

# HP BASIC 6.2 Interface Reference



HP Part No. 98616-90013  
Printed in USA

---

## **Notice**

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

Hewlett-Packard Company (HP) shall not be liable for any errors contained in this document. HP MAKES NO WARRANTIES OF ANY KIND WITH REGARD TO THIS DOCUMENT, WHETHER EXPRESS OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory, in connection with the furnishing of this document or the use of the information in this document.

## **Warranty Information**

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

## **Restricted Rights Legend**

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013.

Use of this manual and magnetic media supplied for this product are restricted. Additional copies of the software can be made for security and backup purposes only. Resale of the software in its present form or with alterations is expressly prohibited.

Copyright © Hewlett-Packard Company 1987, 1988, 1989, 1990, 1991

This document contains information which is protected by copyright. All rights are reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Copyright © AT&T Technologies, Inc. 1980, 1984, 1986

Copyright © The Regents of the University of California 1979, 1980, 1983,  
1985-86

This software and documentation is based in part on the Fourth Berkeley  
Software Distribution under license from the Regents of the University of  
California.

---

## **Printing History**

First Edition - June 1991



# Contents

---

<b>1. Display Interfaces</b>	
Description of Displays . . . . .	1-1
Types of Display Devices for BASIC/WS . . . . .	1-2
How the Default Display Device Is Chosen . . . . .	1-3
Overview of Display Features . . . . .	1-4
Display Regions . . . . .	1-4
Clearing the Screen . . . . .	1-6
The Output Area and the Display Line . . . . .	1-6
Determining Screenwidth and Other Attributes . . . . .	1-8
Changing Pen Colors in Display Regions . . . . .	1-10
Pen Colors for BASIC/WS . . . . .	1-12
Interaction Between Alpha and Graphics . . . . .	1-16
Output to the CRT . . . . .	1-17
Numeric Outputs . . . . .	1-17
String Outputs . . . . .	1-18
Control Characters . . . . .	1-18
Display-Enhancement Characters . . . . .	1-21
Monochrome Enhancement Characters . . . . .	1-22
Color Enhancements . . . . .	1-24
Display Enhancement Guidelines . . . . .	1-25
The Display Functions Mode . . . . .	1-25
Output-Area Memory . . . . .	1-27
Determining Above-Screen Lines . . . . .	1-27
The PRINT Position . . . . .	1-29
Scrolling the Display . . . . .	1-30
Entering from the CRT . . . . .	1-32
Reading a Screen Line . . . . .	1-32
Reading the Entire Output-Area Memory . . . . .	1-33
Final Display . . . . .	1-34
Additional CRT Features . . . . .	1-35

The DISP Line . . . . .	1-35
Changing Pen Colors . . . . .	1-36
Disabling the Cursor Character . . . . .	1-37
Enabling the Insert Mode . . . . .	1-37
Softkey Labels . . . . .	1-38
Softkey Label Colors . . . . .	1-40
Summary of CRT STATUS and CONTROL Registers . . . . .	1-40

## 2. The Keyboard Interface

Description of Keyboards . . . . .	2-1
Types of Keyboards . . . . .	2-1
How the Primary Keyboard Is Chosen for BASIC/WS . . . . .	2-3
HP 98203 Keyboard Compatibility Mode . . . . .	2-4
Re-Configuring HIL Devices . . . . .	2-4
Overview of Keyboard Features . . . . .	2-4
ASCII and Non-ASCII Keys . . . . .	2-5
The Shift and Control Keys . . . . .	2-5
Keyboard Operating Modes . . . . .	2-9
The Caps Lock Mode . . . . .	2-9
The Print All Mode . . . . .	2-10
Disabling Scrolling . . . . .	2-10
Modifying the Repeat and Delay Intervals . . . . .	2-11
Entering Data from the Keyboard . . . . .	2-13
Sending the EOI Signal . . . . .	2-15
Sending Data to the Keyboard . . . . .	2-15
Sending Non-ASCII Keystrokes to the Keyboard . . . . .	2-16
Second Byte of Non-ASCII Key Sequences (String) . . . . .	2-17
Closure Keys . . . . .	2-24
Softkeys . . . . .	2-26
Sensing Knob Rotation . . . . .	2-27
Enhanced Keyboard Control . . . . .	2-29
Trapping Keystrokes . . . . .	2-30
Mouse Keys . . . . .	2-32
Softkeys and Knob Rotation . . . . .	2-33
Disabling Interactive Keyboard . . . . .	2-33
Locking Out the Keyboard . . . . .	2-34
Special Considerations . . . . .	2-35
Keyboard Status and Control Registers . . . . .	2-36

### 3. The HP-IB Interface

Introduction . . . . .	3-1
Initial Installation and Verification . . . . .	3-2
Communicating with Devices . . . . .	3-3
HP-IB Device Selectors . . . . .	3-3
Moving Data Through the HP-IB . . . . .	3-4
Using an Interface in the HP-UX Environment . . . . .	3-5
Locking an Interface to a Process . . . . .	3-6
Using the Burst I/O Mode . . . . .	3-6
General Structure of the HP-IB . . . . .	3-7
Addressing Multiple Listeners . . . . .	3-11
Secondary Addressing . . . . .	3-12
General Bus Management . . . . .	3-12
Remote Control of Devices . . . . .	3-13
Locking Out Local Control . . . . .	3-14
Enabling Local Control . . . . .	3-15
Triggering HP-IB Devices . . . . .	3-15
Clearing HP-IB Devices . . . . .	3-16
Aborting Bus Activity . . . . .	3-16
HP-IB Service Requests . . . . .	3-17
Setting Up and Enabling SRQ Interrupts . . . . .	3-17
Servicing SRQ Interrupts . . . . .	3-18
Polling HP-IB Devices . . . . .	3-19
Configuring Parallel Poll Responses . . . . .	3-19
Conducting a Parallel Poll . . . . .	3-20
Disabling Parallel Poll Responses . . . . .	3-20
Conducting a Serial Poll . . . . .	3-21
Special Case: Serial Polling a Non-Active Controller . . . . .	3-21
Advanced Bus Management . . . . .	3-22
The Message Concept . . . . .	3-22
Types of Bus Messages . . . . .	3-22
Bus Commands and Codes . . . . .	3-24
Address Commands and Codes . . . . .	3-26
Explicit Bus Messages . . . . .	3-28
HP-IB Message Mnemonics . . . . .	3-31
The Computer As a Non-Active Controller . . . . .	3-34
Determining Controller Status and Address . . . . .	3-35
Changing the Controller's Address . . . . .	3-36

Passing Control . . . . .	3-36
Restrictions to Passing Control with BASIC/UX . . . . .	3-37
Interrupts While Non-Active Controller . . . . .	3-37
Addressing a Non-Active Controller . . . . .	3-43
Requesting Service . . . . .	3-44
Responding to Parallel Polls . . . . .	3-45
Responding to Serial Polls . . . . .	3-48
Interface-State Information . . . . .	3-48
Servicing Interrupts that Require Data Transfers . . . . .	3-50
HP-IB Control Lines . . . . .	3-53
Handshake Lines . . . . .	3-54
The Attention Line (ATN) . . . . .	3-54
The Interface Clear Line (IFC) . . . . .	3-55
The Remote Enable Line (REN) . . . . .	3-55
The End or Identify Line (EOI) . . . . .	3-55
The Service Request Line (SRQ) . . . . .	3-56
Determining Bus-Line States . . . . .	3-56
Summary of HP-IB STATUS and CONTROL Registers . . . . .	3-58
HP-IB Status and Control Registers (continued) . . . . .	3-59
HP-IB Status and Control Registers (continued) . . . . .	3-60
HP-IB Status and Control Registers (continued) . . . . .	3-61
HP-IB Status and Control Registers (continued) . . . . .	3-62
HP-IB Status and Control Registers (continued) . . . . .	3-63
HP-IB Status and Control Registers (continued) . . . . .	3-64
Summary of HP-IB READIO and WRITEIO Registers . . . . .	3-65
READIO Registers . . . . .	3-65
HP-IB WRITEIO Registers . . . . .	3-71
Summary of Bus Sequences . . . . .	3-78
ABORT . . . . .	3-79
CLEAR . . . . .	3-79
LOCAL . . . . .	3-80
LOCAL LOCKOUT . . . . .	3-80
PASS CONTROL . . . . .	3-81
PPOLL . . . . .	3-81
PPOLL CONFIGURE . . . . .	3-82
PPOLL UNCONFIGURE . . . . .	3-82
REMOTE . . . . .	3-83
SPOLL . . . . .	3-84

TRIGGER . . . . .	3-85
-------------------	------

#### 4. RS-232C Serial Interfaces

Overview . . . . .	4-1
Asynchronous Data Communication . . . . .	4-2
Character Format . . . . .	4-2
Parity . . . . .	4-3
Error Detection . . . . .	4-4
Data Transfers Between Computer and Peripheral . . . . .	4-5
Overview of Serial Interface Programming . . . . .	4-5
Determining Operating Parameters . . . . .	4-6
Handshake and Baud Rate . . . . .	4-6
Character Format Parameters . . . . .	4-6
Using BASIC/WS Interface Defaults to Simplify Programming . . . . .	4-7
Modem-Line Disconnect Switches . . . . .	4-7
Baud Rate Select Switches . . . . .	4-8
Line-Control Switches . . . . .	4-8
Serial Configuration for BASIC/UX . . . . .	4-9
Defaults for the Serial Interface . . . . .	4-9
Configuring a Serial Interface for BASIC/UX . . . . .	4-10
Using Program Control to Override Defaults . . . . .	4-11
Interface Reset . . . . .	4-11
Selecting the Baud Rate . . . . .	4-12
Setting Character Format and Parity . . . . .	4-13
Transferring Data . . . . .	4-14
Entering and Outputting Data . . . . .	4-14
Outputting Data . . . . .	4-14
Entering Data . . . . .	4-15
Modem Line Handshaking (HP BASIC/WS only) . . . . .	4-15
BASIC/UX Modem Line Handshaking . . . . .	4-16
Incoming Data Error Detection and Handling (BASIC/WS only) . . . . .	4-18
Trapping Serial Interface Errors on BASIC/WS . . . . .	4-19
Advanced Programming Information . . . . .	4-21
RS-232 Software Portability . . . . .	4-21
Sending BREAK Messages . . . . .	4-24
Using the Modem Control Register . . . . .	4-24
Modem Handshake Lines (RTS and DTR) . . . . .	4-24

Programming the DRS Modem Line for BASIC/UX . . . . .	4-25
Programming the DRS and SRTS Modem Lines for BASIC/WS . . . . .	4-25
Configuring the Interface for BASIC/WS Self-test Operations	4-25
READIO and WRITEIO Registers for BASIC/WS . . . . .	4-26
Interface Hardware Registers . . . . .	4-27
UART Registers . . . . .	4-29
Cable Options and Signal Functions . . . . .	4-37
The DTE Cable . . . . .	4-37
Optional Circuit Driver/Receiver Functions . . . . .	4-39
The DCE Cable . . . . .	4-39
RS-232C / CCITT V.24 . . . . .	4-44
HP 98626 and HP 98644 Serial Interface STATUS and CONTROL Registers . . . . .	4-46
Model 216 and 217 Built-In Interface Differences for BASIC/WS	4-55
HP 98644 Interface Differences . . . . .	4-55
Hardware Differences . . . . .	4-56
Card ID Register . . . . .	4-56
Optional Driver Receiver Circuits . . . . .	4-56
Configuration Switches . . . . .	4-57
Coverplate Connector . . . . .	4-57
Cables . . . . .	4-59
BASIC Differences . . . . .	4-59
Card ID Register . . . . .	4-59
Optional Driver/Receiver Registers for BASIC/WS . . . . .	4-59
Baud-Rate and Line-Control Registers for BASIC/WS . . . . .	4-60
Series 300 Built-In 98644 Interface Differences . . . . .	4-61
<b>5. Datacomm Interfaces</b>	
Prerequisites . . . . .	5-2
Protocol . . . . .	5-3
Asynchronous Communication Protocol . . . . .	5-3
Data Link Communication Protocol (BASIC/WS Only) . . . . .	5-4
Data Transfers Between Computer and Interface . . . . .	5-6
Outbound Control Blocks . . . . .	5-6
Inbound Control Blocks for BASIC/UX . . . . .	5-7
Inbound Control Blocks for BASIC/WS . . . . .	5-7
Outbound Data Messages . . . . .	5-9

Inbound Data Messages . . . . .	5-10
Overview of Datacomm Programming . . . . .	5-11
RS-232 Software Portability . . . . .	5-12
Establishing the Connection . . . . .	5-15
Determining Protocol and Link Operating Parameters . . . . .	5-15
Datacomm Configuration for BASIC/UX . . . . .	5-16
Defaults for the Serial Interface . . . . .	5-17
Configuring a Datacomm Interface for BASIC/UX . . . . .	5-17
Resetting the Datacomm Interface . . . . .	5-18
Protocol Selection for BASIC/WS . . . . .	5-19
Datacomm Options for Async Communications . . . . .	5-20
Control Block Contents for BASIC/WS . . . . .	5-21
Modem-initiated ON INTR Branching Conditions for BASIC/WS . . . . .	5-21
Datacomm Line Timeouts . . . . .	5-22
Line Speed (Baud Rate) . . . . .	5-23
Handshake . . . . .	5-23
BASIC/UX Modem Line Handshaking . . . . .	5-25
Handling of Non-Data Characters for BASIC/WS . . . . .	5-25
Protocol Handshake Character Assignment for BASIC/WS . . . . .	5-26
End-Of-Line Recognition for BASIC/WS . . . . .	5-26
Prompt Recognition for BASIC/WS . . . . .	5-27
Character Format Definition . . . . .	5-27
Break Timing for BASIC/WS . . . . .	5-28
Datacomm Options for Data Link Communication for BASIC/WS . . . . .	5-29
Control Block Contents for BASIC/WS . . . . .	5-30
ON INTR Branching Conditions and Line Speed for BASIC/WS . . . . .	5-30
Terminal Identification for BASIC/WS . . . . .	5-31
Handshake for BASIC/WS . . . . .	5-31
Transmitted Block Size for BASIC/WS . . . . .	5-32
Parity for BASIC/WS . . . . .	5-32
Connecting to the Line . . . . .	5-32
Switched (Public) Telephone Links . . . . .	5-33
Private Telecommunications Links . . . . .	5-33
Direct Connection Links . . . . .	5-33
Connection Procedure for BASIC/WS . . . . .	5-33

Dialing Procedure for Switched (Public) Modem Links . . . . .	5-33
Automatic Dialing with the HP 13265A Modem . . . . .	5-34
Initiating the Connection . . . . .	5-36
Connection Procedure for Hayes-Compatible Modems . . . . .	5-36
Setting up the Interrupt System for BASIC/WS . . . . .	5-38
Setting up Softkey Interrupts . . . . .	5-39
Setting Up Program Operator Inputs . . . . .	5-39
Setting Up Datacomm Interrupts . . . . .	5-40
Background Program Routines for BASIC/WS . . . . .	5-41
Interrupt Service Routines . . . . .	5-42
Servicing Datacomm Interrupts . . . . .	5-42
Exit Conditions . . . . .	5-45
Data Formats for Datacomm Transfers . . . . .	5-47
Servicing Keyboard Interrupts . . . . .	5-48
Service Routines for ON KEY Interrupts . . . . .	5-50
Cooperating Programs for BASIC/WS . . . . .	5-51
FORTRAN Program COOP for the HP 1000: . . . . .	5-51
Cooperating BASIC Program for the Desktop Computer: . . . . .	5-53
Program File to be Downloaded from the HP 1000: . . . . .	5-56
Modified Cooperating BASIC Program After Loading: . . . . .	5-56
Results: . . . . .	5-56
Terminal Emulator Example Programs for BASIC/WS . . . . .	5-57
Datacomm Programming Helps for BASIC/WS . . . . .	5-65
Terminal Prompt Messages . . . . .	5-65
Prevention of Data Loss on the HP 1000 . . . . .	5-65
Disabling Auto-poll on the HP 1000 . . . . .	5-67
Prevention of Data Loss on the HP 1000 . . . . .	5-67
Secondary Channel, Half-duplex Communication . . . . .	5-68
Automatic Answering Applications . . . . .	5-71
Communication Between Desktop Computers . . . . .	5-74
Datacomm Error Recovery . . . . .	5-75
Datacomm Error Detection and Program Recovery . . . . .	5-76
Cable and Adapter Options and Functions . . . . .	5-76
DTE and DCE Cable Options . . . . .	5-77
Optional Circuit Driver/Receiver Functions . . . . .	5-79
RS-232C/CCITT V24 . . . . .	5-81
The HP 98642 4-Channel Multiplexer . . . . .	5-83
Specifics on the HP 98642 4-Channel Multiplexer . . . . .	5-83



Using the HP 98642 4-Channel Multiplexer . . . . .	5-84
Keywords Used by the HP 98642 4-Channel Multiplexer . . . . .	5-84
HP 98628 and HP 98642 Datacomm Interface Status and Control Registers . . . . .	5-86

