



# **The HP-IB In HP 1000 Computer Systems**

**User's Manual**

---

**Software Technology Division  
11000 Wolfe Road  
Cupertino, CA 95014-9804**

## NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company.

### RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARs 252.227.7013.

Copyright © 1983, 1984, 1992 by Hewlett-Packard Company

# Printing History

The Printing History below identifies the edition of this manual and any updates that are included. Periodically, update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this printing history page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past updates; however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all updates.

|                       |                |                                |
|-----------------------|----------------|--------------------------------|
| Seventh Edition ..... | Dec 1983 ..... |                                |
| Update 1 .....        | Oct 1984 ..... | Manual Errata                  |
| Reprint .....         | Oct 1984 ..... | Update 1 has been incorporated |
| Eighth Edition .....  | Dec 1992 ..... | Rev. 6000                      |

# Preface

This user's manual describes the Hewlett-Packard Interface Bus (HP-IB), and how to use it in an HP 1000 Computer System. The HP-IB is the Hewlett-Packard implementation of IEEE-488-1975, Digital Interface for Programmable Instrumentation.

The HP-IB is supported by HP 1000 Computer systems with the RTE-A or RTE-6/VM Operating Systems. It can be programmed in BASIC/1000D, BASIC/1000C, FORTRAN, or Macro Assembly language. The first three of these languages are used in this manual.

This manual should be used in combination with the operating system Operator's Guide, the Reference Manual for the programming language that you choose, the Operating and Service Manuals for the bus devices and the bus Interface Card (HP 59310B or HP 12009A), and the HP-IB driver (ID\*37, DVA37) Reference Manual.

# Table of Contents

---

## Chapter 1

### HP-IB and the HP 1000

|  |     |
|--|-----|
| Introduction .....                       | 1-1 |
| A Simple Example: Resistor Testing ..... | 1-2 |
| Languages used with HP-IB .....          | 1-3 |
| Programmable Devices .....               | 1-3 |
| Functional Description .....             | 1-3 |
| Addresses .....                          | 1-4 |
| Specifications .....                     | 1-4 |
| Bus Structure .....                      | 1-5 |
| Data Lines .....                         | 1-5 |
| Handshake Lines .....                    | 1-6 |
| Bus Management Lines .....               | 1-7 |

## Chapter 2

### Installing the HP 1000 HP-IB System

|   |     |
|---|-----|
| Introduction .....                          | 2-1 |
| HP 59310B Interface Card .....              | 2-1 |
| HP 12009A Interface Card .....              | 2-3 |
| Address .....                               | 2-3 |
| System Control and Data Settling Time ..... | 2-3 |
| Select Code .....                           | 2-3 |
| Device Addresses .....                      | 2-5 |
| Dual-Address Devices .....                  | 2-5 |
| Addressable Mode Devices .....              | 2-6 |
| Address Assignment Table .....              | 2-6 |
| Assembling the Bus .....                    | 2-7 |
| Cable Length Restrictions .....             | 2-8 |
| Connector Hardware .....                    | 2-8 |
| Making the Connections .....                | 2-9 |

## Chapter 3

### Logical Unit Numbers and the HP-IB

|  |     |
|--|-----|
| Introduction .....   | 3-1 |
| LU Numbers and Device Addresses .....                                | 3-1 |
| LUs and Auto vs Direct Addressing .....                              | 3-3 |
| LU Assignment for RTE-6/VM .....                                     | 3-4 |
| Assign or Reassign LU – The LU Command .....                         | 3-4 |
| Examine LUs – The LUPRN Utility .....                                | 3-4 |
| Examine Assignment – The SL Command .....                            | 3-5 |
| Check Status – The EQ Command .....                                  | 3-5 |
| WELCOM File Modification .....                                       | 3-6 |
| LU Assignment for RTE-A .....  | 3-6 |
| Examine Assignments and Request Device Status – The IO Command ..... | 3-6 |
| LU Configuration Information .....                                   | 3-6 |
| Device Status .....  | 3-8 |
| Change DP#1 (Bus Address) – The CN Command .....                     | 3-9 |

## Chapter 4 Bus Messages

|  |      |
|--|------|
| Introduction .....                                   | 4-1  |
| Bus Addresses .....                                  | 4-1  |
| Automatic Addressing .....                           | 4-2  |
| Secondary Addressing .....                           | 4-3  |
| BASIC/1000D .....                                    | 4-3  |
| FORTRAN 7X .....                                     | 4-4  |
| BASIC/1000C .....                                    | 4-4  |
| Direct Addressing .....                              | 4-4  |
| Bus Commands .....                                   | 4-5  |
| Addressed Commands .....                             | 4-6  |
| GTL – Go To Local .....                              | 4-6  |
| SDC – Selected Device Clear .....                    | 4-6  |
| PPC – Parallel Poll Configure .....                  | 4-6  |
| GET – Group Execute Trigger .....                    | 4-7  |
| Universal Commands on the DIO Lines .....            | 4-8  |
| LLO – Local Lockout .....                            | 4-8  |
| DCL – Device Clear .....                             | 4-8  |
| PPU – Parallel Poll Unconfigure .....                | 4-9  |
| SPE/SPD – Serial Poll Enable/Disable .....           | 4-9  |
| Universal Commands on the Bus Management Lines ..... | 4-9  |
| EOI and ATN – Initiate Parallel Poll .....           | 4-9  |
| REN – Remote Enable .....                            | 4-10 |
| SRQ – Service Request .....                          | 4-10 |
| ATN – The Command Mode .....                         | 4-11 |
| IFC – Interface Clear .....                          | 4-11 |
| Data Messages .....                                  | 4-11 |
| HP-IB Library Subroutines .....                      | 4-12 |
| Syntax Conventions .....                             | 4-12 |
| Device Control Subroutines .....                     | 4-14 |
| CLEAR .....  | 4-15 |
| RMOTE .....  | 4-15 |
| GTL .....  | 4-16 |
| LLO .....  | 4-16 |
| LOCL .....   | 4-17 |
| TRIGR .....  | 4-17 |
| ABRT .....   | 4-18 |
| Device Communication Subroutines .....               | 4-19 |
| CMDW/CMDR .....                                      | 4-20 |
| SECW/SECR/SECWR/SECRR .....                          | 4-22 |
| READ/ENTER/INPUT/WRITE/PRINT .....                   | 4-22 |
| IOCNT .....  | 4-23 |

## Chapter 5 Managing Service Requests

|   |     |
|---|-----|
| Introduction .....                            | 5-1 |
| Interrupt and Device Status Subroutines ..... | 5-1 |
| SRQ .....                                     | 5-2 |
| SRQSN .....                                   | 5-3 |
| PPSCH .....                                   | 5-4 |

|   |      |
|---|------|
| PPSN .....                                  | 5-4  |
| STATS .....                                 | 5-5  |
| BSTAT .....                                 | 5-5  |
| GETINTR .....                               | 5-6  |
| PPOLL .....                                 | 5-6  |
| Configure One Device .....                  | 5-6  |
| Unconfigure Selected Devices .....          | 5-7  |
| Unconfigure All Devices .....               | 5-7  |
| PSTAT .....                                 | 5-8  |
| Polling Types .....                         | 5-8  |
| The Serial Poll .....                       | 5-9  |
| Automatic Serial Polling .....              | 5-9  |
| Serial Polling by STATS .....               | 5-10 |
| The Parallel Poll .....                     | 5-10 |
| Automatic Parallel Poll Response .....      | 5-11 |
| Parallel Polling by PSTAT .....             | 5-12 |
| Service Requests in BASIC .....             | 5-12 |
| Serial Polling and Service Programs .....   | 5-12 |
| Bus Service Program .....                   | 5-14 |
| Disabling SRQ Response .....                | 5-15 |
| Parallel Polling and Service Programs ..... | 5-16 |
| Disabling Parallel Poll Response .....      | 5-18 |
| Serial Polling in BASIC/1000D .....         | 5-18 |
| Serial Polling in BASIC/1000C .....         | 5-19 |
| Parallel Polling in BASIC/1000D .....       | 5-20 |
| Service Requests in FORTRAN .....           | 5-21 |
| Serial Polling Response .....               | 5-21 |
| Parallel Polling Response .....             | 5-22 |

## Chapter 6 Error Processing

|  |     |
|--|-----|
| Introduction .....                             | 6-1 |
| Error Processing by the Program .....          | 6-1 |
| Error Detection by the System .....            | 6-2 |
| Device Timeout or Transmission Error .....     | 6-2 |
| Illegal Interrupt .....                        | 6-3 |
| Data Transfer Aborted by SRQ Message .....     | 6-4 |
| Specified Program Does Not Exist .....         | 6-4 |
| Equipment Table Extension Overflow .....       | 6-4 |
| IFT Space Too Small for Poll Table Entry ..... | 6-4 |
| Error Processing Example .....                 | 6-5 |

## Chapter 7 Controller Configuration

|  |     |
|--|-----|
| The Configuration Word .....                     | 7-1 |
| Service Requests – The S-Bit .....               | 7-2 |
| Service Requests – The R-Bit .....               | 7-2 |
| Transfer Methods – The D-Bit .....               | 7-3 |
| When to Use the Interrupt Method .....           | 7-3 |
| When to Use DMA .....                            | 7-4 |
| End of Record – The I-, J-, O-, and P-Bits ..... | 7-4 |

|                                    |     |
|------------------------------------|-----|
| Device EOR Requirements .....      | 7-4 |
| Controller EOR Requirements .....  | 7-5 |
| Error Processing – The E-Bit ..... | 7-5 |
| Using the CNFG Subroutine .....    | 7-6 |
| Using CNFG Under RTE-A .....       | 7-6 |
| Using CNFG Under RTE-6/VM .....    | 7-7 |

## **Appendix A RTE-6/VM Generation**

|                                    |     |
|------------------------------------|-----|
| Introduction .....                 | A-1 |
| Generation Procedure .....         | A-2 |
| Program Input Phase .....          | A-2 |
| Parameter Input Phase .....        | A-2 |
| Table Generation Phase .....       | A-2 |
| Equipment Table (EQT) .....        | A-3 |
| Device Reference Table (DRT) ..... | A-4 |
| Interrupt Table .....              | A-4 |
| RTE-6/VM BASIC/1000D .....         | A-5 |

## **Appendix B RTE-A Generation**

|                              |     |
|------------------------------|-----|
| Introduction .....           | B-1 |
| Generation Procedure .....   | B-2 |
| Program Input Phase .....    | B-2 |
| Table Generation Phase ..... | B-2 |
| Interface Table IFT .....    | B-2 |
| Device Table DVT .....       | B-3 |

## **Appendix C Quick Reference Tables**

## **Appendix D Bus Service Program: Example**



## List of Illustrations

|            |  |      |
|------------|--|------|
| Figure 1-1 | HP 1000 System with up to 14 Devices on the HP-IB .....                    | 1-1  |
| Figure 1-2 | Example Control and Testing System with the HP-IB .....                    | 1-2  |
| Figure 1-3 | Control and Data Lines in the HP-IB .....                                  | 1-5  |
| Figure 1-4 | HP-IB Data Transfers – Bit-Parallel, Byte-Serial .....                     | 1-6  |
| Figure 1-5 | Data Byte Transfer Handshake Sequence .....                                | 1-8  |
| Figure 2-1 | The HP 59310 Set-up Switches .....   | 2-2  |
| Figure 2-2 | The HP 12009A U1 DIP Switches .....  | 2-4  |
| Figure 2-3 | Example Device Address Switches on the Rear Panel of an HP-IB Device ..... | 2-6  |
| Figure 2-4 | Address Bits and the Address Switches .....                                | 2-6  |
| Figure 2-5 | Sample Address Assignment Table .....                                      | 2-7  |
| Figure 2-6 | HP-IB Standard Connectors .....  | 2-9  |
| Figure 2-7 | Stacked HP-IB Connectors for Multiple Connections .....                    | 2-10 |
| Figure 3-1 | Bus Hardware/Software Relationship for RTE-6/VM .....                      | 3-2  |
| Figure 3-2 | Bus Hardware/Software Relationship for RTE-A .....                         | 3-3  |
| Figure 3-3 | Excerpt from WELCOM File with Two HP-IBs .....                             | 3-6  |
| Figure 4-1 | Bus Traffic during a Serial Poll .....                                     | 4-9  |
| Figure 4-2 | BASIC/1000D-Compatible String .....  | 4-21 |
| Figure 7-1 | Configuration Word Format with Default Value .....                         | 7-1  |

## Tables

|           |   |      |
|-----------|---|------|
| Table 2-1 | Normal REN, IFC, SHIELD, and PPID Settings for the HP 59310B ....     | 2-2  |
| Table 2-2 | Functions and Normal Settings of the U1 Switches for the HP 12009A .. | 2-4  |
| Table 2-3 | Available HP-IB Cables .....  | 2-5  |
| Table 4-1 | Addressed Commands .....  | 4-6  |
| Table 4-2 | PPE Secondary Command Codes .....                                     | 4-7  |
| Table 4-3 | Universal Commands Transmitted on the DIO Lines .....                 | 4-8  |
| Table 4-4 | Universal Commands Transmitted on the Bus Management Lines .....      | 4-10 |
| Table 4-5 | Device Control Subroutines and Statements .....                       | 4-14 |
| Table 4-6 | Statements and Subroutines to Send and Receive Bus Data .....         | 4-19 |
| Table 5-1 | Interrupt and Service Subroutines and Statements .....                | 5-2  |
| Table 5-2 | Polling Types and Service Language Options in BASIC .....             | 5-13 |
| Table 5-3 | Status Codes for Bus LU .....   | 5-15 |
| Table 7-1 | The I-, J-, O-, and P-Bits – Combinations .....                       | 7-5  |
| Table A-1 | RTE-6/VM HP-IB Software .....   | A-1  |
| Table B-1 | RTE-A HP-IB Software .....  | B-1  |
| Table C-1 | Device Addresses: Binary, Character, Octal, and Decimal .....         | C-1  |
| Table C-2 | ASCII Codes of HP-IB Addresses and Commands .....                     | C-2  |
| Table C-3 | Electrical Characteristics .....                                      | C-3  |

# HP-IB and the HP 1000

## Introduction

The options and accessories of HP 1000 computer systems make them particularly suitable for real-time automatic testing, measurement, and control. These accessories and options include not only the Real-Time Executive operating systems, but also the Hewlett-Packard Interface Bus, a flexible instrument interface that supports a wide range of instruments and peripherals.

In an HP 1000 system, the HP-IB permits two-way communications for up to fourteen devices and a computer, which acts as the system controller. The number of devices on a bus is restricted to fourteen, but an HP 1000 can control several buses at a time, so HP 1000 HP-IB systems can be expanded if necessary. Figure 1-1 shows an HP 1000 system controlling several HP-IB devices.



Figure 1-1. HP 1000 System with up to 14 Devices on the HP-IB

## A Simple Example: Resistor Testing

Figure 1-2 shows an HP 1000 system with an HP-IB performing precision resistor testing. A control program sends data over the bus to step a scanner through the test resistors in a batch. A digital voltmeter, programmed via the bus, returns the measurements to the HP 1000. The bus carries address, command, and data messages between the controller and the instruments.

The following summarizes the actions of the program:

1. The HP 1000 controller tells the voltmeter to switch to remote control. In this mode, the front panel controls are disabled, and the voltmeter can be controlled via the bus. The controller tells the voltmeter to listen for programming information on the bus, and sends a sequence of characters and numbers to the voltmeter to select the resistance measurement function, the measurement range, and to set it to take readings in response to a trigger from the controller. The controller tells the voltmeter to stop listening.
2. The controller tells the scanner to listen and sends codes which tell it to select a channel.
3. The controller tells the voltmeter to listen, then triggers it, telling it to talk to return the reading.
4. Steps 2 and 3 repeat for the rest of the resistors in the batch.

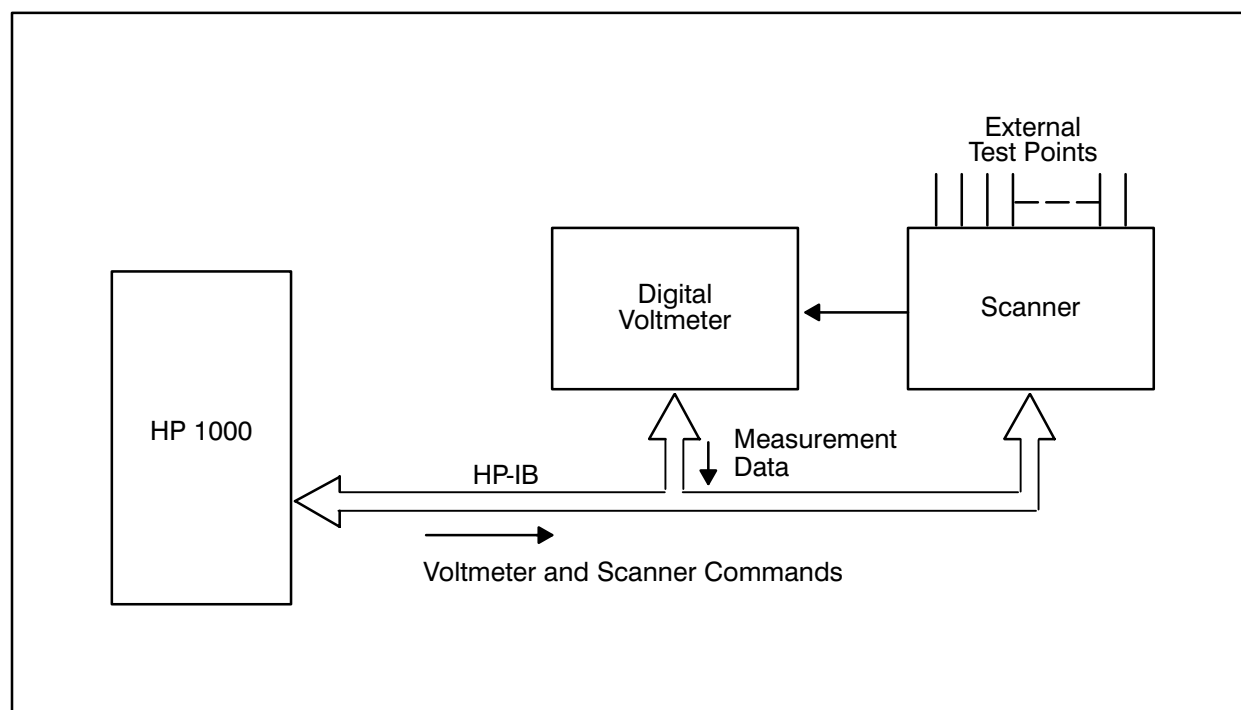


Figure 1-2. Example Control and Testing System with the HP-IB

## Languages used with HP-IB

Three programming languages are commonly used with the HP-IB on an HP 1000: BASIC, FORTRAN, and assembly language. The examples in this manual are in FORTRAN, BASIC/1000D, and BASIC/1000C. Wherever the term BASIC is used, without specifying BASIC/1000D or BASIC/1000C, both languages apply, as in the first sentence of this paragraph. Assembly language HP-IB programming is described in the HP-IB driver manual.

## Programmable Devices

The scanner and the voltmeter in Figure 1-2 are programmable HP-IB devices. Both accept strings of characters which tell them what to do next. If the voltmeter in the example was an HP 3455A DVM, then the programming string might be:

|          |                          |
|----------|--------------------------|
| F1R1M3T2 | F1 = Function 1          |
|          | 2-wire K-ohms            |
|          | R1 = 0.1                 |
|          | M3 = Math functions off  |
|          | T2 = External triggering |

Many HP-IB instruments are programmable by such strings. Many also have remote/local switching that permits the controller to disable the front panel of the device, so it can be controlled via the bus. Some instruments are remote-only devices, with no front panel controls, like the HP 3495A scanner. The instrument operating manuals describe which of the HP-IB functions each device uses.

## Functional Description

The HP-IB itself is passive; all activity on the bus (called bus traffic) is generated by the controller and bus devices. Every device on the bus must be able to perform at least one of the following roles:

- **Talker** – A device that can transmit data over the interface when addressed. Typical talkers are voltmeters returning a reading, tape readers, or disks being read. There can be only one active talker on the bus at a time.
- **Listener** – A device that can receive data from the bus when addressed. Typical listeners are printers, numeric displays, or disks receiving data. There can be up to 14 listeners on the bus.
- **Talker-Listener** – A device that can perform either of the functions just described. Measurement instruments are usually talker-listeners, so they can receive programming strings and transmit measurement data.

- **Controller** – The HP 1000 is the only controller permitted on an HP 1000 HP-IB system. The controller monitors and manages the bus traffic by specifying the talker and listeners for each set of transmissions, by programming the devices on the bus for an application, and by processing service requests.

## Addresses

Each device on the HP-IB, including the controller, has a five-bit device address. The controller uses the addresses to designate talkers and listeners for bus traffic. The device addresses are set by switches on the back of most instruments. A few instruments have the switches inside.

Address 00B is reserved for the E/F-Series controller (RTE-6/VM), 36B is reserved for the A-Series controller (RTE-A), and address 37B is reserved for the UNTALK and UNLISTEN addresses described in the following paragraphs.

Each five-bit device address has two ASCII characters associated with it; they specify the device talk and listen addresses. The controller addresses a device to talk or listen by sending one of its ASCII address characters over the bus. The lower five bits of the character specify the device being addressed, and bits six and seven tell the device to talk or listen.

The talk and listen address characters for address 37B do not address any device to talk or listen. On the contrary, they are the untalk and unlisten addresses: untalk unaddresses any talker on the bus; unlisten unaddresses any listeners. The devices continue to monitor the data and management lines for commands, but they do not transmit or listen for data or programming strings.

Some HP-IB instruments have such complex I/O capacities that the talk and listen addresses are not enough. These instruments have secondary (extended) addresses. Secondary addresses are used to access a subsystem or register within the device. Some devices contain several internal data memories that can be accessed by secondary addresses. The presence and function of secondary addresses are device dependent.

## Specifications

The HP-IB is Hewlett-Packard's implementation of IEEE-488-1978, Digital Interface for Programmable Instrumentation. (The HP 59310B is HP's implementation of IEEE-488-1975.) Any device that conforms to the IEEE standard is compatible with the HP-IB. Devices specifically designated as "Designed For HP-IB Systems" respond as described in this User's Guide. Others respond as described in the standard. The key HP-IB specifications are summarized below:

- **Interconnected Devices** – Up to 14 instruments plus a controller per bus.
- **Network Length** – Up to 20 meters of cable, but no more than two meters per device (average).
- **Signal Lines** – 16 active lines: eight data lines and eight handshaking and bus management lines.

- **Transfer Method** – Byte-serial, bit-parallel, asynchronous data transfer, with a three-wire handshake.
- **Maximum Data Rate** – One megabyte per second over limited path, 250 to 500 kilobytes per second over the full path. The actual rate is determined by the talking and listening devices.
- **Address Capability** – 31 talk and 31 listen addresses, and 31 secondary addresses per talk and listen address, for a total of 961 secondary talk and 961 secondary listen addresses.

## Bus Structure

The interface bus uses 16 lines to connect the bus devices in parallel. Figure 1-3 shows the active control and data lines in the bus. The data lines, labeled DIO1 to DIO8, carry bus commands, device programming strings, and data in byte-serial, bit-parallel format. The DAV, NRFD, and NDAC handshake lines coordinate the movement of data. The remaining five lines are bus management lines. All of the lines are described in the following paragraphs.

## Data Lines

An eight-bit bidirectional bus is used to transfer data and commands between devices on the bus. Normally, the information on the bus is coded in ASCII, with the eighth bit available for parity. The bus standard does not specify the code format. The controller can send commands and data to devices, devices can return data to the controller, or devices can send data to other devices. Again, the standard does not specify the type of information that passes over the bus; it depends on the application and the devices chosen to implement it.

Figure 1-4 shows the byte-serial bit-parallel data transfer scheme. The transfer of the three-byte sequence “BUS” would occur as shown over the data lines.

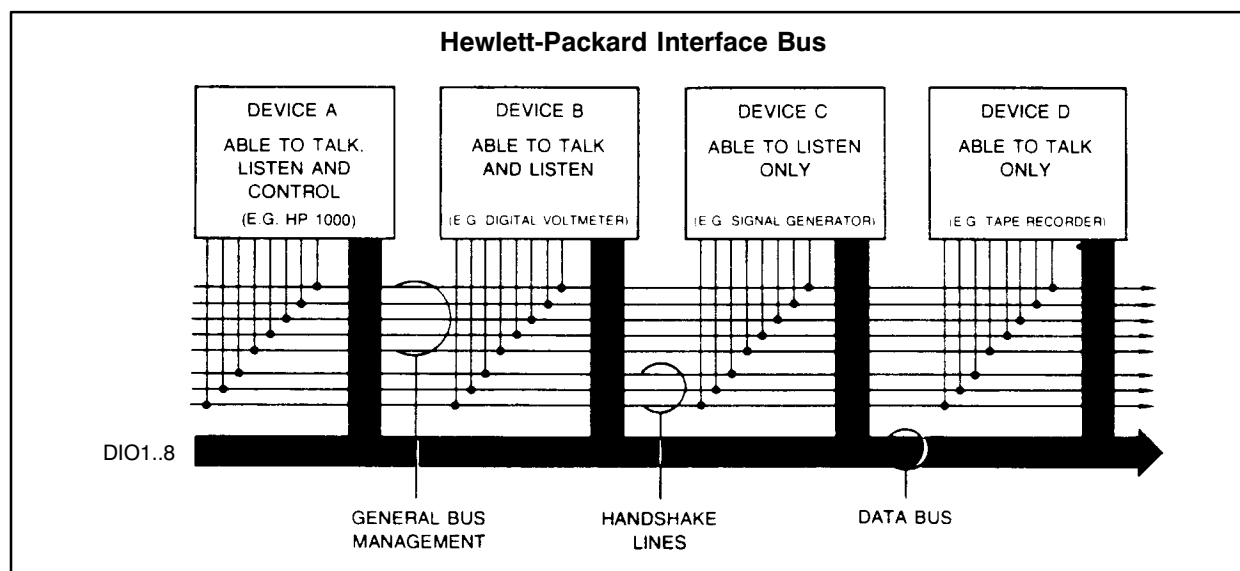


Figure 1-3. Control and Data Lines in the HP-IB

## Handshake Lines

Three lines are used to coordinate the transfer of data from a talker or controller (data source) to one or more listeners on the bus. There is only one talker at a time on the bus, and up to 14 other devices can listen. The handshake lines make sure that all devices have received each message before the next is sent. The three-wire handshake has the following characteristics:

- The data transfer is asynchronous, with the rate of transfers controlled by the speed of the slowest listener on the bus.
- More than one device can accept data at the same time.
- A handshake cycle occurs for every byte transferred.
- The handshake lines are:

**DAV – Data Valid.** This line is controlled by the data source (controller or addressed talker). It indicates that the data on the bus is valid and devices can begin to accept it.

**NRFD – Not Ready For Data.** This line is controlled by all addressed listeners or devices receiving bus commands. Indicates that the device is busy and cannot accept data. The device sets NRFD false when it is ready to receive data.

**NDAC – Not Data Accepted.** This line is controlled by all addressed listeners or devices receiving interface commands. Indicates that the device has not yet accepted the data on the bus. The device sets NDAC false when it accepts the data.

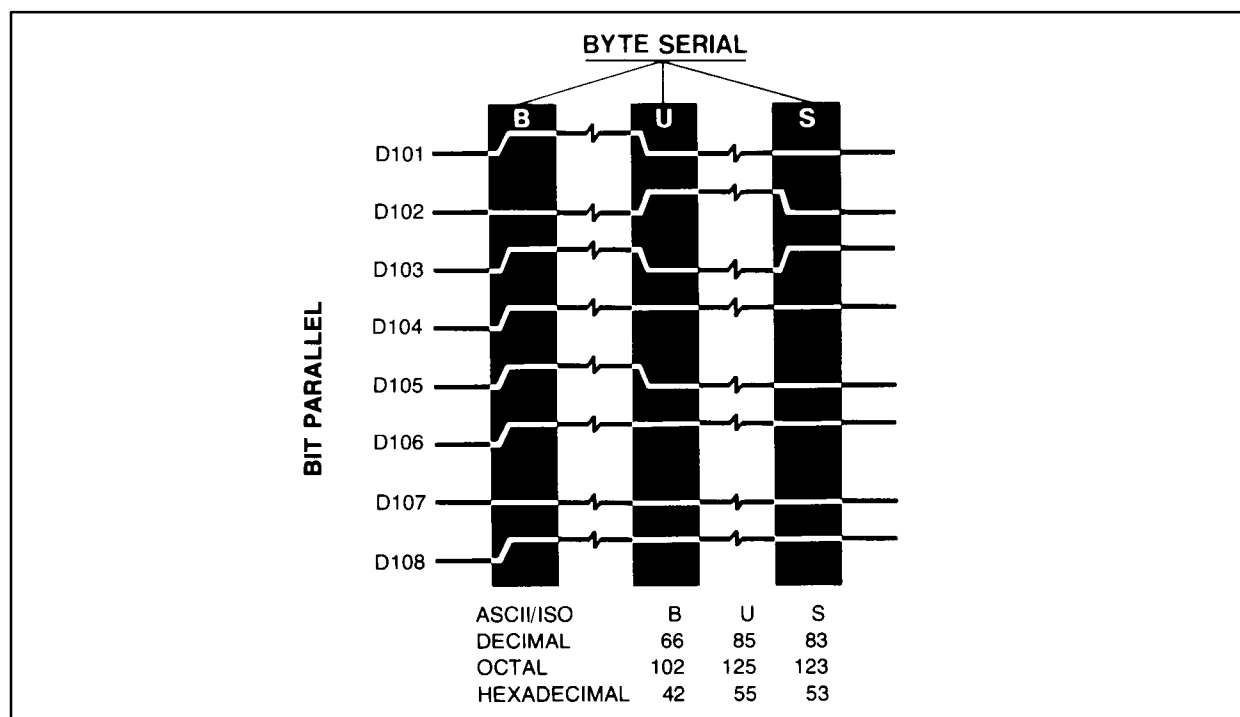


Figure 1-4. HP-IB Data Transfers – Bit-Parallel, Byte-Serial

---

**Note**            The logic sense of the HP-IB signal lines is low-true to implement the wired-OR convention of NRFD and NDAC and for reduced noise susceptibility.