**6. The GPIO Interface**

Introduction . . . . .	6-1
Interface Description . . . . .	6-2
Interface Configuration . . . . .	6-3
Interface Select Code . . . . .	6-4
Hardware Interrupt Priority . . . . .	6-5
Data Logic Sense . . . . .	6-5
Data Handshake Methods . . . . .	6-5
Handshake Lines . . . . .	6-6
Handshake Logic Sense . . . . .	6-6
Handshake Modes . . . . .	6-7
Data-In Clock Source . . . . .	6-7
Optional Peripheral Status Check . . . . .	6-7
Full-Mode Handshakes . . . . .	6-8
Pulse-Mode Handshakes . . . . .	6-11
Interface Reset . . . . .	6-18
Outputs and Enters through the GPIO . . . . .	6-19
ASCII and Internal Representations . . . . .	6-19
Example Statements that Output Data Bytes . . . . .	6-20
Example Statements that Enter Data Bytes . . . . .	6-22
Example Statements that Output Data Words . . . . .	6-25
Example Statements that Enter Data Words . . . . .	6-25
Using a GPIO Interface in the HP-UX Environment . . . . .	6-27
Locking an Interface to a Process . . . . .	6-27
Using the Burst I/O Mode . . . . .	6-28
GPIO Timeouts . . . . .	6-29
Timeout Time Parameter . . . . .	6-29
Timeout Service Routines . . . . .	6-29
Using Alternate Data Representations . . . . .	6-31
BCD Representation . . . . .	6-31
Character Conversions . . . . .	6-34
GPIO Interrupts . . . . .	6-36
Types of Interrupt Events . . . . .	6-36

Setting Up and Enabling Events . . . . .	6-36
Interrupt Service Routines . . . . .	6-37
Designing Your Own Transfers . . . . .	6-40
Full Handshake Transfer . . . . .	6-40
Interrupt Transfers . . . . .	6-42
Ready Interrupt Transfers . . . . .	6-42
Using the Special-Purpose Lines . . . . .	6-44
Driving the Control Output Lines . . . . .	6-44
Interrogating the Status Input Lines . . . . .	6-45
Using the PSTS Line . . . . .	6-45
Summary of GPIO STATUS and CONTROL Registers . . . . .	6-46
Summary of GPIO READIO and WRITEIO Registers . . . . .	6-48
GPIO READIO Registers . . . . .	6-49
GPIO WRITEIO Registers . . . . .	6-50
<b>7. The BCD Interface for BASIC/WS</b>	
Brief Description of Operation . . . . .	7-2
Data Representations and Formats . . . . .	7-2
The BCD Data Representation . . . . .	7-2
Standard Format . . . . .	7-3
Optional Format . . . . .	7-7
The Binary Data Representation . . . . .	7-11
The Binary Mode . . . . .	7-11
Alternate Methods of Entering Data . . . . .	7-14
Outputting Data . . . . .	7-14
Configuring the Interface . . . . .	7-15
Determining Interface Configuration . . . . .	7-16
Setting the Interface Select Code . . . . .	7-17
Setting the Hardware Priority (Interrupt Level) . . . . .	7-18
Setting the Peripheral Status Switches . . . . .	7-18
Setting the Handshake Configuration . . . . .	7-18
Type 1 Timing . . . . .	7-19
Type 2 Timing . . . . .	7-20
Configuring the Cable . . . . .	7-21
Interface Reset . . . . .	7-21
Entering Data Through the BCD Interface . . . . .	7-22
Entering Data from One Peripheral . . . . .	7-23
Entering with BCD-Mode Standard Format . . . . .	7-23

Entering with the Binary Mode . . . . .	7-25
Entering with STATUS Statements . . . . .	7-29
Entering Data from Two Peripherals . . . . .	7-31
Optional Format . . . . .	7-31
Outputting Data Through the BCD Interface . . . . .	7-34
Output Routines Using CONTROL and STATUS . . . . .	7-34
Sending Data with OUTPUT . . . . .	7-35
BCD Interface Timeouts . . . . .	7-37
Timeout Time Parameter . . . . .	7-38
Timeout Service Routines . . . . .	7-38
BCD Interface Interrupts . . . . .	7-40
Setting Up and Enabling Interrupts . . . . .	7-40
Interrupt Service Routines . . . . .	7-41
Summary of BCD STATUS and CONTROL Registers . . . . .	7-42
Summary of BCD READIO and WRITEIO Registers . . . . .	7-45
BCD READIO Registers . . . . .	7-45
BCD WRITEIO Registers . . . . .	7-48

## 8. EPROM Programming for BASIC/WS

Introduction . . . . .	8-1
Accessories Required . . . . .	8-1
Hardware Installation . . . . .	8-1
Brief Overview of Using EPROM Memory . . . . .	8-2
Initializing EPROM Memory . . . . .	8-3
EPROM Programmer Select Code . . . . .	8-3
EPROM Addresses and Unit Numbers . . . . .	8-3
Verifying Hardware Operation . . . . .	8-4
Initializing Units . . . . .	8-8
EPROM Directories . . . . .	8-8
EPROM Catalogs . . . . .	8-9
Programming EPROM . . . . .	8-10
Storing Data . . . . .	8-10
Data Storage Rates . . . . .	8-11
Determining Unused EPROM Memory . . . . .	8-12
Storing Programs . . . . .	8-14
Programming Individual Words and Bytes . . . . .	8-15
Operations Not Allowed . . . . .	8-17
Reading EPROM Memory . . . . .	8-18

Retrieving Data and Programs . . . . .	8-18
Summary of EPROM Programmer STATUS and CONTROL Registers . . . . .	8-19

## 9. HP-HIL Interface

The Interface to HP-HIL Devices . . . . .	9-1
Preview of HP-HIL Devices . . . . .	9-2
Communicating through the HP-HIL Interface . . . . .	9-3
Supported HP-HIL Devices . . . . .	9-6
Selecting HP-HIL Devices . . . . .	9-6
Enabling HP-HIL Devices . . . . .	9-7
Identifying All Devices on the HP-HIL Link . . . . .	9-8
Explanation of the HIL_ID Program . . . . .	9-10
Segment 1 of HIL_ID . . . . .	9-10
Segment 2 of HIL_ID . . . . .	9-12
Segment 3 of HIL_ID . . . . .	9-14
Segment 4 of HIL_ID . . . . .	9-17
HP-HIL Devices . . . . .	9-21
HP-HIL Keyboards . . . . .	9-22
Relative Positioners . . . . .	9-22
Absolute Positioners . . . . .	9-25
Security Device . . . . .	9-26
Other Devices . . . . .	9-26
Communicating with HP-HIL Devices . . . . .	9-29
HP-HIL Device Characteristics . . . . .	9-29
ID Module . . . . .	9-30
Device Characteristics . . . . .	9-30
Interpreting ID Module Data . . . . .	9-31
Note about Installing and Removing ID Modules . . . . .	9-31
Function Box and Vectra Keyboard . . . . .	9-33
Determining Function Box Characteristics . . . . .	9-33
Activating the Function Box . . . . .	9-34
Trapping Key Presses . . . . .	9-36
Assigning Functions to Keys . . . . .	9-39
Using a Touchscreen . . . . .	9-44
Device Characteristics . . . . .	9-44
Plotting Selected Locations . . . . .	9-45
Using a Bar Code Reader . . . . .	9-49

Determining Bar Code Reader Characteristics . . . . .	9-50
Reading a Selected Bar Code . . . . .	9-51
Interaction Among Multiple HP-HIL Devices . . . . .	9-53
Modifying the Interactive Program . . . . .	9-56

## 10. The Parallel Interface

Introduction . . . . .	10-1
Required Software and Hardware . . . . .	10-1
Bus Description . . . . .	10-2
The Data Lines . . . . .	10-3
The Handshake Lines . . . . .	10-4
The Error Lines . . . . .	10-4
The Status Lines . . . . .	10-5
The Reset Line . . . . .	10-5
Summary of Parallel Interface STATUS and CONTROL Registers	10-5
Summary of Parallel Interface READIO and WRITEIO Registers	10-13
Parallel READIO Registers . . . . .	10-13
Parallel WRITEIO Registers . . . . .	10-17

## A. HP-HIL Appendix

HP-HIL Command Reference . . . . .	A-1
Identify and Describe (IDD) . . . . .	A-2
Read Register (RRG) . . . . .	A-2
Write Register (WRG) . . . . .	A-3
Report Name (RNM) . . . . .	A-4
Report Status (RST) . . . . .	A-4
Extended Describe (EXD) . . . . .	A-4
Report Security Code (RSC) . . . . .	A-5
Disable Keyswitch Autorepeat (DKA) . . . . .	A-5
Enable Keyswitch Autorepeat (EKA 1,EKA 2) . . . . .	A-6
Prompt 1 thru Prompt 7 (PRM 1 .. PRM 7) . . . . .	A-6
Prompt (PRM) . . . . .	A-7
Acknowledge 1 thru Acknowledge 7 (ACK 1 .. ACK 7) . . . . .	A-7
Acknowledge (ACK) . . . . .	A-8
Device-Dependent Commands (DDC 128 .. 239) . . . . .	A-8
Device ID Byte . . . . .	A-8
Describe Record . . . . .	A-11
Extended Describe Record . . . . .	A-15

Poll Record . . . . .	A-19
Report Security Code Record . . . . .	A-21
Sample of Report Security Format for a Product Module . . .	A-25
Sample of Report Security Format for An Exchange Module .	A-25
Accessible Keycode Definitions . . . . .	A-27

**Index**

# Display Interfaces

---

This chapter describes programming techniques for sending data to and entering data from display interfaces. For information on using a display for graphics, see the *HP BASIC 6.2 Programming Guide*. Configuring and accessing these devices with I/O statements (OUTPUT, ENTER, STATUS, and CONTROL) is described in this chapter. The display screen is a convenient tool for visually verifying data output before attempting to send it to another device.

---

## Description of Displays

This section provides an overview of the capabilities of display devices available with Series 300 computers. Here are the topics:

- Types of display devices.
- How the “default display device” is chosen (in machines with more than one display installed).
- Overview of display features.

## Types of Display Devices for BASIC/WS

There are essentially two types of displays available with Series 200/300 computers:

- Displays with *separate* alpha and graphics planes:
  - The alpha screen is produced by character-generator hardware.
  - The graphics screen is produced by bit-mapping hardware.
  - Each can be turned on and off independently.
- Displays with *combined* alpha and graphics planes:
  - Alpha and graphics share the same screen.
  - Both are produced by bit-mapping hardware.
  - Alpha and graphics cannot be turned on and off independently.

All Series 200 computers have separate alpha and graphics, except the Model 237 which has a combined display.

All Series 300 computer displays have combined alpha and graphics. However, you can configure Series 300 color (multi-plane) displays to use independent alpha and graphics planes. You can also install the HP 98546 Display Compatibility Interface in a Series 300 computer to get separate alpha and graphics planes (this interface is essentially the display of the Model 217). Configuring the color display and using the compatibility display are both discussed in the *HP BASIC 6.2 Programming Guide*. Interactions between graphics and alpha planes are also described in the *HP BASIC 6.2 Programming Guide*.

HP BASIC/DOS for the HP Measurement Coprocessor emulates a Series 300 bit-mapped display on the PC display.



## How the Default Display Device Is Chosen

Select code 1 (BASIC provides the CRT function, which returns a value of 1) is always assigned to the “default display device.” However, as previously mentioned, Series 300 computers can have more than one display installed at one time. In such cases, the BASIC system has to choose which display will be the default display device. The following list indicates the order that the system chooses this device assuming that both display drivers, CRTA and CRTB, are present. If CRTB is not present, steps 1 and 2 are skipped.

1. The “internal” bit-mapped display.
  - a. A Series 300 bit-mapped display, including Model 362 and 382 built-in display.
  - b. The HP 98700 Display Controller at “internal” select code 6 (i.e., the “internal/external” switch is set to “internal”).
2. An “external” bit-mapped display.
  - a. An HP 98700 Display Controller at the lowest “external” select code (i.e., the “internal/external” switch is set to “external”, and the select code switches are set to a select code in the range 8 through 31).
3. The “internal” non-bit-mapped display (BASIC/WS only).
  - a. The HP 98546 Display Compatibility Interface.

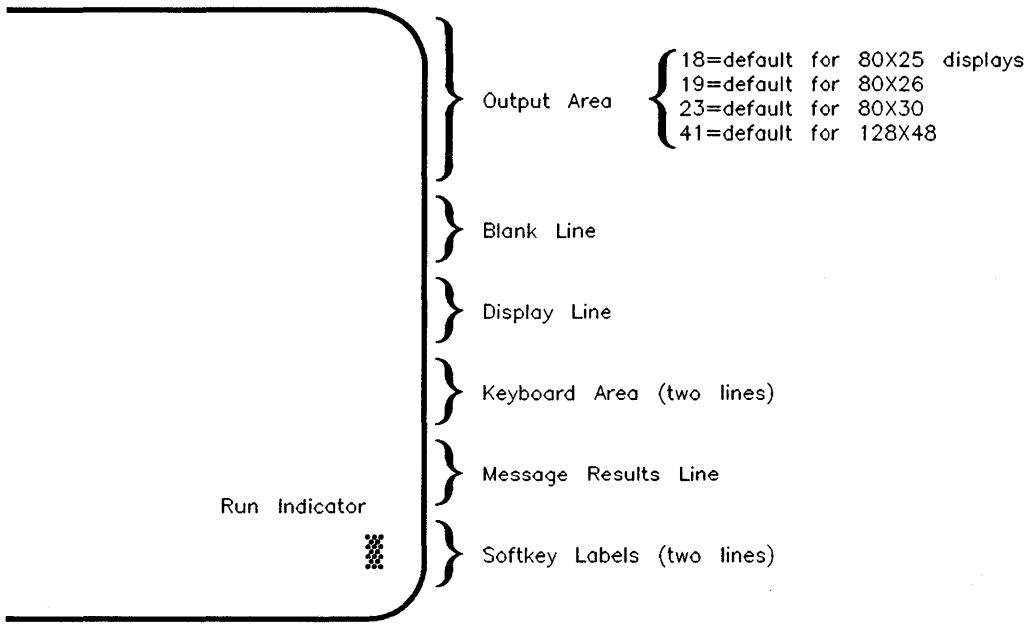
## Overview of Display Features

Even though there are several choices of displays available with Series 300 computers, all have similar alphanumeric display capabilities. Here are the general categories of I/O operations you can perform with the alphanumeric display.

- You can OUTPUT characters to any location on the screen through select code 1 (this applies only to the logical region known as the “output area,” which is described in the next section).
- You can read characters from any location of the output area with the ENTER statement through select code 1. A line-feed character, CHR\$(10), is sent following the last non-blank character on the line. (A simulated HP-IB End-or-Identify, EOI, signal is also sent with the line-feed.)
- You can configure and read the current modes of display operation with CONTROL and STATUS registers.

### Display Regions

The BASIC system logically partitions the alphanumeric display into several regions, each of which has a specific use. Here is a diagram of the regions.



### BASIC Display Organization

Here is a description of each region, from top to bottom of the display.

- The “Output Area” is the logical region on which characters sent to the screen are displayed (characters sent with OUTPUT through select code 1, or with PRINT when PRINTER IS CRT).
- One blank display line separates the output area from other areas.
- One display line is used for output using the DISP statement.
- Two display lines are used for keyboard input and output (input with INPUT, LINPUT, and ENTER through select code 2; OUTPUT through select code 2).
- One display line is used for system “messages” and “results.” These include error messages or results of keyboard computations.

The right end of this line is used by the BASIC system for “annunciators” such as softkey mode (such as System or User 1), CAPS indicator, and

system activity indicator (such as *Running*, *Idle*, or *Command*). For more information, see the *Using HP BASIC* manual for your system.

- Two lines at the bottom of the screen are used for softkey labels. (See the subsequent section called “Softkey Labels” for further information.)

While the number of lines in the output area may vary according to display size, BASIC provides all of these regions on all alpha displays.

## Clearing the Screen

Alpha display memory can be cleared using the `CLEAR SCREEN` statement.

```
CLEAR SCREEN
```

It has the same effect as executing:

```
OUTPUT KBD;CHR$(255)&"K";
```

or pressing the Clear display key.

Note that you can type `CLS` and the statement `CLEAR SCREEN` will appear in the program listing. `CLS` acts as a shorthand method of entering the `CLEAR SCREEN` statement.

## The Output Area and the Display Line

There are two separate areas on the CRT used for displaying characters:

- Characters sent to the CRT with `PRINT` and `OUTPUT CRT; ...` statements are displayed in the output area.
- Characters sent to the display with the `DISP` statement appear in the display line.

Type in and run the following program to see these areas.

```
10 ALPHA HEIGHT 25 ! or CONTROL 1,13;25
20 FOR K=1 TO 18
30 PRINT "This is line #";K;" in the output area."
40 WAIT .5
50 DISP "This is line #";K;" in the display line."
60 WAIT .5
70 NEXT K
80 END
```

The number of Output Area lines available can vary depending on what display you have.

- Series 200 Models 216, 217, 220, 226, and 236 computers with non-bit-mapped displays, and Series 300 computers using an HP 98546A Video Compatibility Interface, provide 18 lines (BASIC/WS only).
- Series 300 medium-resolution (bit-mapped alpha) displays provide 19 lines.
- Series 300 Models 362 and 382 with 640x480 internal displays provide 23 lines.
- Model 237 and Series 300 high-resolution (bit-mapped alpha) displays provide 41 or 44 lines.

The number of lines in the output area can be altered, within display-size limits, by either of these statements:

`ALPHA HEIGHT Number of lines`

or

`CONTROL CRT,13; Number of lines`

The number of lines specifies an area of the display that begins *at the bottom of the screen*. That is, if you have a default Output Area size of 18 and you execute ALPHA HEIGHT 9, the new Output Area will be the 2 lines at the bottom of the screen (just above the Display Line). The lower limit of the alpha height for the root window and CRT is 9. The upper limit is 25, 26, 30, 48, or 51 depending on the size of display the upper limit for your display. Experiment with different parameters for the ALPHA HEIGHT statement in line 10 in the preceding program. Try integers in the range 9 to 51. Error messages alert you to out-of-range situations.

You should experiment with the loop upper limit in line 20. After the program executes, scroll the displayed text up and down to see what happens. Try values such as 10, 30, 48, 49, 51, and 52.

To determine the current height of the alpha area for a CRT or window number, use STATUS register 13:

`STATUS 1,13;Height`

To re-establish the default alpha height, omit the “number of lines” parameter in the preceding ALPHA HEIGHT statement:

```
ALPHA HEIGHT
```

The edit mode in BASIC/WS requires a minimum alpha height of 11 lines. If the alpha height is 9 or 10 lines, going into the edit mode will change it to 11 lines.

Windows created with the `rmb` command (it runs the BASIC interpreter on HP-UX) must have a height of at least 1 for the “`rmb*Geometry`” entry of the `X.defaults` file. Windows with height values less than 10 will have an alpha height of 9, even if the displayed window has fewer lines. This means that output to the windows would be clipped.

## Determining Screenwidth and Other Attributes

A wide variety of displays having different characteristics are available for Series 200/300 computers. Since all programs are transportable between these computers, a program that uses the display extensively should have the ability to distinguish the characteristics of the display it is using. This BASIC language system provides this capability.

Interface select code 1 is used to access the CRT from BASIC programs. You can also use a window number in the range from 600 to 699. Several registers are associated with this interface which allow the interrogation and control of the CRT/window. For example, STATUS register 13 (discussed previously) returns the current alpha height; STATUS register 9 of the CRT interface returns the screenwidth:

```
STATUS 1,9;Screenwidth
```

STATUS register 19 of the CRT interface returns the maximum alpha mask of the CRT:

```
STATUS 1,19;Max_alpha_mask
```

You can also use `SYSTEM$("CRT ID")` to determine screenwidth (and other display attributes). Executing this function returns a string similar to the following:

```
6: 80HCGB15
```

where:

- 6: is a format indicator for Series 300 computers.
- 80 is the CRT width in characters.
- H indicates that CRT highlights are available (blinking, underlining, etc.).
- C indicates that color is available.
- G indicates that graphics is available.
- B indicates that your display is bit-mapped.
- 15 indicates the highest graphics pen number:
  - 1 if monochrome
  - $2^n - 1$  if bit-mapped ( $n$  = number of planes)

The numeric characters following the bit-mapped indicator represent the CRT “max pen” value (highest graphics pen number available on your display: for instance 4-plane color displays have a max pen equal to 15, 6-plane bit-mapped displays have a max pen equal to 63 or  $(2^6 - 1)$ , and so forth).

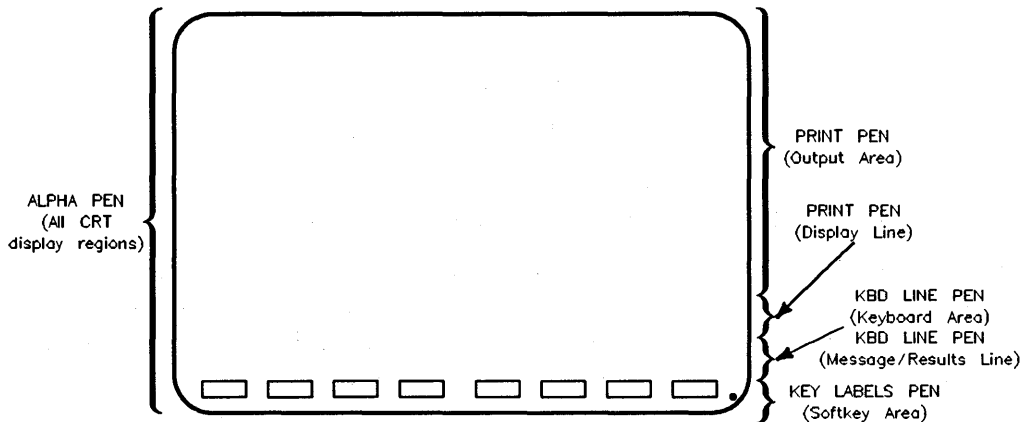
For more information on `SYSTEM$("CRT ID")` results, read the section covering the `SYSTEM$` function in the *HP BASIC Language Reference*.

## Changing Pen Colors in Display Regions

This section covers a set of statements which allow you to change the alpha pen colors in five of the display regions on your CRT/window. Here is a list of the BASIC statements which change the alpha pen colors of the display regions:

- ALPHA PEN *pen value* (same as CONTROL CRT,5; *pen value*)
- KBD LINE PEN *pen value* (same as CONTROL CRT,17; *pen value*)
- KEY LABELS PEN *pen value* (same as CONTROL CRT,16; *pen value*)
- PRINT PEN *pen value* (same as CONTROL CRT,15; *pen value*)

The diagram below shows the areas which these statements affect.



Note that the KBD LINE PEN statement sets the color for more than just the Message/Results Line. This statement also sets the color for the “run-light” and for the program lines listed on screen in the edit mode.



The following table should help to clarify the previous diagram of the display regions affected by pen color statements.

**Display Regions Affected by Pen Color Statements**

<b>Display Region</b>	<b>Written by These Statements</b>	<b>Statements which Affect Pen Colors</b>
Output area	PRINT OUTPUT CAT LIST etc.	ALPHA PEN PRINT PEN
Display line	DISP INPUT (prompt only) LINPUT (prompt only)	ALPHA PEN PRINT PEN
Edit area (replaces Output area and Display line in edit mode)	EDIT	ALPHA PEN KBD LINE PEN
Keyboard area	Keyboard input	ALPHA PEN KBD LINE PEN
Message Results Line	Error messages, keyboard computation results, and annunciators	ALPHA PEN KBD LINE PEN
Softkey Labels	EDIT KEY SET KEY LOAD KEY ON KEY	ALPHA PEN KEY LABELS PEN

## Pen Colors for BASIC/WS

When speaking of alpha pen colors, there are two categories of displays which need to be mentioned:

- Displays with bit-mapped alpha (such as a Series 300 computer using an HP 98543A Color Display). These can be either color or gray scale displays. The examples in this chapter use color displays. For information about gray scales, refer to *HP BASIC 6.2 Advanced Programming Techniques*.
- Displays without bit-mapped alpha (such as a Model 236C).

Displays *without* bit-mapped alpha use a set of alpha pen colors as described in the section entitled "CRT Status and Control Registers" found at the end of this chapter. Note that the alpha and graphics pen colors are *different* for displays without bit-mapped alpha. However, for displays *with* bit-mapped alpha, the alpha and graphics pen colors are the *same*.

### *Pen Colors Example*

The following program shows the use of all the alpha pen color statements previously mentioned.

```

100 ALPHA PEN 1           ! White (for all display regions).
110 ON KEY 1 LABEL "KeyLabel" GOSUB Sftkey_label
120 GOSUB Show_regions
130 WAIT 3
140 !
150 PRINT PEN 2          ! Red (OUTPUT/PRINT and DISP regions).
160 KBD LINE PEN 3       ! Yellow (Keyboard input line).
170 KEY LABELS PEN 4    ! Green (Softkey labels).
180 WAIT 3
190 CLEAR LINE
200 GOSUB Show_regions
210 STOP
220 !
230 Show_regions: !
240 PRINT "PRINT/OUTPUT Area"
250 DISP "DISP Line";
260 OUTPUT KBD;"Message/Results Line";CHR$(255)&"E";
270 OUTPUT KBD;RPT$("KBD Line ",10);
280 Sftkey_label:RETURN
290 END

```

The results on the display (through the WAIT 3 statement on line 180) are output using an alpha pen color of white.

PRINT/OUTPUT Area

DISP Line

KBD Line KBD Line KBD Line KBD Line KBD Line KBD Line KBD Line KBD Line

KBD Line KBD Line

Message/Results Line

User 1

Idle

KeyLabel Continue RUN SCRATCH LOAD "" LOAD BIN LIST BIN RE-STORE

""

""

The above program assumes you are using a medium resolution monitor with a Series 300 computer and that you are in the non-color-map mode. The following is an explanation of the program:

*Line 100* selects white as the alpha pen color.

*Line 120* calls a subroutine called `Show_regions`. This subroutine causes the output of messages to the five CRT display regions. The color of these messages is determined by the alpha pen color statement executed prior to the call to the subroutine. Note that the color may vary from those stated if you are running the program from the X Window environment.

*Line 130* causes the program to wait for 3 seconds before proceeding.

*Line 150* sets the pen color to red for the output area and display line of the CRT (existing text is not affected).

*Line 160* sets the pen color to yellow for the keyboard area and message/results line of the CRT. (Note that the Keyboard lines, annunciators, and "run-light" are updated, the result text is not.)

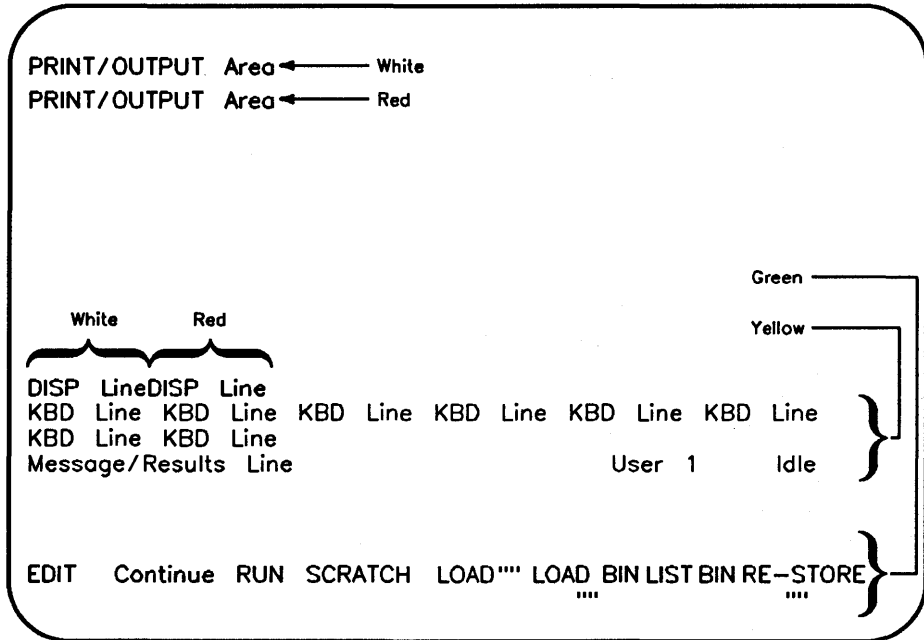
*Line 170* sets the pen color to green for the softkey area of the CRT.

*Line 180* causes the program to wait for 3 seconds before proceeding.

*Line 200* calls the subroutine called `Show_regions`. This subroutine causes the output of messages to the five CRT display regions. The color of these messages is determined by the pen color statements executed prior to the call to the subroutine.

The remaining program lines are the subroutine called `Show_regions` which has been previously explained.

The following display shows what happens after the WAIT 3 statement on line 180.



The above display assumes you have a medium-resolution color monitor.

---

## Interaction Between Alpha and Graphics

On Series 300 computers the alphanumeric and graphic outputs are normally combined. However, it is possible, in multi-plane bit-mapped displays, to make the alpha text independent from the graphics image with the statement **SEPARATE ALPHA FROM GRAPHICS** (see the *HP BASIC 6.2 Language Reference* for more details). With the display in this mode, the user can independently control the alpha and graphic screens by using the corresponding keys on the keyboard or through the **ALPHA ON/OFF** and **GRAPHICS ON/OFF** statements.

For example:

```
10  GINIT
20  SEPARATE ALPHA FROM GRAPHICS
30  PLOTTER IS CRT;"INTERNAL";COLOR MAP
40  CLEAR SCREEN
50  FOR I=1 TO 18
60  PRINT "This is line";I;"of alphanumeric output"
70  NEXT I
80  MOVE 30,30
90  RECTANGLE 50,50 FILL
100 DISP "Use the Alpha and Graphics keys to turn them on and off."
110  END
```

*Line 10* initializes the graphics display parameters.

*Line 20* separates alpha from graphics.

*Line 30* defines the CRT as the output for graphics statements.

*Line 40* erases the alpha screen.

*Lines 50 through 70* fill the output area with text.

*Lines 80 through 90* draw a solid box over the bottom center of the output area.

*Line 100* prompts the user to experiment with the new relationship between alpha and graphics.

---

## Output to the CRT

Data can also be sent to the output area by directing OUTPUT statements to interface select code 1 or a window number. The following example uses an I/O path name to direct the data to the CRT; the default data representation used with the CRT is the ASCII representation.

```

100  ASSIGN @Printer TO 1
110  !
120  CLEAR SCREEN
130  FOR Line=1 TO 18
140      OUTPUT @Printer;"The OUTPUT Area"
150  NEXT Line
160  !
170  END

```

## Numeric Outputs

When numbers are output to the CRT/window by the free-field form of the OUTPUT statement, the standard numeric format is used. ("Standard numeric format" is further described in the chapter "Outputting Data".)

The following statements show how trailing punctuation within the OUTPUT statement affects the item terminators output after each numeric item.

<i>Examples</i>	<i>Results</i>
OUTPUT 1;123,456	123, 456
OUTPUT 1;-123,456	-123, 456
OUTPUT 1;-123,-456	-123,-456
OUTPUT 1;-123;-456	-123-456
OUTPUT 1;123;456	123 456

Leading + signs are replaced by a space.

## String Outputs

Strings are output to the CRT or window in a similar manner with free-field outputs; trailing punctuation in the statement determines whether or not string-item and statement terminators are output. The following examples show how trailing punctuation within the OUTPUT statement affects the output of string-item terminators.

<i>Examples</i>	<i>Results</i>
OUTPUT 1;"One","Two"	One Two
OUTPUT 1;"Three";"Four"	ThreeFour

As with free-field outputs to other devices, a trailing semicolon causes the separator of the item that it follows to be suppressed. In the above case, the carriage-return and line-feed separators which normally follow the output of a string item are suppressed by the semicolon. The next paragraphs describe how the carriage-return and line-feed (control characters) are interpreted by the CRT.

## Control Characters

ASCII characters with codes 0 through 31 are defined to be “control” characters. When one of these characters is sent to a system resource, it is usually interpreted as a *command*, rather than as data. The complete list of control characters and their corresponding codes and definitions is given in the ASCII table in “Useful Tables” of the *HP BASIC 6.2 Language Reference*.

*Four* of these characters are used for controlling the CRT display, and all others are ignored (i.e., are not displayed and cause no special action when received by the CRT). Run the following program and note the results.

```

130 Backspace$=CHR$(8)
140 Line_feed$=CHR$(10)
150 Form_feed$=CHR$(12)
160 Carriage_return$=CHR$(13)
170 !
180 !
190 ASSIGN @Crt TO 1
200 !
210 OUTPUT @Crt;"Back";
220 WAIT 1

```

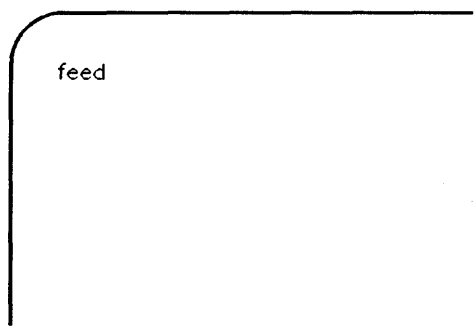


```

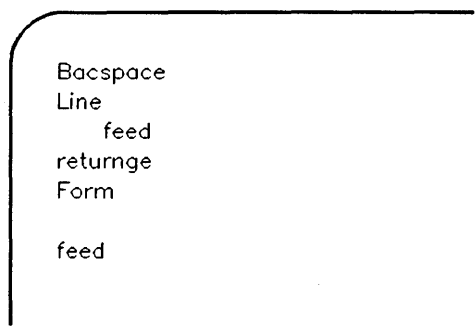
230 OUTPUT @Crt;Backspace$;"space"
240 WAIT 1
250 !
260 OUTPUT @Crt;"Line";
270 WAIT 1
280 OUTPUT @Crt;Line_feed$;"feed"
290 WAIT 1
300 !
310 OUTPUT @Crt;"Carriage";
320 WAIT 1
330 OUTPUT @Crt;Carriage_return$;"return"
340 WAIT 1
350 !
360 OUTPUT @Crt;"Form";
370 WAIT 1
380 OUTPUT @Crt;Form_feed$;"feed"
390 DISP "Scroll down to view previous display."
400 !
410 END

```

Display Before Scroll



Display After Scroll



The following table describes the display functions invoked when the specified control character is sent to the CRT/window (in the "Display functions off" mode). The **print position** is the column and line at which the next character sent to the display will appear.

**Control-Character Functions on the CRT**

Character	ASCII Code	Defined Action
Bell	7	Causes beeper to output the standard tone; no display action is invoked.
Backspace (BS)	8	If the print position was not in column 1, it is moved "back" one character position; if it was in the first column, no action is invoked.
Line-feed (LF)	10	Moves the print position "down" one line.
Form-feed (FF)	12	Prints two blank lines, scrolls the screen "up" as far as possible, and places the print position at column 1 of the second, printed blank line.
Carriage-return (CR)	13	Causes the print position to be moved to the beginning (first column) of the current screen line.
All other control characters	—	Ignored.

Here is another short example program. Type it in and watch the execution. Notice how you can use CONTROL statements and selected registers to control the CRT display.

```

100 CONTROL 1,1;10           ! Go to 10th line
110 PRINT "Let's back up a little." ! Print a sentence
120 WAIT .6                  ! Time to see it.
130 FOR K=24 TO 1 STEP -1    ! Loop for backspacing.
140     CONTROL 1,1;10      ! Stay on 10th line. Could do other ways.
150     CONTROL 1,0;K       ! Move cursor back one space.
160     PRINT CHR$(8);" " ; ! Issue backspace.
170     WAIT .3             ! Lets you see backspacing.
180 NEXT K                  ! Continue loop.
190 END

```

## Display-Enhancement Characters

Characters with codes CHR\$(128)-CHR\$(159) are a second set of “control” characters. Some of these characters are used to implement display enhancements. The others are ignored.