---

## Bus Management Lines

There are five bus management lines. Three are controlled only by the HP 1000 controller. They are ATN, REN, and IFC. The others are used by any of the devices on the bus. They are EOI and SRQ. The bus management lines are explained below:

- **ATN – Attention.** The controller asserts the attention line to put the bus in the command mode. When ATN is asserted, the bus devices expect commands or address bytes on the bus. All instruments must monitor ATN so that the controller can get their attention to send commands and addresses. When the ATN line is not asserted, the bus is in the data mode; that is, all listening devices expect data on the bus.
- **REN – Remote Enable.** The controller asserts the Remote Enable line to permit instruments with remote/local switching to be set for remote programming. The remote/local devices switch to remote mode when they are addressed while REN is true. When REN is false, they return to the local mode. All instruments with remote/local switching must monitor REN.
- **IFC – Interface Clear.** The controller asserts the Interface Clear line to halt all activity on the bus. All devices must monitor IFC. Asserting IFC returns the bus to a predetermined state.
- **SRQ – Service Request.** At any time during bus operation, any device can assert the Service Request line to alert the controller that it requires service. SRQ can be asserted as the result of an error condition or as a normal interrupt in response to the test in progress. The system response to SRQ depends largely upon the controlling program. SRQ can be asserted at any time except when IFC is asserted.
- **EOI – End or Identify.** When the bus is in the data mode (ATN false) and a device or the controller is sending data on the bus, the End or Identify line is asserted to signal the end of the transmission. The active talker controls the EOI line. Not all devices require or use EOI. The timing of the assertion of EOI can vary.

When the bus is in the command mode (ATN asserted), the controller can assert EOI to initiate a parallel poll. Devices that are configured for parallel polling (as described in Chapter 5) can respond to the poll.



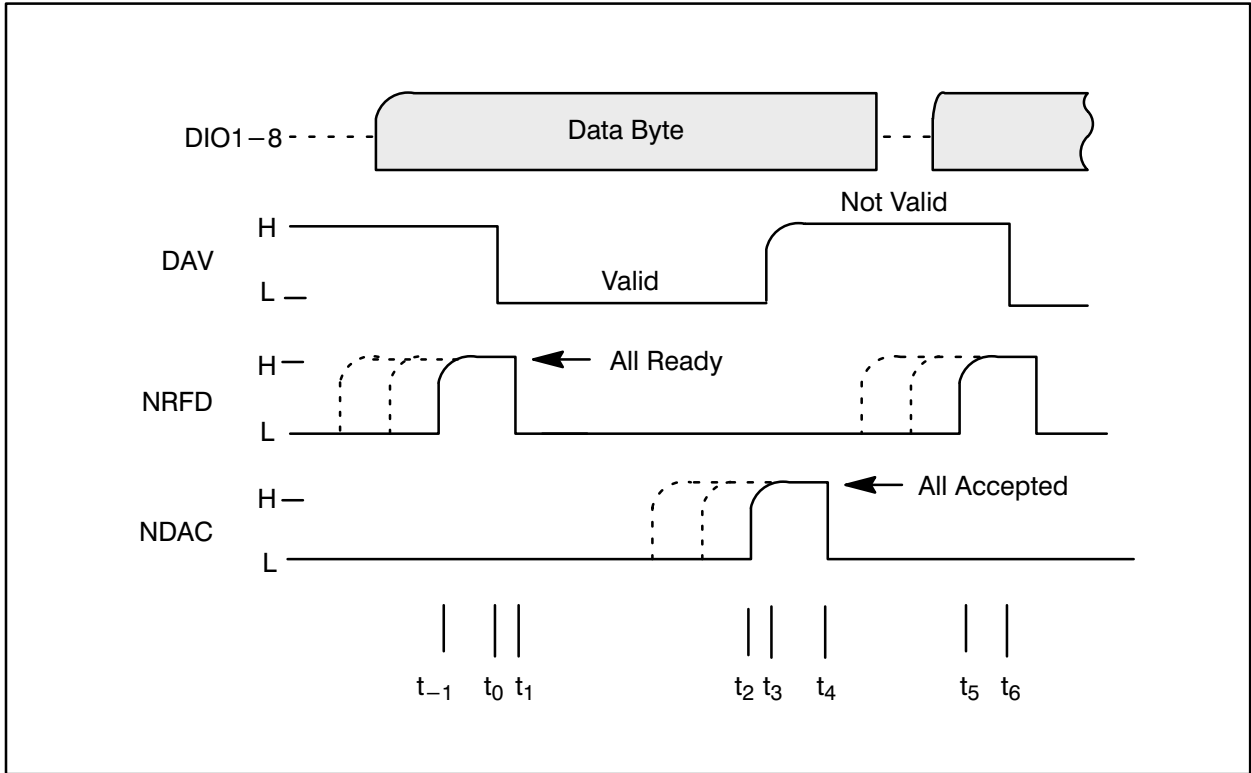


Figure 1-5. Data Byte Transfer Handshake Sequence

# Installing the HP 1000 HP-IB System

---

## Introduction

There are three steps to installing an HP-IB in an HP 1000 system: setting and installing the interface card, selecting and setting the device addresses, and assembling the bus. This chapter is divided according to those steps. Because M/E/F-Series computer systems use the HP 59310B Interface Card and A-Series systems use the HP 12009A card, the interface card installation section of this chapter is divided. The device address and assembly instructions are common to all systems.

## HP 59310B Interface Card

The HP 59310B is the HP-IB interface card for M/E/F-Series HP 1000 computer systems. Two DIP switches are used to set the HP 59310B card for operation. The card address and three operating parameters are set by DIP switch SW2, located near the card's lower extractor handle, as shown in Figure 2-1.

The SW2 switches labelled "1 2 3 4 5" select the interface card address. The address is always 00B, so set all five address switches to "OPEN."

Table 2-1 shows the function and normal settings of the REN, IFC, and SHIELD switches. REN should be set to "REN", and IFC to "ON," to enable the controller to assert the REN and IFC bus management lines. The SHIELD switch should be set to "CNX."

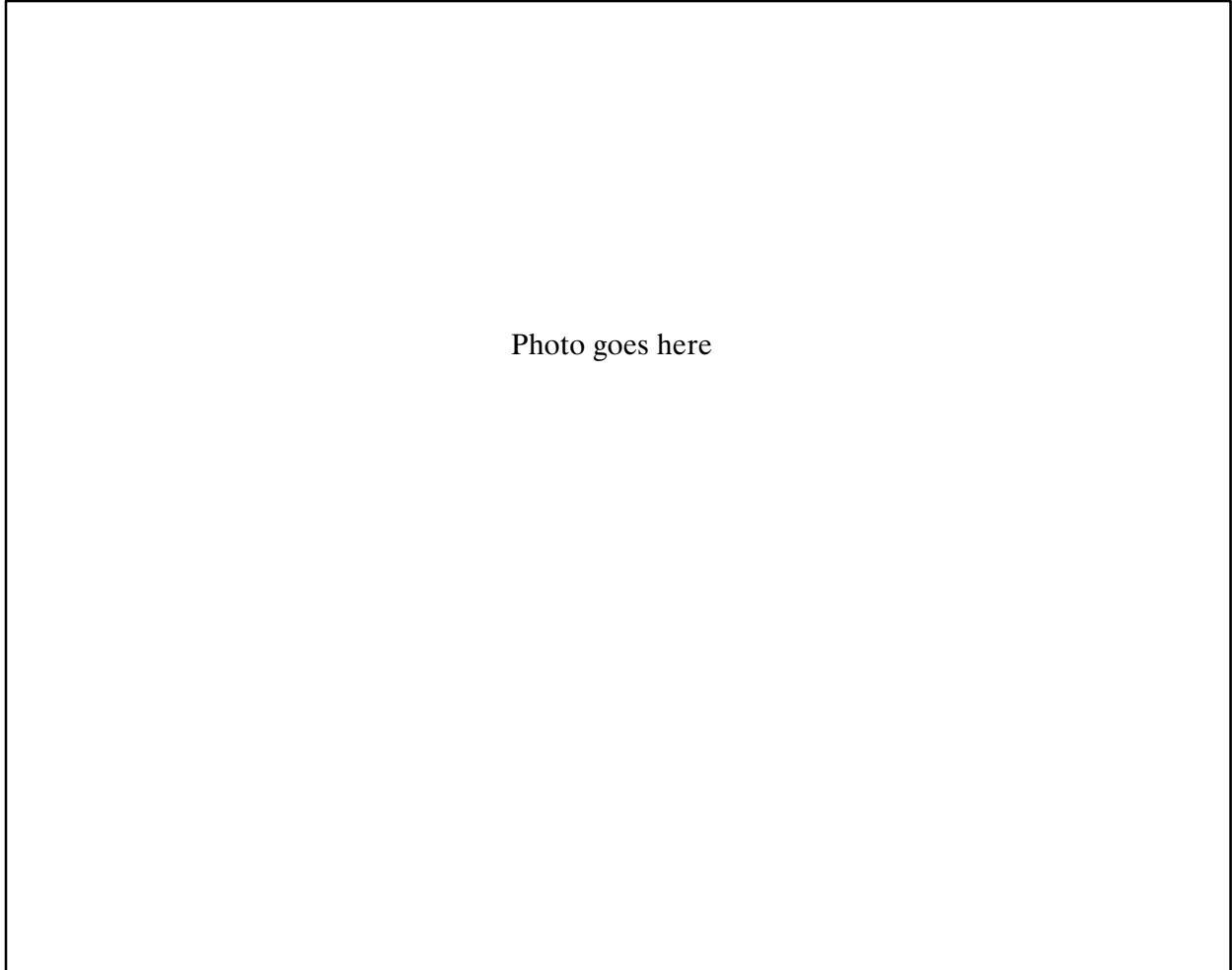
---

**Note** If any other device on the bus has a SHIELD switch, it must be left open. Only the controller's SHIELD switch should be closed.

---

The PPID switches (SW1) are located near the card's upper extractor handle. They are included in Table 2-1. The PPID switches must all be set to "OFF," because the HP 59310B card is always the controller. The PPID switches select which DIO line the HP 59310B uses to respond to parallel polling. As the controller, the card initiates parallel polls; it does not respond to them.

Turn the computer power off and install the card in one of the I/O slots. The location of the I/O slots varies between models, so refer to the *Computer Installation and Service Manual* for your computer for more information.



**Figure 2-1. The HP 59310 Set-up Switches**

**Table 2-1. Normal REN, IFC, SHIELD, and PPID Settings for the HP 59310B**

| <b>Switch</b> | <b>Function</b>   | <b>Normal</b> |
|---------------|---|---------------|
| REN           | ON enables the card to drive bus signal REN.  | REN           |
| IFC           | ON enables the card to drive bus signal IFC.  | ON            |
| SHIELD        | CNX connects the shield to logic circuit common.  | CNX           |
| PPID          | Selects one of eight status bits the card returns in response to a parallel poll. (Not used.) | All OFF       |

# HP 12009A Interface Card

The HP 12009A is the interface card for A-Series HP 1000 computers. The card uses two LSI chips to control communication between the HP 1000 and the bus: the I/O processor (IOP) and the Processor to HP-IB interface chip. The IOP chip is found on all A-Series interface cards. It manages I/O operations to free the processor for other tasks. The HP-IB interface chip performs the conversion and control functions required to let the IOP communicate with the bus.

## Address

The address of the HP 12009A card is set to 36B automatically by the HP-IB driver. There is no need to set the address switches (U16S4 to U16S8). The HP 12009A card is always the system controller, so there is no need to change the settings of U16S1 to U16S3. U16S1 to U16S8 are disabled when U1S1 is open, as described in the following paragraphs.

## System Control and Data Settling Time

DIP switch U1 is located near the extractor handle next to the J1 card edge connector, as shown in Figure 2-2. The functions and normal settings of U1S1 to U1S8 are listed in Table 2-2.

U1S1 must be open to make the HP 12009A card the system controller. When U1S1 is open, the U16S1 to U16S8 are disabled, because the controller address is always 36B, and the functions of U16S1 to U16S3 are meaningless when the HP 12009A card is the system controller.

U1S2 sets the delay between the assertion of data on the bus by the controller and the assertion of DAV. With the switch open, the delay is about 500 nanoseconds. When the switch is closed, the delay is reduced to about 350 nanoseconds. This faster settling time satisfies the IEEE-488-1978 standard for high-speed operation at up to 1 megabyte per second.

---

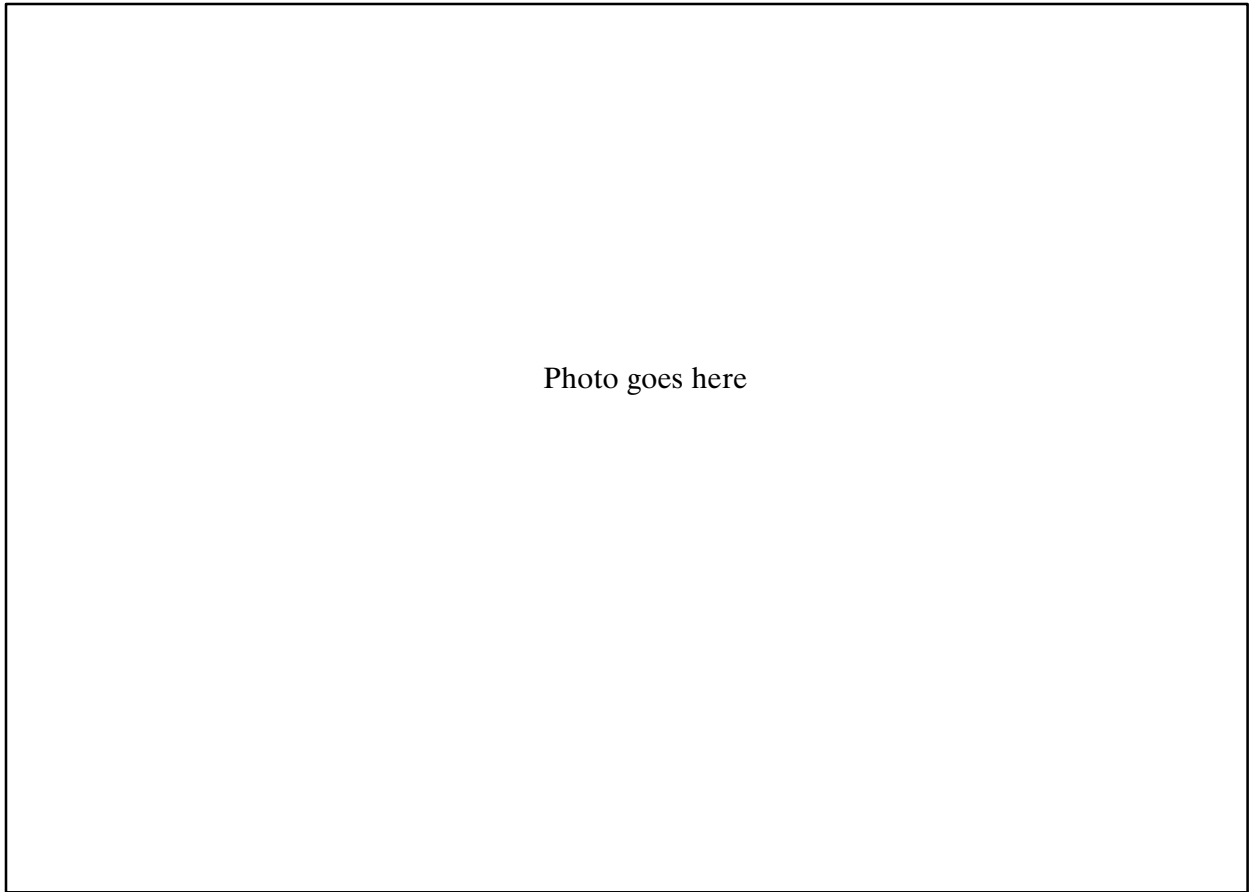
**Note** With the faster settling time, special system restrictions apply. Refer to the *HP 12009A HP-IB Interface Reference Manual*, part number 12009-90001, for a description of the high-speed data cable restrictions.

---

## Select Code

Switches U1S3 through U1S8 specify the select code of the interface card. The select code depends on the system I/O requirements. U1S3 corresponds to the MSB of the select code, and U1S8 to the LSB.

Turn the computer system power off and install the card in an I/O slot of the A-Series computer as described in the *HP 12009A HP-IB Interface Reference Manual*.



**Figure 2-2. The HP 12009A U1 DIP Switches**

**Table 2-2. Functions and Normal Settings of the U1 Switches for the HP 12009A**

| <b>Switch</b>      | <b>Function</b>  | <b>Normal</b> |
|--------------------|--|---------------|
| U1S1               | 0 = 12009A is not the bus controller<br>1 = 12009A is the bus controller (switch open) | 1             |
| U1S2               | 0 = Fast settling time (IEEE 488-1978)<br>1 = Normal settling time (IEEE 488-1975)     | 1             |
| U1S3<br>to<br>U1S8 | 12009A select code   | N/A           |

## Device Addresses

Every device on the bus, including the controller, has a device address. In the last section, you set the HP 59310A controller device address to 00B. The HP 12009A address is automatically set to 36B. In this section, you set the addresses of the devices on the bus.

Most devices have address switches on the rear panel next to the HP-IB connector, as shown in Figure 2-3. The switches select the five-bit device address, and are usually labelled A1 to A5. Refer to the device reference manuals for the number, type, and location of device address switches.

As shown in Figure 2-4, the complete device address is a seven-bit value. Address bits 0 to 4 are selected by the address switches or jumpers labelled A1 to A5. Bits 5 and 6 select the talk and listen addresses. They are set by the controller and represented by ASCII talk and listen address characters.

The positions marked “A” are the device addresses, set by the address switches. For each seven-bit address, there is an ASCII representation that is used to address the device on the bus. The ASCII talk and listen characters for each of the five-bit device addresses are listed in Appendix C.

---

**Note** The device address of disk drives, tape drives, and printers that conform to the “Amigo standard” must be set between 0 and 7. If the address is set above 7, the device cannot respond to parallel polling.

---

## Dual-Address Devices

Some devices have two talk and listen addresses. Dual-address devices have only four address switches, labelled A2 to A5. The A1 position is selected by the controller to make two five-bit addresses. Dual-address devices are addressed to talk and listen by ASCII characters exactly as other devices are. The two addresses give access to two functions in the device, as if there were two related devices in a single unit.

**Table 2-3. Available HP-IB Cables**

| Length    | HP Product Number |
|-----------|-------------------|
| 1 meter   | 10833A            |
| 2 meters  | 10883B            |
| 4 meters  | 10833C            |
| 0.5 meter | 10833D            |

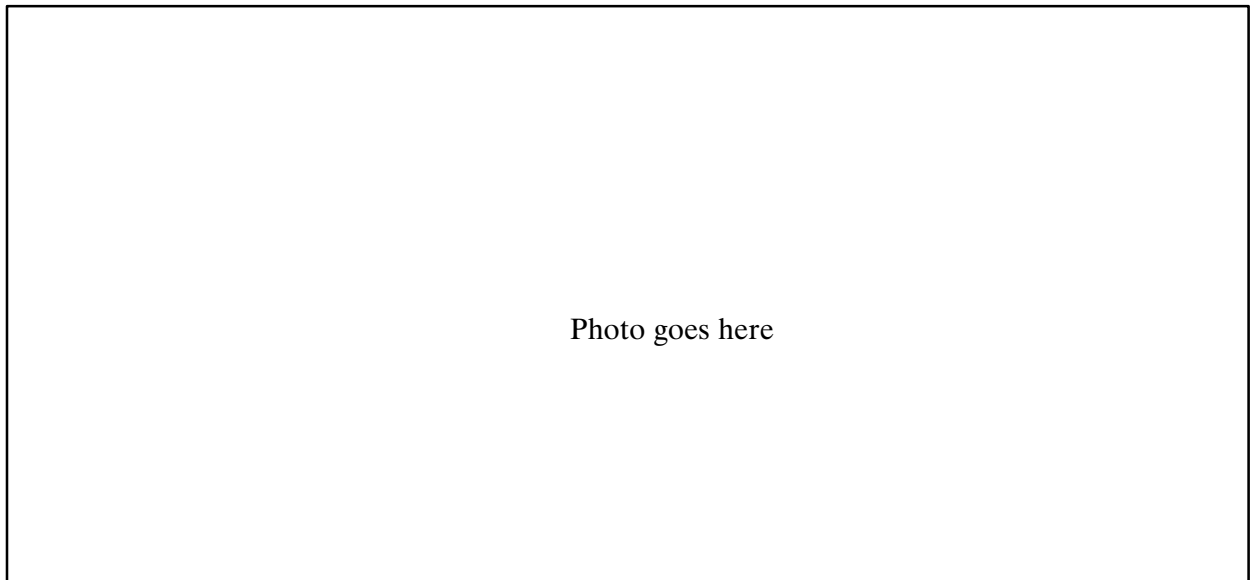
## Addressable Mode Devices

Some devices have a switch, usually on the rear panel, to select the addressable mode or talk- or listen-only mode. For HP-IB systems under HP 1000 control, every device must be addressable, so set the switch to the addressable mode.

## Address Assignment Table

As you set the address for each device in the system, record the addresses for reference as you set up the LUs for the system and during programming. You will need the list of device addresses when you establish LU numbers for the devices in Chapter 3.

Figure 2-5 shows a simple address assignment table with spaces for the device model numbers and names, device addresses, ASCII talk and listen address characters, and the EQT/LU or IFT/DVT/LU assignments.



**Figure 2-3. Example Device Address Switches on the Rear Panel of an HP-IB Device**

| Switch Number  |   |   | A5 | A4 | A3 | A2 | A1 |
|----------------|---|---|----|----|----|----|----|
| Address Bit    | 6 | 5 | 4  | 3  | 2  | 1  | 0  |
| Talk Address   | 1 | 0 | A  | A  | A  | A  | A  |
| Listen Address | 0 | 1 | A  | A  | A  | A  | A  |

**Figure 2-4. Address Bits and the Address Switches**

## Assembling the Bus

You are now ready to connect the devices to the interface card to form the bus. The bus consists of the HP 1000 with an interface card, devices, an interface cable to connect the interface card to one device, and cables to connect the other devices.

Devices are chained together to form the bus. The interface card is connected to one device, which is connected to a second device, and so on. Each device in the chain has two cables connected to it, except the controller and the last device in the chain.

---

**Note** Devices should not be powered up while connected to a working bus. This may cause faulty operation of the bus.

---

---

**Caution** No more than two HP-IB connectors should be stacked on a device. The strain of a stack of three or more connectors can damage the base connector on the device. Ensure that all cables have strain relief to prevent connector damage.

---

| Device | Name | Address<br>A5 A4 A3 A2 A1 | Talk<br>Address | Listen<br>Address | EQT/LU<br>or<br>IFT/DVT/LU |
|--------|------|---------------------------|-----------------|-------------------|----------------------------|
|        |      |                           |                 |                   |                            |
|        |      |                           |                 |                   |                            |
| ⋮      | ⋮    | ⋮                         | ⋮               | ⋮                 | ⋮                          |
| ⋮      | ⋮    | ⋮                         | ⋮               | ⋮                 | ⋮                          |
| ⋮      | ⋮    | ⋮                         | ⋮               | ⋮                 | ⋮                          |

Figure 2-5. Sample Address Assignment Table



## Cable Length Restrictions

The total cable length is restricted to 20 meters for the entire bus, but no more than two meters of cable per device, including the interface card. Four lengths of cable are available for connecting the bus; they are listed in Table 2-3. The restriction of two meters of cable per device is an average. A bus with four instruments and a controller can have up to ten meters of cable. The choice of cables is determined by the mounting requirements.

---

**Note** When using the high-speed data settling time on the HP 12009A Interface Card, shorter cables must be used. Refer to the *HP 12009A HP-IB Interface Reference Manual*, part number 12009-90001, for a description of the high-speed data cabling restrictions.

---

## Connector Hardware

The IEEE-488 standard for HP-IB specifies a 24-pin ANSI MC1.1 connector, as shown on the left in Figure 2-6. The IEC equivalent of IEEE-488 is IEC 625-1. The IEC standard specifies the MIL-C-24308 connector shown on the right in Figure 2-6. It is the same type of connector specified for RS-232.

---

**Caution** Although IEC 625-1 devices use the same connectors as RS-232, they must never be connected to an RS-232 port. Damage to the device can result. Adapters are available for connecting HP-IB cables to IEC 625-1 devices. To prevent mistakes, always use HP-IB cables, with adapters as necessary to connect to IEC 625-1 devices.

---

The electrical specifications of the HP-IB connector and devices are summarized in Appendix C.

---

**Caution** Most HP-IB connectors use black anodized metric fasteners (ISO M3.9x0.6). Some cables manufactured before 1975 have silver colored English fasteners (UNC 6-32). Do not try to mate silver and black fasteners; both can be damaged.

---

## Making the Connections

When you have decided how to mount the instruments, connect the HP-IB cables to the devices as shown in Figure 2-7. Do not use a screwdriver to fasten the connectors together. The screwdriver slot is for removal only, if two fasteners become stuck. Connect one device to the HP 1000 with the cable supplied with the interface card. The HP-IB system is now physically connected to the HP 1000. In the next chapter, the interface is logically connected to the operating system by assigning logical unit (LU) numbers to the bus and its devices.

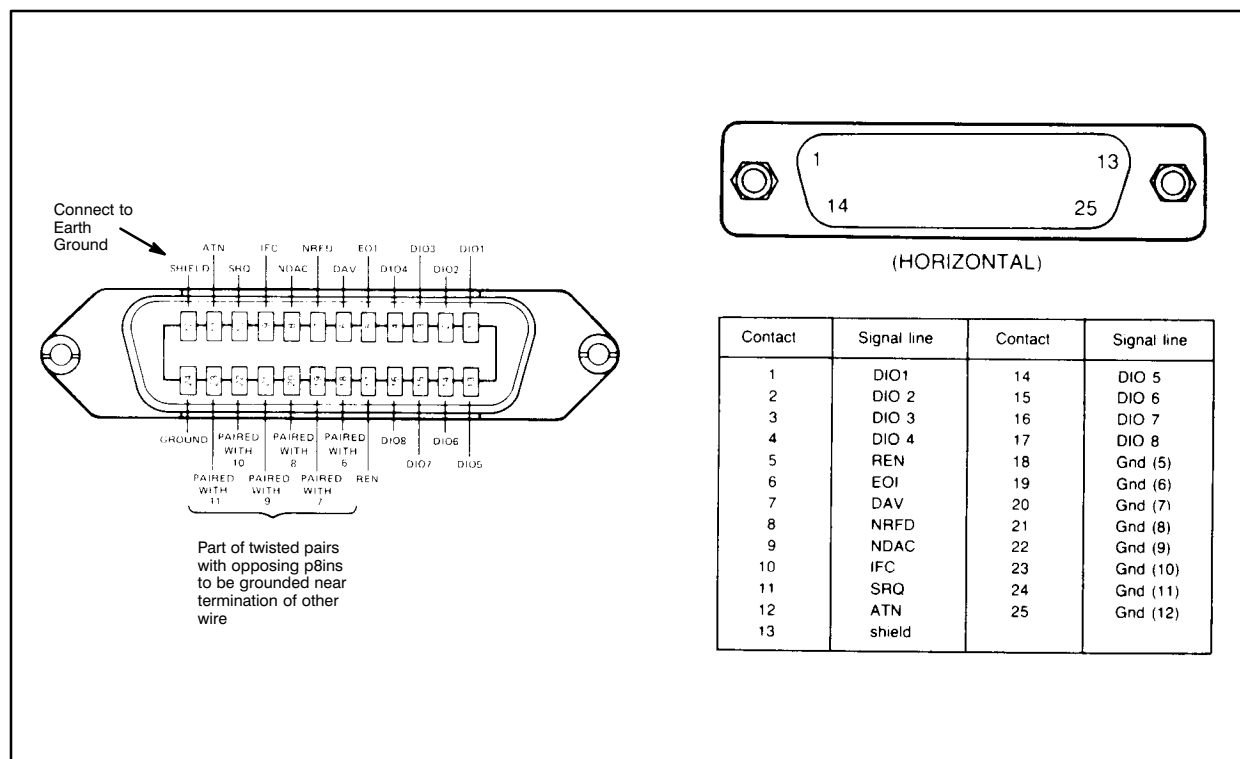
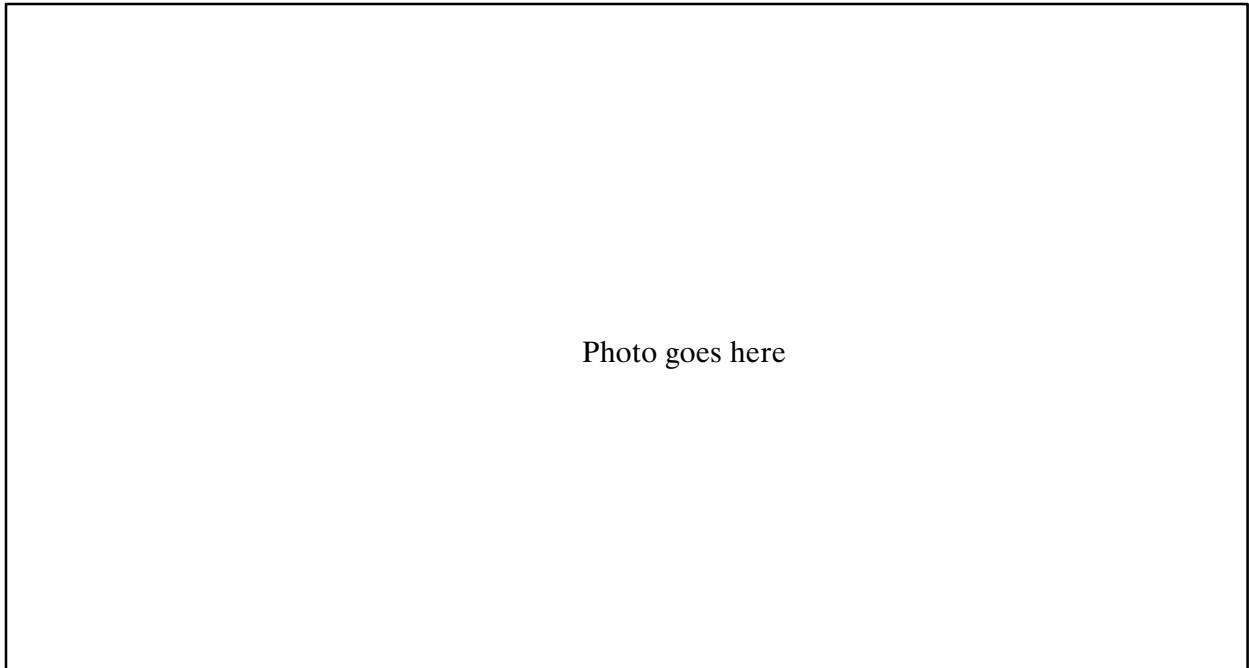


Figure 2-6. HP-IB Standard Connectors



**Figure 2-7. Stacked HP-IB Connectors for Multiple Connections**

# Logical Unit Numbers and the HP-IB

---

## Introduction

This chapter is divided into three main sections. The first describes RTE logical unit numbers (LUs) and their relation to the bus. The second explains how to establish LUs for the bus and its devices on an E/F-Series computer with the RTE-6/VM operating system. The third explains how to establish LUs on an A-Series computer with RTE-A.