Newer revisions of BASIC (6.x and above) support both one- and two-byte display enhancement characters. The one-byte display enhancement characters are the same as those used in BASIC/WS/UX 5.x and previous versions; they are CHR\$(128)-CHR\$(143). The two-byte display enhancement characters are created by adding CHR\$(255) before the one-byte display enhancement character. For example:

```
PRINT CHR$(132);A$;CHR$(128) ! Underline on/off
```

```
PRINT CHR$(255)&CHR$(132);A$;CHR$(255)&CHR$(128) ! Underline on/off
```

One- and two-byte display-enhancement characters are functionally equivalent. One-byte display-enhancement characters are supported for backward compatibility only. Whenever you write a new program, use two-byte display-enhancement characters.

## Monochrome Enhancement Characters

Some displays have the ability to display underlined, blinking, and inverse-video characters. Both the Output Area and the Display Line have these abilities.

### Monochrome Display Enhancements

Character		Resulting Enhancement
<i>One-byte</i>	<i>Two-byte</i>	
CHR\$(128)	CHR\$(255)&CHR\$(128)	All enhancements off
CHR\$(129)	CHR\$(255)&CHR\$(129)	Inverse video on
CHR\$(130)	CHR\$(255)&CHR\$(130)	Blinking on
CHR\$(131)	CHR\$(255)&CHR\$(131)	Inverse video and blinking on
CHR\$(132)	CHR\$(255)&CHR\$(132)	Underline on
CHR\$(133)	CHR\$(255)&CHR\$(133)	Underline and inverse video on
CHR\$(134)	CHR\$(255)&CHR\$(134)	Underline and blinking on
CHR\$(135)	CHR\$(255)&CHR\$(135)	Underline, inverse video, and blinking on

When one of these characters is sent to the CRT/window, it turns on the corresponding enhancement(s). All subsequent characters on the CRT/window are also displayed in the specified enhancement mode; if only a few characters are to be enhanced, a CHR\$(255)&CHR\$(128) must be sent to the display after the last character to be enhanced, which turns off *all* enhancements.

From the preceding table, you may have deduced that certain bits within the character bytes turn on these display modes. The following bit pattern and individual bits control these features.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	0	0	Underline on	Blinking on	Inverse on
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Notice that the upper five bits (7 through 3) must be in the pattern shown (numeric value = 128). Thus, adding the values 4, 2, or 1 enable the Underline, Blinking, and Inverse features. Several examples of using these enhancements are shown in the following program.

```

100  PRINTER IS 1
110  Off=128
120  Underline=4
130  Blinking=2
140  Inverse=1
150  !
160  PRINT CHR$(255)&CHR$(Off);"Normal"
170  PRINT
180  PRINT CHR$(255)&CHR$(128+Inverse);"Inverse"
190  PRINT "carries over onto"
200  PRINT "subsequent lines"
210  PRINT
220  PRINT CHR$(255)&CHR$(128+Underline);"Underline"
230  PRINT "also remains on until turned off"
240  PRINT
250  PRINT CHR$(255)&CHR$(128+Blinking);"Blinking"
260  PRINT "is the same"
270  PRINT
280  PRINT CHR$(255)&CHR$(Off);"Back to normal"
290  PRINT
300  END

```

## Color Enhancements

Color displays recognize eight additional control codes for selecting the first eight alpha pen colors. Both the Output Area and the Display Line have this ability.

### Color Display Enhancements

Character		Resulting Enhancement	
<i>One-byte</i>	<i>Two-byte</i>	<i>Model 236C</i>	<i>Bit-mapped</i>
CHR\$(136)	CHR\$(255)&CHR\$(136)	White	Pen 1
CHR\$(137)	CHR\$(255)&CHR\$(137)	Red	Pen 2
CHR\$(138)	CHR\$(255)&CHR\$(138)	Yellow	Pen 3
CHR\$(139)	CHR\$(255)&CHR\$(139)	Green	Pen 4
CHR\$(140)	CHR\$(255)&CHR\$(140)	Cyan	Pen 5
CHR\$(141)	CHR\$(255)&CHR\$(141)	Blue	Pen 6
CHR\$(142)	CHR\$(255)&CHR\$(142)	Magenta	Pen 7
CHR\$(143)	CHR\$(255)&CHR\$(143)	Black	Pen 8

When using BASIC/UX from the X Window environment, the colors will correspond to the first eight colors of the X Window color map. For information about programming for gray scale displays, refer to the *HP BASIC 6.2 Advanced Programming Techniques* manual.

These same features can also be placed in strings by using the **f7** key (Any Char) while typing in string literals. Keep in mind that, even though these characters are not shown on the screen, they are counted in the length of the string. Dimension string variables accordingly.

## Display Enhancement Guidelines

For maximum portability between localized and non-localized BASIC, always use the two-byte form of display enhancement characters. Two-byte display enhancements are compatible with both one- and two-byte languages. One-byte display enhancement characters are *not* compatible with many two-byte languages.

## The Display Functions Mode

The preceding program showed the control characters which are defined to invoke a special display function when sent to some CRTs or windows. To display *all* control characters sent to the CRT/window, rather than have the CRT/window interpret them as commands, turn the Display Functions mode *on* by pressing the **Display Fctns** key (**F6** in the System menu). Repeatedly pressing this key toggles this display mode between “on” and “off”. The same thing can be accomplished programmatically using the statement `DISPLAY FUNCTIONS ON/OFF`. Using the CRT/window with `DISPLAY FUNCTIONS ON` is very useful when you need to see exactly which control characters have been output. An asterisk is visible in the Display Functions softkey label when the Display Functions mode is on.

Except for the carriage-return character, all subsequent control characters sent to the display (while in this mode) do not invoke their defined function, but are *only displayed*. The carriage-return is *both* displayed *and* causes the print position to move to the beginning of the next line (both CR and LF functions invoked).

The `DISPLAY FUNCTIONS` mode can also be enabled from BASIC programs with the use of the `CONTROL` statement. The following program shows how this is accomplished. Notice that the carriage-return invokes both carriage-return and line-feed functions.

```

100  DISPLAY FUNCTIONS ON  ! CONTROL CRT,4;1
110  !
120  ! First send with default CR/LF sequence.
130  OUTPUT 1;"DISPLAY FUNCTIONS ON"
140  !
150  ! Then suppress the CR/LF (with ";"").
160  OUTPUT 1;CHR$(12);
170  END

```

Notice that the `DISPLAY FUNCTIONS ON` message normally displayed when the key is pressed is not automatically displayed when the mode is changed by one of these statements; instead, the program must display the message, if so desired.

The following program uses the CRT mapping register (lines 150 and 270) and the `DISPLAY FUNCTIONS` statements (lines 160, 210, 230, and 260) to display the logical and physical CRT character sets.

```

100  ! Output available character sets to CRT
110  PRINT "If you want to see the ROM contents, THEN ENTER any character"
120  PRINT "ELSE, to see the legitimate character set, ENTER a null string."
130  INPUT A$
140  PRINT CHR$(12)      ! Issue a form feed.
150  CONTROL 1,11;LEN(A$) ! Identify which character set.
160  DISPLAY FUNCTIONS ON ! Set display functions mode.
170  ! Provide loop to display character set.
180  FOR I=0 TO 255
190    PRINT USING "#,DDD,X,A,X";I,CHR$(I)
200    IF NOT (I MOD 13) THEN
210      DISPLAY FUNCTIONS OFF
220      PRINT
230      DISPLAY FUNCTIONS ON
240    END IF
250  NEXT I
260  DISPLAY FUNCTIONS OFF
270  CONTROL 1,11;0
280  PRINT
290  END

```

The logical and physical CRT character sets are the same on all bit-mapped displays, including the HP 98204B Video Output Interface and the HP 98546A Display Compatibility Interface, when the "character-set select" switch is set to the ASCII/ROMAN 8 position.

Access `DISPLAY FUNCTIONS` functionality from windows other than the root window with:

```
CONTROL 699,4;1
```

where 699 is the device selector for window 699.



---

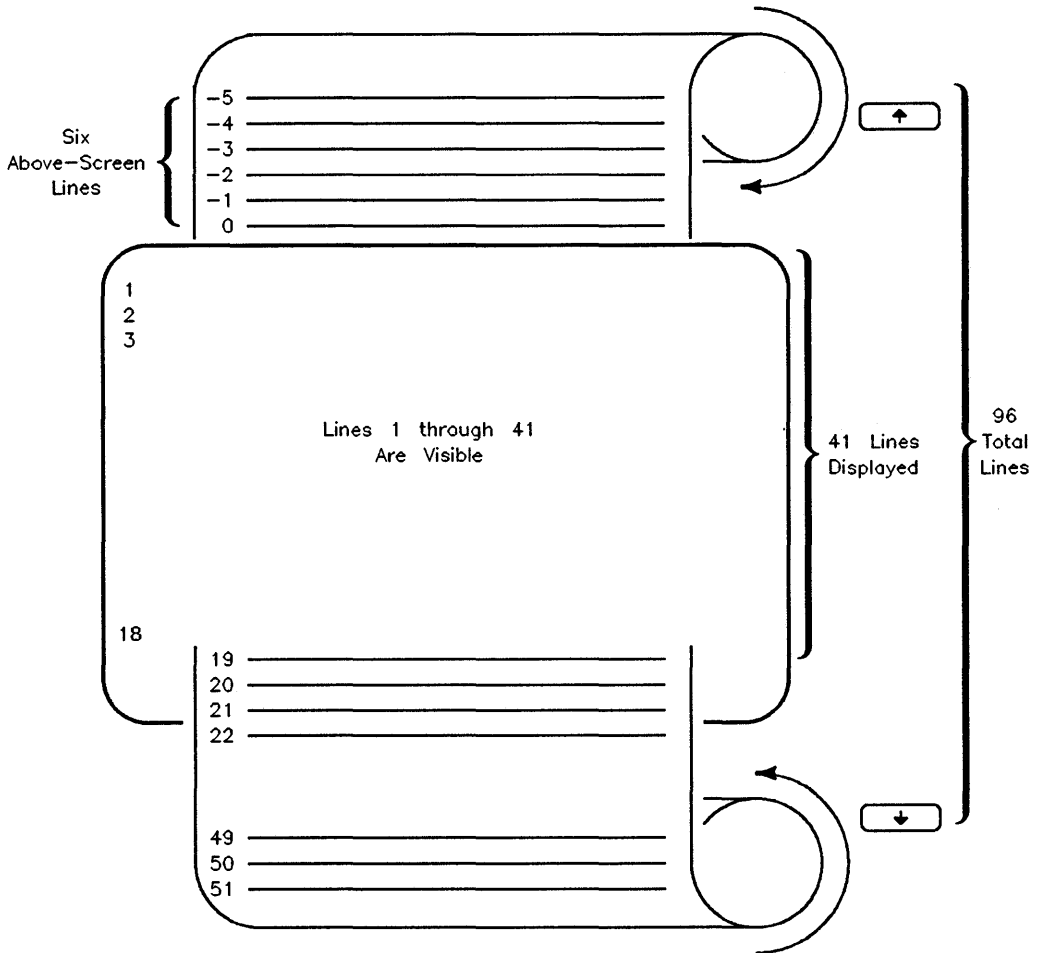
## Output-Area Memory

In addition to the visible display lines in the output area, there may be additional lines available within output-area memory. These additional lines of display memory can be viewed by running the following program and then scrolling the display down. The *visible* lines of output area memory will hereafter be called the **screen**.

```
100  ! Example to show scrolling.
110  !
120  PRINTER IS 1                ! PRINT on CRT.
130  !
140  FOR Line=1 TO 48           ! Write 48 lines.
150    PRINT Line
160  NEXT Line
170  ! Now scroll manually.
180  END
```

### Determining Above-Screen Lines

Scrolling the display up and down allows you to view different portions of the lines within output-area memory. If the display is scrolled *down* as far as possible, there are no lines “above screen”. Similarly, if the display is scrolled *up* as far as possible, there are no lines “below screen”. The following drawing illustrates the screen of an HP 98782A color monitor with six lines above the screen.



### Line Positions of the Output Area

The values could vary if you have a different CRT/window.

The number of lines that are above screen can be determined from BASIC programs by reading STATUS register 3 of interface select code 1. The returned value is the number of lines currently *above screen*.

If the screen has just been cleared (**Clear display**), the following program displays:

```
0 lines above screen.
```

Running the program a second time displays:

```
41 lines above screen
```

Subsequently running the program produces similar results, until the following message is displayed continually (on a 48-line CRT):

```
96 lines above screen
```

```
100  FOR Line=1 TO 41
110    OUTPUT 1;Line
120  NEXT Line
130  !
140  STATUS 1,3;Lines_above
150  DISP Lines_above;"  lines above screen"
160  END
```

The final value will be different for different displays.

## The PRINT Position

All of the characters in output-area memory can be addressed individually by the character's screen column and line. The character in the upper left corner of the screen is in column 1 and line 1, and the character in the lower right corner varies depending on what monitor you have. The addresses of the characters "off screen" are limited by the number of lines currently above screen.

The screen addresses (both column and line) at which a subsequent character sent to the display will appear on the screen are known as the **print position**. The current print position is automatically changed as characters are output to the display. For instance, the print-position column is incremented each time a character is sent; when the last character is sent to a line, the print-position column is reset to 1 and the print-position line is incremented, sending the next character to the next line. The following program shows how the print-position line varies during output to the CRT/window.

```

100  FOR Line=1 TO 48
110    OUTPUT 1;Line
120    STATUS 1,1;Print_line
130    DISP "Print-position line = ";Print_line
140    IF Line<25 THEN WAIT .2
150  NEXT Line
160  !
170  END

```

Notice that *the print-position line is always relative to the first line of the current screen*. This accounts for the print position (read with STATUS) remaining at a value of 19 while the 19th through 48th lines are being printed. When the print position is *off screen*, the display is scrolled (when it receives a character) so that the character appears on the screen. When the display is finished scrolling, all line addresses are again relative to the *new* top screen line. The next section describes using this feature to scroll the display from the program.

## Scrolling the Display

A program can scroll the display up and down by changing the print position to a location *off screen* and then outputting character(s) to the CRT/window. In order to *scroll up*, values greater than the number of lines in the output area must be written to register 1. Assuming an alpha height of 25, this number is 18. If the screen is to be scrolled up 4 lines, the following statements can be used. In this case, the OUTPUT statement outputs the “Null” control character so that no characters will be overwritten.

```

100  CONTROL 1,1;18+4  ! Move print position off screen;
110  OUTPUT 1;CHR$(0); ! scrolling takes place when next
120                                ! character sent to the CRT.

```

The screen is not scrolled up until the OUTPUT statement actually writes to the CRT/window at the current print position (even though, in this case, no visible character is actually output to the display).

In order to *scroll down*, a non-positive number must be written into register 1. For instance, to scroll down one line, a 0 would be written into register 1. Again, the display is not actually scrolled until an OUTPUT (or PRINT) to the CRT is executed.

The only *restriction* on the value of the line number is that it must not attempt to scroll the screen down *past* the first line of output-area memory. In other words, to scroll down as far as possible, the following value would be used; using smaller values results in an error.

Top line's address = - (number of lines above screen) + 1

Thus, if no lines are above screen, the top line's address is 1.

An example of scrolling down "as far as possible" is shown in the following program.

```

100  FOR Line=1 TO 48
110    OUTPUT 1;Line
120  NEXT Line
130  !
140  STATUS 1,1;Line_pos
150  DISP "Print-position line =";Line_pos;" after OUTPUT,"
160  WAIT 2
170  !
180  STATUS 1,3;Lines_above  ! Find # lines above screen.
190  DISP "and";Lines_above;" lines are above screen"
200  WAIT 3
210  !
220  CONTROL 1,1;-Lines_above+1  ! Change line-pos.
230  OUTPUT 1;"Line 1"          ! Scroll made when 1st.
240                                ! character is sent.
250  !
260  STATUS 1,3;Lines_above
270  DISP "Now, number of lines above screen =";Lines_above
280  END

```

## Entering from the CRT

Data is entered from the CRT or a window beginning at the current print position. As characters are read from the screen (from left to right), the print position is updated. When the ENTER statement attempts to read past the last non-blank character on a line, the CRT/window driver sends a line-feed character accompanied by a (simulated) EOI signal, and the print position is advanced to the beginning of the next line.

Display-enhancement characters, CHR\$(128) through CHR\$(143), cannot be entered from the CRT/window memory. When these characters are shown on the screen (because they were displayed while DISPLAY FUNCTIONS was on), they can be read with an ENTER statement. However, if they are instead "executed" (because DISPLAY FUNCTIONS is off), they are not read with ENTER.

## Reading a Screen Line

The following program uses the line-feed accompanied by EOI to terminate entry into a string variable. Since the free-field ENTER statement is used, only one line can be read because of the EOI sent with the line-feed character.

```

100 CONTROL 1;5,8           ! Move print position to
110                        ! 5th column of line 8,
120 OUTPUT 1;"ABCDEFGH"    ! then OUTPUT (with CR/LF).
130 !
140 OUTPUT 1;"IJKLMNOP"    " ! OUTPUT to line 9 with
150                        ! trailing spaces.
160                        !
170 CONTROL 1,1;8         ! Move print position back
180                        ! to 1st column of line 8.
190                        !
200 ENTER 1;Line_8$
210 DISP LEN(Line_8$);"characters read from line 8"
220 WAIT 2
230 !
240 ENTER 1;Line_9$
250 DISP LEN(Line_9$);"characters read from line 9"
260 END

```

This feature of the CRT/window is very useful when simulating entry from the HP-IB interface; however, keep in mind that *no spaces can be read after the last visible character at the end of each line*. Notice in the preceding example that the leading space characters in a string which are sent to the display were read by ENTER and trailing space characters sent to the display were *not* read back by the ENTER statement. These trailing characters are treated as “blanks” by the CRT, which sends the line-feed with EOI when the ENTER statement attempts to read the first one.

## Reading the Entire Output-Area Memory

In order to read all lines within output-area memory, an ENTER statement that uses an image must be used to prevent the EOI signal from terminating the statement prematurely (since the EOI signal acts as an *item* terminator during ENTER-USING-image statements which contain no “%” image specifiers). The following program shows the entire contents of output-area memory being read.

```

100  OPTION BASE 1
110  DIM Memory$(48)[50]           ! To read 48 lines.
120  !
130  FOR Screen_line=1 TO 48
140    OUTPUT 1;"Line";Screen_line
150  NEXT Screen_line
160  WAIT 1
170  !
180  STATUS 1,3;Lines_above
190  CONTROL 1,1;-Lines_above+1   ! Scroll to read
200  ENTER 1 USING "K";Memory$(*) ! entire memory.
210  !
220  FOR Screen_line=1 TO 48      ! Display all lines;
230    PRINT Memory$(Screen_line);" "; ! no CR/LF.
240  NEXT Screen_line
250  END

```

## Final Display

```
Line 36
Line 37
Line 38
Line 39
Line 40
Line 41
Line 42
Line 43
Line 44
Line 45
Line 46
Line 47
Line 48
Line 1 Line 2 Line 3 Line 4 Line 5 Line 6 Line 7 L
ine 8 Line 9 Line 10 Line 11 Line 12 Line 13 Line
14 Line 15 Line 16 Line 17 Line 18 Line 19 Line 20
Line 21 Line 22 Line 23 Line 24 Line 25 Line 26 L
ine 27 Line 28 Line 29 Line 30 Line 31 Line 32 Lin
e 33 Line 34 Line 35 Line 36 Line 37 Line 38 Line
39 Line 40 Line 41 Line 42 Line 43 Line 44 Line 45
Line 46 Line 47 Line 48
```

Notice that the print position was moved to the top line before attempting to read memory contents, since the ENTER statement reads characters beginning at the print position. If the print position is not at the “top line” of memory before attempting to read all 57 lines, the lines above screen will not be read. However, the statement executes with no errors, because the CRT sends line-feeds (with EOI) for each line that does not really exist “below screen”. For instance, if the print position is at line 10 when the ENTER begins, only the last 47 lines of output-area memory will be read (and placed into the first 47 elements of Memory\$). When the ENTER statement attempts to fill the last ten elements of Memory\$, the CRT sends only line-feeds accompanied by EOI because the print position is past the last non-blank character.



---

## Additional CRT Features

This section describes the remainder of features of the CRT/window display controllable by BASIC programs. *Interrupt and timeout events are not available with the CRT interface.*

### The DISP Line

BASIC programs can output characters to the DISP Line with the DISP statement, as described in the *HP BASIC 6.2 Language Reference*. As with the output-area's print position, the position (column) within the DISP line at which subsequent characters will appear can be read and changed explicitly by BASIC programs. This *DISP-line position* can be read and changed with STATUS register 8 and CONTROL register 8 (of interface select code 1), respectively. Note that the CONTROL register 8 statement can be replaced with the DISP TAB statement as shown in the following program. However, the DISP TAB statement and the CONTROL register 8 statement are slightly different. For example, if there were characters in the first 45 columns of the DISP Line, those characters would be blanked when the DISP TAB statement is executed. The CONTROL register 8 statement allows the characters to remain, and as the FOR loop decrements its count the characters in columns 1 through 45 would not be blanked.

```

100  FOR Disp_pos=46 TO 1 STEP -1
110    DISP TAB(Disp_pos),"HELLO"  ! or 110  CONTROL 1,8;Disp_pos
120                                     !    120  DISP "HELLO"
130                                     !
140    WAIT .2
150  NEXT Disp_pos
160  END

```

Keep in mind that if trailing carriage-return and line-feed characters are output to the DISP line, the carriage-return returns the DISP-line position to column 1. A subsequent DISP statement clears the entire line. However, if these trailing characters are suppressed, the DISP-line position is left unchanged. Run the following program to see these effects.

```

100 PRINT "First with trailing CR/LF,"
110 DISP "HI"
120 WAIT .5
130 DISP " THERE"
140 WAIT 1
150 !
160 PRINT "then with no CR/LF."
170 DISP "HI";
180 WAIT .5
190 DISP " THERE"
200 END

```

Also notice that if a DISP attempts to send characters to the DISP line so that any character will be past the last column (50, 80, or 128 depending on your CRT/window), the entire line is shifted left so that all of the new characters will be displayed (i.e., so that the last character written will end up in the last column).

```

100 A$=SYSTEM$("CRT ID")
110 X=VAL(A$[3])-10 ! Screen width minus 10.
120 DISP TAB(X),"CHARACTERS"; ! No CR/LF.
130 WAIT 1
140 DISP " SHIFTED LEFT"
150 !
160 END

```

The display-enhancement characters produce the same effects in the DISP Line as in the OUTPUT Area.

### Changing Pen Colors

The pen color of the OUTPUT Area and DISP Line of the CRT/window can be changed using either the ALPHA PEN or PRINT PEN statement followed by a *pen value*. Information for both of these commands can be found in the preceding section of this chapter entitled "Changing Pen Colors in Display Regions."

## Disabling the Cursor Character

BASIC programs even have control over whether any cursor is displayed (during all computer modes, such as during EDITs and other keyboard-entry modes). The cursor is *disabled* with the following statement.

```
CONTROL 1,10;0
```

Any *non-zero* value written to this register *re-enables* the cursor to be displayed. Resetting the computer also re-enables the cursor being displayed.

```
CONTROL 1,10;1
```

## Enabling the Insert Mode

The insert mode of the keyboard area can be enabled and disabled with STATUS and CONTROL statements. If any *non-zero* numeric value is written to register 2, the insert mode is *enabled*. All subsequent characters typed into this area are “inserted” between the cursor and the character to its immediate left, and characters to its right are shifted appropriately.

The following program turns insert mode on for approximately five seconds. During this time, use the arrow keys to move the cursor left and right while typing in characters from the keyboard.

```
100  Insert_mode=1
110  CONTROL 1,2;Insert_mode
120  !
130  DISP "Insert mode is now being used."
140  BEEP 200,.2
150  WAIT 5
160  !
170  Insert_mode=0
180  CONTROL 1,2;Insert_mode
190  DISP "Now the mode has changed to overwrite."
200  BEEP 100,.2
210  WAIT 5
220  !
230  BEEP 50,.2
240  DISP "Program ended."
250  END
```

## Softkey Labels

Softkeys can be defined as typing-aid keys or as keys that initiate program (ON KEY) branches. In any usage, two display lines (near the bottom of the CRT/window) can be used for key labels. The topic of typing-aid keys is discussed in the *Using HP BASIC* manual for your system. The topic of using softkeys to initiate program branches is discussed in the *HP BASIC 6.2 Programming Guide*.

Softkey labels can be turned off and on by writing to CRT or window CONTROL Register 12 or using the KEY LABELS ON/OFF statement. The values written to the register have the following effects:

### Turning Softkey Labels Off/On

Value of CRT Register 12	Effect on Key Labels
0	Typing-aid key labels are <i>displayed until the program is run</i> , at which time they are turned off (until at least one ON KEY is executed). Annunciators, if present, stay on. System menu softkeys are displayed even when a program is running. (This is the default for systems with an HP 98203A/B/C keyboard.)
1	Typing-aid and softkey labels are <i>not displayed at any time</i> .
2	Typing-aid and softkey labels are <i>displayed at all times</i> . (This is the default for systems with an ITF keyboard.)

The default value of this register is 2 for BASIC/UX, since it uses an ITF keyboard. The default is restored at power-on and when SCRATCH A is executed. The register's current contents can be determined by reading STATUS Register 12.

Here is an illustrative program which cycles through Register 12 using the values 0, 1, and 2. Note that loading the binary CRTX allows you to use the statements KEY LABELS ON and KEY LABELS OFF in place of the CONTROL register 12 statements.

```

100  ! Toggle key displays
110  PRINT "Softkey labels are toggled."
120  WAIT 2
130  ! Set up toggle loop.
140  FOR J=1 TO 3
150    FOR Toggle=0 TO 2 ! 1= KEY LABELS ON, and 2= KEY LABELS OFF
160      CLEAR SCREEN
170      PRINT "Flag value =";Toggle
180      CONTROL 1,12;Toggle ! KEY LABELS ON and OFF.
190      WAIT 1.5
200    NEXT Toggle
210  NEXT J
220  !
230  STOP
240  END

```

Try running the program with one of the three User menus and then with the System menu.

You can use CONTROL register 2 (of select code 2) to cycle the menus: 0=System, 1=User 1, 2=User 2, and 3=User 3. For example:

```
CONTROL 2,2;3
```

OR

```
USER 3 KEYS
```

displays the menu for User 3 softkeys. Another method of bringing the System menu and User menus up is to use the:

- SYSTEM KEYS statement for the System menu.
- USER *menu number* KEYS statement with the appropriate user *menu number* for the three User menus.

CONTROL register 14 (of select code 2) can be used to set softkey bases 0 or 1 (i.e., **f1** is KEY 0, **f2** is KEY 1, etc.) The default is 0, which means the softkeys start at 1. Changing the flag value to 1 starts the softkeys at 0. You might need to deal with this because the softkeys on HP 98203 Keyboards are labeled from 0 to 9, and from 1 to 8 on ITF Keyboards.

Note that you can draw a solid line between the two lines of key labels (on machines with ITF keyboards). To cause a line to be drawn, put a CHR\$(132) as the first character of the label (third character if the first two are inverse-video **K** (character code 255) and “#”, which represents a Clear line key). An example of this technique is the System key labeled **Clr Tab** (above the line) and **Set Tab** (below the line). For examples, see the *Using BASIC* manual for your system.

## Softkey Label Colors

Softkey pen color changes are made using the KEY LABELS PEN statement followed by the *pen value*. For a detailed description of *pen values* and the use of this statement, read the preceding section in this chapter entitled “Changing Pen Colors in Display Regions.”

---

## Summary of CRT STATUS and CONTROL Registers

<i>STATUS Register 0</i>	Current print position (column)
<i>CONTROL Register 0</i>	Set print position (column). See also TAB and TABXY.
<i>STATUS Register 1</i>	Current print position (line)
<i>CONTROL Register 1</i>	Set print position (line). See also TABXY.
<i>STATUS Register 2</i>	Insert-character mode
<i>CONTROL Register 2</i>	Set insert character mode if non-0  (Error 713 is given if a window number is specified instead of a select code.)
<i>STATUS Register 3</i>	Number of lines “above screen”.
<i>CONTROL Register 3</i>	Undefined
<i>STATUS Register 4</i>	Display functions mode

*CONTROL Register 4*

Set display functions mode if non-0. To perform the same function, use the statement DISPLAY FUNCTIONS ON/OFF.

*STATUS Register 5*

Returns the CRT alpha color value set (or default). This does not reflect changes due to printing CHR\$(*x*), where  $136 \leq x \leq 143$ .

*CONTROL Register 5*      Set default alpha color.

For Alpha Displays:

Value	Result
< 16	The number is evaluated MOD 8 and resulting values produce the following:  0—black 1—white 2—red 3—yellow 4—green 5—cyan 6—blue 7—magenta
16 to 135	Ignored
136	White
137	Red
138	Yellow
139	Green
140	Cyan
141	Blue
142	Magenta
143	Black
144 to 255	Ignored

For Bit-Mapped Displays:

Values 0 thru 255 which correspond to the graphics pens. The values are treated as MOD  $2^n$  where  $n$  is the number of display planes.

CONTROL CRT,5; $n$  sets the values of the CRT registers 15, 16, and 17, but the converse is not true. That is, STATUS CRT,5 may not accurately reflect the CRT state if CONTROL 15, 16, and/or



17 have been executed. Note that to perform the same function as `CONTROL CRT,5;n`, you can use the `ALPHA PEN` statement.

*STATUS Register 6*

ALPHA ON flag

(Error 713 is given if a window number is specified instead of a select code.)

*CONTROL Register 6*

Undefined

(Error 713 is given if a window number is specified instead of a select code.)

*STATUS Register 7*

GRAPHICS ON flag

(Error 713 is given if a window number is specified instead of a select code.)

*CONTROL Register 7*

Undefined

(Error 713 is given if a window number is specified instead of a select code.)

*STATUS Register 8*

Display line position (column)

(Error 713 is given if a window number is specified instead of a select code.)

*CONTROL Register 8*

Set display line position (column). See also `TAB`.

(Error 713 is given if a window number is specified instead of a select code.)

*STATUS Register 9*

Screenwidth (number of characters). Also available in the `SYSTEM$("CRT ID")` function result.

*CONTROL Register 9*

Undefined

*STATUS Register 10*

Cursor-enable flag

(Error 713 is given if a window number is specified instead of a select code.)

<i>CONTROL Register 10</i>	Cursor-enable: 0=invisible cursor. non-0=cursor visible.  (Error 713 is given if a window number is specified instead of a select code.)
<i>STATUS Register 11</i>	CRT character mapping flag
<i>CONTROL Register 11</i>	Disable CRT character mapping (if non-0)
<i>STATUS Register 12</i>	Key labels display mode.  (Error 713 is given if a window number is specified instead of a select code.)
<i>CONTROL Register 12</i>	Set key labels display mode: 0 = typing-aid key labels displayed unless program is running. 1 = key labels always off (or use KEY LABELS OFF). 2 = key labels displayed at all times (or use KEY LABELS ON).  (Error 713 is given if a window number is specified instead of a select code.)
<i>STATUS Register 13</i>	CRT height (number of lines to be used for alpha display).
<i>CONTROL Register 13</i>	Set CRT height (must be $\geq 9$ ). Alternately use the ALPHA HEIGHT statement.
<i>STATUS Register 14</i>	Display replacement rule currently in effect (BASIC/WS only).

*CONTROL Register 14*      Set display replacement rule (BASIC/WS only)  
(with bit-mapped alpha displays only)

**Note**



This register is not processed for the 9863C display, nor for the Model 362/382 internal display because they do not have replacement rule support hardware. Any updates made to this register are ignored for those displays.

- 0—0
- 1—source AND old
- 2—source AND NOT old
- 3—source;default
- 4—NOT source AND old
- 5—old
- 6—source EXOR old
- 7—source OR old
- 8—source NOR old
- 9—source EXNOR old
- 10—NOT old 11—source OR NOT old
- 12—NOT source
- 13—NOT source OR old
- 14—source NAND old
- 15—1

*It is strongly recommended that you do not change the default display replacement rule.*

*STATUS Register 15*      Return the value set (or the default) for the color in the PRINT/DISP area. This does not reflect changes due to printing CHR\$(*x*), where  $136 \leq x \leq 143$ .

*CONTROL Register 15*      Set PRINT/DISP color (or use the PRINT PEN statement). Similar to CRT control register 5 but specific to CRT PRINT/DISP areas; that is, it does not affect the areas covered by CRT registers 16 and 17.

- STATUS Register 16* Return the value set (or the default) for the softkey label color.  
(Error 713 is given if a window number is specified instead of a select code.)
- CONTROL Register 16* Set key labels color (or use the KEY LABELS PEN statement). Similar to CRT control register 5 but only affects the softkey labels. Does not affect the areas covered by CRT registers 15 and 17.  
(Error 713 is given if a window number is specified instead of a select code.)
- STATUS Register 17* Return the value set (or the default) for the color of the “non-enhance” area. This includes the keyboard entry line, runlight, system message line, annunciators, and edit screen.
- CONTROL Register 17* Set “non-enhance” color (or use the KBD LINE PEN statement). This includes the keyboard entry line, runlight, system message line, annunciators, and edit screen. Similar to CRT control register 5 but does not affect the areas covered by CRT control registers 15 and 16.
- STATUS Register 18* Read the alpha write-enable mask.
- CONTROL Register 18* Set alpha write-enable mask to a bit pattern (or use the SET ALPHA MASK statement). When running BASIC/UX in the X Window environment, this CONTROL register is *not* supported.
- STATUS Register 19* Returns the maximum value for ALPHA MASK argument.
- CONTROL Register 19* Undefined.
- STATUS Register 20* Read the alpha display-enable mask.  
(Error 713 is given if a window number is specified instead of a select code.)

*CONTROL Register 20* Set alpha display-enable mask to a bit pattern (or use the SET DISPLAY MASK statement).

(Error 713 is given if a window number is specified instead of a select code.)

*STATUS Register 21* Active CRT binary identity. See CONTROL register 21 for a table of CRT binary identification codes.

*CONTROL Register 21* Specify which loaded CRT binary BASIC will attempt to activate. Each CRT binary is represented by one of the following values:

Value	Binary
0	default search
1	CRTA
2	CRTB
3	reserved
4	CRTD (single width)
5	CRTD (double width)

If 0 is sent to CONTROL register 21, BASIC searches all the loaded binaries in a default order and activates the first one found that is compatible with the installed hardware. The default search order is CRTD, then CRTB, then CRTA.

Sending a new value to CONTROL register 21 effectively initializes the alpha display and executes GINIT and PLOTTER IS CRT, "INTERNAL". BASIC/UX does not support switching between non-bit-mapped and bit-mapped displays, but the initialization is still done.

*STATUS Register 22* Undefined.

*CONTROL Register 22* Raises a window to the top of the window stack if non-zero; pushes a window to the bottom of the stack if zero.

<i>STATUS Register 23</i>	Returns terminal compatibility mode.
<i>CONTROL Register 23</i>	Sets terminal compatibility mode.