## LU Numbers and Device Addresses

Every I/O device in an HP 1000 system has a logical unit (LU) number. In an E/F-Series system with RTE-6/VM, each LU is associated with an equipment table (EQT) entry. Several LUs may be associated with the same EQT entry. In an A-Series system with RTE-A, each LU is associated with a device table (DVT). There is one LU per DVT. Several DVTs may be associated with one interface table (IFT).

The HP-IB is a device with subchannels. For RTE-6/VM systems, there is an EQT entry for the bus interface card, with an LU assigned to it. The interface card EQT entry has subchannel entries for the bus devices, with an LU assigned for each device. The EQT subchannel numbers match the HP-IB device addresses.

For RTE-A systems, there is a DVT for the interface card and one for each device on the bus. Every DVT has an assigned LU number, and all the DVTs are associated with a single IFT for the bus. The first driver parameter (DP#1) for each bus device DVT specifies the HP-IB device address.

There is one LU number per device address on the bus, plus a bus LU. Device LU numbers generally do not match device addresses. LU numbers are used in READ, ENTER, INPUT, WRITE, and PRINT statements in BASIC and FORTRAN programs to refer to bus devices. When you use the HP-IB subroutines CMDW and CMDR, however, you use the device talk and listen ASCII address characters. Refer to Chapter 4 for a description of addressing and the HP-IB subroutines.

To program bus devices, you need a list of the devices, with their HP-IB addresses and ASCII talk and listen characters, and the LU numbers of each device address. An address assignment table like the one in Figure 2-5 contains this information.

Figure 3-1 diagrams an E/F-Series computer with two HP-IB systems. The lower half of the figure shows the bus hardware, consisting of two HP-IB interface cards, cables, and six bus devices. Each card controls three bus devices.

The upper half of the figure shows the software structure of the system. The driver has two EQTs, one for each bus. Each EQT has three subchannels for the devices attached to the interface card. Each EQT has an assigned LU number (the bus LU), and there is an LU for each subchannel (device LUs).

Figure 3-2 diagrams an A-Series computer with two HP-IB systems. The lower half of the figure shows the bus hardware, consisting of two HP-IB interface cards, cables, and six bus devices. Each card controls three bus devices.

The upper half of the figure shows the software structure of the system. The interface driver has two IFTs, one for each bus. For some bus devices, there are device drivers that process I/O requests for specific devices and pass the processed data to the interface driver. The device drivers are shown between the interface driver and the devices they control, although they actually pass data between the operating system and the interface driver. Each IFT has four associated DVTs, one for each bus and each bus device. There is an LU number assigned to each DVT.

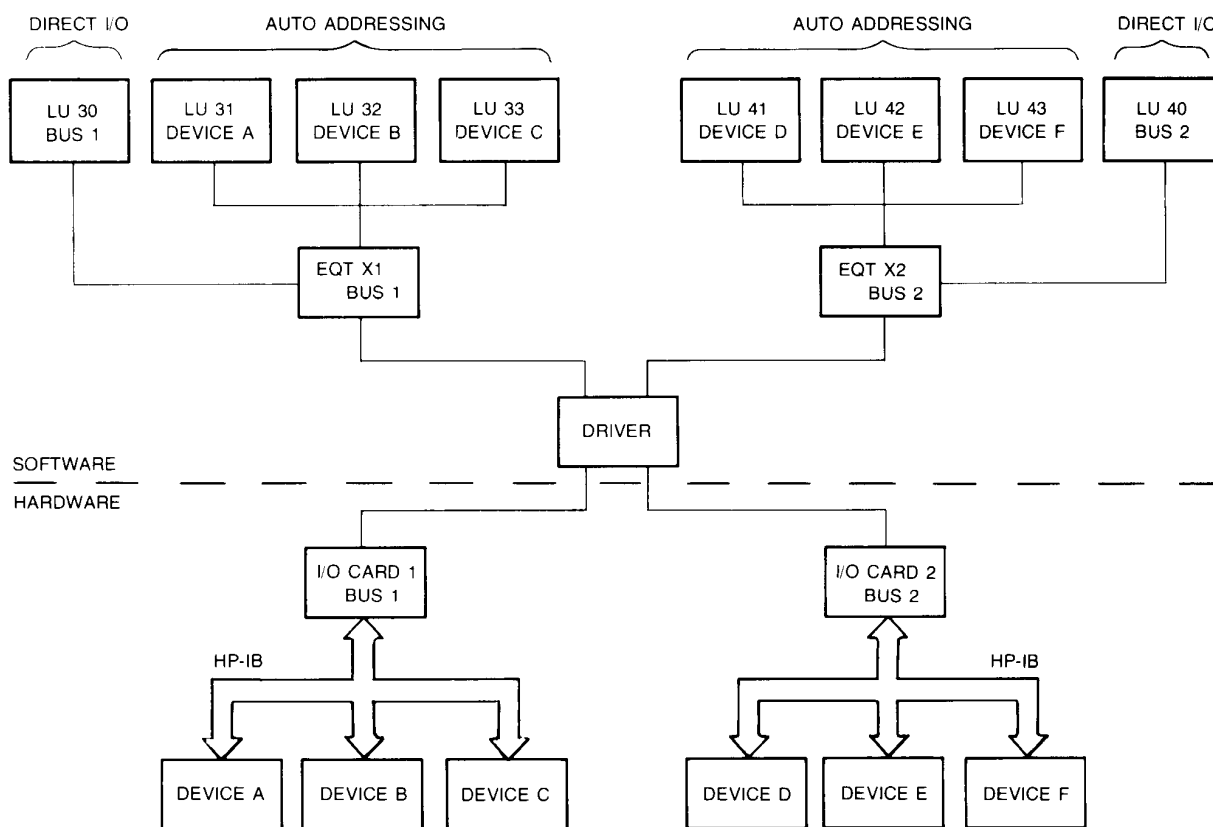


Figure 3-1. Bus Hardware/Software Relationship for RTE-6/VM

## LUs and Auto vs Direct Addressing

When a program uses autoaddressing, see Chapter 4 for a description of addressing, it uses an LU to identify a specific device. If the program statement is a WRITE or PRINT statement, the driver finds the device address in the bus EQT for RTE-6/VM systems, or in the first driver parameter of the DVT for RTE-A systems. The driver then creates a string containing the ASCII talk address of the controller and the listen address of the device, plus the data in the WRITE or PRINT statement. If the statement is a READ or ENTER, the driver creates a string containing the ASCII talk address of the device, and the listen address of the controller, so the computer can receive the expected data. Autoaddressing puts the driver in control of the device traffic.

When a program uses direct addressing, it specifies the LU of the bus and creates a string containing the ASCII talk and listen addresses of the devices to be addressed, and sends the LU and address information to the driver, along with any data to be transferred. The driver puts the addresses and data directly on the bus, without referring to the EQT or DVT to find device addresses. Direct addressing puts the program in control of the device traffic. Because the program supplies all addressing information, device LUs and DVT or EQT entries for bus devices are not needed. One LU for the bus is all that is required.

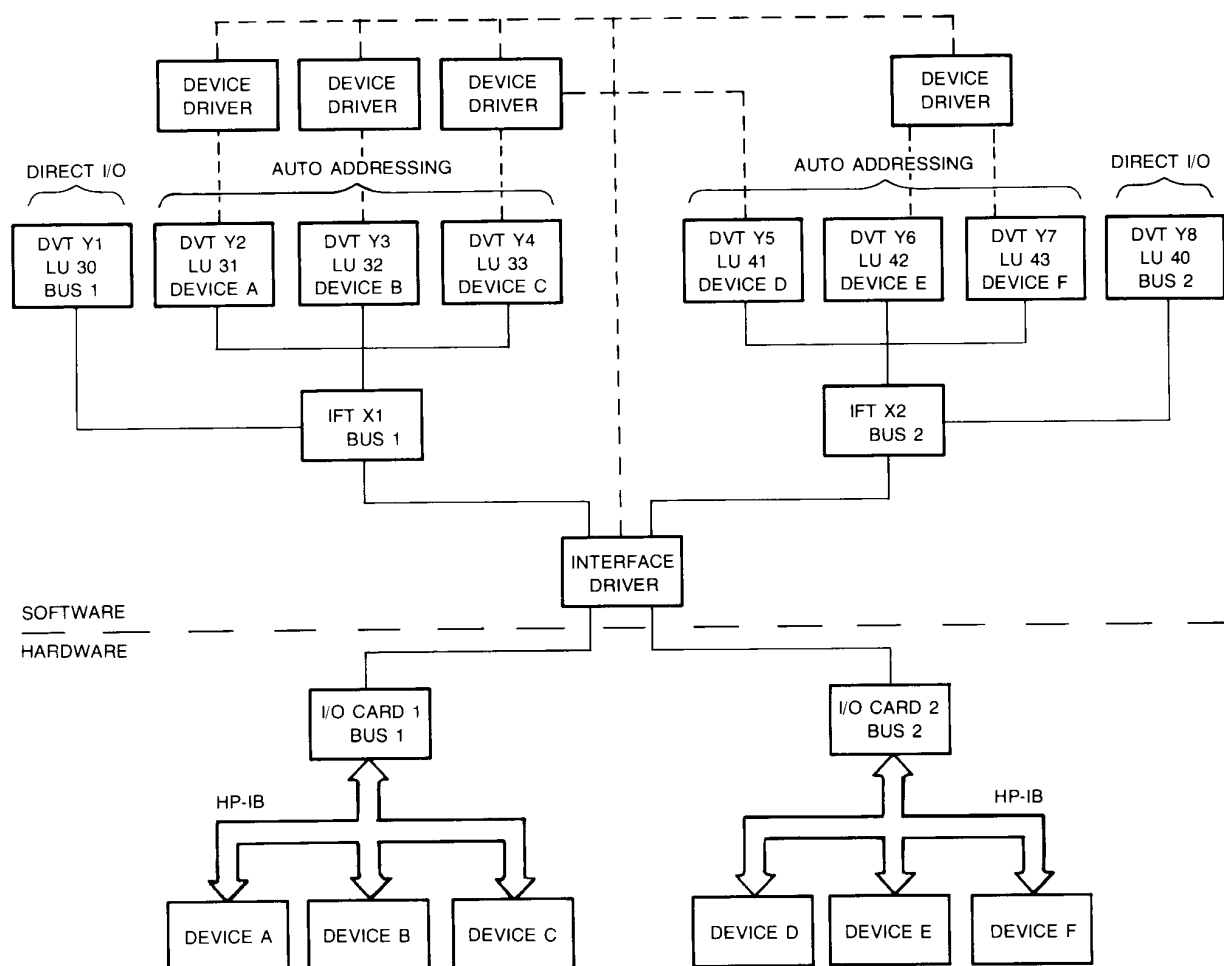


Figure 3-2. Bus Hardware/Software Relationship for RTE-A

## LU Assignment for RTE-6/VM

At system generation, spare LUs are usually set up to permit I/O expansion. When an HP-IB is installed in the computer, this becomes particularly important. The bus and its devices take up EQT space and LU numbers. New devices can be added to the bus only if there are available LU numbers. Appendix A describes HP-IB generation requirements for RTE-6/VM in detail. The following section assumes that the system has been generated with enough LUs for the application. RTE-6/VM have operator commands that let you make or modify LU assignments, examine existing assignments, and check device status.

### Assign or Reassign LU – The LU Command

To change a system LU assignment from one device to another, or to assign a spare system LU number to a device, use the reassignment command. The LU command requires a capability level of 60.

```
S=xx COMMAND ?LU ,lu,eqt,schnl
```

or

```
:SYLU ,lu,eqt,schnl
```

where *lu* = the system LU number that you want to reassign or assign, *eqt* is the EQT number of the HP-IB interface, and *schnl* is the device address of the instrument on the bus. This assignment command cancels any previous assignment of *lu*. Always use the SL command as described in the Examine Assignment section of this chapter to make sure that you do not cancel a necessary assignment. Never reassign LU numbers 1 through 6. They are reserved for assignment by the system.

If the system is rebooted from disk after you have made assignments by the LU command, the assignments are lost. A typical assignment might be:

```
:SYLU , 45 , 2 , 5
```

where 45 is the LU number to be assigned to the HP-IB instrument at device address 5 on the bus with EQT number 2.

### Examine LUs – The LUPRN Utility

The LUPRN utility, described in the *RTE-6/VM Utility Programs Reference Manual*, part number 92084-90007, lets you list the system LUs, sorted by any of several keys. It can be used to examine the LU assignments, or to examine the system LUs to find unassigned LUs for the bus devices.

## Examine Assignment – The SL Command

This request returns the session LU, system LU, and EQT/subchannel linkages for every device assigned to the session. It requires a capability level of only 10, so any user can examine the linkages. A typical SL command and the system response are shown below:

```
:SL  
  
SLU    1=LU    55 = E14  
SLU    2=LU     2 = E 1  
SLU    3=LU     3 = E 2  
SLU    4=LU   142 = E14 S 1 D  
SLU    6=LU     6 = E 6  
SLU    8=LU     8 = E 8  
SLU   13=LU    13 = E 1 S 5  
SLU   16=LU    16 = E 1 S 8  
SLU   20=LU    20 = E 2 S 1  
SLU   26=LU    26 = E 2 S 7  
SLU   36=LU    36 = E 2 S17  
SLU   47=LU    47 = E 2 S28  
:
```

If any device is down, the letter “D” appears after the EQT or subchannel number, as in the entry for SLU 4 above.

## Check Status – The EQ Command

To check the status of the bus, once you know the EQT, use the EQ command, which, like SL, requires a capability level of only 10:

```
S=xx COMMAND ? EQ,eqt
```

or

```
:SYEQ,eqt
```

the request and the system response might be:

```
:SYEQ,2  
24 DV.37 0 B U0 0
```

2 is the EQT number of the bus, 24 is the bus select code, DV.37 identifies the driver as DVA37, the first 0 indicates that DMA is not being used for the bus, B means that the buffering option is selected, U0 indicates that subchannel 0 was the last addressed, and the final 0 is the status of the interface (it is available). The status can be any of these four values:

- 0 = Available
- 1 = Down, unavailable
- 2 = Busy, unavailable
- 3 = Waiting for a DMA channel



## WELCOM File Modification

If the HP-IB system is not likely to change, and if you need to reboot the system, you should modify the system WELCOM file to assign the bus and device LUs. Figure 3-3 shows an excerpt from a WELCOM file that assigns LUs for two HP-IBs and the devices on each.

```
:SYLU, 40, 10, 0      (assign first bus LU)
:SYLU, 41, 10, 1      (assign LU for device address 1)
:SYLU, 42, 10, 2      (assign LU for device address 2)
:SYLU, 43, 10, 3      (assign LU for device address 3)
:SYLU, 50, 20, 0      (assign second bus LU)
:SYLU, 51, 20, 1      (assign LU for device address 1)
:SYLU, 52, 20, 2      (assign LU for device address 2)
:SYLU, 53, 20, 3      (assign LU for device address 3)
```

Figure 3-3. Excerpt from WELCOM File with Two HP-IBs

## LU Assignment for RTE-A

At system generation, spare LU numbers and DVT entries should be created to allow I/O expansion. When the system includes an HP-IB, this becomes even more important, because the system requires a DVT entry and an LU for every device on the bus, and for the bus itself. This section assumes that the system has enough DVT entries and enough LUs for the HP-IB system. Appendix B describes RTE-A generation requirements for the HP-IB.

## Examine Assignments and Request Device Status – The IO Command

The IO command on RTE-A displays LU configuration information and I/O device status.

### LU Configuration Information

Check assignments by listing the LU configuration information for the entire LU list or for a range of LUs with the IO command:

```
CI> io -c [firstlu] [lastlu]
```

or

```
CI> io -cx [firstlu] [lastlu]      (the -x option gives extended information)
```

The parameters *firstlu* and *lastlu* are optional and may be used to specify a range of LUs. If omitted, the entire LU list of the system is displayed.

For example, to display extended configuration information of the entire LU list:

```
CI> io -cx
lu  device name                select  bus  dvt dvt  interface
   device name                code   addr nbr addr type
1  terminal (5)                20     3   36102 0
   fifo queuing; timeout: 32767 csecs; priority: 63; buffered
   buffer limits: lower = 32 words, upper = 384 words
2  CS/80 disk (33)            0      0   1 36000 37
   priority queuing; timeout: 0 csecs; priority: 63; unbuffered
   unit: 0 block: 0 tracks: 0 sectors/track: 0
3  CS/80 disk (33)            0      0   2 36041 37
   priority queuing; timeout: 0 csecs; priority: 63; unbuffered
   unit: 0 block: 0 tracks: 0 sectors/track: 0
4  not assigned
5  streaming tape drive (24)  26     3  12 36540 37
   fifo queuing; timeout: 500 csecs; priority: 0; unbuffered
6  printer (12)               30     2  17 36757 37

More io...  {Press the space bar to continue the listing.}
```

The I/O configuration report listed in the example above displays the following information:

- the interface select code (or 0 if none)
- the name of the interface and the interface type (the interface type is in parentheses next to the name of the interface)
- the bus address
- the DVT number and address
- the interface type
- extended configuration information which includes (displayed only with the `-x` option):

**queuing** – `priority` indicates requests are queued in the order of the priority of the requesting program; `fifo` indicates requests are queued in order of arrival.

**timeout** – the number of milliseconds that a device has to respond to a request before timing out, or 0 if no timeout is specified for this device.

**priority** – the interface priority of requests for this device.

**buffered/unbuffered** – specifies whether the device is buffered or unbuffered. If buffered, the buffer limits are displayed.

## Device Status

Obtain the status of the entire LU list or a range of LUs with the IO command as follows:

```
CI> io [firstlu] [lastlu]
```

The parameters *firstlu* and *lastlu* are optional and may be used to specify a range of LUs. If omitted, the entire LU list of the system is displayed. For example, to display device status for LUs 10 through 24:

```
CI> io 10 24
```

```
lu device name          status
10 SCSI DAT             idle
11 330MB SCSI: Product LUidle
12 330MB SCSI: User LU  idle
13 not assigned
14 not assigned
15 not assigned
16 HP-IB 7914 - Boot Diskidle lu 19 in node list is busy
17 HP-IB 7914           idle lu 19 in node list is busy
18 HP-IB 7914           idle
19 HP-IB 7914           idle
20 HP-IB 7914           idle
21 HP-IB 7914           idle
22 not assigned
23 not assigned
24 7914's CTD
```

The device status report includes the following information:

- the LU number of the device.
- device name corresponding to the device type.
- the status of the device which can be one of the following: idle, down, busy.
- additional status information.

For more detailed information on using the IO command, read the online help by entering the following command:

```
CI> ? io
```

## Change DP#1 (Bus Address) – The CN Command

When you set or change the device address switch on an instrument, you must set or change DP#1 (the bus address) to inform the operating system of the address by using the CN command:

```
CI> cn lu 24b dev_addr
```

Where *lu* is the LU number that identifies the DVT to be affected, 24B is a code to specify changing DP#1 (the CN command has other uses), and *dev\_addr* is the new device address.

---

**Note** This command can only be used with instruments. Device drivers for other HP-IB devices, such as disk and tape drives, do not support this function.

---

In the following example, the IO command is used to show the LU assignment, including the DVT number and the bus address before and after the bus address is changed:

```
CI> io -cx 25
lu device name          select  bus  dvt dvt  interface
                        code   addr nbr addr type
25 instrument (77)      31     1  81  42417 37
   priority queuing; timeout: 50 csecs; priority: 63; unbuffered
```

```
CI> cn 25 24b 3
```

```
CI> io -cx 25
lu device name          select  bus  dvt dvt  interface
                        code   addr nbr addr type
25 instrument (77)      31     3  81  42417 37
   priority queuing; timeout: 50 csecs; priority: 63; unbuffered
```

# Bus Messages

---

## Introduction

There are three kinds of bus messages. This chapter describes each of the three types in detail, then lists the HP-IB library subroutines that make it easy for you to program devices to transmit and receive messages.

Messages on the bus are called bus traffic. Bus traffic consists of all three kinds of messages:

- **Bus addresses** – ASCII characters transmitted on the DIO lines that select devices to send or receive bus commands or data. All addressing originates at the controller and is accompanied by assertion of the ATN bus management line.
- **Bus commands** – ASCII characters transmitted on the DIO lines with the ATN bus management line asserted, or signals on the other bus management lines, that control the bus devices. With the exception of the request service message (SRQ), all command messages originate at the controller.
- **Bus data** – Measurement results, programming information, and any other communications on the bus that are not accompanied by assertion of the ATN line. Bus data make up the bulk of bus traffic, and are usually encoded in ASCII. Bus data are always transmitted on the DIO lines, and can originate at any device with talk capability.

Bus addresses and commands transmitted over the DIO lines are always accompanied by assertion of the ATN line. When ATN is asserted, the bus is in the command mode, and all devices interpret signals on the DIO lines as addresses or commands. When ATN is not asserted, the bus is in the data mode, and signals on the DIO lines are interpreted as data for the devices or controller.

## Bus Addresses

Before any bus commands or data can be transmitted, one device to send the commands or data and one or more receivers must be addressed. The sender is called the talker, and the receivers are called listeners.

The controller transmits all addresses. The controller itself can be the talker or a listener, or it might not be involved at all in the bus traffic after it addresses a talker and one or more listeners.

From the viewpoint of the programmer, there are two kinds of addressing:

- **Automatic** – used to send a single command or data message from the controller to one device or to read a single data message from one device to the controller. The ASCII address strings are created by the HP-IB driver.
- **Direct** – used to address a talker and one or more listeners, often without addressing the controller in either role. Direct addressing is usually used for multiple messages. The ASCII address strings are created by the program, so you have complete control over their contents.

## Automatic Addressing

Automatic addressing is performed by READ, ENTER, INPUT, WRITE, or PRINT statements, or HP-IB subroutine calls that specify a device LU.

The HP-IB driver uses the device LU to look up the HP-IB device address. The driver sends the ASCII unlisten address to prepare the bus for new traffic. The ASCII address string created by the driver for WRITE or PRINT statements or HP-IB command message subroutines also includes the listen address of the specified device and the controller talk address. The ASCII address string created by the driver for READ, ENTER, or INPUT statements or HP-IB device status subroutines also contains the talk address of the specified device and the listen address of the controller.

As soon as a talker is addressed, any previous talker is automatically unaddressed. The ASCII untalk address character is only used to clear all talkers during a clear or abort sequence.

For example, a BASIC/1000D PRINT statement that uses automatic addressing to send a programming string (bus data) to an HP 3455A Digital Voltmeter at LU 34 looks like this:

```
250 PRINT# 34; "F1R1M3"
```

The driver uses the device LU (34) to look up its HP-IB device address. Then it creates an ASCII address string that unaddresses the previous listeners, addresses the HP 3455A to listen, and addresses the controller to talk. The ASCII address string is sent over the bus with the ATN bus management line asserted. The controller releases ATN and sends the bus data to the listening voltmeter. The programming string sets the voltmeter to measure DC voltage (F1) in the lowest range (R1), and turns the mathematics functions off (M3).

The following example shows the BASIC/1000C statements that create the same addressing string and program the voltmeter. BASIC/1000C uses I/O specifiers to identify devices. I/O specifiers are special variables that can store LU numbers, secondary addresses (if any), and an optional list of device attributes. The LU, secondary address, and attributes are stored in an I/O specifier by the ASSIGN statement. I/O specifiers are used in I/O statements, as in the PRINT statement in the following example:

```
250 ASSIGN @Dvm TO 34
260 PRINT @Dvm; "F1R1M3"
```

Line 250 assigns LU 34 to I/O specifier @Dvm, and line 250 uses it to send the programming string "F1R1M3" to the DVM at LU 34 as in the BASIC/1000D example.

## Secondary Addressing

Many instruments have secondary addresses for internal subsystems. A form of automatic addressing is used to access the secondary addresses. The device LU and a secondary address number are specified in one of four HP-IB secondary addressing subroutines, or in the I/O statements of some languages.

The four HP-IB subroutines for secondary addressing are:

- **SECW and SECR** – (secondary write and secondary read) for integers in BASIC/1000D and real values in FORTRAN 4.
- **SECWR and SECRR** – (secondary write of real and secondary read of real) for real values in BASIC/1000D only. They are required because of the way real numbers are represented internally.

Secondary addressing in BASIC/1000D, FORTRAN 4X and 7X, and BASIC/1000C is described in the following paragraphs.

### BASIC/1000D

A SECW call from a BASIC/1000D program that sends a string of data to secondary address 5 of the device at LU 44 looks like this:

```
310 SECW(44,5,A,25)
```

The first parameter is the device LU; the second parameter is the secondary address. A is an integer array that contains the data and 25 is the number of words in A. The HP-IB driver creates a string of HP-IB commands that unaddresses all devices on the bus, addresses the device at LU 44 to listen via secondary address 5, addresses the controller to talk, and sends the data message.

## FORTRAN 7X

Secondary addressing is implemented in FORTRAN 7X without the need for SECW or SECR. Secondary addressing can be performed by the WRITE and READ statements, as shown in the following example, which reads data from the secondary address 2 of the device at LU 30, and responds to the data by sending the value of 16.23 to subaddress address 3 of the device at LU 25.

```
FTN7X, L
  PROGRAM RSPOND
  ILUA=30
  ILUB=25
  ISECA=2
  ISECD=3

  READ(ILUA:ISECA, 100) VAL

100  FORMAT (F14.7)
     .
     .
     .
     DATA = 16.23
     WRITE(ILUB:ISECB, 101) DATA

101  FORMAT (F14.7)
     .
     .
     .
     END
```

The ILUA:ISECA and ILUD:ISECB parameters permit you to specify the device LU and the secondary address of the bus devices. As with the SECW and SECR subroutines, these READ and WRITE statements use a form of automatic addressing to access the device.

## BASIC/1000C

BASIC/1000C performs secondary addressing with its I/O specifiers. The ASSIGN statement below assigns secondary address 5 of LU 25 to I/O specifier @Dvm:

```
250 ASSIGN @Dvm TO 25:5
260 PRINT @Dvm; A$
```

The ASSIGN statement and I/O specifiers are described in the *BASIC/1000C Reference Manual*, part number 92857-90001.

## Direct Addressing

Direct addressing is used when the programmer wants to control all device addressing. In direct addressing subroutine calls, the bus LU is specified, rather than a device LU. The program creates ASCII address strings to unaddress previous listeners and address new talkers and listeners. The driver passes the address string to the bus without adding to it or changing it.



The CMDR and CMDW message subroutines are direct addressing subroutines. The subroutines are described in detail later in this chapter, but the following BASIC/1000D example shows how a CMDW command is used to address one talker and two listeners:

```
230 CMDW(30,"?D#*",0)
```

The first parameter is the bus LU. The second parameter is a string containing the unlisten address, the talk address of the device at HP-IB device address 04B, and the listen addresses of the devices at addresses 03B and 12B. The last parameter is a zero, to indicate that the controller is not sending any data with these addresses. The ASCII addressing characters that represent device addresses are listed in Appendix C.

After this subroutine is executed, the devices at addresses 03B and 12B listen to the talker at device address 04B. The addressing string ("?D#\*") is similar to the string that the driver creates when it is called upon to perform automatic addressing, except that you have complete control of its contents. If you want to add listeners to the bus without unlistening the current listeners, for example, you can transmit the new listener addresses as the second CMDW parameter without transmitting the unlisten address.

If you want the controller to talk or listen on an A-Series system, the command string must also contain the talk or listen address of the HP-IB interface chip (which acts as the bus controller). The HP-IB interface chip talk address is 76B (>), and the listen address is 136B (^). If you want the controller to talk or listen on an E/F-Series system, the talk address is 40B (space) and the listen address is 100B (@).

The advantage of direct addressing is that you specify all bus traffic. The driver does not create any of the traffic. This gives you control over all bus activity, but it adds the responsibility for accuracy in specifying talk and listen address characters. Also, because device LUs are not used in direct addressing, there does not have to be an LU number for each device on the bus, just one for the bus itself (refer to Chapter 3).

## Bus Commands

The bus traffic that follows addressing often includes bus commands that tell the devices on the bus what to do.

Like bus address messages, most bus commands are ASCII characters sent over the DIO lines while the ATN bus management line is active. Other bus commands are sent as signals on the bus management lines. This section describes two groups of command messages:

- Addressed commands that affect only devices addressed to listen when the command is transmitted. Addressed commands are always transmitted over the DIO lines.
- Universal commands that affect every device on the bus that is capable of performing the function of the command, whether addressed to listen or not. Universal commands are transmitted over the DIO lines or the bus management lines.