# The Keyboard Interface

---

As with displays, access to the keyboard can be made with OUTPUT, ENTER, CONTROL, and STATUS statements. This chapter describes I/O programming techniques for “interfacing” to the keyboard.

---

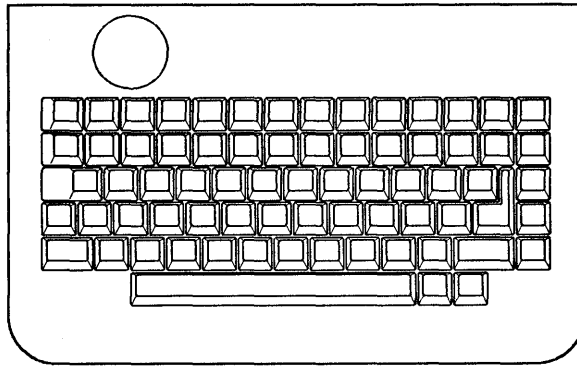
## Description of Keyboards

This section introduces you to the different types of keyboards available with Series 200/300 computers, and provides an overview of their capabilities. Here are the topics covered:

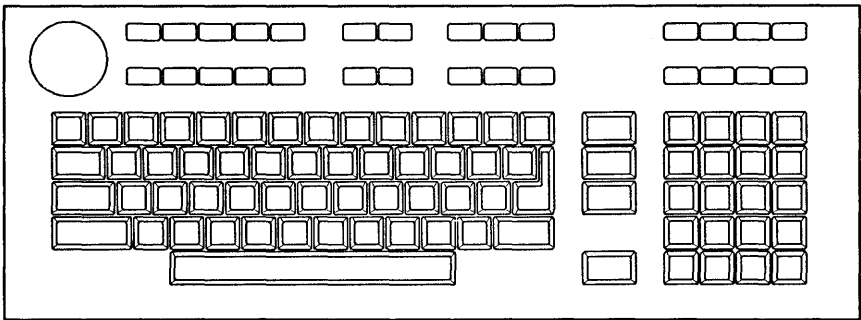
- Types of keyboards.
- How the “primary” keyboard is chosen (in machines with more than one keyboard installed).
- Overview of keyboard features.

## Types of Keyboards

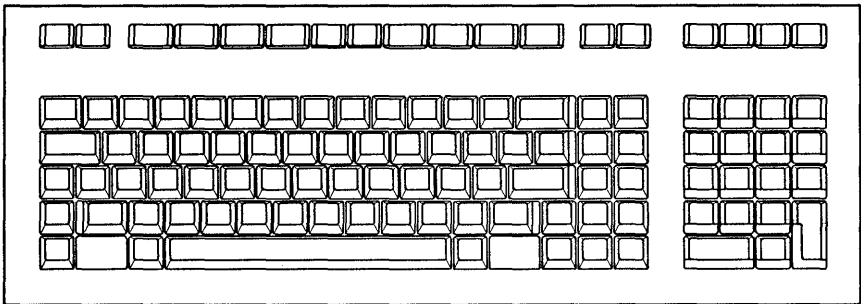
There are essentially three types of keyboards available with Series 200/300 computers:



**HP 98203A Keyboard (BASIC/WS Only)**



**HP 98203B and 98203C Keyboards (BASIC/WS Only)**



**ITF Keyboards (HP 46020 and HP 46021)**



Series 200 Model 216 computers may have the smaller HP 98203A keyboard or optionally the larger 98203B keyboard. Model 220 computers have options which allow either the HP 98203A or HP 98203B Keyboards, or the HP 98203C or ITF Keyboards. Models 226 and 236 computers have built-in 98203B keyboards. Models 217 and 237 and all Series 300 computers have ITF keyboards as the standard, but you can also order the 98203C keyboard as an option.

Complete descriptions of the BASIC definitions of each key of every keyboard is provided in the *Using HP BASIC* manual for your system.

## How the Primary Keyboard Is Chosen for BASIC/WS

Select code 2 is always assigned to the keyboard interface. However, Series 200/300 computers can have more than one keyboard installed at one time. In such cases, the BASIC system has to choose which one will be the primary keyboard. Here is the order that the system chooses this keyboard:

1. If there is an “internal” keyboard, then it is chosen as the primary keyboard. Examples are as follows:
  - a. The 98203 keyboard on a Series 200 computer.
  - b. The HIL keyboard on a Series 200 or 300 computer.
2. If there is an “external” HIL keyboard, and no “internal” keyboard interface, then it will be chosen as the primary keyboard. An example is:
  - a. A keyboard connected to the HIL port of an HP 98700 Display Controller.

Only one primary keyboard and one HIL interface will be recognized by the BASIC system.

Note that the primary keyboard determines the keyboard language and which softkey labels are chosen. Thus, if two keyboards with different languages are connected to the computer (and recognized by BASIC), then the language and softkey labels of the primary keyboard are used. This effect may cause some keys on the secondary keyboard to produce incorrect characters.

## 2 HP 98203 Keyboard Compatibility Mode

There is also a mode of operation, enabled and disabled via keyboard CONTROL register 15 (or the KBD CMODE statement), in which ITF Keyboards can emulate an HP 98203B keyboard; see the “Keyboard Status and Control Register Summary” section at the end of this chapter for values and effects. Details of using this mode are provided in the *HP BASIC 6.2 Programming Guide*.

### Re-Configuring HIL Devices

If you add or remove HIL devices while the BASIC system is in the computer, you must re-configure in order for BASIC to properly recognize all devices. Executing SCRATCH A initiates this re-configuration.

### Overview of Keyboard Features

Series 200/300 computer keyboards are controlled by their own separate processors, which allows many more capabilities than most other desktop-computer keyboards. These keyboards are devices which reside at select code 2. (BASIC provides the KBD function which returns a value of 2.) Here is a brief list of keyboard capabilities:

- You can use the ENTER statement to enter data from the keyboard, and thus simulate devices for debugging purposes.
- You can monitor keys and the “knob” (rotary pulse generator), if present, and enable them to interrupt BASIC programs; the BASIC program can contain a segment of code to read and use this input.
- You can OUTPUT commands to the keyboard, simulating an operator entering them. You can also OUTPUT data to the keyboard which the operator can then edit and send back.

Note, however, that the INTR and TIMEOUT event-initiated branches *cannot* be sensed by the keyboard.

## ASCII and Non-ASCII Keys

The keys of a Series 200/300 computer keyboard can be generally grouped by function into the ASCII and non-ASCII keys.

ASCII (or alphanumeric) keys	all produce an ASCII character when pressed, and include the character entry and numeric keys.
non-ASCII (or non-alphanumeric) keys	do not produce characters but initiate specific actions when pressed; the <b>Return</b> and <b>Back space</b> keys are considered to be non-ASCII keys for this reason. Non-ASCII keys also include all program control, editing, cursor control, and system control keys.

## The Shift and Control Keys

The **Shift**, **CTRL**, and **Extend char** keys are not really either type of key because they cannot cause action on their own; instead, they are used only with the other types of keys. Pressing the **Shift** key with another key *qualifies* the other keypress, allowing the other key to have a second meaning. For instance, in the “Caps lock off” mode, pressing an alphabetic ASCII key generates a lowercase alphabetic character. Pressing the **Shift** key *simultaneously* with an alphabetic key in the “Caps lock off” mode generates an uppercase character. The **Shift** key is used similarly with the non-ASCII keys, allowing many of those keys to have a second function.

The **Extend char** key is held down while you press other keys from the main typewriter section to generate the rest of the available 256 ASCII characters. It also has a special use with the softkeys when in keyboard compatibility mode; see the *HP BASIC 6.2 Programming Guide*.

The **CTRL** (Control) key is also used to further qualify *both* ASCII and non-ASCII keypresses. Pressing the **CTRL** key simultaneously with an ASCII key generates an ASCII control character in the display, and is often faster than using the **Any char (f7)** softkey. The following table shows how to generate control characters by simultaneously pressing the **CTRL** key and a

key as listed. This is particularly useful when you need to include a control character in a string.

### Generating Control Characters with CTRL and ASCII Keys

Character Code	ASCII Character	Character Description	Key(s) Pressed with CTRL	Character on CRT
0	NUL	Null	(space bar)	N <sub>u</sub>
1	SOH	Start of Header	(A)	S <sub>h</sub>
2	STX	Start of Text	(B)	S <sub>x</sub>
3	ETX	End of Text	(C)	E <sub>x</sub>
4	EOT	End of Transmission	(D)	E <sub>t</sub>
5	ENQ	Enquiry	(E)	E <sub>q</sub>
6	ACK	Acknowledgement	(F)	A <sub>k</sub>
7	BEL	Bell	(G)	(bell)
8	BS	Backspace	(H)	B <sub>s</sub>
9	HT	Horizontal Tab	(I)	H <sub>t</sub>
10	LF	Line-feed	(J)	L <sub>f</sub>
11	VT	Vertical Tab	(K)	V <sub>t</sub>
12	FF	Form-feed	(L)	F <sub>f</sub>
13	CR	Carriage-return	(M)	C <sub>r</sub>
14	SO	Shift Out	(N)	S <sub>o</sub>
15	SI	Shift In	(O)	S <sub>i</sub>

**Generating Control Characters with CTRL and ASCII Keys  
(continued)**

Character Code	ASCII Character	Character Description	Key(s) Pressed with CTRL	Character on CRT
16	DLE	Data Link Escape	<b>P</b>	D <sub>1</sub>
17	DC1	Device Control	<b>Q</b>	D <sub>1</sub>
18	DC2	Device Control	<b>R</b>	D <sub>2</sub>
19	DC3	Device Control	<b>S</b>	D <sub>3</sub>
20	DC4	Device Control	<b>T</b>	D <sub>4</sub>
21	NAK	Neg. Acknowledgement	<b>U</b>	N <sub>k</sub>
22	SYN	Synchronous Idle	<b>V</b>	S <sub>y</sub>
23	ETB	End of Text Block	<b>W</b>	E <sub>b</sub>
24	CAN	Cancel	<b>X</b>	C <sub>n</sub>
25	EM	End of Media	<b>Y</b>	E <sub>m</sub>
26	SUB	Substitute	<b>Z</b>	S <sub>b</sub>
27	ESC	Escape	<b>I</b>	E <sub>c</sub>
28	FS	File Separator	<b>Shift-I</b>	F <sub>s</sub>
29	GS	Group Separator	<b>I</b>	G <sub>s</sub>
30	RS	Record Separator	<b>▲</b>	R <sub>s</sub>
31	US	Unit Separator	<b>Shift-/</b>	U <sub>s</sub>

Pressing the **ESC** key is an alternative to **CTRL-I**. The keys listed in the preceding table are *not the only* ways to generate control characters, but are generally the simplest and most easily memorized method. For instance, to generate a line-feed character, press the **CTRL** and the **J** keys simultaneously.

**CTRL-J**

Pressing the **CTRL** key with a non-ASCII key is used to generate and store non-ASCII keystrokes within strings and is further discussed in “Outputs to the Keyboard”.

The display enhancement control codes can be generated by pressing **CTRL**, **Extend char**, and a key from the following table simultaneously.

**Generating Control Characters with CTRL, Extend char, and  
ASCII Keys**

Character Code	Character Description	Key(s) Pressed with CTRL and Extend char	Character on CRT
128	Clear enhancements	Ⓐ	C <sub>L</sub>
129	Inverse video	Ⓑ	I <sub>V</sub>
130	Blinking	Ⓒ	B <sub>G</sub>
131	Inverse blinking	Ⓓ	I <sub>B</sub>
132	Underline	Ⓔ	U <sub>L</sub>
133	Underline and Inverse	Ⓕ	I <sub>V</sub>
134	Underline and Blinking	Ⓖ	B <sub>G</sub>
135	Underline, Inverse, and Blinking	Ⓗ	I <sub>B</sub>
136	White	Ⓚ	W <sub>H</sub>
137	Red	Ⓛ	R <sub>D</sub>
138	Yellow	Ⓜ	Y <sub>E</sub>
139	Green	Ⓝ	G <sub>R</sub>
140	Cyan	Ⓣ	C <sub>Y</sub>
141	Blue	Ⓨ	B <sub>U</sub>
142	Magenta	Ⓤ	M <sub>G</sub>
143	Black	Ⓢ	B <sub>K</sub>

## Keyboard Operating Modes

The keyboard has three operating modes which can be changed within a program with the `CONTROL` statement. This section describes changing these modes from the program.

### The Caps Lock Mode

Pressing the `[Caps]` key toggles the keyboard between the “Caps lock on” and “Caps lock off” modes. In the “Caps lock on” mode, pressing any alphabetic key causes an uppercase letter to be displayed on the screen; in the “Caps lock off” mode, these keys generate lowercase letters. This mode can be changed with the `CONTROL` statement and sensed with the `STATUS` statement.

Writing any *non-zero* numeric value into register 0 (of interface select code 2) sets the caps lock mode *on*; writing a zero into this register sets the mode off.

```

100 STATUS 2;Caps_lock ! Check mode.
110 !
120 PRINT "Initially, ";
130 IF Caps_lock=1 THEN
140     Mode$="ON"
150 ELSE
160     Mode$="OFF"
170 END IF
180 !
190 PRINT "CAPS LOCK was "&Mode$&CHR$(10) ! Skip line.
200 BEEP
210 WAIT 1
220 !
230 CONTROL 2;1
240 PRINT "CAPS LOCK now ON"
250 PRINT "Type in a few characters"&CHR$(10)
260 WAIT 4
270 !
280 CONTROL 2;0
290 PRINT "CAPS LOCK now OFF"
300 PRINT
310 BEEP
320 END

```

## The Print All Mode

Pressing the **Prt all** softkey (**f4** in the System menu) toggles the “Print all” mode “on” and “off”. The “Print all” mode can also be sensed and changed by reading and writing to STATUS register 1 and CONTROL register 1 (of interface select code 2). Writing a *non-zero* numeric value into this register sets the “Print all” mode *on*; writing a value of zero turns this mode “off”. The following statement turns the “Print all” mode off.

```
CONTROL 2,1;0
```

## Disabling Scrolling

If there are results you *do not* want to accidentally scroll off the screen after or while executing a program, keyboard CONTROL register 16 can be used to prevent this from happening. The “scrolling keys” which keyboard register 16 affects are:

- **▲** and **Shift-▲**
- **▼** and **Shift-▼**
- **Prev** and **Shift-Prev**
- **Next** and **Shift-Next**
- **▽** and **Shift-▽**

including implied **▲** and **▼** arrows from knobs and mice, OUTPUT KBD of these keys, and typing-aid softkey definitions which contain these keycodes.

To disable the keys mentioned above, execute the following statement:

```
CONTROL KBD,16;1
```

You can re-enable these keys by writing a 0 (the default state) into this register.

```
CONTROL KBD,16;0
```



The “scrolling keys” are also re-enabled when you:

- power-up your computer.
- press **Shift-Reset**.
- execute either the SCRATCH or SCRATCH A statement.

If you are not sure of the status of the scrolling keys previously mentioned, you can execute the following statement in a program:

```
100 STATUS KBD,16;A
110 END
```

The results returned will be a 1 if the keys are disabled and a 0 if they are enabled.

Note that keyboard register 16 has no effect when you are in the EDIT mode.

Also, programmatic scrolling will still occur as a result of executing TABXY or CONTROL CRT,1; ... or printing more lines than fit in the OUTPUT Area.

## Modifying the Repeat and Delay Intervals

The keyboard has an auto-repeat feature which allows you to hold a key down to repeat its function rather than pressing and releasing it repeatedly. Holding a key down will cause it to be repeated every 40 milliseconds for as long as it is held down, resulting in a repeat rate of approximately 25 characters per second. However, you may have noticed that the initial delay between the key being pressed and the key being repeated is longer than successive delays between repeats; the initial delay before a key is repeated for the first time is 300 milliseconds (3/10 second).

These intervals can be changed from the program, if desired, by writing different values into CONTROL registers 3 and 4 (of interface select code 2). Register 3 contains the parameter that controls the auto-repeat interval, and register 4 contains the parameter that controls the initial delay. The values of these parameters, multiplied by 10, give the respective intervals in milliseconds with the following exception; if register 3 is set to 256, the auto-repeat is disabled.

The following program sets up softkeys 1, 4, 6, 8 to change these parameters. Run the program and experiment with these intervals to optimize them for your own preferences and needs.

**Note**

Softkey labels (on the keycaps) are  $\overline{f1}$  through  $\overline{f8}$  on ITF keyboards. In default mode, the correspondence between key labels ( $\overline{f1}$ ,  $\overline{f2}$ , etc.) and KEY numbers (in ON KEY and with typing-aid softkeys) is  $\overline{f1}$ =KEY 1,  $\overline{f2}$ =KEY 2, etc. You can change this correspondence by writing a 1 into KBD CONTROL register 14; the new correspondence will be  $\overline{f1}$ =KEY 0,  $\overline{f2}$ =KEY 1, etc.

```

100  ON KEY 1 LABEL "Faster" GOSUB Decr_interval
110  ON KEY 4 LABEL "Slower" GOSUB Incr_interval
120  ON KEY 6 LABEL "Sooner" GOSUB Decr_delay
130  ON KEY 8 LABEL "Later" GOSUB Incr_delay
140  !
150  Interval=40    ! Defaults.
160  Delay=300
170  !
180  DISP "Interval=";Interval;" Delay= ";Delay
190  GOTO 180    ! Loop.
200  !
210  Incr_interval:Interval=Interval+10*(Interval<2560)
220          CONTROL 2,3;Interval/10
230          RETURN
240          !
250  Decr_interval:Interval=Interval-10*(Interval<>10)
260          CONTROL 2,3;Interval/10
270          RETURN
280          !
290  Incr_delay:Delay=Delay+10*(Delay<2560)
300          CONTROL 2,4;Delay/10
310          RETURN
320          !
330  Decr_delay:Delay=Delay-10*(Delay>10)
340          CONTROL 2,4;Delay/10
350          RETURN
360          !
370  END

```

---

## Entering Data from the Keyboard

When the keyboard is specified as the source of data in an ENTER statement, the computer executes the process just as if entering data from any other device. The computer signals to the keyboard that the keyboard is to be the sender of data. The keyboard in turn signals that it is not ready to send data and waits for you to type in and edit the desired data.