## Addressed Commands

Table 4-1 lists the five addressed commands. They affect only those devices that are addressed to listen when the command is sent, and have the capacity to respond to the command.

**Table 4-1. Addressed Commands**

| Command                 | Octal Code | Mnemonic | Function  |
|-------------------------|------------|----------|---|
| Go to Local             | 01         | GTL      | Return listening devices to local mode.                       |
| Selected Device Clear   | 04         | SDC      | Set listening devices to predetermined state.                 |
| Parallel Poll Configure | 05         | PPC      | Configure listening device to respond to parallel polling.    |
| Group Execute Trigger   | 10         | GET      | Trigger all listening devices to perform a programmed action. |
| Take Control            | 11         | TCT      | Pass control to listening device.*                            |

\* The Take Control command is not implemented on the HP 1000, because the HP 59310B or HP 12009A HP-IB Interface Card is always the controller; it cannot pass control to any other device.

### GTL – Go To Local

Many bus devices can operate in either the local or remote mode. In the remote mode, the device is controlled by the HP-IB. In the local mode, the device is controlled by its front panel. For normal HP-IB use, devices are kept in the remote mode by holding the REN line active. Refer to the REN – Remote Enable section of this chapter for more information. The GTL command permits you to temporarily put a device in the local mode while REN is true, so an operator can make adjustments at the front panel. The device automatically returns to the remote mode the next time it is addressed.

### SDC – Selected Device Clear

All HP-IB devices have a clear state, defined for each device in its operation manual. The SDC command lets you return all listening devices to the clear state. On most instruments, it is a “safe” state, such as all relays open on a switching device.

### PPC – Parallel Poll Configure

The PPC command configures a single device to respond to a parallel poll by setting one of the eight DIO lines to logic 1 or 0. It can also unconfigure one or more devices from parallel poll response.

Only one device should be addressed to listen when the PPC command is used to configure parallel polling response. The DIO line and logic level for the response are specified by a secondary command transmitted over the DIO lines after the PPC command. The PPE (parallel poll enable) secondary command must immediately follow the PPC command. The PPE secondary command is an ASCII character that is interpreted as shown in Table 4-2.

**Table 4-2. PPE Secondary Command Codes**

| Octal PPE Value | ASCII PPE Char. | DIO Line | Logic Level | Octal PPE Value | ASCII PPE Char. | DIO Line | Logic Level |
|-----------------|-----------------|----------|-------------|-----------------|-----------------|----------|-------------|
| 140             | '               | 1        | 0           | 150             | h               | 1        | 1           |
| 141             | a               | 2        | 0           | 151             | i               | 2        | 1           |
| 142             | b               | 3        | 0           | 152             | j               | 3        | 1           |
| 143             | c               | 4        | 0           | 153             | k               | 4        | 1           |
| 144             | d               | 5        | 0           | 154             | l               | 5        | 1           |
| 145             | e               | 6        | 0           | 155             | m               | 6        | 1           |
| 146             | f               | 7        | 0           | 156             | n               | 7        | 1           |
| 147             | g               | 8        | 0           | 157             | o               | 8        | 1           |

The PPC command can also unconfigure parallel polling response from one or more devices when it is followed by the PPD (parallel poll disable) secondary command. The PPD secondary command has only one value, 160B.

Remember that only one device should be addressed to listen to the PPC/PPE pair, but more than one can be addressed to listen to the PPC/PPD pair. Parallel polling is described in detail in Chapter 5 of this manual.

### **GET – Group Execute Trigger**

If several bus devices must perform their functions simultaneously, as in an automated test set, where several measurement devices must sample a group of test points at the same time, the GET command lets you trigger them together after programming them individually. All addressed listeners with triggering capability perform a preset function when they receive the GET command.

## Universal Commands on the DIO Lines

Table 4-3 lists the five Universal commands that are transmitted over the DIO lines. They affect every device on the bus that has the capacity to respond to the command, whether or not the device is addressed to listen.

**Table 4-3. Universal Commands Transmitted on the DIO Lines**

| Command                   | Octal Code | Mnemonic | Function  |
|---------------------------|------------|----------|---|
| Local Lockout             | 21         | LLO      | Disables “local” switch on local/remote devices.                  |
| Device Clear              | 24         | DCL      | Returns all devices to predetermined state.                       |
| Parallel Poll Unconfigure | 25         | PPU      | Disables response to parallel polling for all configured devices. |
| Serial Poll Enable        | 30         | SPE      | Prepares bus for serial polling.                                  |
| Serial Poll Disable       | 31         | SPD      | Returns bus to normal operation after serial polling.             |

### LLO – Local Lockout

Most devices with remote and local modes have a pushbutton on the front panel to return the device to the local mode. The LLO command disables the button so the device cannot be manually returned to the local mode from the remote mode. The LLO command is universal, so every device that recognizes LLO responds, whether it is addressed to listen or not.

The local lockout is released when the device is returned to the local mode. Refer to the REN – Remote Enable section of this chapter for more information on the remote and local modes.

### DCL – Device Clear

This command is similar to the SDC command, except that it clears every device on the bus that can be cleared, whether addressed to listen or not. The clear state for most devices is a “safe” state, such as all relays open on a switching device. The exact clear state of each device is described in its operating manual. Individual devices are cleared with the SDC command.

## PPU – Parallel Poll Unconfigure

Every device on the bus that has been configured to respond to parallel polling is unconfigured by this command, whether addressed to listen or not. Individual devices are unconfigured by the PPC command with the PPD secondary command. Parallel polling is described in detail in Chapter 5.

## SPE/SPD – Serial Poll Enable/Disable

Serial polling is a method of locating a device in need of service. The SPE command prepares the bus for serial polling, and the SPD command returns it to normal operation. Between the SPD and SPE commands, the address of a device that recognizes serial polling and its status byte pass over the bus. The bus traffic generated during a typical serial poll is shown in Figure 4-1.

The controller can perform automatic serial polling in response to the SRQ command. Refer to the SRQ – Service Request section of this chapter to find the device that requires service. The controller automatically generates the SPE and SPD commands and device addresses and reads the status bytes of the devices that have been set up for automatic serial polling. Serial polling is described in detail in Chapter 5.

| Sender     | Traffic                   | Receiver    |
|------------|---------------------------|-------------|
| Controller | Unlisten Address          | All Devices |
| Controller | Controller Listen Address | Controller  |
| Controller | Spe Command               | All Devices |
| Controller | Device Talk Address       | Device      |
| Device     | Status Byte               | Controller  |
| Controller | Spd Command               | All Devices |

Figure 4-1. Bus Traffic during a Serial Poll

## Universal Commands on the Bus Management Lines

Table 4-4 lists the universal commands that are transmitted on the bus management lines. The ATN line is always asserted when bus commands are sent over the DIO lines. It is not asserted when bus commands are sent over the DIO lines, except when ATN and EOI are asserted together to initiate a parallel poll.

## EOI and ATN – Initiate Parallel Poll

When the ATN and EOI bus management lines are asserted together, all devices configured for parallel polling (refer to the PPC – Parallel Poll Configure section of this chapter) respond by setting one of the eight DIO lines to logic 1 or logic 0. Parallel polling is a method of finding devices in need of service. Parallel polling is described in detail in Chapter 5.

## REN – Remote Enable

Many HP-IB devices can operate in the local or remote mode. In the local mode, a device is controlled by its front panel. The device ignores signals on the bus. In the remote mode, the front panel is disabled, and the device is controlled by the bus. Devices are set to the remote mode by asserting the REN line (sending the REN command), and addressing the devices to listen while it is asserted.

REN is a universal command, but addressing is required to put remote/local devices in the remote mode when REN is asserted. All devices with remote and local operation must monitor the REN line.

Devices in the remote mode can be temporarily returned to local operation by the GTL command. All devices can be returned to the local mode by releasing the REN line.

Remember that the REN command only enables remote operation. Individual devices do not enter the remote mode unless they are addressed to listen while REN is true. The same is true for returning to local mode; individual devices remain in the remote mode until REN is released.

**Table 4-4. Universal Commands Transmitted on the Bus Management Lines**

| Command         | Signal Line | Function                                 |
|-----------------|-------------|--|
| Parallel Poll   | EOI & ATN   | Initiate parallel poll.                  |
| Remote Enable   | REN         | Set addressed devices to remote mode.    |
| Service Request | SRQ         | Interrupt controller to request service. |
| Command Mode    | ATN         | Establish command mode.                  |
| Interface Clear | IFC         | Stop all bus traffic.                    |

## SRQ — Service Request

This is the only bus command that does not originate at the controller. Devices with service request capability interrupt the controller by asserting the SRQ line (sending the SRQ command).

The controller can respond to the request by performing an automatic serial poll to find the requesting device. Because all devices are connected in parallel to SRQ, the controller must poll all of the devices that have been enabled for serial polling to determine which device asserted the line. Serial polling is performed with the SPE and SPD commands, and is described in detail in Chapter 5.

## **ATN – The Command Mode**

All devices on the bus are required to monitor the ATN line at all times. When ATN is asserted, the bus is in the command mode. Signals on the DIO lines are interpreted as bus address or command messages. The controller asserts ATN and EOI together to initiate a parallel poll. When ATN is high, the bus is in the data mode. Signals on the DIO lines are interpreted as data for devices or the controller, and the EOI line is used to indicate the end of a data transfer.

## **IFC – Interface Clear**

A pulse on the IFC line suspends all bus operations and returns each bus device to a state defined in its operation manual. The IFC command clears the bus immediately. It interrupts any data transfers in progress and clears a service request on some devices (causes the device to release SRQ).

## **Data Messages**

When the ATN line is not asserted, all characters transmitted on the DIO lines are interpreted as data. When a measurement device is programmed to take a reading from a test point and triggered to take the reading, it returns the result to the controller or to another bus device as data.

Data messages consist of eight-bit bytes, usually encoded as seven-bit ASCII characters, with or without an eighth parity bit. Bus commands originate at the controller, but data messages can originate at any device on the bus that is addressed to talk.

# HP-IB Library Subroutines

The bus control program manages the transmission of bus address, bus command, and data messages through calls to the HP-IB library subroutines. Whether you are programming in BASIC/1000D, BASIC/1000C, or FORTRAN, the library subroutines make bus programming simple. The subroutines are divided into three groups:

- **Device control subroutines** that transmit bus commands, sometimes with address messages and secondary commands, to control the actions of bus devices.
- **Data communications subroutines** that transmit address messages and usually transmit or receive data.
- **Interrupt and device status subroutines** that perform serial and parallel polling and process service requests.

The device control and data communications subroutines are described in this chapter. The interrupt and device status subroutines are described in the first section of Chapter 5.

Most of the HP-IB subroutines have corresponding BASIC/1000C statements, but it is sometimes necessary to call HP-IB library subroutines from BASIC/1000C. The subroutine descriptions that follow include descriptions of the corresponding BASIC/1000C statements, and explain when it is necessary to use a subroutine call rather than a BASIC/1000C statement. Refer to the *BASIC/1000C Reference Manual*, part number 92857-90001, for detailed descriptions of the HP-IB statements.

## Syntax Conventions

The subroutine call syntax specifications and examples in this chapter and in the first part of Chapter 5 use the following conventions. Subroutine calls from BASIC/1000C are the same as in BASIC/1000D and FORTRAN, except as explained below.

In FORTRAN and BASIC/1000C, the subroutine call must include the word `CALL`:

```
CALL subname(arguments)
```

In FORTRAN and BASIC/1000C, there can be a space between the subroutine name and the left parenthesis:

```
CALL subname (arguments)
```

or

```
CALL subname(arguments)
```

In BASIC/1000D, the word `CALL` is optional, but no blanks are allowed between the subroutine name and the left parenthesis:

```
subname (arguments)
```

or

```
CALL subname(arguments)
```



The first argument of every HP-IB subroutine call is the LU number of the device (automatic addressing form) or the LU number of the bus (direct addressing form).

*devicelu* = LU number of the device (automatic addressing)  
*bushu* = LU number of the bus (direct addressing)

Arguments are defined the first time they appear in this chapter. If an argument in a subroutine description is not defined, look for its definition in an earlier description in the chapter.

For the BASIC/1000C statement examples and syntax specifications, the following conventions are used:

- Devices are referred to by I/O specifiers, which are the character “@” followed by any legal variable name, as in the following ASSIGN statement:

```
ASSIGN @Specifier TO 10
```

- The value assigned to an I/O specifier has three parts:

```
ASSIGN @Specifier TO devicelu[:secondary][;attriblist]
```

- The *devicelu* is the LU number of an instrument or the bus. In the syntax specifications, device specifiers and bus specifiers are always identified as such, because the actions performed by the statements are different for the bus than for devices. The two specifiers are shown as:

```
REMOTE devspec
```

or

```
REMOTE busspec
```

- The optional *:secondary* makes the I/O specifier point to a device secondary address, rather than the entire device. No *:secondary* can be specified for a *busspec*.
- The optional *attriblist* is a list of device attributes, separated by commas, as described in the *BASIC/1000C Reference Manual*, part number 92857-90001.
- The I/O specifiers do not appear inside parentheses for statements, but they do for functions:

```
TRIGGER devspec
```

vs.

```
SPOLL ( devspec )
```

## Device Control Subroutines

The device control subroutines transmit bus commands. The commands always originate at the controller, and are sent either over the eight DIO lines in the command mode (ATN asserted) or they are sent over one or more of the bus management lines. Messages sent over the DIO lines are transferred with the necessary handshaking.

In most of the subroutine descriptions, the terms “automatic addressing form” and “universal command form” appear. Subroutines that take the automatic addressing form transmit addressed commands, and use automatic addressing to select the device to receive the command. Refer to the Bus Addresses section of this chapter. The LU of the device is specified in the subroutine call. Subroutines that take the universal command form transmit universal commands. The LU of the bus is specified in the subroutine call. Most subroutines can take either form; the only difference in the subroutine call is the LU.

Not all devices respond to all bus commands, so check the manuals for the devices on the bus. The subroutine descriptions explain whether the subroutine transmits universal or addressed commands. Table 4-5 lists the device control subroutines.

**Table 4-5. Device Control Subroutines and Statements**

| Message       | Subroutine | Subroutine Function   |
|---------------|------------|---|
| Clear         | CLEAR      | Issues selected device clear command SDC to one or more addressed devices, or<br>Issues universal device clear command DCL to all devices on the bus. |
| Remote        | RMOTE      | Sets remote enable line REN true, enabling remote operation for all devices responding to the REN control line.                                       |
| Trigger       | TRIGR      | Issues group execute trigger command GET to one or more addressed devices.  |
| Local         | GTL        | Issues go to local command GTL to addressed devices on the bus.   |
| Local Lockout | LLO        | Issues local lockout command LLO to all devices on the bus.   |
| Clear Lockout | LOCL       | Sets remote enable line REN false, removing all devices from local lockout mode and returning them to local control.                                  |

## CLEAR

The CLEAR subroutine can use automatic addressing to clear a single device or all listening devices, or it can take the form of a universal bus command to clear all devices on the bus. The CLEAR subroutine can take any of the following forms:

CLEAR(*devicelu*,1)

CLEAR(*bushu*,1)

CLEAR(*bushu*,2)

The first form issues an SDC bus command to the device specified by *devicelu*. The subroutine unaddresses all listeners, addresses the device at *devicelu* to listen and the controller to talk, and sends SDC.

The second form issues an SDC to all devices on the bus that are already addressed to listen. No addresses are transmitted.

The third form issues a universal device clear (DCL) command to clear all devices on the bus, whether or not they are addressed.

The exact effect of the SDC and DCL messages depends on the device, so refer to the device operating manuals.

In BASIC/1000C, the clear function is performed by the CLEAR statement. The statement has two forms:

CLEAR *devspec*

or

CLEAR *busspec*

These two forms correspond to the first and third forms of the CLEAR subroutine call. *devspec* and *busspec* are I/O specifiers for a bus device and the bus itself. Refer to the BASIC/1000C reference manual for more information about HP-IB I/O specifiers. There is no BASIC/1000C equivalent for the second form of the call, so use the subroutine call as shown above.

## RMOTE

Many instruments can be controlled either locally, by their front panel, or remotely, via the HP-IB. The RMOTE subroutine sets all listening devices equipped with remote/local switching to the remote mode. Instruments with remote/local switching that are not addressed enter the remote mode when they are addressed and the REN line has been asserted.

The RMOTE subroutine can use automatic addressing or it can be a universal bus command, so it can take either of these forms:

RMOTE(*devicelu*)

or

RMOTE(*bushu*)

In the automatic addressing form, RMOTE sends the REN bus command by setting the REN line true, unaddresses all devices on the bus, and addresses the device specified by *devicelu* to listen, which puts it in the remote mode, if it has remote/local switching.

In the universal bus command form, RMOTE simply sets REN true on the bus at *bustlu*. Any listening devices with remote/local switching are set to the remote mode.

Any remote/local devices addressed after REN is set by either form of this call enter the remote mode.

The BASIC/1000C REMOTE statement performs the same functions as the RMOTE subroutine calls. The forms of the REMOTE statement are shown below:

```
REMOTE devspec
```

*or*

```
REMOTE busspec
```

## **GTL**

The GTL subroutine temporarily sets addressed remote/local devices to the local mode while REN is true. It is usually used to give front panel control of a single device to the operator of a test system so the operator can adjust a test parameter. The GTL subroutine can use automatic addressing or it can send the GTL command to previously-addressed devices, so it has two forms:

```
GTL(devicelu)
```

*or*

```
GTL(bustlu)
```

The automatic addressing form unaddresses all devices on the bus, addresses the device at *devicelu* to listen, and transmits the GTL bus command. The second form simply transmits the GTL command. Any remote/local devices addressed to listen enter the local mode. The next time that the device set to local by GTL is addressed, it returns to the remote mode, because REN is still true.

A form of the BASIC/1000C LOCAL statement performs the function of the first form of the GTL subroutine. To perform this function, the LOCAL statement takes this form:

```
LOCAL devspec
```

There is no BASIC/1000C statement to perform the second form on the GTL bus command, so use the subroutine call shown above.

## **LLO**

Remote/local devices often have a switch marked LOCAL on their front panel. It is used to manually return control of the device to the local mode. To prevent this manual control, the LLO message subroutine transmits the LLO bus command message. The LLO subroutine is a universal bus command only, so it has only one form:

```
LLO(bustlu)
```

The universal LLO message that is transmitted by LLO locks out the local mode of any remote/local devices capable of responding to this universal command. No other instruments are affected. It can be cleared only by the LOCL subroutine and a form of the LOCAL statement in BASIC/1000C, which release the REN line. LOCL and this form of LOCAL are described in the next section.

In BASIC/1000C, the local lockout function is performed by the LOCAL LOCKOUT statement, shown below:

```
LOCAL LOCKOUT busspec
```

## LOCL

Once the LLO message has been transmitted, no device that responds to LLO can be manually returned to the local mode until the lockout has been cleared. The universal bus command subroutine LOCL clears the lockout by releasing the REN line. All remote/local devices respond by returning to the local mode, and clearing the lockout. Devices which do not respond to the GTL subroutine can respond to LOCL.

```
LOCL(bushu)
```

The REN line is set false. Any devices with remote/local switching, listening or not listening, enter the local mode, clearing the lockout.

The function of the LOCL subroutine is performed by a form of the BASIC/1000C LOCAL statement. It is shown below:

```
LOCAL busspec
```

## TRIGR

It is often necessary to trigger several devices to act at the same instant, as in a test system, where a group of measurement devices must be activated at once. Many bus devices can be programmed to perform their function when they receive the GET bus command. Not all devices can be triggered by GET, so check the device manuals for the instruments. The TRIGR subroutine uses two forms to send the GET message. The TRIGR subroutine calls have the following forms:

```
TRIGR(devicelu)
```

*or*

```
TRIGR(bushu)
```

The automatic addressing call unaddresses all devices on the bus, addresses the instrument specified by *devicelu* to listen, and sends the GET message, which triggers the addressed device, if it has triggering capability. The second form simply transmits the GET message, which triggers all addressed devices with triggering capability.

BASIC/1000C programs can trigger devices with the TRIGGER statement, which performs the same function as the automatic addressing form of the TRIGR subroutine. The TRIGGER statement is shown below:

```
TRIGGER devspec
```

There is no BASIC/1000C statement to perform the second form of the TRIGR subroutine call, so call the subroutine as shown above.

## ABRT

The ABRT subroutine gives you three ways to reset the bus. All three take the universal bus command form. The ABRT calls are shown below:

```
ABRT(bushu , 1 )
```

```
ABRT(bushu , 2 )
```

```
ABRT(bushu , 3 )
```

When the second parameter is 1, ABRT sends the IFC command by pulsing the IFC line, which halts all bus traffic.

When the second parameter is 2, ABRT sends the IFC message and the universal device clear (DCL) message.

When the second parameter is 3, ABRT unaddresses all talkers and listeners by sending the untalk and unlisten (UNT, UNL) address messages.

Two of the three ABRT subroutine functions can be performed by BASIC/1000C statements. The ABORT statement is equivalent to the ABRT subroutine call with the second parameter equal to 1. It is shown below:

```
ABORT busspec
```

Two BASIC/1000C statements are necessary to perform the functions of the ABRT subroutine call when the second parameter equal to 2. The statements are shown below:

```
ABORT busspec  
CLEAR busspec
```

The third form of the ABRT subroutine is not implemented in BASIC/1000C as a statement. Use the ABRT subroutine call with the second parameter equal to 3.

## Device Communication Subroutines

The device communication subroutines transmit address strings to select a talker and one or more listeners for subsequent data transfers. If the talker or listener is the controller, the subroutines also transmit or receive the data in the subsequent transfers. If the controller is not the talker or a listener, the subroutine only transmits the address strings.

Table 4-6 lists the device communication subroutines and statements, and describes the function of each.

**Table 4-6. Statements and Subroutines to Send and Receive Bus Data**

| <b>Subroutine<br/>or<br/>Statement</b> | <b>Function</b>  |
|--|--|
| CMDW                                   | Write commands and data to the bus. (ASCII)  |
| SECW                                   | Secondary write for FORTRAN and BASIC/1000C programs or secondary write of integer data from BASIC/1000D. (Binary) |
| SECWR                                  | Secondary write of real data from BASIC/1000D. (Binary)  |
| CMDR                                   | Read commands and data from the bus. (ASCII)   |
| SECR                                   | Secondary read for FORTRAN and BASIC/1000C programs or secondary read of integer data from BASIC/1000D. (Binary)   |
| SECRR                                  | Secondary read of real data from BASIC/1000D. (Binary)   |
| WRITE                                  | Write data to the bus from a FORTRAN program. (ASCII)  |
| PRINT                                  | Write data to the bus or an HP-IB device from a BASIC program. (ASCII)   |
| READ                                   | Read data in FORTRAN or BASIC/1000D programs from the bus or an HP-IB device. (ASCII)                              |
| ENTER/ INPUT                           | Read data from the bus or an HP-IB device to a BASIC/1000C program.  |
| IOCNT                                  | Return number of bytes/words in last data transfer.  |

## CMDW/CMDR

The CMDW and CMDR subroutines always use direct addressing. They have the following form:

```
CMDW(buslu, addr_cmd,datawrite)
```

```
CMDR(buslu, addr_cmd,dataread)
```

*addr\_cmd* – a string variable or literal in BASIC/1000D, a BASIC/1000C string variable with the string length packed in the first word, or a BASIC/1000D-compatible string in FORTRAN, containing the ASCII-encoded addresses of bus devices, and often bus commands.

The addressing part of the string should begin with the unlisten address, which removes all addressed listeners from the bus prior to selecting new listeners. Addressing a new talker automatically untalks any addressed talker on the bus. The talker can be any device on the bus with talk capability. If no commands are to be transmitted, then *addr\_cmd* contains only addressing characters.

*addr\_cmd* – can also contain bus commands. When no addressing is required, the string consists only of bus commands. The bus commands can be addressed or universal commands. The advantage of using CMDR/CMDW to send commands, rather than the device control subroutines, is that you can specify a group of listeners and commands for those listeners in a single call.

*datawrite* – a BASIC/1000D string variable or literal, a BASIC/1000C string variable with the string length packed in the first word, or a BASIC/1000D-compatible string in FORTRAN, containing data to be sent to the listeners addressed by *addr\_cmd*, if the controller is the addressed talker. If the controller is not the addressed talker, *datawrite* = 0.

*dataread* – the name of a BASIC/1000D string variable, a BASIC/1000C string variable with the string length packed in the first word, or a BASIC/1000D-compatible string variable in FORTRAN.

*dataread* is a buffer that lets the controller receive data from addressed talker. The ASCII EOR convention is used, and no more than 255 bytes can be read. If the controller is not addressed as a listener, *dataread* = 0.

---

### Note

A “BASIC/1000D-compatible string” is an integer array containing the character count in the first word, and packed ASCII in subsequent words. The HP-IB subroutines expect string variables to appear in this form. Figure 4-2 shows a BASIC/1000D-compatible string.

---

To build a BASIC/1000D-compatible string in BASIC/1000C, create a variable of type CHARACTER, and append the string length as the first word of the variable before passing the string to an HP-IB subroutine.

When calling CMDR from FORTRAN, BASIC/1000D or BASIC/1000C, the character count of *dataread* must first be set to the maximum expected length of the string to be read, expressed in characters.

In FORTRAN, the first word of the array is simply assigned the length of the expected character string.

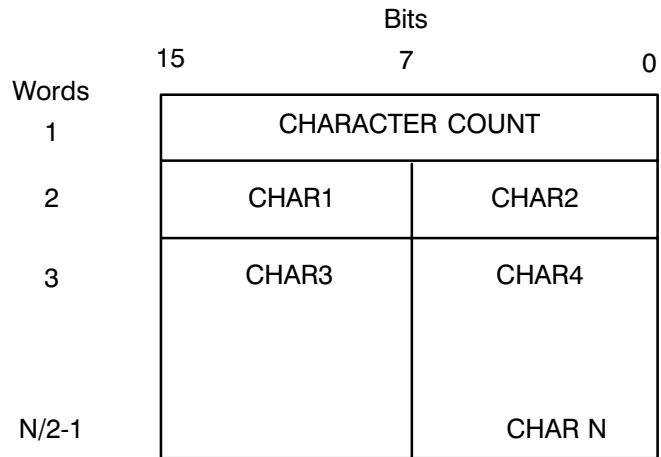


In a call to CMDR from BASIC/1000D, the first word of *dataread* is set to the expected string length by filling *dataread* with as many blanks as the expected length. The following example fills a *dataread* variable named D\$ with 20 blanks to set the character count to 20. D\$ can then be used to receive data in a CMDR subroutine call.

```

100 D$=" "
110 CMDR(LU,A$,D$)
100 BLK$(20,D$)
or
110 CMDR(LU,A$,D$)

```



**Figure 4-2. BASIC/1000D-Compatible String**

In BASIC/1000C, the string length is “packed” into a CHARACTER variable with the PACK USING and PACKFMT statements, as shown in the following example:

```

100 Length = LEN(String$)
110 PACK USING Format; Dataread$
120 Format: PACKFMT Length,String$[1,Length]

```

Line 100 sets Length to the current length of String\$, a string variable long enough to contain the longest expected data message. Line 110 packs the variables listed in line 120 (identified by its label “Format”) into Dataread\$. After the statement is executed, Dataread\$ contains the length of String\$ in the first word, and String\$ in the remaining words. Dataread\$ can then be used to receive data in a CMDR subroutine call.

## **SECW/SECR/SECWR/SECRR**

The secondary addressing subroutines always use automatic addressing. Secondary addressing lets you access device subsystems and internal registers or ports.

SECW and SECR send and read integer or real data in FORTRAN or integer data in BASIC/1000D. SECWR and SECRR are used in BASIC/1000D programs only, to send and read real data. SECWR and SECRR are used because of the way that real data is represented in BASIC/1000D. The secondary addressing subroutines have the following forms:

SECW ( *devicelu,secadd,datawrite,length* )

SECR ( *devicelu,secadd,dataread,length* )

SECWR ( *devicelu,secadd,datawrite,length* )

SECRR ( *devicelu,secadd,dataread, length* )

*secadd* is the secondary address in the device at *devicelu*.

*length* is a positive number of words or a negative number of characters to be sent from *datawrite* or read into *dataread*, or 0 if the subroutine is being used to address a secondary device address to talk or listen without sending or reading any data.

Secondary addressing in FORTRAN 4X and FORTRAN 7X programs is performed by the WRITE and READ statements directly, without the need to call SECW and SECR. Refer to the Bus Addresses section of this chapter and the FORTRAN 4X or FORTRAN 7X language manuals.

Secondary addressing in BASIC/1000C is performed by including the secondary device address in the special HP-IB form of the ASSIGN statement. The I/O specifier in the ASSIGN statement then points to a secondary address of a bus device. Refer to the HP-IB programming chapter of the *BASIC/1000C Reference Manual*.

## **READ/ENTER/INPUT/WRITE/PRINT**

The syntax of these BASIC and FORTRAN statements are defined in the language manuals, but it is important to remember that these statements use automatic addressing only, so specify the device LU, not the bus LU. In FORTRAN 4X and FORTRAN 7X, WRITE and READ can perform secondary addressing, if necessary. In BASIC/1000C, I/O specifiers are used to specify a device LU or a secondary address in the ENTER and PRINT statements. The BASIC/1000C INPUT statement can be used only for device LUs without secondary addresses.

## IOCNT

With IOCNT, you can retrieve the length of the last data message sent to or received from the bus or a device. IOCNT can also return the length of the last message transferred to or from the bus LU. The IOCNT call has two forms:

$$I = \text{IOCNT}(\textit{devicelu})$$

*or*

$$I = \text{IOCNT}(\textit{buslu})$$

In the automatic addressing form, the variable  $I$  receives the positive number of words or negative number of bytes transferred in the last message sent to or received from the device at *devicelu*. In the second form, the variable  $I$  receives the positive number of words or negative number of bytes transferred in the last message sent to or read from the bus at *buslu*.

# Managing Service Requests

---

## Introduction

This chapter is divided into three major sections. In the first, the interrupt and device status HP-IB library subroutines are described in a format similar to the subroutine descriptions in Chapter 4. The second section describes serial and parallel polling, and how each is used automatically by the controller or in a bus control program. In the last section, the first two sections are brought together to show how to use the subroutines to set up automatic or programmatic polling, and how each device on the bus can request service from the controller by sending the service request bus command (SRQ) or by responding to a parallel poll initiated by the controller.

## Interrupt and Device Status Subroutines

The controller is programmed to perform serial or parallel polling and to respond to positive poll results by the HP-IB interrupt and device status subroutines, listed in Table 5-1.

The SRQ bus command is the only command that does not originate at the controller. Devices on the bus interrupt the controller with the SRQ command to let it know that they require service. The controller can respond by polling the devices to find the interrupting device, and by scheduling a service program or routine to process the interrupt.

Devices can also request service by responding to a parallel poll, if they have been configured for parallel polling. Again, the controller can respond by scheduling a service program or routine when it determines what device requires service.

**Table 5-1. Interrupt and Service Subroutines and Statements**

| Message         | Subroutine                 | Subroutine Function   |
|-----------------|----------------------------|---|
| Require Service | SRQ                        | Activates or disables a service request program for the bus or for a device on the bus. If active, the program is automatically scheduled when its associated device requests service.                                  |
|                 | SRQSN                      | For BASIC/1000D programs only: activates a service request trap entry.  |
|                 | PPSCH<br>(RTE-A only)      | Activates or disables a parallel poll program for a device: system software automatically schedules an active program when the device requests service.   |
|                 | PPSN<br>(RTE-A only)       | For BASIC/1000C/1000D programs only: sets up a parallel poll entry.   |
| Status Byte     | STATS                      | Returns a byte of status information from a specific device, or returns a byte of status information from a designated HP-IB interface card. NOTE: Aborts current transfer if sent to <i>buslu</i> in RTE-A.            |
|                 | BSTAT<br>(RTE-A only)      | For BASIC/1000D programs only: retrieves actual interrupt status of device requesting service.  |
| Status Bit      | PPOLL                      | Enables a device to respond to a parallel poll and configures its response; disables a device or group of devices from responding to a parallel poll; or resets all parallel poll devices to a predetermined condition. |
|                 | PSTAT<br>(E/F-Series only) | Returns up to eight bits of information from up to eight responding devices or groups which were previously parallel poll enabled.  |

## SRQ

The SRQ subroutine sets up service programs to run when SRQ interrupts are received from the bus. The SRQ subroutine for RTE-6/VM systems is similar to the PPSCH subroutine for A-Series systems, except that the SRQ subroutine sets up the bus for an automatic serial poll in response to an interrupt, instead of relying on continuous parallel polling as in the A-Series systems.

Each device on the bus can have its own service program, or one service program can respond for more than one device. The call has the following form in BASIC and FORTRAN programs:

```
SRQ ( lu,value,programe )
```

*lu* is the LU number of a device on the bus, or the LU number of the bus itself.

*value* is an integer number to be passed to *progrname*. This number can be any value you want to use as an indicator to the service program, except that the value  $-1$  directed to the bus LU turns off all interrupts, and a value other than  $-1$  directed to the bus LU turns interrupts back on.

*progrname*, the name of the service program, must be a BASIC/1000D string variable, a BASIC/1000D-compatible string in a FORTRAN program, or a BASIC/1000C string with the string length packed in the first word. This is the name of the service program to be scheduled if the device at *lu* sends an SRQ interrupt. If *progrname* is zero in the call, automatic request servicing is disabled for the device.

Note that if the service program has been disabled ( $SRQ(lu, value, 0)$ ), you will not be able to turn interrupts off by setting *value* not equal to  $-1$ . Therefore, when disabling service programs, you must first turn off interrupts to that LU or you can use the following call (where *value* and *progrname* are set to 0) which turns off interrupts and disables the service program:

```
CALL SRQ(lu, 0, 0)
```

If *progrname* does not have an ID segment, an error occurs when the SRQ call is made. Chapter 6 describes error processing.

SRQ program scheduling can only schedule programs that are not in a session environment. If an SRQ-scheduled program attempts to access a cartridge that is attached to a session, a “cartridge not mounted” error will occur. If you need to access a cartridge, the cartridge should be a system cartridge as opposed to a session cartridge.

As soon as you configure a service program by calling SRQ you enable the bus interface card to interrupt on SRQ from any device on the bus. Also, the call to SRQ sets up automatic response to the interrupt. The HP-IB driver automatically conducts a serial poll of all bus devices that have been assigned service programs. The serial poll finds the device that interrupted, and schedules the service program that was set up for that device. You will find details under Serial Polling and Service Programs, later in this Chapter.

## SRQSN

Service requests can be processed within BASIC/1000D programs without suspending the program and scheduling a service program by the SRQSN subroutine and a TRAP statement. The SRQSN subroutine can only be used in BASIC/1000D, which must have TRAP capability. SRQSN associates a device LU with one of the 16 TRAP numbers available in BASIC/1000D. SRQSN has the following form:

```
SRQSN( devicelu , trapnum )
```

*trapnum* must be a number between 1 and 16, and it must not be associated with any other device. For more information on the TRAP statement and its use with SRQSN, refer to the third section of this chapter. Automatic request servicing can be disabled by setting *trapnum* equal to 0.

Service request response within a BASIC/1000C program is set up by the ON INTR and ENABLE INTR statements, as shown below:

```
ON INTR devspec direction  
ENABLE INTR devspec
```

*devspec* is an I/O specifier for a bus device, assigned by the ASSIGN statement. Refer to the *BASIC/1000C Reference Manual*, part number 92857-90001, for more information on ASSIGN.

*direction* is a GOSUB, CALL, or GOTO that directs program flow to a service subroutine or subprogram.

The first line associates the device at *devspec* with the *direction*. The second turns on interrupt processing. When the device at *devspec* asserts SRQ, the controller conducts a serial poll to find the requesting device. When it finds the requesting device, it looks up the associated *direction* and executes it.

Automatic service request processing is disabled in BASIC/1000C with the OFF INTR statement, described in the last section of this chapter.

## PPSCH

Available only with A-Series systems with BASIC/1000D, PPSCH is similar to SRQ, except that it enables the controller to schedule a service program when the continuous parallel polling feature of the HP 12009A Interface Card receives a positive response.

PPSCH is used with the PPOLL subroutine to set up automatic response to a positive parallel poll by scheduling a service program to process the request. The PPSCH call can appear in FORTRAN or BASIC programs, and has the following form:

```
PPSCH ( devicelu, value, progrname )
```

*devicelu* is the LU number of a device which can assert SRQ.

*value* is an integer to be passed to *progrname*. The exact use of *value* is up to the programmer.

*progrname* is a BASIC/1000D string variable, a BASIC/1000D-compatible string in a FORTRAN program, or a BASIC/1000C string with the string length packed in the first word. *progrname* contains the name of a service program to be scheduled if the device at *devicelu* requests service by asserting SRQ on the bus. If *progrname* is replaced with zero, automatic request servicing is disabled for the device.

If *progrname* does not have an ID segment, an error occurs when the SRQ call is made. Chapter 6 describes error processing.

In BASIC/1000C, PPSCH is used with the PPOLL CONFIGURE statement, described later in this chapter.

## PPSN

Also limited to A-Series systems with BASIC/1000D, PPSN is similar to SRQSN, except that it enables the controller to jump to a BASIC/1000D service subroutine when the HP 12009A continuous parallel polling receives a positive response.

PPSN is used with the PPOLL subroutine and a BASIC/1000D TRAP statement. PPSN associates a device with a BASIC/1000D trap number. PPOLL associates the device with a DIO line for parallel poll response, and the TRAP statement associates the trap number with a service subroutine.

When the device requests service by responding to a parallel poll, the associations created by PPSN, PPOLL, and TRAP let the controller find and branch to the service routine. PPSN is used in BASIC/1000D programs only. The PPSN call has the following form:

```
PPSN ( devicelu, trapnum )
```

The arguments are the same as for SRQSN.

## STATS

A FORTRAN or BASIC/1000D program can conduct a serial poll to determine if any device requires service and to decide how to service requests by calling the STATS routine. STATS reads a status byte from a bus device just as the controller does in a serial poll. If bit 6 of the status byte for a device is set, the device requires service. The other seven bits of the status byte are device-dependent. The device that is requesting service releases SRQ when the STATS call reads its status byte. The STATS call has the following forms:

```
STATS ( devicelu, status )
```

*status* is a variable to receive the status byte from the device at *devicelu*.

On E/F-Series systems, STATS can be called to read the status of the HP 59310B Interface Card, by specifying the bus LU rather than a device LU. On A-Series systems, a STATS call to the bus LU aborts the current bus operation without returning any information.

BASIC/1000C can conduct a serial poll with its SPOLL function. SPOLL reads the status byte from a device just as the controller does during a serial poll. As with STATS, when SPOLL reads the status byte from a device that requires service, the device releases SRQ. The SPOLL function call is shown below:

```
status = SPOLL ( devspec )
```

Like STATS, SPOLL can read the status of the HP 59310B Interface Card in an E/F-Series system, by specifying *busspec* instead of *devspec*. A SPOLL of the bus LU on an A-Series system merely aborts the current bus operation, without returning any information.

## BSTAT

Service subroutines in BASIC/1000D programs must check the status of the interrupting device to service the request properly. BSTAT retrieves the status of the requesting device as it was stored by BASIC/1000D. BSTAT has the following form:

```
BSTAT ( trapnum, value, status )
```

The status of the device associated with *trapnum* is returned to *status*. The variable *value* returns the integer passed by the last call to SRQ or PPSCH, or zero, if SRQSN or PPSN have been called since.



## GETINTR

In BASIC/1000C, the GETINTR function returns the status byte from a specified device, so the program can determine how to process the request. The GETINTR function has the following syntax:

```
GET INTR(devspec)
```

GETINTR returns a decimal integer representation of the device status byte stored as a result of the most recently-detected interrupt. If no ON INTR statement has been executed for the device or if the device never interrupted, GETINTR returns zero.

## PPOLL

PPOLL has three functions, selected by the second argument of the call.

When the second argument is 1, PPOLL enables parallel polling and configures one device to respond to parallel polling by setting one of the DIO lines to logic 1 or logic 0.

When the second argument is 2, PPOLL disables parallel polling for one or more devices.

When the second argument is 3, it unconfigures all devices that are configured to respond to parallel polling.

### Configure One Device

PPOLL can take two forms to configure a device for parallel polling response; they are:

```
PPOLL(devicelu,1, dataline)
```

or

```
PPOLL(buslu,1, dataline)
```

In the automatic addressing form, PPOLL unaddresses all devices on the bus, addresses the controller to talk and the instrument at *devicelu* to listen, and configures the device to request service via the DIO line identified by *dataline*. If the value of *dataline* is positive, the device requests service by setting the line to logic 0. If *dataline* is negative, the device requests service by setting the line to logic 1. The controller then enables parallel polling for the device.

In the second form, PPOLL configures any listening device to request service via the DIO line identified by *dataline*, and enables parallel polling for the device. The interpretation of *dataline* is the same as the automatic addressing form.

Only one device should be configured to respond on each DIO line so the controller can know which device is asserting the line.

In BASIC/1000C, the automatic addressing form of the PPOLL configuration call is implemented by the statement shown below:

```
PPOLL CONFIGURE devspec; dataline
```

*dataline* is an integer. The lower three bits (bits 0 to 2) of *dataline* determine the DIO line for parallel poll response, and the fourth bit (bit 3) determines the response. If bit 3 is a 0, the device sets the DIO line to logic 0 to request service; if it is 1, the device sets the DIO line to logic 1.

The second form of the PPOLL subroutine is not implemented as a BASIC/1000C statement, so call PPOLL exactly as in FORTRAN or BASIC/1000D.

---

**Note** Disk drives, tape drives, and some printers respond to parallel polling according to the “Amigo standard.” Refer to the “Parallel Polling and Service Programs” section of this chapter for a description of the restrictions on device configuration required by the Amigo standard.

---

## Unconfigure Selected Devices

The second function of the PPOLL subroutine is the disabling of one or more selected devices from parallel poll response. The PPOLL call for this second function takes the following forms:

```
PPOLL( devicelu , 2[ , 0 ] )
```

or

```
PPOLL( buslu , 2[ , 0 ] )
```

The third parameter is optional in FORTRAN, but required in BASIC. In the automatic addressing forms, PPOLL unaddresses all devices on the bus, addresses the controller to talk and the device at *devicelu* to listen, configures it to respond on a nonexistent data line, and disables parallel polling for the device.

The second form configures all listening devices to respond on a nonexistent data line, and disables parallel polling for those devices.

In BASIC/1000C, a single device is unconfigured by the PPOLL UNCONFIGURE statement shown below:

```
PPOLL UNCONFIGURE devspec
```

The parallel poll response configuration of the device at *devspec* is cancelled, and parallel polling is disabled for the device. There is no BASIC/1000C equivalent for the second form of the PPOLL unconfiguration call, so use the call as shown above (with the optional 0) to cancel configuration and disable parallel polling for all listening devices.

## Unconfigure All Devices

The third function of the PPOLL subroutine is unconfiguration of all devices that have been configured for parallel polling response. This form of the PPOLL call has the following syntax:

```
PPOLL( buslu , 3[ , 0 ] )
```

Again, the third parameter is required in BASIC, but optional in FORTRAN. This call unconfigures all parallel poll configured devices, whether addressed to listen or not.

This same function is implemented in BASIC/1000C as the PPOLL UNCONFIGURE statement shown below:

```
PPOLL UNCONFIGURE busspec
```

All devices configured for parallel polling are unconfigured by this statement. It looks very much like the single-device PPOLL UNCONFIGURE statement explained earlier in this section, except that a *busspec* is specified rather than a *devspec*. Make sure that you specify *devspec* to unconfigure one device, and *busspec* to unconfigure all devices.

## PSTAT

Available only on RTE-6/VM, the PSTAT subroutine performs a parallel poll of the bus from a BASIC or FORTRAN program. The subroutine returns the parallel poll status byte, which shows the levels of the eight DIO lines. The PSTAT call takes the following form:

```
PSTAT(buslu, stat)
```

The parallel poll status byte for the bus at *buslu* is returned to the variable *stat*.

In BASIC/1000C, the equivalent of the PSTAT subroutine is the PPOLL function, shown below:

```
status = PPOLL(busspec)
```

## Polling Types

Devices on the bus can interrupt the controller to request service by sending the SRQ bus command, or, on A-Series systems, by responding affirmatively to the automatic parallel poll.

On all RTE systems, devices can request service via the SRQ command. If either of the SRQ or SRQSN subroutines has been called and the controller receives an SRQ command, it automatically conducts a serial poll to find the requesting device. In a serial poll, the controller reads a status byte from each device that has been enabled to respond to serial polling, until it finds the requesting device. The controller then schedules a service program or routine.

On A-Series systems, the controller performs continuous parallel polling when it is idle. In a parallel poll, eight bus devices are asked to set one of the DIO lines to logic 1 or 0 to request service. If the controller receives a positive response from one of the devices, it schedules a service program or routine.

The following sections describe serial and parallel polling, and explain how to use the interrupt and device status subroutines in BASIC or FORTRAN programs to detect and process service requests.

## The Serial Poll

Devices on the bus send the Required Service message to the controller by asserting the SRQ control line. When the controller notices that SRQ has been asserted, it needs to know what device is requesting service so it can respond. If the controller has been set up for automatic request servicing by the SRQ subroutine, it begins its response by conducting a serial poll, as described in the following paragraphs. If the controller is not set up for automatic serial polling, the program can conduct a serial poll, as described in the Serial Polling by STATS section of this chapter.

### Automatic Serial Polling

An automatic serial poll begins when the controller sends the universal Serial Poll Enable (SPE) command, which sets all devices on the bus to the serial poll mode. The controller sequentially addresses each device that has been enabled for serial polling, and reads the status byte. After reading its status byte, the requesting device releases the SRQ line. If bit 6 of the status byte is 1, the controller knows that the device requires service. The controller sends the universal serial poll disable (SPD) command over the bus, which returns all devices to the normal mode.

If serial polling was enabled by the SRQ or SRQSN routines, then after the controller finds the requesting device, it looks up the name of the service program to be scheduled. In RTE-6/VM, the controller passes the EQT address and subchannel number of the requesting device, the status byte, and the value of the second parameter of the SRQ call to the service program. In RTE-A, the controller passes the LU number of the requesting device, the status byte, and the second SRQ parameter.

The service program can obtain the status byte of the requesting device by calling the RTE subroutine RMPAR. In RTE-6/VM, RMPAR also returns the HP-IB device address and the EQT address. The controller only checks bit 6 of the status byte. The service program should check the seven other device-dependent bits to decide how to process the request. Table 5-3 provides the possible status codes returned if the LU is the bus LU.

If serial polling was enabled by the SRQSN routine and a TRAP statement in a BASIC/1000D program, then after controller finds the requesting device, it schedules the trap setting program, which causes a branch to the service subroutine. The service subroutine should call BSTAT to obtain the status byte as it was recorded by the controller when the interrupt was trapped. The service subroutine should use the status byte to determine how to service the request.

If serial polling was enabled by the ON INTR and ENABLE INTR statements in a BASIC/1000C program, then after the controller finds the requesting device, it performs the direction given in the ON INTR statement. If the direction is branch to a subroutine, then control passes to the subroutine, which services the request. The service subroutine reads the device status with the GETINTR function, and uses it to determine how to service the request.

It is possible for two or more devices to request service at the same time. Upon completion of the scheduling of the interrupt handler for the first interrupt, the controller notices that SRQ is still asserted. It continues serial polling until it reads the status byte of the second requesting device, causing it to release SRQ.

If two or more devices use the same service program, and the program is busy when a second request is made, the second request is delayed 500 milliseconds (on an RTE-A operating system) or 1000 milliseconds (on an RTE-6/VM operating system) to give the program time to complete. If the service program is still busy after the delay on an RTE-A system, the second request is not processed. For this reason, service programs for multiple devices on those systems should complete within 500 milliseconds.

If the service program is still busy after the delay on RTE-6/VM system, the request is delayed again and retried after the delay. The delay-and-retry cycle continues until the program is available.

If the service program on an RTE-A system requires operator response, it will almost certainly take more than 500 milliseconds to complete. Write a short program to schedule the longer service program with queued no-wait schedule. The short scheduling program always completes within 500 milliseconds, so no service requests are lost.

## **Serial Polling by STATS**

If the controller is not set up to automatically conduct a serial poll to find the requesting device (if the SRQ and SRQSN subroutines have not been called), then the FORTRAN or BASIC/1000D program can use the STATS subroutine to conduct a serial poll. STATS reads the status byte from a device, as the controller would in an automatic serial poll: it sends the SPE message, addresses the device to talk, reads the status byte, and sends the SPD message. If the device was asserting SRQ, it releases it when polled. The program must examine the status byte as the controller does to determine if bit 6 is set. If it is set, the device is requesting service. The program should use the other seven status bits to determine how to process the request.

In BASIC/1000C, the SPOLL function performs serial polling from the program. The description of STATS, above, applies to SPOLL in BASIC/1000C.

## **The Parallel Poll**

Parallel polling allows devices to request service by asserting one of the eight DIO lines in response to a parallel poll request from the controller. By reading a single byte from the bus after the parallel poll request, the controller can determine if any device requires service.

The HP 12009A HP-IB Interface Card for A-Series systems performs continuous parallel polling when it is otherwise idle. If the card finds an asserted DIO line, it schedules the enabled service request program or the BASIC/1000D trap setting routine.

Not all devices respond to parallel polling. If the bus includes devices that do not respond, you can use a combination of parallel and serial polling to detect service requests. The devices that respond to parallel polling can request service by asserting a DIO line during a parallel poll. The devices that do not respond to parallel polling can request service by the SRQ message.

Only one device should be configured to respond on each DIO line, because the controller needs to know exactly what device is requesting service so it can pass the device EQT/subchannel or LU number to the service program or BASIC/1000D trap setting routine. When several devices respond on the same line, there is no way for the controller to know which device is asserting the line.

## **Automatic Parallel Poll Response**

The PPSCH and PPSN subroutines enable the controller to schedule a service program or the BASIC/1000D trap setting program when the HP 12009A Interface Card receives a positive parallel poll response. PPSCH associates a service program with a device LU. PPSCH can be called from FORTRAN, BASIC/1000D, or BASIC/1000C programs running under RTE-A. PPSN associates a BASIC/1000D trap number with a device LU. PPSN can be called only from BASIC/1000D programs running under RTE-A.

The PPOLL subroutine and PPOLL CONFIGURE statement are used with PPSCH and PPSN to configure a device to respond to parallel polling by asserting one of the eight DIO lines. PPOLL can be called from FORTRAN or BASIC/1000D. PPOLL CONFIGURE can be called only from BASIC/1000C.

The A-Series controller checks the DIO lines in ascending order, from DIO 1 to DIO 8, and stops looking for asserted lines as soon as it finds one, so DIO 1 has a higher priority than DIO 8. The E/F-Series controller checks the lines in descending order, from DIO 8 to DIO 1, so DIO 8 has a higher priority than DIO 1. If two devices assert lines simultaneously, the device on the higher-priority DIO line is serviced first. The second request is serviced as soon as the controller conducts the next parallel poll, usually within a few milliseconds.

When the controller finds an asserted DIO line in response to a parallel poll, it searches the linkages created by PPSCH or PPSN and PPOLL or PPOLL CONFIGURE to find the device associated with the asserted DIO line. The controller then schedules either the enabled service program for the asserted DIO line or the BASIC/1000D trap setting routine. The controller passes either the EQT and sub-channel number of the requesting device (RTE-6/VM) or its LU number (RTE-A) to the service program or BASIC/1000D trap setting routine. If PPSCH was used to enable a service program, then the controller also passes the second parameter of the PPSCH call to the program.

If more than one device uses a service program, and the service program is busy when a second request is made, the second request is delayed 500 milliseconds (on an RTE-A operating system) or 1000 milliseconds (on an RTE-6/VM system) to give the first time to complete. If the program is not complete after the delay, the request is not processed (RTE-A) or the delay-and-retry cycle repeats (RTE-6/VM) until the program completes. RTE-A service programs for multiple devices should, therefore, complete within 500 milliseconds.

If operator action is needed by an RTE-A service program, the service will almost certainly take more than a half-second to complete. In that case, the enabled service program should make a queued no-wait schedule of another program, and complete. While the called program services the request, the service program can begin to process the second request.

## Parallel Polling by PSTAT

If you want to conduct a parallel poll from a program, use PSTAT. PSTAT initiates a parallel poll and returns the status bit message (a byte containing the status of the eight DIO lines) to the program. The program should examine the status bit message to determine if any of the eight DIO lines are asserted. If any line is asserted, the program should service the interrupt. The program can use the STATS or SPOLL routines to read the status byte of the interrupting device to determine how to process the interrupt if the interrupting device has both parallel and serial polling capability.

In BASIC/1000C, the PSTAT subroutine is replaced by the PPOLL function. The PPOLL function conducts a parallel poll and returns the status bit message from the bus, just as PSTAT does. The description of PSTAT, above, applies to the BASIC/1000D PPOLL function.

## Service Requests in BASIC

With BASIC, you can choose to call an external service program upon receipt of SRQ or a positive parallel poll response, or to jump to a service subroutine within the program. Table 5-2 summarizes the statements and subroutines that you can use to select the service response best suited to the application.

### Serial Polling and Service Programs

If the bus control program is written in BASIC/1000D or BASIC/1000C, if you want the speed of an external service program, and if you want to use serial polling, then use the SRQ subroutine. SRQ associates the name of a service program with a device LU, and enables SRQ detection and serial polling. A typical call to SRQ looks like this:

```
100 SRQ(25,V,"SSERV")
```

The call associates the program name SSERV with the device at LU 25. If SRQ specifies a program that is not restored, an error is reported. The call also enables serial polling for the device at LU 25, and saves the value of V, to be passed to SSERV.

When the device at LU 25 asserts SRQ, the controller conducts a serial poll to find the requesting device. It reads the status byte of each device enabled for serial polling until it finds bit 6 of the status byte of the device at LU 25 active. It stops polling and schedules SSERV.

**Table 5-2. Polling Types and Service Language Options in BASIC**

| <b>Polling Type</b> | <b>Service Language</b>                      | <b>Subroutines/Statements</b>   |
|---------------------|--|---|
| Serial              | FORTTRAN, PASCAL,<br>or MACRO                | SRQ( <i>devicelu,value,programe</i> )   |
| Serial              | BASIC/1000D                                  | SRQSN( <i>devicelu,trapnum</i> )<br>TRAP <i>trapnum</i> GOSUB <i>subline</i>  |
| Serial              | BASIC/1000C                                  | ON INTR <i>devspec direction</i><br>ENABLE INTR <i>devspec</i>  |
| Parallel            | FORTTRAN,PASCAL,<br>or MACRO<br>(RTE-A only) | PPSCH( <i>devicelu,value,programe</i> )<br>PPOLL( <i>devicelu,1,dataline</i> )  |
| Parallel            | BASIC/1000D<br>(RTE-A only)                  | PPSN( <i>devicelu,trapnum</i> )<br>PPOLL( <i>devicelu,1,dataline</i> )<br>TRAP <i>trapnum</i> GOSUB <i>sublinenum</i> |

SSERV needs the status byte of the device to determine how to process the request. The status byte contains information about the device at the time that it requested service. The contents of the status byte varies between instruments. It is documented in the instrument manuals. The RTE library subroutine RMPAR can be used to retrieve the status byte and other parameters. These other parameters are operating system dependent.

The following lines from SSERV show how the RMPAR call returns those values:

```
FTN7X, L
PROGRAM SSERV
DIMENSION IPRAM(5)           Allocate a five-word array.

CALL RMPAR(IPRAM)           The call to RMPAR must be the first
.                             executable statement in the program.
.
.
service the request
.
.
CALL EXEC(6,0,1)           All service programs under RTE-A must terminate
                             "saving resources."
```



The operating system dependencies of the RMPAR return parameters are described as follows:

Under RTE-A:

```
IPRAM( 1 ) = device LU
IPRAM( 2 ) = value from SRQ call
IPRAM( 3 ) = status byte
IPRAM( 4 ) = 0
IPRAM( 5 ) = 0
```

If SSERV processes service requests from several devices or from devices on several buses, then the device LU helps SSERV identify the device to process the request correctly. Remember that service programs for multiple devices should complete within 500 milliseconds, or a second request that schedules the program could be left unserved.

Under RTE-6/VM, four parameters are available through RMPAR: the status byte, the HP-IB device address, the EQT address of the bus, and the value of the second argument to the SRQ call.

```
IPRAM( 1 ) = status byte
IPRAM( 2 ) = device address
IPRAM( 3 ) = EQT address
IPRAM( 4 ) = value from SRQ call
IPRAM( 5 ) = 0
```

If SSERV processes service requests from several devices, then the device address is necessary to service the request correctly. If SSERV processes requests from more than one bus, then the EQT address is also necessary, to identify the bus correctly. The status byte lets the program determine how to process the request. The use of *value* is determined by the application.

## Bus Service Program

If you are operating under RTE-IVB or RTE-6/VM, you can set up a bus service program in much the same way you set up device service programs. For example:

```
200 SRQ( 30 , V , "BSERV" )
```

In this example, 30 is the LU number of the bus interface card and V is a number to be passed to bus service program BSERV when BSERV calls system subroutine RMPAR. For the bus service program, the five words returned by RMPAR are:

```
IPRAM( 1 ) = bus status code
IPRAM( 2 ) = subchannel address of interrupting device
IPRAM( 3 ) = address of bus EQT
IPRAM( 4 ) = value from SRQ call
IPRAM( 5 ) = 0
```

The bus status codes (first parameter, above) are listed in Table 5-3. Your bus service program can report the error or take any other action appropriate to conditions on the bus.

**Table 5-3. Status Codes for Bus LU**

| Status Code   | Meaning                                    |
|---|--|
| 1   | I/O device timeout                         |
| 2   | IFC detected during I/O request            |
| 3   | SRQ aborted I/O request                    |
| 4   | Non-existent alarm program                 |
| 5   | Illegal I/O request                        |
| 6   | EQTX is full                               |
| * 10B   | Unclaimed SRQ interrupt                    |
| * 37B   | Timeout during SRQ serial poll disable     |
| * 40B+xx  | Timeout during SRQ serial poll address     |
| * 140B+xx   | Timeout during SRQ serial poll status read |
| 240B+xx   | Buffered request failed                    |
| 340B+xx   | E-bit not set and request failed           |
| <p>* For these events the driver disables the I/O card's SRQ interrupt capability, and runs the bus service program, if a service program has been configured (via subroutine SRQ) to run. In order to re-enable the interrupt capability of the I/O card, the bus service program must reschedule itself by calling the SRQ subroutine with a value other than minus one for the second argument of the call. Appendix D shows an example of such a program.</p> |  |
| <p>xx HP-IB address of device on the bus. Note that the error codes that contain the device address end in 40B, for which the least-significant five bits are zero. Use bit masking to strip the device address (1B–36B) out of the lower five bits.</p>  |  |

## Disabling SRQ Response

The second argument in the SRQ call (fourth parameter, Value, above) has special significance when the SRQ call is to the bus LU. If Value is minus one (–1), the call disables interrupts from the interface card. With interrupts thus disabled, you can use the SRQ call to set up service programs for bus devices without engendering unclaimed interrupts from devices that have not yet been set up. When all devices that can interrupt have had service programs assigned, a call to SRQ using the bus LU and a Value other than –1 will restore normal interrupt capability to the interface card.

Note that the unclaimed interrupt error and the timeout errors in themselves disable the SRQ capability of the bus. This is done because such an SRQ interrupt would otherwise remain in force, keeping the controller fully occupied answering interrupts just as if there were a new one each time it finished searching for (and not finding) the last.

The unclaimed interrupt occurs because the controller conducts its serial poll by polling only those devices that are enabled for serial polling. The devices that are enabled are those that have had service programs assigned to them. If no service programs have been set up, the controller will not poll the devices at all; therefore, any interrupt would result in an error 10B (see Table 5-3), and the controller would immediately disable the interrupt capability of the bus.