The characters you type in appear in the keyboard area of the display, but they are not automatically sent to the computer. As long as you can see the characters, you can edit them before sending them to the computer, *just as during an INPUT statement*. Available characters include all 256 characters that can be generated either with keystrokes or with the **Any char** key (softkey **f7** in the System menu on the ITF keyboard).

Pressing any of the following keys signals the keyboard that the data is to be sent to the computer:

- **Return** key (ITF keyboard).
- **Enter** key (HP 98203 keyboard).
- **Step** key (**f1** in the System menu on the ITF keyboard).
- **Continue** key (**f2** in the System menu, and **f2** User 1 and User 2 menus on the ITF keyboard—User menu softkeys require the KBD binary).

The data is then sent byte-serially according to an agreed-upon handshake convention. The computer enters the data in byte-serial fashion and processes it according to the specified variable(s), type of ENTER statement, and image (if it is an ENTER USING statement).

The differences in pressing the keys or softkeys in the above paragraph are as follows. Keep in mind that the ENTER statement is still being executed as long as the “?” appears in the lower right corner of the display.

**Return** or **Enter** or **Step** All of the characters displayed in the keyboard area are sent to the computer, followed by carriage-return and line-feed characters. These last two characters *usually* terminate entry into the current item in the ENTER statement. In addition, the **Step** key causes the computer to remain in the single-step mode after the ENTER statement has been completely executed.

**Continue** All of the characters displayed in the keyboard area are sent to the computer for processing; *no* trailing carriage-return and line-feed characters are sent. The **Continue** key is pressed if more characters are to be entered into the current variable in the destination list of the ENTER statement.

Type in and run the following program. Experiment with how entry into each variable item is terminated by using the different keys (i.e., the **Continue** key versus **Return**, **Enter**, or **Step** keys). Pressing the **Return**, **Enter**, or **Step** key terminates entry into the current variable, while pressing the **Continue** key allows additional characters to be entered into the current variable.

```

100 DIM String_array$(1:3)[100]
110 ASSIGN @Device_simulate TO 2
120 !
130 ENTER @Device_simulate;String_array$(*)
140 !
150 OUTPUT 1;String_array$(*)
160 !
170 END

```

This use of the keyboard is very powerful when tracing the cause of an error in an ENTER operation. With this tool, you can “debug” or verify any type of ENTER statement, including ENTER statements whose source is intended to be a device on the HP-IB interface. The next section describes this topic.

## Sending the EOI Signal

The EOI signal is implemented on the HP-IB interface. This line ordinarily signals to the computer that the data byte being received is the last byte of the item; thus, it is either an item terminator or a terminating condition for the ENTER statement. (See the chapter "Entering Data" for a further explanation of the EOI signal's effects during ENTER.)

The EOI signal can be simulated from the keyboard when this feature is properly enabled. CONTROL register 12 of interface select code 2 controls this feature; the following example statement shows how to enable this feature.

```
CONTROL 2,12;1
```

To simulate the EOI signal with a character, press the **CTRL** and **E** (or **Shift**-**\*** on the numeric keypad) keys *simultaneously* before the character to be accompanied with EOI is typed. For instance, if the characters "DATA" are to be entered and the EOI is to accompany the last "A", the following key sequence should be pressed before pressing the **Return**, **Enter**, **Step**, or **Continue** key (or softkey).

```
D A T CTRL-E A
```

The same result can be obtained by placing an ENQ character (ASCII control character CHR\$(5), Eq) in front of the character to be accompanied by the EOI signal (see the previous section for further details).

## Sending Data to the Keyboard

Characters output to the keyboard are indistinguishable from characters typed in from the keyboard. All characters output to the keyboard, *including control characters*, are displayed in the keyboard area. The following program outputs the BEEP statement to the keyboard. Read on to see how it works.

```
100 OUTPUT 2;"BEEP"; ! No CR/LF
110 !
120 END
```

## 2 Sending Non-ASCII Keystrokes to the Keyboard

The preceding program sent the characters BEEP to the keyboard, but the statement *was not executed*. Pressing the **Return** or **Enter** key after the program has ended executes the statement. Modify the program to “press” the **Return** or **Enter** key by typing **CTRL-Return** (or **CTRL-Enter**) following the BEEP. Sending this special two-character sequence to the keyboard is equivalent to the operator pressing the corresponding key. Thus, *in general*, to store a non-ASCII “keystroke” within a program line, press the **CTRL** key while simultaneously pressing the desired non-ASCII key.

Since `CHR$(255)` does not generate the same character on most printers as it does on the computer’s display, it is recommended that some explicit means of documenting these character sequences be employed. For instance, string variables can be defined to contain these sequences; then when the program is listed on an external printer, it will be much easier to determine which non-typing keys are being represented. The **CTRL** key is still used with the non-ASCII key to generate the two-character sequence, but the special character should be changed to a `CHR$(255)`.

```
100 Enter_key$=CHR$(255)&"E"  
110 Printall_key$=CHR$(255)&"A"  
120 !  
130 OUTPUT 2;Printall_key$; ! Use ";" to suppress CR/LF.  
140 OUTPUT 2;"BEEP"&Enter_key$;  
150 END
```

---

### Note



Since this type of output can be used to send immediately executed commands (such as `SCRATCH A`), it is important that you use care when outputting commands to the keyboard and when editing statements and commands sent to the keyboard. Undesirable results may occur if the wrong non-ASCII key sequences are output by a program.

---

The table in the next section shows the resultant characters that follow `CHR$(255)` in the two-character sequences generated by these keystrokes. The table can be used to look up which non-ASCII key is to be output if the second character is known or vice-versa.

## Second Byte of Non-ASCII Key Sequences (String)

Holding the **CTRL** key and pressing a non-ASCII key generates a two-character sequence on the CRT. For example,

**CTRL** **Clear line**

produces the following character on the CRT:

**K**

Non-ASCII keypresses can be simulated by outputting these two-byte sequences to the keyboard. For example,

```
OUTPUT KBD;CHR$(255)&"%";
```

produces the same result as shown above. The decimal value of the first byte is 255 (on some computers this is the "inverse-video" **K**).

The following table can be used to look up the key that corresponds to the second character of the sequence.

Normally on an ITF keyboard, **f1** corresponds to ON KEY 1 ... , **f2** corresponds to ON KEY 2 ... , etc. However, you can use CONTROL KBD,14;1 to change this relationship so that **f1** corresponds to ON KEY 0 ... , **f2** corresponds to ON KEY 1, etc.

With 98203 keyboard compatibility (KBD CMODE ON), the ITF keyboard softkeys **f1** thru **f4**, the **Menu** and **System** keys, and **f5** thru **f8** correspond to 98203 softkeys **k0** thru **k9**, respectively. See the *HP BASIC 6.2 Programming Guide* for further information about this mode.

The terms System and User in the *ITF Key* column refer to the softkey menu which is currently active on an ITF keyboard.

Char.	Val.	ITF Key	98203 Key	Closure Key
space	32	1	1	
!	33	Shift-Stop	SHIFT-CLR I/O	Yes
"	34	1	1	
#	35	Shift-Clear line	CLR LN	
\$	36	System (f7)	ANY CHAR	Yes
%	37	Clear line	CLR→END	Yes
&	38	Select <sup>3</sup>	2	
'	39	Prev	2	Yes
(	40	Shift-Tab	SHIFT-TAB	
)	41	Tab	TAB	
*	42	Insert line	INS LN	Yes
+	43	Insert char	INS CHR	
,	44	Next	2	Yes
-	45	Delete char	DEL CHR	
.	46	2	2	

<sup>1</sup> These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR\$(255) in an output to the keyboard, an error is reported (Error 131 Bad non-alphanumeric keycode.).

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error *is not* reported. Instead, the system will perform as much of the indicated action as possible.

<sup>3</sup> These keys have no system meaning, and will BEEP if not trapped by ON KBD.



Char.	Val.	ITF Key	98203 Key	Closure Key
/	47	Delete line	DEL LN	Yes
0	48	User 3 f8	k0	Yes
1	49	User 1 f1	k1	Yes
2	50	User 1 f2	k2	Yes
3	51	User 1 f3	k3	Yes
4	52	User 1 f4	k4	Yes
5	53	User 1 f5	k5	Yes
6	54	User 1 f6	k6	Yes
7	55	User 1 f7	k7	Yes
8	56	User 1 f8	k8	Yes
9	57	User 2 f1	k9	Yes
:	58	System Shift-f6 <sup>3</sup>	2	
;	59	System Shift-f7 <sup>3</sup>	2	
<	60	←	←	
=	61	Result <sup>4</sup>	RESULT	

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error *is not* reported. Instead, the system will perform as much of the indicated action as possible.

<sup>3</sup> These keys have no system meaning, and will BEEP if not trapped by ON KBD.

<sup>4</sup> This ITF key is located in the System Control Key Group just above the Numeric Keypad Group. Note that these keys have no labels on their keycaps; however, they do have labels on the BASIC keyboard overlay for the ITF keyboard. For information on the BASIC keyboard overlay for the ITF keyboard, read the *Using HP BASIC* manual for your system.

Char.	Val.	ITF Key	98203 Key	Closure Key
>	62			
?	63	Recall <sup>4 5</sup>		
Ⓞ	64			
A	65	System		Yes
B	66			
C	67	System		
D	68	<sup>2</sup>		
E	69			Yes
F	70	System		Yes
G	71			
H	72			
I	73			
J	74	(Katakana) <sup>2</sup>	(Katakana) <sup>2</sup>	

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error *is not* reported. Instead, the system will perform as much of the indicated action as possible.

<sup>4</sup> This ITF key is located in the System Control Key Group just above the Numeric Keypad Group. Note that these keys have no labels on their keycaps; however, they do have labels on the BASIC keyboard overlay for the ITF keyboard. For information on the BASIC keyboard overlay for the ITF keyboard, refer to the *Using HP BASIC* manual for your system.

<sup>5</sup> also System

<sup>6</sup> also System

<sup>7</sup> Or

Char.	Val.	ITF Key	98203 Key	Closure Key
K	75	Clear display	CLR SCR	Yes
L	76	Graphics <sup>4</sup>	GRAPHICS	Yes
M	77	Alpha <sup>4</sup>	ALPHA	Yes
N	78	Dump Graph <sup>4</sup>	DUMP GRAPHICS	Yes
O	79	Dump Alpha <sup>4 8</sup>	DUMP ALPHA	Yes
P	80	Stop	PAUSE	Yes
Q	81	1	1	
R	82	System f3	RUN	Yes
S	83	System f1	STEP	Yes
T	84	Shift-↓	SHIFT-↓	Yes
U	85	Caps	CAPS LOCK	Yes
V	86	↓	↓	Yes
W	87	Shift-▲	SHIFT-↑	Yes

<sup>1</sup> These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR\$(255) in an output to the keyboard, an error is reported (Error 131 Bad non-alphanumeric keycode.).

<sup>4</sup> This ITF key is located in the System Control Key Group just above the Numeric Keypad Group. Note that these keys have no labels on their keycaps; however, they do have labels on the BASIC keyboard overlay for the ITF keyboard. For information on the BASIC keyboard overlay for the ITF keyboard, refer to the *Using HP BASIC* manual for your system.

<sup>8</sup> Also **Print**

Char.	Val.	ITF Key	98203 Key	Closure Key
x	88	2	<b>EXECUTE</b>	Yes
y	89	(Roman) <sup>2</sup>	(Roman) <sup>2</sup>	Yes
z	90	1	1	
[	91	System <b>f5</b>	<b>CLR TAB</b>	
\	92	<b>▼</b>	2	Yes
]	93	System <b>Shift-f5</b>	<b>SET TAB</b>	
^	94	<b>▲</b>	<b>↑</b>	Yes
_	95	System <b>Shift-▼</b>	2	Yes
‘	96	1	1	
a	97	User 2 <b>f2</b>	<b>SHIFT-k0</b>	Yes
b	98	User 2 <b>f3</b>	<b>SHIFT-k1</b>	Yes
c	99	User 2 <b>f4</b>	<b>SHIFT-k2</b>	Yes
d	100	User 2 <b>f5</b>	<b>SHIFT-k3</b>	Yes
e	101	User 2 <b>f6</b>	<b>SHIFT-k4</b>	Yes
f	102	User 2 <b>f7</b>	<b>SHIFT-k5</b>	Yes

<sup>1</sup> These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR\$(255) in an output to the keyboard, an error is reported (Error 131 Bad non-alphanumeric keycode.).

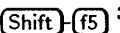
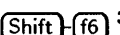






<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error *is not* reported. Instead, the system will perform as much of the indicated action as possible.

Char.	Val.	ITF Key	98203 Key	Closure Key
g	103	User 2 (f8)	(SHIFT)-(k6)	Yes
h	104	User 3 (f1)	(SHIFT)-(k7)	Yes
i	105	User 3 (f2)	(SHIFT)-(k8)	Yes
j	106	User 3 (f3)	(SHIFT)-(k9)	Yes
k	107	User 3 (f4)	2	Yes
l	108	User 3 (f5)	2	Yes
m	109	User 3 (f6)	2	Yes
n	110	User 3 (f7)	2	Yes
o	111	System (Shift)-(f1) <sup>3</sup>	2	
p	112	System (Shift)-(f2) <sup>3</sup>	2	
q	113	System (Shift)-(f3) <sup>3</sup>	2	
r	114	System (Shift)-(f4) <sup>3</sup>	2	
s	115	User (Shift)-(f1) <sup>3 9</sup>	2	
t	116	User (Shift)-(f2) <sup>3 9</sup>	2	
u	117	User (Shift)-(f3) <sup>3 9</sup>	2	
v	118	User (Shift)-(f4) <sup>3</sup>	2	

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error *is not* reported. Instead, the system will perform as much of the indicated action as possible.

<sup>3</sup> These keys have no system meaning, and will BEEP if not trapped by ON KBD.

<sup>9</sup> These keys are also generated by the HP 46060A/B and HP 46095A (HP mouse devices) buttons unless GRAPHICS INPUT IS is using them.

Char.	Val.	ITF Key	98203 Key	Closure Key
w	119	User  <sup>3</sup>	2	
x	120	User  <sup>3</sup>	2	
y	121	User  <sup>3</sup>	2	
z	122	User  <sup>3</sup>	2	
{	123		2	Yes
	124		2	Yes
}	125		2	Yes
-	126		2	Yes
	127	1	1	

<sup>1</sup> These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR\$(255) in an output to the keyboard, an error is reported (Error 131 Bad non-alphanumeric keycode.).

<sup>2</sup> Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error *is not* reported. Instead, the system will perform as much of the indicated action as possible.

<sup>3</sup> These keys have no system meaning, and will BEEP if not trapped by ON KBD.

## Closure Keys

Several of the non-ASCII keys are known as **closure keys**. Closure keys are so named because they close (block) further keyboard input until processed. *The computer can only process two closure keys between program lines during a running program.* If more than two appear in the data output to the keyboard, the extra keys will be deferred until the next end-of-line is encountered and two more closure keys can be processed. See the table on the preceding pages to determine which keys are “closure keys”.

As an example, the following program sends four closure keys to the keyboard with a single OUTPUT statement. Only the first two closure keys are processed *after* this OUTPUT statement (but *before* DISP "Next BASIC line" is executed). The third and fourth closure keys are processed after DISP "Next BASIC line" is executed (but before DISP "2nd BASIC line" is executed). This accounts for the following display after running the program, since the "Printall" command was not executed until after DISP "Next BASIC line" was executed.

```
100 ! Define non-ASCII keys.
110 En$=CHR$(255)&"E" ! ENTER or Return key.
120 Up$=CHR$(255)&"^" ! Up arrow key.
130 Prt$=CHR$(255)&"A" ! PRT ALL key or softkey.
140 !
150 CONTROL 2,1;0 ! Turn PRINTALL off.
160 CONTROL 1,1;1 ! Begin on top screen line.
170 OUTPUT 1;"Line 1"
180 OUTPUT 1;"Line 2"
190 OUTPUT 1;"Line 3"
200 WAIT 1
210 !
220 ! Now send statement with 4 closure keys.
230 OUTPUT 2;"DISP ""Hello""";En$;Up$;Up$;Prt$;
240 DISP "Next BASIC line" ! PRT ALL still off.
250 DISP "2nd BASIC line" ! Now PRT ALL is on.
260 !
270 END
```

```
Line 3  
2nd BASIC line
```

```
2nd BASIC line
```

```
Printall on
```

In addition, if the last character sent to the keyboard is a `CHR$(255)`, the next character typed in by the user will give unexpected results. Again, it is important to exercise care when using this feature.

---

## Softkeys

The keys on the upper-left portion of the keyboard are called “softkeys.” These keys can be defined by BASIC programs to initiate program branches. In addition, these keys can be defined as typing-aid keys, which produce keystrokes just as if you had typed them in yourself.

Brief examples of using the softkeys have already been presented in the “Interface Events” chapter, and in the section found earlier this chapter



entitled “Modifying the Repeat and Delay Intervals”. Typing-aid softkeys are discussed in the *Using HP BASIC* manual for your system.