If you no longer want a particular service program to respond to service requests, use the SRQ call shown below:

```
520 SRQ(25,V,0)
```

Response to service requests from the device at LU 25 is disabled by this call to SRQ. The value of *v* is ignored. If the device at LU 25 generates a service request after response has been disabled, under RTE-6/VM, the bus LU reports an unclaimed interrupt and the interface card stops interrupting. Under RTE-A, an illegal interrupt error message is generated and all future service requests are ignored until the request is cancelled and service requests are enabled again (with the original SRQ call).

## Parallel Polling and Service Programs

If the bus control program is written in BASIC/1000D or BASIC/1000C, if you want the speed of an external service program, and you want to use parallel polling, then use the PPSCH subroutine. PPSCH is available only on A-Series systems that use the HP 12009A HP-IB Interface Card, because automatic scheduling of service programs for parallel polling is only available with the continuous polling feature of that card.

From BASIC/1000D, it takes two subroutine calls to establish automatic response by a service program to a positive parallel poll, as shown in the following lines of BASIC/1000D code:

```
100 D = 5
110 PPSCH(25,D,"PSERV")
120 PPOLL(25,1,D)
```

Line 100 sets the variable *D* equal to 5. *D* represents the DIO line on which the device is to respond. Line 110 associates LU 25 with the service program PSERV, and saves the value of *D* to be passed to PSERV. PSERV uses it to re-enable parallel polling after a service request. Line 120 enables parallel polling, and configures the device at LU 25 to respond to a parallel poll by setting DIO 5 to logic 0.

When you configure a device for parallel polling, remember that the DIO lines are assigned priority, from DIO 1 down to DIO 8 on an A-Series system, or from DIO 8 down to DIO 1 on an E/F-Series system. Only one device can be assigned to each DIO line, and if the service program has not been restored, the PPSCH call generates an error.

---

### Note

Disk drives, tape drives, and some printers respond to parallel polling according to the "Amigo standard." These devices are automatically configured to respond to parallel polling by their device drivers. Amigo standard devices respond to parallel polling by asserting the DIO line determined by the following formula:

$$\text{DIO line} = 8 - \text{device address}$$

A disk drive at device address 5, for example responds to parallel polling on DIO 3.

---

In BASIC/1000C, parallel polling response is configured by a call to PPSCH and a PPOLL CONFIGURE statement, as in the following lines from a BASIC/1000C program:

```
320 Dataline = 5
330 ASSIGN @Dev25 TO 25
340 CALL PPSCH(25,Dataline,"PSERV")
350 PPOLL CONFIGURE @Dev25;Dataline
```

Line 320 sets the variable *Dataline* equal to 5. Line 330 assigns LU 25 to the device I/O specifier @Dev25. Line 340 associates @Dev25 with the service program PSERV, and saves the value of Dataline to be passed to PSERV (PSERV uses it to re-enable parallel polling response after a service request). Line 350 configures @Dev25 to respond to a parallel poll by setting DIO 5 to logic 0, and enables parallel polling.

The PPOLL CONFIGURE statement interprets the lowest three bits of *Dataline* as the DIO line, and the fourth bit as the logic state (1 or 0) of a positive response. Because *Dataline* = 5, bits 0 to 3 represent DIO 5, and bit 4 is logic 0, so the device requests service by setting DIO 5 to logic 0. If *Dataline* = 13, then the device would respond by setting DIO 5 to logic 1 to request service, because bit 4 is logic 1.

If the device at LU 25 requires service, it sets DIO 5 to logic 0 when the controller conducts a parallel poll. The controller searches a table for the device configured to respond on DIO 5, and finds LU 25. It searches another table, finds the program PSERV, and schedules it.

PSERV should call STATS to read the status of the interrupting device, so it can determine how to service the request. The call to STATS clears the service request. PSERV should also call RMPAR, to read the device LU and the value of the second argument to the PPSCH call. When a device requests service via a parallel poll, it is disabled from responding to further polling. This is to prevent the device from reporting the same request twice. PSERV must call PPOLL to re-enable polling for the device so it can respond in the future. The following lines from PSERV show the calls to STATS, RMPAR, and PPOLL:

```
FTN7X,L
PROGRAM PSERV
DIMENSION IPRAM(5)

CALL RMPAR(IPRAM)
CALL STATS(IPRAM(1), ISTAT)
.
.
service the request
.
.
CALL PPOLL(IPRAM(1), 1, IPRAM(2)) Parallel polling response should be
re-enabled after the request is processed.
.
.
CALL EXEC(6, 0, 1) All service programs under RTE-A must
terminate "saving resources."
```

RMPAR returns the following values to IPRAM:

```
IPRAM( 1 ) = device LU
IPRAM( 2 ) = value from PPSCH call
IPRAM( 3 ) to IPRAM( 5 ) = 0
```

The STATS call needs the device LU, obtained by RMPAR, to retrieve the status of the requesting device and to clear its service request. PSERV interprets the status byte in ISTAT to determine how to process the request. Bit 6 of the status byte is set to logic 1 to indicate that the device requires service. The other seven bits in the status byte are device-dependant.

Like STATS, the PPOLL call needs the device LU obtained by RMPAR. It also uses the value of the second parameter of the PPSCH call, which is set in the BASIC/1000D and BASIC/1000C examples above to 5, to represent DIO 5, on which LU 25 responds to parallel polling. The PPOLL call should come after the request is serviced, to prevent the device from reporting the same request twice.

## Disabling Parallel Poll Response

When you no longer want a device to respond to parallel polling, call PPOLL as shown below:

```
600 PPOLL( 25, 270 )
```

This call prevents the device at LU 25 from responding to parallel polls. It does not alter the association of LU 25 with the service program.

In a BASIC/1000C program, parallel polling response by a device is disabled by the PPOLL UNCONFIGURE statement:

```
600 PPOLL UNCONFIGURE @Dev25
```

## Serial Polling in BASIC/1000D

Service requests can be processed without leaving the BASIC bus control program. In BASIC/1000D, the SRQSN subroutine and a TRAP statement cause the system to execute a serial poll upon receipt of a service request. When the controller finds the requesting device, it schedules the BASIC/1000D trap setting routine, which executes a branch to a service subroutine within the program. The SRQSN call and TRAP statement are available only in BASIC/1000D, and take the following form:

```
100 L = 25
110 T = 5
120 SRQSN( L, T )
130 TRAP T GOSUB 1000
```

When the device at LU 25 asserts SRQ to request service, the controller conducts a serial poll. It finds bit 6 of the status byte for the device at LU 25 asserted, so it stops the poll and schedules the BASIC/1000D trap setting routine. The routine searches the trap table for the trap number associated with LU 25. When it finds trap number 5, it executes a GOSUB to the service subroutine at line 1000.

---

**Note**

Simultaneous users of BASIC/1000D share a set of 16 trap numbers. Two BASIC/1000D programs that use traps simultaneously must cooperatively share the 16 trap numbers. Refer to *the BASIC/1000D Reference Manual*.

---

To decide how to process the service request, a BASIC/1000D service subroutine should call the BSTAT library subroutine. Unlike a service program, a BASIC/1000D service subroutine does not need to call RMPAR to determine which device caused the request, because each device can have its own service subroutine, and the subroutine has access to the same variables as the main program. The following example shows a call to BSTAT and the use of a variable from the main program in the subroutine:

```
1000 REM *** Service Routine for DVM
1010 BSTAT(25,V,S)
1020 PRINT "SRQ detected on device LU ";L;" Device status=";S
.
.
.
service the request
.
.
.
1190 RETURN
```

BSTAT returns the status byte of the device at LU 25 as it was recorded when the controller conducted the serial poll, and the value of the second argument of the last SRQ or PPSCH call. If the program does not contain a call to SRQ or PPSCH, or if the program has called SRQSN or PPSN to set up trapping, then the value returned to V by BSTAT is zero. The variable L, which identifies the LU that interrupted, is a variable from the main program used in the service subroutine.

## Serial Polling in BASIC/1000C

In a BASIC/1000C bus control program, the ON INTR and ENABLE INTR statements perform a function similar to SRQSN and TRAP in BASIC/1000D. The statements are shown below:

```
260 ASSIGN @Dev25 TO Devlu
270 ON INTR @Dev25 GOSUB 1080
280 ENABLE INTR @Dev25
```

Line 260 assigns LU 25 to the I/O specifier @Dev25. Line 270 associates the device at @Dev25 with the direction GOSUB 1080. Line 280 enables interrupt response for the device at @Dev25. If the device asserts SRQ, the controller conducts a serial poll. When it finds the requesting device, it schedules the BASIC/1000C interrupt routine, which searches for the direction associated with LU 25 (GOSUB 1080), and executes it.

BASIC/1000C service subroutines obtain the device status with the GETINTR function. GETINTR returns the device status as an integer. A BASIC/1000C service subroutine call to GETINTR is shown below:

```
1050 REM *** DVM service
1060 Devstat = GETINTR(Dvm)
1070 PRINT "SRQ detected at ";Devlu;" Device Status = ";Devstat
.
.
.
service the request
.
.
.
1220 RETURN
```

Serial poll response and service can be disabled as explained in the Disabling Serial Poll Response section, earlier in this chapter.

## Parallel Polling in BASIC/1000D

If you have an A-Series system and you want to use the continuous parallel polling feature of the HP 12009A Interface Card to detect service requests, and if you want to process requests in the BASIC/ 1000D bus control program, then use the PPSN subroutine with PPOLL and a TRAP statement to set up parallel poll response. PPSN is available only on BASIC in A-Series systems. The following lines from a BASIC/1000D program set up automatic parallel poll response by a service subroutine:

```
110 L = 25
120 T = 5
130 D = 3
140 PPSN(L,T)
150 PPOLL(L,1,D)
160 TRAP T GOSUB 1000
```

Line 110 sets L to 25 to represent LU 25; line 120 sets T to 5 to represent trap number 5; line 130 sets D to 3 to represent DIO 3. Line 140 associates LU 25 with trap number 5. Line 150 configures LU 25 to respond to a parallel poll by setting DIO line 3 to logic 0, and enables parallel polling for the device at LU 25. Line 160 associates trap number 5 with the service routine at line 1000.

If LU 25 asserts DIO line 3 during a parallel poll, the controller schedules the BASIC/1000D trap setting routine, which uses the linkages set up in the PPSN and PPOLL calls to find trap number 5 associated with the device at LU 25. It uses the trap number to find the service routine, and executes a GOSUB to line 1000.

---

**Note**

Simultaneous users of BASIC/1000D share a set of 16 trap numbers. BASIC/1000D programs that use traps simultaneously must cooperatively share the 16 trap numbers. Refer to the *BASIC/1000D Reference Manual*.

---

The service subroutine should call the STATS subroutine to read the status byte from the device at LU 25 and to clear its request. Parallel polling response can be disabled as explained in the Disabling Parallel Poll Response section, earlier in this chapter.

## Service Requests in FORTRAN

If the bus control program is written in FORTRAN, then all service requests can be processed by external service programs. The SRQ and PPSCH subroutines can be called from FORTRAN programs to establish the linkages necessary to schedule a service program in response to a service request. SRQ is available under any RTE operating system; PPSCH is only available on RTE-A systems. Since drivers can only schedule programs outside of session, cartridges should be mounted outside of session.

### Serial Polling Response

To associate a service program with a device LU, use the SRQ library subroutine. The call is the same as in BASIC, except that the word “CALL” is required, as shown in the following lines from a FORTRAN program:

```
DIMENSION IPROG(4)

DATA IPROG/5,2HSS,2HER,2HV /
IV=5

CALL SRQ(25,IV,IPROG)
```

The system response to this call and the SSERV subroutine are described in the Serial Polling and Service Programs section earlier in this chapter. Whether it is called from a FORTRAN or BASIC program, the linkages established by the SRQ subroutine are the same. The external service program (such as SSERV) is completely independent of the program that causes it to be scheduled, so it can service requests from FORTRAN or BASIC.

SRQ response can be disabled by calling SRQ with the value 0 in the place of IPROG. SRQ response disabling is described in more detail, in the Disabling Service Request Response section earlier in this chapter.



## Parallel Polling Response

If the system is an A-Series, then you can use the continuous parallel polling feature of the HP 12009A Interface Card and the PPSCH and PPOLL library subroutines to detect and process service requests from devices capable of responding to parallel polling. Not all devices can respond to parallel polling. For those devices, use serial polling, as described in the previous section. PPSCH enables parallel polling for a device and associates a service program with the device. A typical PPSCH call is shown in the following lines of FORTRAN code:

```
DIMENSION IPROG(4)

DATA IPROG/5,2HPS,2HER,2HV /
ILINE=-3

CALL PPSCH(25,IVAL,IPROG)
CALL PPOLL(25,1,ILINE)
```

The call to PPSCH associates the program PSERV with LU 25. PSERV must have an ID segment or an error occurs when PPSCH is called. The PPOLL call configures the device at LU 25 to respond to a parallel poll by setting DIO line 3 to logic 1 (ILINE = -3) in response to a parallel poll when it requires service. If the device requests service by setting line 3 to logic 1, the controller searches for the LU configured to respond on that line, and when it finds it, schedules the service program associated with the LU.

The function PSERV is described in the Parallel Polling and Service Programs section earlier in this chapter. Response to parallel polling is disabled as described in the Disabling Parallel Poll Response section, also described earlier in this chapter.

# Error Processing

---

## Introduction

There are two ways to check for HP-IB errors:

1. Perform periodic status requests from the bus control program, and jump to an error-processing program or subroutine when errors are discovered.
2. Let the operating system perform error checking and display error messages when errors occur.

This chapter explains the differences between these two error checking methods, describes possible errors, and explains how the system responds to each error with each of the two error checking methods.

## Error Processing by the Program

If the E-bit of the configuration word is set to logic 1, the program can detect and process HP-IB I/O errors. Refer to Chapter 7 for a description of the configuration word and the function of the E-bit. If the bus is unbuffered, the program can process I/O errors. If it is buffered, the program cannot process them.

If the bus is unbuffered and the E-bit of the configuration word is set to logic 1, then the program is ready to find and process errors. To detect errors, the program must read the error code after every transfer, by calling the IBERR subroutine. The IBERR function returns the error code from the last I/O transfer to or from an LU to the program. The IBERR call takes the following form:

```
I = IBERR(L)
```

Where *I* receives the error code from the last I/O transfer to or from LU *L*. If LU *L* is not configured for user error processing (if the E-bit of the configuration word of LU *L* is 0), the system aborts the program and displays the following message:

```
I/O NR LU lu E eqt 5 addr error code
```

The *error code* returned to I is defined as follows:

- †\* 0 = No error.
  - †\* 1 = Device timeout or transmission error detected.
  - † 2 = Illegal Interrupt.
  - † 3 = Data transfer aborted by SRQ message.
  - †\* 4 = Specified service program does not exist.
  - † 6 = Equipment Table overflow.
  - \* 20 = IFT extension space too small for poll table entry.
- 
- † Applies to RTE-6/VM systems
  - \* Applies to RTE-A systems

The errors are described in detail in the following paragraphs, with instructions for recovery. The system error messages shown in the descriptions are displayed only if the E-bit is set to 0.

## Error Detection by the System

If the E-bit of the configuration word is set to 0, the operating system automatically performs error checking and aborts the program if an I/O error occurs. Refer to Chapter 7 for a complete description of the function of the E-bit. In the error descriptions in the following section, the error messages appear only when the E-bit is set to 0.

Some errors cannot be processed by the program. The most common is specification of an illegal LU number in an HP-IB library subroutine call. The operating system aborts the program and reports the error with this message:

```
ILL HP-IB LU PROG ABORTED
```

### Device Timeout or Transmission Error

A timeout occurs when the controller addresses a device and does not receive a response within a preset period of time. For every LU or EQT on the system, there is a timeout value.

If the E-bit is not set, the RTE-6/VM system aborts the program and reports the device timeout with the following message:

```
I/O NR LU xx E xx S xx error code
```

RTE-A systems abort the program and display the following message when a timeout occurs and the E-bit is not set:

```
I/O TO LU xx
```

If the data transfer was carried out without DMA, only the first byte of the message is affected. If DMA is in effect, the entire message is affected. To recover from a timeout error, first make sure that the timeout value for the LU is long enough. Refer to the device instruction manual to

determine how long it takes to respond. If the timeout value is too short, change it. Refer to the *Operating System User's Manual* for instructions on changing timeout values. If the timeout value is correct, then make sure the device is connected properly to the bus, turned on, and is otherwise ready for operation. If the device is a printer, for example, make sure its "ON LINE" switch is on. When the device is ready, bring the LU up (with the CI or FMGR UP command), and try the transfer again.

If the E-bit is set, the program can respond by prompting the operator to perform these actions.

---

**Note** If you are using an HP 59401A Bus Analyzer, make sure it is not in the HALT mode. If it is, then device timeouts will almost certainly occur on bus data transfers.

---

## Illegal Interrupt

This message occurs if an error results during the processing of a service request. The illegal interrupt message displayed on RTE-IVB and -6/VM systems when the E-bit = 0 is shown below (*xx* = the select code of the bus):

```
ILL INT xx
```

RTE-A systems use the following message, with *xx* = the select code of the bus:

```
**ILL INT SCxx
```

This message can occur if one or more devices on the bus are enabled for SRQ response (by the SRQSN or SRQ subroutines), and another device, which is not enabled for SRQ response, asserts the SRQ line. All SRQ processing is disabled, and the message is displayed. To recover, reset the device to release the SRQ line, and re-enable SRQ response for the devices that were originally enabled.

If a device receives a serial poll request, but does not respond, the following message appears on RTE-A systems:

```
POLL ERR: NO RESP LU xx
```

Make sure the device has serial polling capability. This error can also occur if a service program specified in a SRQ or PPSCH call has been removed from the system since the call, and is not available to service a request, or on the second scheduling of a service program that did not terminate "saving resources" after the first scheduling (RTE-6/VM).

On RTE-A systems, the following message is displayed on the second call to a service program that did not terminate "saving resources" after the first scheduling:

```
POLL ERR: NO PROG LU xx
```

On RTE-A systems, the following message is displayed if the interrupt program on the HP-IB LU is busy:

```
POLL ERR: NO SCHED LU xx
```

## Data Transfer Aborted by SRQ Message

If the SRQ line is set while a data transfer is in progress, and if the S-bit of the configuration word (on RTE-6/VM systems only) is set to 1, then the data transfer is aborted and the service request is immediately processed. Because the S-bit cannot be set to 1 on RTE-A operating systems, interrupts never abort transfers, so this message never occurs. Most devices cannot recover from an aborted data transfer, so the S-bit should be set only when immediate service request processing is required, regardless of the cost of an aborted transfer. The ability to recover from an SRQ-aborted data transfer depends upon the devices involved in the transfer.

## Specified Program Does Not Exist

If a program calls the SRQ or PPSCH subroutines and specifies a non-existent program (one without an ID segment), this error occurs. Make the program available and try the call again. RTE-A systems abort the calling program and report the following message if this error occurs and the E-bit of the configuration word is not set:

```
program ABORTED IO07 addr
```

*program* is the name of the calling program; *addr* is the address at which it was aborted.

## Equipment Table Extension Overflow

This error occurs only on RTE-6/VM systems. It occurs the first time the program accesses a device when the EQT extension is full. The first time a device is accessed in an I/O statement or a message subroutine, it is configured in the system. If the EQT table extension is full, the device cannot be configured and this error occurs. This error can be prevented by making sure that the EQT table extension is large enough for all the devices on the bus.

## IFT Space Too Small for Poll Table Entry

On RTE-A operating systems, the following message is displayed if there is not enough room in the IFT extension space for another device to be enabled for serial polling. Refer to the system generation planning manual for recovery information.

```
IFT Space too small for poll table entry
```

## Error Processing Example

The program shown below sets the E-bit of the configuration word to 1, performs a data transfer, and checks the error code. If an error occurs, it branches to an error processing subroutine. If none occurs, it continues with the normal program flow.

```
FTN4X, L
  PROGRAM WRITR
  CALL CNFG(25,1,17400B)           Sets configuration word.

50  WRITE(25,500)                 Send a message to LU 25.
500 FORMAT("F3R5M3T3")

  IERR = IBERR(25)               Read the error code.

  IF(IERR.EQ.0) GO TO 100         If there is no error, go on with the program.
  .
  .
  .                               The error processing routines check
  process the error             IERR and respond accordingly.
  .
  .
  STOP                            Stop if error prevents further action.

100 CONTINUE                     Continue normal program.
  .
  .
  remainder of program
  .
  .
  END
```

Because the E-bit is set, the program does not abort if an error occurs. Refer to Chapter 7 for a full description of the function of the E-bit. The program reads the error code through IBERR, and either continues if no error is reported, or falls into an error processing routine if an error occurs. The error processing routine should perform a series of “IF(IERR.EQ.xx)” checks, with xx = possible error codes, to decide how to process the error.

# Controller Configuration

---

You can modify the way that the RTE-6/VM operating system transfers bus data, uses EOR, processes service requests, and handles error-checking by programming the configuration words for each device on the bus and for the entire bus. On RTE-A operating systems, the only characteristic that can be configured is how the system handles error-checking (the E-bit).

This chapter briefly defines the bits of the configuration word for RTE-6/VM operating systems, and the E-bit for RTE-A operating systems. For all these operating systems, the default values in Figure 7-1 show how they process interrupts, use DMA and EOR signals, and respond to errors.

## The Configuration Word

The eight bits of the configuration word which can be changed are listed below:

S = SRQ data transfer abort

0 = Disable or

1 = Enable driver to abort any current bus I/O request to process a service request.

R = Restart aborted I/O request

0 = Disable or

1 = Enable driver to attempt to restart a bus I/O request that was aborted by an SRQ message. The R-bit is ignored when S = 0.

D = DMA transfer

0 = Disable or

1 = Enable driver to use DMA for I/O transfers.

I = EOR signal indication (from device)

0 = Do not require or

1 = Require an EOR signal from the device at the end of a data transmission.

|    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| S  | R  | D  | I  | J  | O  | P | E | X | X | X | X | X | X | X | X |
| 0  | 0  | 0  | 1  | 1  | 1  | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |




  


Figure 7-1. Configuration Word Format with Default Value

J = EOR signal timing (from device)

0 = Expect an EOR signal from device after or

1 = Expect an EOR signal from device at the same time as the last byte of a data transmission.

O = EOR signal indication (from controller)

0 = Do not issue or

1 = Issue an EOR signal at the end of a data transmission.

P = EIO timing (from controller)

0 = Issue an EOR signal after or

1 = Issue an EOR signal at the same time as the last byte of a data transmission.

E = Error-detection abort

0 = Allow or

1 = Do not allow an error to abort current program. Let the program process the error.

## Service Requests – The S-Bit

The S-bit determines whether a device I/O transfer can be aborted when any other device on the bus asserts the SRQ line. If the S-bit of the configuration word for a device is set and the device is performing a data transfer, and another device on the bus asserts SRQ, the data transfer of the first device is immediately aborted. If the S-bit of the configuration word of the first device is not set, the HP-IB driver lets the data transfer complete before it responds to the SRQ message.

If there is a high-priority device on the bus that must have its service requests processed without delay, then set the S-bit in the configuration word of every other device on the bus. When the high-priority device requests service, any other bus traffic is aborted. If the high priority device must never be interrupted, do not set its S-bit, because any device can abort the current data transfer of a device with the S-bit set.

Few devices can easily recover from an aborted I/O request, so reserve setting the S-bit for cases where the cost of aborting the current data transfer is less than the cost of delaying SRQ response.

## Service Requests – The R-Bit

The R-bit is ignored when the S-bit is set to logic 0. When S and R are both set, the controller restarts the operation that was aborted to process a service request, or attempts to restart it. The R-bit should be set only on devices that can recover from an aborted request.

If the S-bit is set and R is not, or if both are set and the restart fails, then an error condition exists. Controller response to this condition depends on whether the bus is buffered or unbuffered and the status of the E-bit, described later in this chapter. If the bus is buffered, any error aborts the program. If it is unbuffered, the controller response depends on the setting of the E-bit.

Refer to the system generation records to determine if the bus is buffered or unbuffered. Buffering is a system generation option.



## Transfer Methods – The D-Bit

There are two ways to transfer information between the controller and the bus: the interrupt method and Direct Memory Access (DMA). The DMA option is selected by setting the D-bit in the configuration word for a device to 1, for RTE-6/VM systems only; RTE-A always uses DMA. The difference between the interrupt method and DMA is illustrated by the following example. A voltmeter takes a measurement and returns the following string to the controller:

```
+3.141593E+00 <cr><lf>
```

<cr> represents a carriage return/line feed (ASCII 12B and 15B).

There are two ways the controller can accept this information. The first is by interrupting the CPU for each word received:

```
i + 3i .1 i 41 i 59 i 3E i +0 i 0<cr> i <lf> i
```

*i* represents an interrupt. To transfer this short message, the CPU is interrupted nine times. The longer the message, the more times the CPU is interrupted. Because interrupts take processing time away from other tasks, this method is costly in terms of overhead, especially for transfers of long buffers. This is the interrupt method.

The second method generates only two interrupts:

```
i +3.141593E+00 <cr><lf> i
```

No matter what the length of the buffer, there are only two interrupts generated by this second method. There is no added overhead for longer buffers. This is Direct Memory Access (DMA).

## When to Use the Interrupt Method

Before you select DMA to increase transfer speed, consider how many other devices in the system are using DMA. There are only two DMA channels on an E/F-Series system, so delays can occur if there are a number of DMA requests at once. Consider also the types of devices that will use DMA. If DMA is used to transfer data to or from a very slow device, the channel can be tied up for a long time. DMA should be reserved for fast devices.

On an A-Series system, there is a DMA channel for each I/O card, so none of the following restrictions apply. Use the interrupt method for a device on an E/F-Series system under the following conditions:

- When it is likely that both DMA channels will be busy much of the time.
- If the device is slow, and would keep the DMA channel busy for extended periods.
- If the device has a short buffer (about a dozen characters). For short buffers, the cost of using a DMA channel is greater than the overhead savings.

## When to Use DMA

You can take advantage of the higher speed of a DMA transfer anytime on an A-Series system, or under the following conditions on an E/F-Series system:

- When there is little chance that the DMA channels will be often busy.
- When the device is medium- to high-speed, and has a long buffer.

The DMA method is selected by setting the D-bit of the configuration word to logic 1.

## End of Record – The I-, J-, O-, and P-Bits

The HP-IB driver must determine when the controller has finished sending a data message to a device and when a device has finished sending data to the controller. The end of record (EOR) requirements for most devices follow the default values of the configuration word. There are devices with special EOR characteristics, and for those devices, the I-, J-, O-, and P-bits can be changed.

The HP-IB has two kinds of EOR signals:

1. The ASCII codes *<crlf>* (12B and 15B).
2. The bus control signal EOI.

Devices that do not follow the default EOR settings can send and receive EOR signals differently. For that reason, the device and controller EOR characteristics can be altered for each device on an E/F-Series system.

## Device EOR Requirements

An HP-IB compatible device can require any of the following EOR conditions when data is sent to the device from the controller:

1. Known buffer length, and no EOR signal required (*<crlf>* or EOI).
2. Known buffer length, EOR signal sent at the same time as the last byte transferred (EOI only, since *<crlf>* must be sent over the data lines).
3. Known buffer length, EOR signal sent after the last byte transferred (either *<crlf>* or EOI).

Refer to the operating manuals of the bus devices to determine their EOR requirements. Table 7-1 lists the possible combinations of I, J, O, and P, and the EOR characteristics they represent.

## Controller EOR Requirements

The controller can expect data sent from devices to follow any of the following EOR conventions:

1. Known buffer length, EOR signal sent after the last byte of the data transfer (<crLf> or EOI).
2. Known buffer length, EOR signal transmitted at the same time as the last byte in the data transfer (EOI only, because <crLf> is transferred on the data lines).
3. Known buffer length, and no EOR signal (<crLf> or EOI).
4. Unknown buffer length, EOR signal transmitted after the last byte of the transfer (<crLf> or EOI).
5. Unknown buffer length, EOR signal transmitted at the same time as the last byte of the transfer (EOI only).
6. Unknown buffer length, no EOR signal (RTE-6/VM only).

**Table 7-1. The I-, J-, O-, and P-Bits – Combinations**

| I | J | O | P | EOR Characteristics                      |
|---|---|---|---|--|
| 0 | 0 | X | X | No EOR expected from device              |
| 1 | 0 | X | X | EOR expected from device after last byte |
| 1 | 1 | X | X | EOR expected from device with last byte  |
| X | X | 0 | 0 | No EOR sent to device                    |
| X | X | 1 | 0 | EOR sent to device after last byte       |
| X | X | 1 | 1 | EOR sent to device with last byte        |

The worst case of the above is number 6, because the controller has no way of knowing when the device has finished sending data. For such devices, the only way to determine when the device is finished is to wait a specified length of time after the device stops sending data, and assume it is finished. For all five of the other methods, the controller can be certain of the end of data. Table 7-1 lists the possible combinations of the I-, J-, O-, and P-bits, and the EOR characteristics they represent.

## Error Processing – The E-Bit

Controller response to HP-IB errors is determined by the setting of the E-bit. The E-bit is the only configuration bit that can be programmed in RTE-A systems. Refer to Chapter 6 of this manual for further description of error processing.

## Using the CNFG Subroutine

The HP-IB library subroutine CNFG lets the bus control program change the configuration word of the bus or any bus device. This section describes the use of CNFG in RTE-A systems to set and clear the E-bit; then it provides its more complete use in RTE-6/VM systems.

### Using CNFG Under RTE-A

The subroutine CNFG has four forms, for setting the E-bit of one device or the bus, or clearing it. The forms are shown below:

|  |                                   |
|--|-----------------------------------|
| CALL CNFG( <i>devicelu</i> ,1, <i>config</i> ) | <i>Set E-bit for one device</i>   |
| CALL CNFG( <i>devicelu</i> ,2[ , 0 ])          | <i>Clear E-bit for one device</i> |
| CALL CNFG( <i>buslu</i> , 1 , <i>config</i> )  | <i>Set E-bit for the bus</i>      |
| CALL CNFG( <i>buslu</i> ,2[ , 0 ])             | <i>Clear E-bit for the bus</i>    |

*devicelu* is the LU number of the device to be configured or unconfigured.

*buslu* is the LU number of the bus to be configured or unconfigured.

The second parameter determines whether this is a configuration or unconfiguration request. When the second parameter = 1, the configuration request sets the E-bit to the value of the config parameter. When the second parameter = 2, the unconfiguration request clears the LU flag.

*config* is 400B (decimal 256), to set the flag (enable program error handling), or 0 to clear it (disable program error handling); *config* must be expressed in decimal for BASIC.

[ , 0 ] is required only in BASIC. In FORTRAN, the unconfiguration request does not need a third parameter.

The two CNFG calls shown below clear the E-bit for the device at LU 25 (disable program error handling) except that the second one also disables interrupt programs:

|                         |  |
|-------------------------|--|
| CALL CNFG( 25 , 1 , 0 ) | <i>Disable program error handling</i>                            |
| CALL CNFG( 25 , 2 )     | <i>Disable program error handling and<br/>interrupt programs</i> |

The call shown below sets the E-bit for the device at LU 32 from a FORTRAN program:

```
CALL CNFG( 32 , 1 , 400B )
```

In a BASIC program, the third parameter must be expressed in decimal:

```
CALL CNFG( 32 , 1 , 256 )
```

## Using CNFG Under RTE-6/VM

There are four forms of the CNFG subroutine for changing the configuration word: the device configuration and unconfiguration forms, and the bus configuration and unconfiguration forms:

```
CALL CNFG(devicelu , 1 , config)           Configure one device
CALL CNFG(devicelu , 2 [ , 0 ] )           Unconfigure one device
CALL CNFG(buslu , 1 , config)           Configure the bus
CALL CNFG(buslu , 2 [ , 0 ] )           Unconfigure the bus
```

*devicelu*, the second parameter, *buslu*, and [ , 0 ] are described in the Setting and Clearing the LU Flag section, above.

*config* is the octal representation of the desired bit pattern of the configuration word for the device at *devicelu*. In BASIC, it is the decimal representation of the bit pattern.

The default configuration word value is 17000B or decimal 7680. Suppose you want to enable DMA (D-bit = 1) for the device at LU 25 without changing any other bits in the configuration word. The following FORTRAN call to CNFG shows how it is done:

```
CALL CNFG( 25 , 1 , 37000B )
```

The same setting is accomplished in BASIC with the following call, in which 37000B is expressed as decimal 15872:

```
CALL CNFG( 25 , 1 , 15872 )
```

The following call in a BASIC program unconfigures the bus at LU 15:

```
CALL CNFG( 15 , 2 , 0 )
```

In FORTRAN, the “ , 0 ” is not required, and the call takes this form:

```
CALL CNFG( 15 , 2 )
```

## RTE-6/VM Generation

---

### Introduction

This appendix describes RTE-6/VM generation for the HP-IB software, supplementing the information in the operating system programming and operating manual. Table A-1 lists the HP-IB software and the standard file names under which these software modules are stored on your disk for RTE-6/VM. The HP-IB Library subroutines can be called from BASIC and FORTRAN programs.

When using the Loader or Linker (LINK) to relocate an HP-IB application program, the HP-IB library (\$IB6A) should be searched, (not relocated) to satisfy the HP-IB subroutine calls. This is especially important if your system does not include the BASIC libraries, because unsatisfied externals may then result, which will abort the loading process. The following example shows the search being performed for \$IB6A:

```

: RU,LOADR           Run Interactive Loader
/LOADR: RE,%PROG    Relocate the Application Program
/LOADR: SE,$IB6A    Search $IB6A
/LOADR: EN

```

or, using LINK:

```
CI> link %prog $ib6a::libraries
```

If \$IB6A was specified as a library during system generation, it is automatically be searched by the loader.

**Table A-1. RTE-6/VM HP-IB Software**

| File Name | HP Part No. | Description                                   |
|-----------|-------------|---|
| %6DV37    | 92084-16592 | HP-IB Interface Driver with SRQ capability    |
| %6DA37    | 92084-16593 | HP-IB Interface Driver without SRQ capability |
| \$IB6A    | 92084-12036 | RTE-6/VM HP-IB Subroutine Library             |
| %SRQ*P    | 59310-16005 | BASIC SRQ/TRAP Service Program                |
| %BAMLB    | 92101-12002 | BASIC TRAP Tables                             |
| RINTR     | 92857-16128 | BASIC/1000C Interrupt Routine                 |

## Generation Procedure

The generation procedure given here follows the same sequence and format as that given in the operating system manuals.

### Program Input Phase

After selecting the HP-IB driver to be used, load the driver during system generation as described in the generation manuals for your system. The HP-IB Library can be loaded during this phase; this will eliminate having to load it with each user program during online loading. If BASIC/1000D is to be used, load the SRQ/TRAP program and the BASIC/1000D Memory Resident Library at this time also. The BASIC/1000D Subroutine Library can also be loaded during this phase. This avoids having to load it into a file and specify it during Branch and Mnemonic Table online generation.

### Parameter Input Phase

During the parameter input phase of system generation, make the following entries if using BASIC:

```
TTYEV, 17  
TRAP, 30
```

### Table Generation Phase

The table generation phase of system configuration is divided into three parts: Equipment Table entries, Device Reference Table entries, and Interrupt Table entries. Take the steps given in the following paragraphs to configure the HP-IB driver into the system.

## Equipment Table (EQT)

Make an EQT entry for each interface card as follows:

1. Determine the I/O slot for the HP-IB Interface Card and note the *select code* of the slot. Select codes are the numbers marked on the slots that run from octal 10 to octal 77. Select codes below 10 are used internally by the system.
2. Determine the number of EQT extension words required for your application. Calculate the number of extension words required as follows:

$$\text{Number of EQT extension words} = 7n + 27 \text{ (Maximum} = 255)$$

where  $n$  = the number of autoaddressable devices to be connected to the HP-IB. The maximum number of devices on a particular bus is 14; therefore, the maximum recommended number of EQT extension words is 118.

Be sure to include enough extension words to allow for adding devices to the system at a later date. Adding extension words requires regenerating the system, which is a more time-consuming procedure than simply adding the new information to available extension area.

3. Determine the maximum timeout value for the slowest device on the bus.  $T = xxxx$  in tens of milliseconds. Always supply a timeout value for each equipment table entry.

DMA is not specified at generation time, because the driver bases DMA selection on the configuration word, the device configuration word for autoaddressing and the bus interface card configuration word for direct addressing.

Make each equipment table entry as follows:

```
*EQUIPMENT TABLE ENTRY
.
.
EQT n?
sc , DVA37 , , T=xxxx , X=yy
```

where:

$n$  = the EQT number assigned by the RTE system.  
 $sc$  = the select code of the bus interface card.  
 $T$  = timeout.  
 $X$  = EQT extensions.

Repeat the above steps for each bus interface card in your system.



## Device Reference Table (DRT)

The Device Reference Table cross-references the device logical unit number (LU) to the EQT entries. Make the appropriate DRT entries as follows:

```
*DEVICE REFERENCE TABLE
.
.
lu=EQT#?
n,m
.
.
```

where:

*lu* = Logical unit number to be assigned to a bus device.  
*n* = EQT entry number you assigned to the bus interface card.  
*m* = EQT subchannel number = decimal device address.

Note the following:

1. Assign an LU number and subchannel 0 to the bus interface card. Subchannel 0 need not agree with the address set on the card; the driver uses it to enable the card to control the bus remote enable line.
2. For the auto-addressing mode, assign an LU number and a non-zero subchannel number to each bus device. Note that the software subchannel number of each device must equal its hardware address and must be a decimal number from 1 through 30.
3. Direct I/O and auto-addressing modes can both be used as long as EQT subchannel 0 is assigned to the bus itself and non-zero EQT subchannels are assigned to the individual bus devices.

## Interrupt Table

The Interrupt Table allows you to establish interrupt links that tie the select code to its EQT number. Make the table entries as indicated below:

```
*INTERRUPT TABLE
.
.
.
sc, EQT, n
.
.
```

where:

*sc* = the select code of the bus interface card.  
*n* = EQT entry number previously assigned to the card.

Repeat the table generation entries for each bus interface card in your system.

## RTE-6/VM BASIC/1000D

Configuration of the HP-IB Message Subroutine into BASIC is done by adding the subroutine calls to the Branch and Mnemonic Table. During subroutine table generation, enter the appropriate RTETG commands as listed below. If you are not using SRQ, omit SRQ and SRQSN.

The following RTETG commands are necessary to provide BASIC access to the HP-IB Message Subroutines. In order to use these subroutines and functions, their names must be entered into the Branch and Mnemonic Tables. Refer to the *System Configuration Manual for BASIC/1000D*, part number 92060-90016, for using RTETG to build the Branch and Mnemonic tables.

|                  |                          |                  |            |
|------------------|--------------------------|------------------|------------|
| TRIGR(I),        | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| CLEAR(I,I),      | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| RMOTE(I),        | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| GTL(I),          | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| LLO(I),          | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| LOCL(I),         | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| STATS(I,IV),     | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| PPOLL(I,I,I),    | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| PSTAT(I,IV),     | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| CNFG(I,I,I),     | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| ABRT(I,I),       | OV= <i>n</i> , SZ=4,     |                  | FIL=\$IB6A |
| HPIB(I,I,I),     | OV= <i>n</i> , SZ=4,     | ENT=HPIB,        | FIL=\$IB6A |
| CMDR(I,RA,RVA),  | OV= <i>n</i> , SZ=4,     | ENT=CMDR,        | FIL=\$IB6A |
| CMDW(I,RA,RA),   | OV= <i>n</i> , SZ=4,     | ENT=CMDW,        | FIL=\$IB6A |
| IBERR(I),        | OV= <i>n</i> , SZ=4,     | INTG, ENT=IBERR, | FIL=\$IB6A |
| IBSTS(I),        | OV= <i>n</i> , SZ=4,     | INTG, ENT=IBSTS, | FIL=\$IB6A |
| SRQ(I,I,RA),     | OV= <i>n</i> , SZ=4,     | ENT=SRQ,         | FIL=\$IB6A |
| SRQSN(I,I),      | OV= <i>n</i> , SS, SZ=4, | ENT=SRQSN,       | FIL=\$IB6A |
| SECR(I,I,I,VA,I) | OV= <i>n</i> , SZ=4,     | ENT=SECR,        | FIL=\$IB6A |
| SECRR(I,I,RVA,I) | OV= <i>n</i> , SZ=4,     | ENT=SECRR,       | FIL=\$IB6A |
| SECW(I,I,IA,I)   | OV= <i>n</i> , SZ=4,     | ENT=SECW,        | FIL=\$IB6A |
| SECWR(I,I,RA,I), | OV= <i>n</i> , SZ=4,     | ENT=SECWR,       | FIL=\$IB6A |
| IOCNT(I),        | OV= <i>n</i> , SZ=4,     | INTG, ENT=IOCNT, | FIL=\$IB6A |

where *n* is the overlay (OV) number and \$IB6A is the name of the FMGR file containing the HP-IB Message Subroutines, assuming that the routine was not loaded at system generation time. The above calls should all be included in the same overlay number *n*. If the HP-IB Utility was loaded at RTGEN time, FIL=\$IB6A can be omitted.

When you are configuring the HP-IB software into BASIC, it may be useful to include the data-conversion and bit-manipulation calls provided with BASIC. For convenience, these calls are listed below. Note that a FMGR file name is not required if the BASIC Subroutine Library has been loaded during system generation. The DCODE routine should reside in a separate overlay number as indicated by the symbols *p* and *q* below:

|                         |                            |           |
|-------------------------|----------------------------|-----------|
| DCODE (RVA ,RVA ,RA ) , | OV= <i>p</i> ,SZ=4 ,BP ,   | ENT=CODE  |
| NUM (RA ) ,             | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=NUM   |
| CHRS ( I ,RVA ) ,       | OV= <i>q</i> ,SZ=4 ,       | ENT=CHRS  |
| IBSET ( I , I ) ,       | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=BBSET |
| IEOR ( I , I ) ,        | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=BEOR  |
| OR ( I , I ) ,          | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=BIOR  |
| AND ( I , I ) ,         | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=BAND  |
| NOT ( I ) ,             | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=BNOT  |
| ISHFT ( I , I ) ,       | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=BSHFT |
| IBTST ( I , I ) ,       | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=BBTST |
| IBCLR ( I , I )         | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=BBCLR |
| ISETC (RA ) ,           | OV= <i>q</i> ,SZ=4 ,INTG , | ENT=ISETC |
| DEB\$ (RVA ) ,          | OV= <i>q</i> ,SZ=4 ,       | ENT=DEB\$ |
| BLK\$ ( I ,RVA ) ,      | OV= <i>q</i> ,SZ=4 ,       | ENT=BLK\$ |

Refer to the *BASIC Reference Manual* for further information on subroutine table generation.

## RTE-A Generation

---

### Introduction

This appendix describes RTE-A generation information for the HP-IB software, supplementing the information provided in the operating system programming and operating manual. Table B-1 lists the HP-IB software and the standard file names under which these software modules are stored on your disk for each operating system.

The HP-IB Subroutine libraries make a set of calls available from BASIC and FORTRAN for controlling the bus.

**Table B-1. RTE-A HP-IB Software**

| <b>File Name</b> | <b>HP Part No.</b> | <b>Description</b>                   |
|------------------|--------------------|--------------------------------------|
| %ID*37           | 92077-16696        | HP-IB Interface Driver               |
| \$HPIB           | 92077-12021        | HP-IB Subroutine Library for FORTRAN |
| RINTR            | 92857-16128        | BASIC/1000C Interrupt Routine        |

# Generation Procedure

The generation procedure given here follows the same sequence and format as that given in the RTE-A system manuals.

## Program Input Phase

Load the HP-IB driver during system generation as described in the generation manual. If BASIC is to be used, load the TRAP Module into a labelled system common in RTE-A. Also, the driver service routine, %SRV.L, must be online, loaded, and RPed upon boot-up. To load %SRV.L:

```
CI> RU, LINK
      LCOM          to access TRAP
      REL, %SRV.L   to load the service routine
```

## Table Generation Phase

The table generation phase of system configuration is divided into two parts: Interface Table entries, and Device Table entries. Take the steps given in the following paragraphs to configure the HP-IB driver into the system being generated.

### Interface Table IFT

Make an IFT entry for each interface card as follows:

1. Determine the select code of the HP-IB interface card. *sc:xx* where *xx* is the octal select code (10 through 77 octal). Select codes below 10 are used internally by the system.
2. Determine the number of IFT extension words required for your application. Calculate the number of extension words required as follows:

$$\text{Number of IFT extension words} = 4n + 130 \text{ (maximum} = 250\text{)}$$

where *n* = the number of auto-addressable devices to poll for an SRQ. The maximum number of devices is 30; therefore the maximum number of IFT extension words is 250.

Be sure to include enough extension words to allow for adding devices to the system at a later date. Adding extension words requires regenerating the system, which is a more time consuming procedure than simply adding the new information to available extension area.

3. Make each Interface Table entry as follows:

```
IFT, %ID*37, SC:XXB[ , TX:XX] [ , IT:XXB]
```

where:

SC = Select code of bus interface card

TX = Number of IFT extension words (Set to 134 by ID.37)

IT = Interface Type (Set to 37 octal by ID.37)

Repeat the above steps for each bus interface card in your system.

## Device Table DVT

The Device Table cross-references the device logical unit (LU) number to the DVT entries. Make the appropriate DVT entries as follows:

```
DVT, File Namr, Model #, LU:xx [ , TO:xx] [ , DT:xxB] [ , TX:xx] [ , BL:xx:xx] [ , DX:xx] [ , DP:n:xx] [ , PR:xx]
```

where:

*File Namr* is the file name of a device driver, or two commas if not used.

*Model #* is the model number of the device, if a device driver exists and supports multiple devices, must begin with M or two commas if not used.

LU is the Logical Unit number to be assigned to a bus device.

TO is the timeout value (default is zero, no timeout).

DT is the device Type (37B for HP-IB Controller, 70B for instrumentation; defaults to driver number).

TX is the number of DVT extension words (defaults to zero).

BL is the Buffer Limits (defaults to zero).

DX is the number of driver parameters all zero filled.

DP:n is the Driver Parameter: for HP-IB devices, the user must provide the HP-IB address which is the first driver parameter (one octal word per parameter).

PR is the priority of the device, 1 through 63 (defaults to 63).

The following is an example for assigning an Interface Table entry to a bus interface card set to select code 26.

```
IFT,%ID*37,SC:26B,TX:162
```

where:

%ID\*37 is the HP-IB Interface driver.

SC:26B indicates select code 26.

TX:162 adds table space for 8 devices.

If your application requires Direct I/O, then a DVT must be assigned for the bus controller. Octal 36 in driver parameter one enables the card to control the bus remote enable line.

```
DVT,, ,LU:23,TO:50,DT:37B,TX:0,DX:1,DP:1:36B,PR:0
```

where:

HP-IB Bus LU = 23

Bus Timeout = 50 msec

Device Type = 37B

Table Extension = zero words

Driver Extension = 1 word

Driver parameter 1 = 36 octal (HP-IB Controller address)

Priority = zero

For the auto-addressing mode, assign a DVT and an HP-IB address to each bus device using the above controller. Driver parameter one contains the HP-IB device address and must be a decimal in the range of 0 through 29. This example assigns two HP-IB devices to the above controller.

```
DVT,, ,LU:24,TO:500,DT:70B,DX:1,DP:1:0
```

where:

Device LU = 24

Device Timeout = 500 msec (Set for specific device)

Device Type = 70 octal (Instrumentation)

Driver Extension = 1 word

Driver parameter 1 = (HP-IB address 0)

```
DVT,, ,LU:25:TO:500,DT:70B,DX:1,DP:1:1
```

where:

Device LU = 25

Driver parameter 1 = 1 (HP-IB address 1)

## Quick Reference Tables

**Table C-1. Device Addresses: Binary, Character, Octal, and Decimal**

|                     | Address Switch Numbers |   |   |   |    | Octal Value | Decimal Value | Address Characters |        |
|---------------------|------------------------|---|---|---|----|-------------|---------------|--------------------|--------|
|                     | 5                      | 4 | 3 | 2 | 1  |             |               | Talk               | Listen |
| RTE-6/VM Controller | 0                      | 0 | 0 | 0 | 0  | 00          | 00            | @                  | SP     |
|                     | 0                      | 0 | 0 | 0 | 1  | 01          | 01            | A                  | !      |
| Devices             | 0                      | 0 | 0 | 1 | 0  | 02          | 02            | B                  | "      |
|                     | 0                      | 0 | 0 | 1 | 1  | 03          | 03            | C                  | #      |
|                     | 0                      | 0 | 1 | 0 | 0  | 04          | 04            | D                  | \$     |
|                     | 0                      | 0 | 1 | 0 | 1  | 05          | 05            | E                  | %      |
|                     | 0                      | 0 | 1 | 1 | 0  | 06          | 06            | F                  | &      |
|                     | 0                      | 0 | 1 | 1 | 1  | 07          | 07            | G                  | '      |
|                     | 0                      | 1 | 0 | 0 | 0  | 10          | 08            | H                  | (      |
|                     | 0                      | 1 | 0 | 0 | 1  | 11          | 09            | I                  | )      |
|                     | 0                      | 1 | 0 | 1 | 0  | 12          | 10            | J                  | *      |
|                     | 0                      | 1 | 0 | 1 | 1  | 13          | 11            | K                  | +      |
|                     | 0                      | 1 | 1 | 0 | 0  | 14          | 12            | L                  | ,      |
|                     | 0                      | 1 | 1 | 0 | 1  | 15          | 13            | M                  | -      |
|                     | 0                      | 1 | 1 | 1 | 0  | 16          | 14            | N                  | .      |
|                     | 0                      | 1 | 1 | 1 | 1  | 17          | 15            | O                  | /      |
|                     | 1                      | 0 | 0 | 0 | 0  | 20          | 16            | P                  | 0      |
|                     | 1                      | 0 | 0 | 0 | 1  | 21          | 17            | Q                  | 1      |
|                     | 1                      | 0 | 0 | 1 | 0  | 22          | 18            | R                  | 2      |
|                     | 1                      | 0 | 0 | 1 | 1  | 23          | 19            | S                  | 3      |
|                     | 1                      | 0 | 1 | 0 | 0  | 24          | 20            | T                  | 4      |
|                     | 1                      | 0 | 1 | 0 | 1  | 25          | 21            | U                  | 5      |
|                     | 1                      | 0 | 1 | 1 | 0  | 26          | 22            | V                  | 6      |
|                     | 1                      | 0 | 1 | 1 | 1  | 27          | 23            | W                  | 7      |
|                     | 1                      | 1 | 0 | 0 | 0  | 30          | 24            | X                  | 8      |
|                     | 1                      | 1 | 0 | 0 | 1  | 31          | 25            | Y                  | 9      |
|                     | 1                      | 1 | 0 | 1 | 0  | 32          | 26            | Z                  | :      |
|                     | 1                      | 1 | 0 | 1 | 1  | 33          | 27            | [                  | ;      |
|                     | 1                      | 1 | 1 | 0 | 0  | 34          | 28            | \                  | <      |
| 1                   | 1                      | 1 | 0 | 1 | 35 | 29          | ]             | =                  |        |
| RTE-A Controller    | 1                      | 1 | 1 | 1 | 0  | 36          | 30            | ^                  | >      |
| Untalk/Unlisten     | 1                      | 1 | 1 | 1 | 1  | 37          | 31            | -                  | ?      |



**Table C-2. ASCII Codes of HP-IB Addresses and Commands**

| BITS |    |    |    | 0 0              | 0 0               | 0 1                 | 0 1                 | 1 0               | 1 0               | 1 1                    | 1 1                    |
|------|----|----|----|------------------|-------------------|---------------------|---------------------|-------------------|-------------------|------------------------|------------------------|
| B7   | B6 | B5 |    | 0                | 0                 | 1                   | 1                   | 0                 | 0                 | 1                      | 1                      |
| B4   | B3 | B2 | B1 | addr<br>cmds     | univ<br>cmds      | listen<br>addresses | listen<br>addresses | talk<br>addresses | talk<br>addresses | Secondary<br>addr/cmds | Secondary<br>addr/cmds |
| 0    | 0  | 0  | 0  | 0<br>NUL         | 20B<br>DLE        | 40B<br>SP<br>L00    | 60B<br>0<br>L16     | 100B<br>0<br>T00  | 120B<br>P<br>T16  | 140B<br>,<br>PPE1/0    | 160B<br>P<br>PPD       |
| 0    | 0  | 0  | 1  | 1<br>SOH<br>GTL  | 21B<br>DC1<br>LLO | 41B<br>!<br>L01     | 61B<br>1<br>L17     | 101B<br>A<br>T01  | 121B<br>Q<br>T17  | 141B<br>a<br>PPE2/0    | 161B<br>q              |
| 0    | 0  | 1  | 0  | 2<br>STX         | 22B<br>DC         | 42B<br>"<br>L02     | 62B<br>2<br>L18     | 102B<br>B<br>T02  | 122B<br>R<br>T18  | 142B<br>b<br>PPE3/0    | 162B<br>r              |
| 0    | 0  | 1  | 1  | 3<br>ETX         | 23B<br>DC3        | 43B<br>#<br>L03     | 63B<br>3<br>L19     | 103<br>C<br>T03   | 123B<br>S<br>T19  | 143B<br>c<br>PPE4/0    | 163B<br>s              |
| 0    | 1  | 0  | 0  | 4<br>EOT<br>SDC  | 24B<br>DC4<br>DCL | 44B<br>\$<br>L04    | 64B<br>4<br>L20     | 104B<br>D<br>T04  | 124B<br>T<br>T20  | 144B<br>d<br>PPE5/0    | 164B<br>t              |
| 0    | 1  | 0  | 1  | 5<br>ENQ<br>PPC  | 25B<br>NAK<br>PPU | 45B<br>%<br>L05     | 65B<br>5<br>L21     | 105B<br>E<br>T05  | 125B<br>U<br>T21  | 145B<br>e<br>PPE6/0    | 165B<br>u              |
| 0    | 1  | 1  | 0  | 6<br>ACK         | 26B<br>SYN        | 46B<br>&<br>L06     | 66B<br>6<br>L22     | 106B<br>F<br>T06  | 126B<br>V<br>T22  | 146B<br>f<br>PPE7/0    | 166B<br>v              |
| 0    | 1  | 1  | 1  | 7<br>BEL         | 27B<br>ETB        | 47B<br>'<br>L07     | 67B<br>7<br>L23     | 107B<br>G<br>T07  | 127B<br>W<br>T23  | 147B<br>g<br>PPE8/0    | 167B<br>w              |
| 1    | 0  | 0  | 0  | 10B<br>BS<br>GET | 30B<br>CAN<br>SPE | 50B<br>(<br>L08     | 70B<br>8<br>L24     | 110B<br>H<br>T08  | 130B<br>X<br>T24  | 150B<br>h<br>PPE1/1    | 170B<br>x              |
| 1    | 0  | 0  | 1  | 11B<br>HT<br>TCT | 31B<br>EM<br>SPD  | 51B<br>)<br>L09     | 71B<br>9<br>L25     | 111B<br>I<br>T09  | 131B<br>Y<br>T25  | 151B<br>i<br>PPE2/1    | 171B<br>y              |
| 1    | 0  | 1  | 0  | 12B<br>LF        | 32B<br>SUB        | 52B<br>*<br>L10     | 72B<br>:<br>L26     | 112B<br>J<br>T10  | 132B<br>Z<br>T26  | 152B<br>j<br>PPE3/1    | 172B<br>z              |
| 1    | 0  | 1  | 1  | 13B<br>VT        | 33B<br>ESC        | 53B<br>+<br>L11     | 73B<br>;<br>L27     | 113B<br>K<br>T11  | 133B<br>[<br>T27  | 153B<br>k<br>PPE3/1    | 173B<br>{              |
| 1    | 1  | 0  | 0  | 14B<br>FF        | 34B<br>FS         | 54B<br>,<br>L12     | 74B<br><<br>L28     | 114B<br>L<br>T12  | 134B<br>\<br>T28  | 154B<br>l<br>PPE4/1    | 174B<br>               |
| 1    | 1  | 0  | 1  | 15B<br>CR        | 35B<br>GS         | 55B<br>-<br>L13     | 75B<br>=<br>L29     | 115B<br>M<br>T13  | 135B<br>]`<br>T29 | 155B<br>m<br>PPE6/1    | 175B<br>}              |
| 1    | 1  | 1  | 0  | 16B<br>SO        | 36B<br>RS         | 56B<br>.<br>L14     | 76B<br>><br>L30     | 116B<br>N<br>T14  | 136B<br>^<br>T30  | 156B<br>n<br>PPE7/1    | 176B<br>~              |
| 1    | 1  | 1  | 1  | 17B<br>SI        | 37B<br>US         | 57B<br>/<br>L15     | 77B<br>?<br>UNL     | 117B<br>O<br>T15  | 137B<br>_<br>UNT  | 157B<br>o<br>PPE8/1    | 177B<br>DEL            |

## Legend

|     |  |          |                           |    |       |                      |
|-----|--|----------|---------------------------|----|-------|----------------------|
| Key | <table border="1"> <tr> <td>xxB</td> </tr> <tr> <td>CHR</td> </tr> <tr> <td>FN</td> </tr> </table> | xxB      | CHR                       | FN | xxB = | Octal representation |
|     |  | xxB      |                           |    |       |                      |
| CHR |  |          |                           |    |       |                      |
| FN  |  |          |                           |    |       |                      |
|     |  | CHR =    | ASCII character           |    |       |                      |
|     |  | FN =     | Function                  |    |       |                      |
|     |  | Lyy =    | Listen address yy         |    |       |                      |
|     |  | Tyy =    | Talk address yy           |    |       |                      |
|     |  | GTL =    | Go to Local               |    |       |                      |
|     |  | LLO =    | Local Lockout             |    |       |                      |
|     |  | SDC =    | Selected Device Clear     |    |       |                      |
|     |  | DCL =    | Device Clear              |    |       |                      |
|     |  | PPC =    | Parallel Poll Configure   |    |       |                      |
|     |  | PPU =    | Parallel Poll Unconfigure |    |       |                      |
|     |  | GET =    | Group Execute Trigger     |    |       |                      |
|     |  | SPE =    | Serial Poll Enable        |    |       |                      |
|     |  | TCT =    | Take Control              |    |       |                      |
|     |  | SPD =    | Serial Poll Disable       |    |       |                      |
|     |  | PPEd/l = | Parallel Poll Enable      |    |       |                      |
|     |  |          | d = DIO line              |    |       |                      |
|     |  |          | l = level (0/1)           |    |       |                      |
|     |  | UNT =    | Untalk                    |    |       |                      |
|     |  | UNL =    | Unlisten                  |    |       |                      |

**Table C-3. Electrical Characteristics**

|   |   |                    |                      |           |                     |          |                    |
|---|---|--------------------|----------------------|-----------|---------------------|----------|--------------------|
| <b>General</b>  |   |                    |                      |           |                     |          |                    |
|   | <table> <tr> <td><b>Logic Level</b></td> <td><b>Voltage Level</b></td> </tr> <tr> <td>0 (False)</td> <td><math>\geq +2.0V</math> (High)</td> </tr> <tr> <td>1 (True)</td> <td><math>\leq +0.8V</math> (Low)</td> </tr> </table> | <b>Logic Level</b> | <b>Voltage Level</b> | 0 (False) | $\geq +2.0V$ (High) | 1 (True) | $\leq +0.8V$ (Low) |
| <b>Logic Level</b>  | <b>Voltage Level</b>  |                    |                      |           |                     |          |                    |
| 0 (False)   | $\geq +2.0V$ (High)   |                    |                      |           |                     |          |                    |
| 1 (True)  | $\leq +0.8V$ (Low)  |                    |                      |           |                     |          |                    |
| <b>Sender Types</b>   |   |                    |                      |           |                     |          |                    |
| <p><b>Open Collector Only</b><br/> SRQ, NRFD, NDAC<br/> DIO1-8 (Parallel Poll devices)</p> <p>*Tristate useful to reach data rates above 250,000 bytes/sec.</p>                                       | <p><b>Open Collector or *Tristate</b><br/> ATN, IFC, REN, EOI, DAV<br/> DIO1-8 (non-Parallel Poll devices)</p>  |                    |                      |           |                     |          |                    |
| <b>Sender Specifications</b>  |   |                    |                      |           |                     |          |                    |
| <p><math>V_{OL} &lt; +0.5V</math> @ 48 mA continuous sink (tristate or open collector)<br/> <math>V_{OH} \geq 2.4V</math> @ 5.2 mA source (tristate)<br/> see dc Load Line Graph (open collector)</p> |   |                    |                      |           |                     |          |                    |
| <b>Receiver Specifications</b>  |   |                    |                      |           |                     |          |                    |
| <p><b>Preferred (Schmitt-type)</b><br/> <math>V_{OL} = V_{tneg} \geq +0.8V</math><br/> <math>V_{OH} = V_{tpos} \leq +2.0V</math><br/> Hysteresis: <math>V_{tpos} - V_{tneg} \geq +0.4V</math></p>     | <p><b>Allowed (non-Schmitt-type)</b><br/> <math>V_{OL} \leq +0.8V</math><br/> <math>V_{OH} \geq +2.0V</math></p>  |                    |                      |           |                     |          |                    |

## Bus Service Program: Example

---

The following example of an HP-IB bus service program is designed to show what kinds of things the bus service program is capable of doing. This is an example of a program you would configure to run whenever there is an unclaimed interrupt on the bus. Since any bus error that will run the bus service program will also turn off all interrupts from the bus (Table 5-3), one of the duties of the service program (statement 100, below) is to turn interrupts back on. The comments in the program describe other features of the example.

```
FTN7X,L
PROGRAM SRQ00(3,89)
DIMENSION IP(5), INAMB(4), INAMD(4)
C
C*****
C
C      IP will receive the parameters returned by the call to RMPAR.
C      The values passed in are supplied by the driver:
C
C      IP(1) = Status code of bus.
C      IP(2) = Five-bit HP-IB address (subchannel) of interrupting device.
C      IP(3) = EQT address of associated bus.
C      IP(4) = User-defined value to be passed to the service program.
C
C*****
C
C      Initialize names and LUs
C
C      CALL RMPAR(IP)
C
C      Set up the HP-IB bus service program name. Note that the first
C      element in the array is the number of characters to follow.
C
C      INAMD(1)=5
C      INAMD(2)=2HSR
C      INAMD(3)=2HQ0
C      INAMD(4)=2H0
C
C      Set up the device alarm program names. One example is given here. You
C      may want more than one so each can perform a device-specific function.
C
C      INAMD(1)=5
C      INAMD(2)=2HSR
C      INAMD(3)=2HQ1
C      INAMD(4)=2H0
C
C      Set up the bus lu in IBLU and a device lu in IDLU. Note that IDLU is
```

```

C      used to make HP-IB library calls with the subchannel dynamically
C      altered by subroutine MESSXX.  This means that IDLU should not also be
C      used for a real device on the HP-IB bus.  This extra LU is needed to do
C      control calls to the bus in the event that the real device at the HP-IB
C      address of concern is down, blocking further I/O to that LU.
C
      IBLU=20
      IDLU=29
C
C      Set up EQTX fixed size and logging device.
C
      IFIXSZ=20
      LOG=6
C
C      Come back here on every schedule
C
5     CONTINUE
C
C*****
C      Note that the following lines are for debugging purposes only.
C
      WRITE(LOG,10)
D 10  FORMAT("SUBCHANNEL 0 ALARM PROGRAM HAS BEEN ACTIVATED!  ")
D      WRITE(LOG,20),IP(1),IP(2),IP(3),IP(4)
D 20  FORMAT("STATUS=",K3/"DEV ADD=",K3/"EQT="K4/,"IV=",K6)
C
C*****
C
C      Set up EQT pointers
C
      IEQT=IP(3)
C
C      Get EQT words 12 and 13
C
      IEQT13=IGET(IEQT+12)
      IEQT12=IGET(IEQT+11)
C
C      Compute the EQT number.
C      Base page location 1650B is the first word address of the EQT table.
C
      IEQTN=(IEQT-IGET(1650B))/15+1
C
C      Start to test the reason for the service program schedule.
C
      IF(IP(1).LT.40B.OR.IP(1).GE.240B) GOTO 30
C
C      Remove device from POLL table.  Device did not respond to serial poll.
C      Make sure just to look are the low five bits for the HP-IB address.
C
      IADDR=IAND(IP(1),37B)
C
C      Call subroutine MESSX to set IDLU to the EQT and subchannel of the
C      problem device to bypass system downing of the device normal LU.
C      All calls made to the HP-IB library are control requests that alter
C      EQT extension values; they do not depend on device operation.
C
      CALL MESSX(IDLU,IEQTN,IADDR)

```

```

C
C   Disable service program for this device LU.
C
CALL SRQ(IDLU,0,0)
WRITE(LOG,21)IADDR
21  FORMAT(1X,"BUS DEVICE ADDR",13," REMOVED FORM POLL TABLE")
C
C   Go to turn HP-IB BUS SRQs back on.  The driver will disable SRQ
C   interrupts until a CALL SRQ (BUSLU,0,INAMB) is done so that SRQ00
C   can maintain control of the HP-IB bus and avoid continuous interrupt
C   conditions.
C
GOTO 100
C
C   Test for Serial Poll Disable timeout and report error.  This case is
C   ignored otherwise in this sample program.
C
30  IF (IP(1).NE.37B) GOTO 32
WRITE(LOG,31)
31  FORMAT("SPD TIMEOUT OCCURRED")
GOTO 100
C
C   If not unclaimed SRQ, then 200
C
32  IF (IP(1).NE.10B) GOTO 200
C
C   unclaimed SRQ has occurred
C
C   First, calculate EQTX size for the loop counter and set IFOUND to 0
C
ICOUNT=(IAND(IEQT12,225)-IFIXSZ)/7
IDEX=IAND(IEQT13,77777B)+IFIXSZ
IFOUND=0
C
C   Find out who pulled the SRQ line but is not currently enabled in the
C   EQTX with a service program.  The device that pulled SRQ should have
C   previously existed in the poll table kept by the driver in the EQTX.
C   Presumably, it has been removed by SRQ00 due to a previous event, such
C   as the device failing to answer a serial poll.
C
C   This program does not test all possible addresses by a status call,
C   because if the device does not exist a bus timeout must occur to get
C   back to SRQ00.  Also, if all devices are tested, the STATS call in the
C   HP-IB library will fill in an EQTX entry and use up the EQTX area.
C   This implies that this program will not find a device that was never
C   configured by a call to SRQ.
C
C   Go through EQT Extension, checking each device for source of interrupt.
C
DO 40 I=1,ICOUNT
IVAL=IGET(IDEX)
IADDR=IAND(37B,IVAL)
C
C   If HP-IB address = 0, it is an empty poll table entry, so ignore it.
C
IF(IADDR.EQ.0) GOTO 35
IVAL=IGET(IDEX+1)
C
C   If program name <> 0, then device is not causing problem.

```

```

C
IF(IVAL.NE.0) GOTO 35
C
C Found an address in poll table with cleared name, so test if it has
C SRQ status and configure the device back in the poll table if it is
C requesting SRQ service.
C
CALL MESSX(IDLU,IEQTN,IADDR)
WRITE(LOG,33)IADDR
33 FORMAT(1X,"TESTING STATUS ON ADDR",13)
C
C Test device status.
C Subroutine STATS does a serial poll of the device with device errors
C returning a status of 0 after a timeout period.
C
CALL STATS(IDLU,ISTAT)
C
C If not asserting SRQ, then ignore
C
IF(IAND(ISTAT,100B).EQ.0) GOTO 35
WRITE(LOG,34)IADDR
34 FORMAT("ADDING DEVICE ADDR",I3," TO POLLING TABLE")
C
C Add device to poll table. This program assumes all devices on the
C HP-IB bus are serviced by SRQ10. You may want to keep a table of
C device LU and service program names in an array that can be indexed
C into for the call to SRQ.
C
CALL SRQ(IDLU,0,INAMD)
C
C Schedule the program to handle the device because the driver did not do
C this for you.
C
CALL EXEC (24,INAMD(2),ISTAT,IADDR,IEQT,0)
C
C Set a flag indicating that a device has been found.
C
IFOUND=1
C
C Go to the next device in the Extension.
C
35 IDEX>IDEX+7
40 CONTINUE
C
C The loop has tested all devices in the poll table that had their alarm
C program names zeroed out. If no such device was found, then that fact
C is logged, but this program example does nothing about it. However,
C this condition can cause looping if you subsequently add a device to
C the bus and it SRQs before it has been configured into the EQTX or
C poll table (by a call to the SRQ subroutine). If you want to shut
C down SRQ interrupts, then a call SRQ (IBLU,-1,IBNAM) will hold off SRQ
C interrupts for system startup purposes.
C
C The driver also disables SRQ interrupts for serial poll related problems
C by setting the PRAM4 word for the bus alarm program in the EQTX to -1.
C
IF(IFOUND.EQ.1) GOTO 100
WRITE(LOG,46)
46 FORMAT(1X,"UNCLAIMED SRQ COULD NOT BE FOUND")

```

```

C
C
C   Calling SRQ to the bus with zero in the second argument of the call
C   rearms the bus, turning on SRQ interrupts.
C
C
100 CALL SRQ (IBLU,0,INAMB)
    GOTO 900
C
C   A buffered request failed and is only logged in this program.
C
200 IF(IP(1).GE.240B.AND.IP(1).LT.340B) WRITE(LOG,201)IP(1)
201 FORMAT(1X,"BUFFERED REQUEST FAILED ISTAT="I4)
C
C   E-bit not set and request failed is logged.
C
    IF(IP(1).GE.340B) WRITE(LOG,202) IP(1)
202 FORMAT(1X,"E-BIT NOT SET AND REQUEST FAILED ISTAT="I4)
    IF(IP(1).GT.6) GOTO 900
C
C   The rest of the error conditions are vectored to by a completed goto.
C
    GOTO (310,320,330,340,350,360) IP(1)
C
C   Device TO occurred.
C
310 WRITE(LOG,311)
311 FORMAT(1X,"IO DEVICE TIMEOUT")
    GOTO 900
C
C   IFC detected during an I/O request.
C
320 WRITE(LOG,321)
321 FORMAT(1X,"IFC DETECTED DURING I/O REQUEST")
    GOTO 900
C
C   SRQ aborted I/O request
C
330 WRITE(LOG,331)
331 FORMAT(1X,"SRQ ABORTED I/O REQUEST")
    GOTO 900
C
C   Service program named in SRQ call has no ID Segment
C
340 WRITE(LOG,341)
341 FORMAT(1X,"NON-EXISTENT SRQ PROGRAM")
    GOTO 900
C
C   Illegal I/O request was attempted.
C
350 WRITE(LOG,351)
351 FORMAT(1X,"ILLEGAL I/O REQUEST")
    GOTO 900
C
C   The driver has found that the poll table area (EQTX) is full.
C
360 WRITE(LOG,361)
361 FORMAT(1X,"EQTX FULL")
C

```

```

C      Go dormant
C
C
900  CALL EXEC(6,0,1)
C
C      On wake-up, pick up parameters and start again.
C
      CALL RMPAR(IP)
      GOTO 5
      END
C
C*****
C
C      This subroutine sends a MESSS call to the system to point
C      IDLU at the IEQTN with a subchannel of IADDR.
C
      SUBROUTINE MESSX(IDLU,IEQTN,IADDR)
      DIMENSION IBUFF(14)
C
C      Select one of the three methods for formatting a MESSS call
C
C*****
C
C      If using FTN4 compiler, then use call CODE.
C
      CALL CODE
      WRITE(IBUFF,11)IDLU,IEQTN,IADDR
C
C*****
C
C      If using FTN4x, then use ENCODE.
C
      ENCODE(12,11,IBUFF)IDLU,IEQTN,IADDR
C
C*****
C
C      If using FTN7X, then use internal files.
C
      CHARACTER*28 CIBUFF
      EQUIVALENCE (CIBUFF, IBUFF(1))
      WRITE (CIBUFF,11) IDLU,IEQTN,IADDR
C
C*****
C
11  FORMAT("LU," ,I3," ,",I2," ,",I2)
      CALL EXEC(2,1,IBUFF,14)
      IA=MESSS( IBUFF,12)
      IF( IA.GE.0)RETURN
      CALL EXEC(2,LOG,IBUFF,IA)
      RETURN
      END$

```



# Glossary

---

## Address

See **Device address**.

## Bus management lines

The five lines used to control operation of the HP-IB. See Chapter 1. The lines are:

- **Attention (ATN)**—Selects command or data mode on the bus. See **Command mode** and **Data mode**.
- **End or Identify (EOI)**—In the data mode (ATN false), a signal on EOI indicates the end of a data transmission. In the command mode (ATN true), it initiates a parallel poll. See **Parallel polling**.
- **Service Request (SRQ)**—Interrupts the controller to indicate that a bus device needs attention.
- **Remote Enable (REN)**—Permits devices with remote/local switching to be set to the remote mode.
- **Interface Clear (IFC)**—Halts all bus traffic and returns bus to a predetermined state.

## Command mode

The state of the bus when the ATN bus management line is true. All signals on the DIO lines are interpreted by the bus devices as commands. Compare **Data mode**, see Chapter 4.

## Commands

ASCII codes on the DIO lines accompanied by an active signal on the ATN line, or signals on the bus management lines, which initiate bus device activities. See Chapter 4.

## Controller

The HP 12821A, HP 59310B, or HP 12009A Interface card in an HP 1000 computer system. In this manual, the term “controller” is used for the entire HP 1000 computer, including the interface card, the RTE operating system, and the software driver for the interface. See Chapter 2.

## Data mode

The state of the bus when the ATN bus management line is false. All signals on the DIO lines are interpreted by bus devices as data. Compare **Command mode**, see Chapter 4.

## Down

The state of a non-functioning I/O device that has been made unavailable by the operating system. When the fault is corrected, the device is made available by resetting its EQT (RTE-6/VM) or LU (RTE-A) status. Compare **Up**.

## Device address

A number between 0 and 31, set by switches on a device, that identifies the device on the bus. The device address corresponds to the EQT subchannel for RTE-6/VM, and the first driver parameter (DP#1) for RTE-A. The device address is a 5-bit code making up the lower bits of the 7-bit talk or listen address. See **Talk address**, **Listen address**, and Chapters 2 and 3.

**Device reference table (DRT)**

An RTE-6/VM system table that associates LU numbers with EQT entries. See Chapter 3.

**Device table (DVT)**

An RTE-A system table for each LU on the bus, that contains the device type, HP-IB address, timeout value, and driver type (if there is a device driver). Device LUs are associated with DVTs in the logical unit table (LUT). Each DVT has an entry in the interface table (IFT). See Chapter 3.

**DIO lines**

The eight data lines of the bus. Commands, data, and device status bytes pass over the DIO lines. See Chapter 1.

**Driver**

The software that lets the operating system communicate with the interface card and the devices attached to it. The HP-IB interface drivers are ID.37, DVA37, and DVR37. See Chapter 3.

**Equipment table (EQT)**

An RTE-6/VM system table, with a 15-word entry for each interface card, that contains the select code of the card, the interface driver type, and other driver and interface card information. The EQT entries are set at generation. See Chapter 3.

**Handshake**

A sequence of signals on the three handshake lines that accompanies every byte transferred on the DIO lines. See Chapter 1. The handshake lines are:

- **Data Valid (DAV)**—Indicates that a new byte is on the bus and devices can now accept it.
- **Not Ready For Data (NRFD)**—Indicates that a device is busy and cannot accept new data yet. The device sets NRFD false when it is ready.
- **Not Data Accepted (NDAC)**—Indicates that a device has not yet accepted the newest byte on the bus. The device sets NDAC false when it accepts the byte.

**Interface card**

A circuit board in the HP 1000 that lets the computer send and receive data. There are a number of interface card types for HP 1000 computers. The HP-IB Interface Card for E/F-Series computers is the HP 12821A or the HP 59310B; the A-Series HP-IB Interface Card is the HP 12009A. See Chapter 2.

**Interface table (IFT)**

An RTE-A system table for each interface card, that contains the interface driver type, timeout queueing, driver parameters, and an extension area for passing information to device drivers. See Chapter 3.

**Listen address**

An ASCII character that selects a device to listen, to receive data or commands. See Chapter 4.

**Logical unit (LU) number**

A number, unique to a peripheral I/O device, that identifies the device in I/O operations. See Chapter 3.

**Logical unit table (LUT)**

An RTE-A system table that associates LU numbers with DVTs. See Chapter 3.

**Parallel polling**

A method of checking up to eight devices at a time to determine if any need to be serviced. See Chapter 5.

**Polling**

A method of identifying devices in need of service. There are two types of polling on the HP-IB: serial and parallel. See **Serial polling**, **Parallel polling**, and Chapter 5.

**RTE – Real-Time Executive**

The operating system of an HP 1000 computer system. HP-IB is supported by RTE-6/VM and RTE-A.

**Secondary address**

A number between 0 and 31 that identifies a subchannel within a device, such as an I/O port, internal register, or status word. See Chapter 4.

**Select code (SC)**

A number between 10B and 77B that identifies an interface card. On E/F-Series computers, the select codes identify the card slots in the I/O card cage. On A-Series computers, each interface card has a select code switch. See Chapter 3.

**Serial Polling**

A method of locating a device in need of service. If serial polling is enabled and a device asserts the SRQ bus management line to request service, the controller performs a serial poll to find the requesting device. See Chapter 5.

**Talk Address**

An ASCII character that selects one of the devices on the bus to talk, or transmit data. See Chapter 2.

**Timeout**

The length of time that the operating system waits for a device to respond before it sets the device status to down.

**Up**

The status of a device that is functioning normally and is available for access. Compare **Down**.

# Index

---

## A

ABI chip, 2-3, 4-5  
ABORT statement, 4-18  
aborted data transfer, 6-4  
ABRT subroutine, 4-18  
address  
    and LU numbers, 3-1  
    ASCII representation, 2-5, 4-1  
    assignment table, 2-6  
    definition, 1-4, 4-1  
    device, 2-5  
    dual-address devices, 2-5  
    HP 12009A interface card, 2-3  
    HP 59310B interface card, 2-1  
    listen, 1-3  
    secondary, 1-4  
    subchannels, 3-1  
    switches, 2-3  
    talk, 1-3  
addressable mode devices, 2-6  
addressed commands, 4-5  
addressing, 4-1  
    automatic, 3-3, 4-22  
    direct, 3-3, 4-2, 4-17  
    secondary, 4-2, 4-22  
Amigo standard, 2-5, 5-7, 5-16  
ASCII address strings, 4-2, 4-5  
assembling the bus, 2-7  
ATN bus management line, 1-7

## B

BASIC/1000C  
    and BASIC/1000D strings, 4-20  
    automatic addressing, 4-22  
    CHARACTER variable type, 4-20  
    device control, 4-14  
    I/O specifiers, 4-13  
    secondary addressing, 4-3, 4-22  
    service requests, 5-9  
    syntax, 4-12  
BASIC/1000D  
    automatic addressing, 4-22  
    device control, 4-14  
    secondary addressing, 4-3, 4-22  
    strings, 4-20  
    syntax, 4-12  
BSTAT subroutine, 5-2, 5-5

## bus

assembly, 2-7  
command messages, 4-5  
functional description, 1-3  
interface card HP 12009A, 2-3  
interface card HP 59310B, 2-1  
LU numbers, 3-1  
operation example, 1-2  
programming languages, 1-3  
RTE-6/VM LU assignment, 3-4  
RTE-A LU assignment, 3-6  
service program, 5-11, D-1  
specifications, 1-4  
structure, 1-5

## C

cable  
    connector hardware, 2-8  
    installation, 2-7  
    length restrictions, 2-3  
CLEAR statement, 4-15  
CLEAR subroutine, 4-15  
CMDW/CMDR subroutines, 4-20  
CNFG subroutine, 7-6  
command  
    addressed, 4-6  
    ATN—The Command Mode, 4-11  
    DCL—Device Clear, 4-8, 4-15  
    EOI and ATN—Parallel Poll, 4-9  
    GET—Group Execute Trigger, 4-7, 4-17  
    GTL—Go To Local, 4-6, 4-16  
    IFC—Interface Clear, 4-11  
    LLO—Local Lockout, 4-8, 4-16  
    mode, 1-7, 4-1, 4-5  
    PPC—Parallel Poll Configure, 4-6  
    PPU—Parallel Poll Unconfigure, 4-9  
    REN—Remote Enable, 4-10, 4-15  
    SDC—Selected Device Clear, 4-6, 4-15  
    SPD—Serial Poll Disable, 4-9  
    SPE—Serial Poll Enable, 4-9  
    SRQ—Service Request, 4-10  
configuration, 7-1  
    CNFG subroutine, 7-6  
configuration word, 7-1  
    D bit, 7-3  
    E bit, 7-5  
    I bit, 7-4  
    J bit, 7-4  
    O bit, 7-4  
    P bit, 7-4  
    R bit, 7-2  
    S bit, 7-2

- controller, 1-3
  - address, 2-1
  - as talker/listener, 4-5
  - configuration, 7-1
  - EOR requirements, 7-4

## D

- data
  - lines, 1-5
  - messages, 4-11
  - mode, 1-7, 4-1, 4-11
  - settling time, 2-3
- DAV handshake line, 1-6
- DCL bus command, 4-8
- device address, 2-5
  - address assignment table, 2-6
  - addressable mode devices, 2-6
  - and LU numbers, 3-1
  - and subchannels, 3-1
  - dual-address devices, 2-5
  - switches, 2-3
- device communications
  - CMDW/CMDR subroutines, 4-20
  - data messages, 4-11
  - IOCNT subroutines, 4-23
  - PACK USING statement, 4-21
  - READ/ENTER/INPUT/WRITE/PRINT statements, 4-22
  - SECW/SECR/SECWR/SECRR subroutines, 4-22
- device control
  - CLEAR statement, 4-15
  - CLEAR subroutine, 4-15
  - GTL subroutine, 4-16
  - LLO subroutine, 4-16
  - LOCAL LOCKOUT statement, 4-17
  - LOCAL statement, 4-16
  - LOCL subroutine, 4-17
  - REMOTE statement, 4-16
  - RMOTE subroutine, 4-15
  - TRIGGER statement, 4-17
  - TRIGR subroutine, 4-17, 4-18
- device table (DVT), 3-1
- DIO lines and parallel polling, 4-7
- direct addressing, 4-20

## E

- ENABLE INTR statement, 5-3, 5-19
- EOI and ATN bus command, 4-9
- EOI bus management line, 1-7
- EOR
  - requirements for controller, 7-5
  - requirements for devices, 7-4
- EQT overflow, 6-4
- equipment table (EQT), 3-1

- error
  - aborted data transfer, 6-4
  - codes, 6-1
  - EQT overflow, 6-4
  - IBERR subroutine, 6-1
  - IFT space too small, 6-4
  - illegal interrupt, 6-3
  - processing
    - by program, 6-1
    - by system, 6-2
    - example, 6-5
  - specified program does not exist, 6-4
  - timeout or transmission, 6-2

## F

- FORTRAN
  - and BASIC/1000D strings, 4-20
  - automatic addressing, 4-20
  - device control, 4-12
  - secondary addressing, 4-3, 4-22
  - syntax, 4-12
- function
  - GETINTR, 5-6, 5-20
  - PPOLL, 5-6
  - SPOLL, 5-5

## G

- GET bus command, 4-7
- GETINTR function, 5-6
- GTL bus command, 4-6
- GTL subroutine, 4-14

## H

- handshake lines, 1-6
- high-speed operation, 2-3

## I

- I/O specifiers, 4-2, 4-13
- IBERR subroutine, 6-1
- IEC 625-1 (equivalent), 2-8
- IEEE-488-1978, 1-4
- IFC bus command, 4-10
- IFC bus management line, 1-7
- IFT space too small, 6-4
- illegal interrupt, 6-3
- installation, 2-9
- interface card
  - HP 12009A, 2-3
    - address, 2-3
    - select code, 2-3
  - HP 59310B, 2-1
    - address, 2-1
- interface table (IFT), 3-1

interrupt and device status  
  BSTAT subroutine, 5-5  
  ENABLE INTR statement, 5-3  
  GETINTR function, 5-6  
  ON INTR statement, 5-3  
  PPOLL CONFIGURE statement, 5-6  
  PPOLL subroutine, 5-4  
  PPOLL UNCONFIGURE statement, 5-6  
  PPSCH subroutine, 5-4  
  PPSN subroutine, 5-4  
  PSTAT subroutine, 5-8  
  SPOLL statement, 5-5  
  SRQ subroutine, 5-2  
  SRQSN subroutine, 5-3  
  STATS subroutine, 5-5  
IOCNT subroutine, 4-23

## L

listener, 1-3, 4-2  
LLO bus command, 4-8  
LLO subroutine, 4-14  
LOCAL LOCKOUT statement, 4-17  
LOCAL statement, 4-16, 4-17  
LOCL subroutine, 4-17  
LU numbers, 3-1  
  and device addresses, 3-1  
  and EQTs, DVTs, and IFTs, 3-1  
  and hardware, 3-2  
  CN command, 3-9  
  EQ command, 3-5  
  IO command, 3-6  
  LU command, 3-4  
  LUPRN utility, 3-4  
  of the bus, 3-1  
  RTE-A assignment, 3-6  
  RTE-6/VM assignment, 3-4  
  SL command, 3-5

## M

message, 4-1  
  command, 4-5  
  data, 4-11  
  service request, 5-2

## N

NDAC handshake line, 1-6  
NRFD handshake line, 1-6

## O

ON INTR statement, 5-3

## P

PACK USING statement, 4-21  
parallel polling, 5-10, 5-16  
  automatic response, 5-11  
  BASIC/1000D subroutines, 5-16  
  disabling, 5-18  
  PPC bus command, 4-6  
  PPOLL function, 5-6  
  PSTAT subroutine, 5-8  
  service programs, 5-9  
PHI chip, 2-3, 4-5  
polling  
  parallel, 5-7, 5-10, 5-12, 5-16, 5-17, 5-20, 5-22  
  serial, 5-1, 5-8, 5-9, 5-10, 5-12, 5-15, 5-18  
  types, 5-8  
PPC bus command  
  PPD secondary command, 4-7  
  PPE secondary command, 4-7  
PPOLL CONFIGURE statement, 5-6  
PPOLL function, 5-7  
PPOLL subroutine, 5-4  
PPOLL UNCONFIGURE statement, 5-7, 5-8  
PPSCH subroutine, 5-4  
PPSN, subroutine, 5-4  
PPU bus commands, 4-8  
PSTAT subroutine, 5-8

## R

READ/ENTER/INPUT/WRITE/PRINT statements, 4-22  
remote mode, 4-16  
REMOTE statement, 4-16  
REN bus command, 4-16  
REN bus management line, 1-7  
RMOTE subroutine, 4-15  
RMPAR subroutine, 5-9, 5-13, 5-14

## S

SDC bus command, 4-6  
secondary addressing, 4-22  
secondary command  
  PPD—Parallel Poll Disable, 4-7  
  PPE—Parallel Poll Enable, 4-7  
SECW/SECR/SECWR/SECRR subroutines, 4-22  
select code, 2-3  
serial polling, 5-9, 5-18  
  automatic, 5-8  
  BASIC subroutines, 5-10, 5-18, 5-19  
  disabling, 5-15  
  GETINTR function, 5-6  
  service programs, 5-9, 5-12  
  SPOLL function, 5-5  
  STATS subroutine, 5-5  
service programs, 5-12  
  bus, 5-12  
service request  
  from BASIC, 5-9, 5-12  
  from FORTRAN, 5-21

- signal lines
  - bus management, 1-7
  - data, 1-5
  - handshake, 1-6
- SPD command, 4-9
- SPE commands, 4-9
- specifications, 1-4
- specified program does not exist, 6-4
- SPOLL function, 5-5
- SRQ bus command, 4-10, 5-2
- SRQ bus management line, 1-7
- SRQ subroutine, 5-2
- SRQSN subroutine, 5-3
- statement
  - ABORT, 4-18
  - CLEAR, 4-15
  - ENABLE INTR, 5-3, 5-19
  - LOCAL, 4-17
  - LOCAL LOCKOUT, 4-17
  - ON INTR, 5-3, 5-19
  - PACK USING, 4-21
  - PPOLL CONFIGURE, 5-6
  - PPOLL UNCONFIGURE, 5-7
  - READ/ENTER/INPUT/WRITE/PRINT, 4-22
  - REMOTE, 4-15
- subroutine
  - ABRT, 4-18
  - BSTAT, 5-5
  - call syntax, 4-12
  - CLEAR, 4-15
  - CMDW/CMDR, 4-20
  - CNFG, 7-6
  - GTL, 4-16
  - IBERR, 6-1
  - IOCNT, 4-23
  - LLO, 4-16
  - LOCL, 4-17
  - PPOLL, 5-6
  - PPSCH, 5-4
  - PPSN, 5-4
  - PSTAT, 5-8
  - RMOTE, 4-15
  - RMPAR, 5-9, 5-17
  - SECW/SECR/SECWR/SECRR, 4-22
  - SRQ, 5-2
  - SRQSN, 5-3
  - STATS, 5-5
  - TRIGR, 4-17
- syntax, 4-12
- TRAP, 5-18, 5-19
- TRIGGER, 4-17

**T**

- talker, 1-3, 4-1
- timeout for transmission error, 6-2
- TRAP statement, 5-18
- TRIGGER statement, 4-17
- TRIGR subroutine, 4-17

**U**

- universal commands, 4-8

**W**

- WELCOM file, 3-6