---

## Sensing Knob Rotation

Your computer system may or may not have a knob (built-in, or HP 46083) or a mouse (HP 46060). In any event, the programs below are illustrative of how knob and mouse movements can be trapped in a program. It is assumed that you will use the techniques and apply them to your programming situation.

The “event” of the knob (rotary pulse generator) being rotated can be sensed by a program. The branch location, interval at which the computer interrogates the knob for the occurrence of rotation, and branch priority are set up with a statement such as the following:

```
ON KNOB Interval,Priority CALL Knob_turned
```

In addition to the program being able to sense rotations of the knob, it can also determine how many pulses the knob has produced and whether or not either or both of the **CTRL** or **Shift** keys are being pressed. This ability to “qualify” the use of the knob allows it to be used for up to four different purposes. The following program shows how to set up the branch, how to determine the number of pulses, and how to determine the direction of rotation.

---

### Note



HIL devices do not set the “CTRL” bit, although they do set the “SHIFT” bit (if the last record processed was “y-axis” data). Consequently, you should not depend on the value of keyboard status register 10.

---

```

100  ON KNOB .25 GOSUB Knob  ! Check knob every 1/4 sec.
110  !
120  FOR Iteration=1 TO 400
130    WAIT .2
140    DISP Iteration
150  NEXT Iteration
160  !
170  STOP
180  !
190 Knob:  STATUS 2,10;Key_with_knob
200        PRINT KNOBX;" pulses ";KNOBY;" pulses ";
201        DISP TAB(40),"Status = ";Key_with_knob
210        IF Key_with_knob=0 THEN
220          PRINT
230        ELSE
240          IF Key_with_knob=1 THEN PRINT "with SHIFT"
250          IF Key_with_knob=2 THEN PRINT "with CTRL"
260          IF Key_with_knob=3 THEN PRINT "with SHIFT and CTRL"
270        END IF
280        RETURN
290  END

```

If any pulses have occurred since the last branch, the specified branch will be initiated.

One full rotation of the knob produces 120 pulses. The service routine calls the KNOBX and KNOBY functions to determine how many pulses (only *net* rotation) have been generated *since the last call* to this function. If the number is positive, a net clockwise rotation has occurred; a negative number signifies that a net counterclockwise rotation has occurred. Since the pulse counter (on built-in knobs) can only sense +128 to -127 pulses *during the specified interval*, the interval parameter should be chosen small enough to interrogate the knob before the pulse counter reaches one of these values. Experiment with this parameter to adjust it for your particular application. (Note that HIL devices can count from 32 767 to -32 768 pulses during the interval.)

The next program illustrates the use of an ON KNOB with a mouse (HP 46060). Note changes in iteration as you move the mouse.

```

10  COM /Knob/ Kx,Ky
20  Kx=0
30  Ky=0
40  ON KNOB 1 CALL Knob
50  PRINT TABXY(1,1);" "
60  FOR I=1 TO 1.E+6
70    DISP I
80    PRINT TABXY(1,2);Kx;Ky;"      "
90  NEXT I
100 END
110 SUB Knob
120  COM /Knob/ Kx,Ky
130  INTEGER Knx,Kny
140  Knx=KNOBX
150  Kny=KNOBY
160  Kx=Kx+Knx
170  Ky=Ky+Kny
180  PRINT TABXY(1,5);Knx;Kny;"      "
190 SUBEND

```

You can also trap mouse keys with ON KBD and KBD\$ function (see the subsequent section for details on using these keywords). These keys produce the same codes as the **Shift-f1**, **Shift-f2**, etc. keys on ITF keyboards (while in any User menu).

---

## Enhanced Keyboard Control

Normally, the BASIC operating system handles all keyboard inputs. Several BASIC statements allow programs to handle inputs from the keyboard; examples are the INPUT, LINPUT, ENTER, ON KEY, and ON KNOB statements. Additional keyboard statements provide BASIC programs with a means of intercepting both ASCII and non-ASCII keystrokes for processing by the program. The statements are:

- ON KBD** sets up and enables keystrokes to be trapped.
- ON KBD ALL** includes **Pause**, **Stop**, **Clr I/O**, **System**, **User**, **Menu**, **Shift-Menu** and sofkeys. See the key tables in the section of this chapter entitled "Second Byte of Non-ASCII Key Sequences (String)" for appropriate ITF key labels.

**KBD\$** returns keystrokes trapped in the buffer.  
**OFF KBD** resumes normal keystroke processing.

ON KBD allows terminal emulation, keyboard masking, and special data inputs. Each keystroke produces unique code(s) that allow the program to differentiate between different keys being pressed. The program can also determine whether the **Shift** or **CTRL** keys are being pressed with most keys, but these keystrokes cannot be detected by themselves. Also, the **Reset** key cannot be trapped by ON KBD.

## Trapping Keystrokes

The ON KBD statement sets up a branch that is initiated when the keyboard buffer becomes "non-empty". The service routine may then interrogate the buffer as desired, processing the keystrokes as determined by the program. The keyboard buffer contains up to 256 characters. Calling the KBD\$ function does two things: it returns all keystrokes trapped since the last time the buffer was read, and it then clears the keyboard buffer.

The following program uses ON KBD, KBD\$, and OFF KBD to trap and process keystrokes, rather than allowing the operating system to do the same. The program defines each keystroke to print a complete word.

```

100 OPTION BASE 1
110 DIM String$(26)[6]
120 READ String$(*)
130 !
140 DATA A,BROWN,CAT,DOG,EXIT,FOX,GOT
150 DATA HI,IN,JUMPS,KICKED,LAZY,MY
160 DATA NO,OVER,PUSHED,QUICK,RED,SMART
170 DATA THE,UNDER,VERY,WHERE,XRAY,YES,ZOO
180 !
190 PRINTER IS 1
200 PRINT "Many ASCII keys have been"
210 PRINT "defined to produce words."
220 PRINT
230 PRINT "Press the following keys."
240 PRINT "T Q B F J O T L D ."
250 !
260 ON KBD GOSUB Process_keys
270 !
280 LOOP

```

```

290 EXIT IF Word$="EXIT"
300 END LOOP
310 !
320 STOP
330 !
340 Process_keys:Key$=KBD$      ! Read buffer.
350 !
360 REPEAT ! Process ALL keys trapped.
370   Key_code=NUM(Key$[1;1]) ! Calculate code.
380   !
390   SELECT Key_code         ! Choose response.
400   !
410   CASE 65 TO 90           ! CASE "A" TO "Z".
420     Word$=String$(Key_code-64)
430     Key$=Key$[2]         ! Remove processed key.
440     !
450   CASE 97 TO 122         ! CASE "a" TO "z".
460     Word$=String$(Key_code-96)
470     Key$=Key$[2]         ! Remove processed key.
480     !
490   CASE 255               ! CASE non-ASCII key.
500     IF Key$[2;1]<>CHR$(255) THEN
510       Word$=Key$[1,2]    ! Non-ASCII key alone,
520       Key$=Key$[3]      ! so take 2 codes.
530     ELSE
540       Word$=Key$[1,3]    ! Non-ASCII w/ CTRL,
550       Key$=Key$[4]      ! so take 3 codes.
560     END IF
570   CASE ELSE              ! CASE all others.
580     Word$=""
590     Key$=Key$[2]        ! Remove processed key.
600     !
610   END SELECT
620   !
630   ! Execute response.
640   Defined=LEN(Word$)<>0
650   IF Defined THEN
660     PRINT Word$;" ";
670     DISP
680   ELSE
690     BEEP 100,.05
700     DISP "Key undefined."
710   END IF
720   !
730 UNTIL LEN(Key$)=0      ! Until ALL keys processed.

```

```

740      !
750  RETURN
760      !
770 Quit:END

```

Notice that all non-ASCII keys produce two-character sequences: CHR\$(255) followed by an ASCII character. Pressing the **CTRL** key with non-ASCII keys produce three-character sequences: another CHR\$(255) character preceding the two-character sequence produced by pressing the non-ASCII key by itself. See the tables in the section entitled “Second Byte of Non-ASCII Key Sequences (String)” for a listing of the sequences produced by non-ASCII keys.

BASIC programs can output ASCII keystrokes to the keyboard, via OUTPUT 2, without initiating an ON KBD branch; however, outputting non-ASCII “closure” keys followed by other keys will initiate the ON KBD branch. For example, executing the following statement (in a program line):

```
OUTPUT 2;"32*2";CHR$(255);"E";"KBD";
```

causes the characters KBD which follow the closure key to be placed in the KBD\$ buffer, which also initiates the ON KBD branch. The **Return** keycode which was sent to the keyboard executes the numeric expression 32\*2 before the branch is initiated. OUTPUT to the keyboard while ON KBD is in effect should contain at most one closure key, and that key should be at the end, in order to avoid this “recirculation” of closure keys.

ON KBD branching is disabled by DISABLED, deactivated by OFF KBD, and temporarily deactivated when the program is executing LINPUT, INPUT, or ENTER KBD statements. Note that the keyboard input line can be read without deactivating ON KBD by using the SYSTEM\$(“KBD LINE”) function.

## Mouse Keys

You can also trap mouse keys with this technique. The keys produce CHR\$(255) followed by “s”, “t”, and so forth.

## Softkeys and Knob Rotation

When ON KNOB is not in effect, knob rotation is also trapped by ON KBD. Rotation of the knob will produce “cursor” keystrokes. A clockwise rotation of the knob produces CHR\$(255) followed by “>”, while a counter-clockwise rotation produces CHR\$(255) followed by “<”. When using the HP 46083 Rotary Control Knob, pressing the **Shift** key and rotating the Knob clockwise produces CHR\$(255) followed by “^”, and rotating the Knob counter-clockwise produces CHR\$(255) followed by “V”. These same results can be produced when using the HP 46060A Mouse; however, the results are dependent on the “toggle” state for the Rotary Control Knob and “horizontal” and “vertical” movements for the HP Mouse.

ON KBD ,ALL allows softkey trapping (“overrides” ON KEY) but does not change the softkey labels.

## Disabling Interactive Keyboard

Another group of statements is used to disable the interactive keyboard functions:

SUSPEND INTERACTIVE

ignores the **Pause**, **Stop**, **Step**, and **Clr I/O** keys (see the table in the section entitled “Second Byte of Non-ASCII Key Sequences (String)” for equivalent ITF keys) and disables live keyboard execution.

SUSPEND INTERACTIVE,RESET

ignores **Reset** (see the table in the section entitled “Second Byte of Non-ASCII Key Sequences (String)” for equivalent ITF key) too.

RESUME INTERACTIVE

returns to normal operation.

SUSPEND INTERACTIVE can be used to prevent interruption of programs which gather data or which control other systems.

Special care should be taken when using SUSPEND INTERACTIVE,RESET. If an “infinite loop” is executed while interactive keyboard functions are disabled, only the power switch will stop execution of the program.

```

110 ! This program cannot be stopped by
120 ! Pause, Stop, or Reset keys
130 ! before its normal completion.
140 !
150 !
160 SUSPEND INTERACTIVE, RESET ! Ignore keyboard.
170 !
180 PRINT "COUNTDOWN IS "
190 PRINT
200 I=10 ! Initial value.
210 REPEAT
220 PRINT " T minus ";I ! Print count.
230 I=I-1 ! Decrement count.
240 WAIT 1 ! Wait one second.
250 UNTIL I<0
260 !
270 PRINT
280 BEEP 100,1
290 PRINT "Done"
300 RESUME INTERACTIVE ! Return to normal.
310 !
320 END

```

---

## Locking Out the Keyboard

There are certain times during program execution when it is expedient to prevent the operator from using the keyboard, such as during a critical experiment which cannot be disturbed. Then the knob and groups of keyboard keys can be enabled and disabled separately.

Setting bit 0 of register 7 (of interface select code 2) *disables* all keys (excluding the **Reset** key) and the knob. The following program first sets up the KNOB and KEY events to initiate program branches. It is assumed that the keyboard is already enabled; if you are not sure, press the **Reset** key. When the program is run, the keyboard and knob remain enabled for about five seconds, after which they are disabled. The program then displays the time of day indefinitely; the only way to stop the program is to press the **Reset** key.



```

100 ON KEY 1 LABEL "SFK 1" GOSUB Key1
110 ON KNOB .2 GOSUB Knob
120 !
130 PRINT "You've got 5 seconds. GO! "
140 FOR Iteration=1 TO 20
150   WAIT .25
160 NEXT Iteration
170 !
180 Reset_disable=0 ! Reset key remains ENABLED.
190 Ky_knb_disable=1 ! DISABLE reset of kbd.
200 CONTROL 2,7;2*Reset_disable+Ky_knb_disable
210 PRINT "Time's up!"
220 BEEP
230 !
240 Loop: DISP TIME$(TIMEDATE)
250   GOTO Loop
260   !
270   !
280 Key1: PRINT "Special function key 1 pressed."
290   RETURN
300   !
310 Knob: PRINT "Knob rotation sensed."
320   RETURN
330 END

```

If the value of the variable `Reset_disable` is set to 1 in the preceding program, the only way to *stop* the program is to turn off power to the computer, losing the program and all data currently in computer memory.

## Note



Use care when locking out *both* the **Reset** key and the keyboard keys. If both are locked out, the *only* way to prematurely stop the program is to turn the computer off.

## Special Considerations

Disabling keyboard interrupts by locking out the keyboard will also block the use of other HP-HIL devices. For example, if an HP-HIL Graphics Tablet is the current graphics input device and keyboard interrupts are disabled, executing a DIGITIZE statement will cause the system to hang, waiting for a response it cannot receive. Attempting to execute an HIL SEND statement while keyboard interrupts are disabled will cause an error to occur.

## 2 Keyboard Status and Control Registers

<i>STATUS Register 0</i>	CAPS lock flag
<i>CONTROL Register 0</i>	Set CAPS lock if non-0
<i>STATUS Register 1</i>	PRINTALL flag
<i>CONTROL Register 1</i>	Set PRINTALL if non-0
<i>STATUS Register 2</i>	Function key menu.
<i>CONTROL Register 2</i>	Function key menu: 0 = System menu (or SYSTEM KEYS statement) 1-3 = User menu 1 thru 3 (or USER <i>n</i> KEYS statement along with the appropriate menu number)
<i>STATUS Register 3</i>	Undefined
<i>CONTROL Register 3</i>	Set auto-repeat interval. If 1 thru 255, repeat interval in milliseconds is 10 times this value. 256 = turn off auto-repeat. (Default at entry to BASIC/UX or SCRATCH A is the value that was in effect before entry to BASIC/UX.) (Default at power-up or SCRATCH A is 40 ms for BASIC/WS.)
<i>STATUS Register 4</i>	Undefined
<i>CONTROL Register 4</i>	Set delay before auto-repeat. If 1 thru 256, delay in milliseconds is 10 times this value. (Default at entry to BASIC/UX or SCRATCH A is the value that was in effect before entry to BASIC/UX.) (Default at power-up or SCRATCH A is 300 ms for BASIC/WS.)
<i>STATUS Register 5</i>	KBD\$ buffer overflow register. 1 = overflow. Register is reset when read.
<i>CONTROL Register 5</i>	Undefined

<i>STATUS Register 6</i>	Typing aid expansion overflow register. 1 = overflow. Register is reset when read.
<i>CONTROL Register 6</i>	Undefined
<i>STATUS Register 7</i>	Interrupt Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	Initialize timeout interrupt disabled	Reserved for future use	Reserved for future use	Reset key interrupt disabled	Keyboard and knob interrupt disabled
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*CONTROL Register 7*      Interrupt Disable Mask

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used			Initialize timeout	Reserved for future use	Reserved for future use	Reset key	Keyboard and knob
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*STATUS Register 8*

## Keyboard Language Jumper

0-US ASCII	7-United Kingdom	13-Swiss German
1-French	8-Canadian French	14-Latin(Spanish)
2-German	9-Swiss French	15-Danish
3-Swedish	10-Italian	16-Finnish
4-Spanish	11-Belgian	17-Norwegian
5-Katakana	12-Dutch	18-Swiss French*
6-Canadian English		19-Swiss German*

\* Alternate version

See also SYSTEM\$(“KEYBOARD LANGUAGE”) which requires the LEX binary. Note that the STATUS statement when used with this register does not require the LEX binary.

*CONTROL Register 8*      Undefined

*STATUS Register 9*      Keyboard Type

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Internal use		1=HIL keyboard interface 0=non-HIL	1=No keyboard 0=Keyboard present	1=n-key rollover 0=2 or less rollover	0	0	0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bits 5, 1, and 0 of STATUS Register 9 and the following table can be used to determine the Keyboard Type.

Bit 5	Bit 1	Bit 0	Keyboard Type
0	0	0	HP 98203B or built-in (unsupported)
0	0	1	HP 98203A (unsupported)
1	0	0	ITF (such as the HP 46020A and 46021A)
1	1	0	HP 98203C (unsupported)

*CONTROL Register 9*            Undefined  
*STATUS Register 10*            Status at Last Knob Interrupt

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	CTRL key pressed	SHIFT key pressed
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Note that bit 1 is *always* 0 for keyboards and all HP-HIL mice and knobs (e.g., HP 46083A Rotary Control Knob and HP 46085 Control Dials).

*CONTROL Register 10*            Undefined  
*STATUS Register 11*            0=horizontal-pulse mode; 1=all-pulse mode.  
*CONTROL Register 11*            Set knob pulse mode. (This CONTROL register is *not* supported with BASIC/UX, because the KNB2.0 binary is unsupported on BASIC/UX.)  
*STATUS Register 12*            "Pseudo-EOI for CTRL-E " flag  
*CONTROL Register 12*            Enable pseudo-EOI for CTRL-E if non-0  
*STATUS Register 13*            Katakana flag

- CONTROL Register 13* Set Katakana if non-0
- STATUS Register 14* Numbering of softkeys on ITF keyboard:  
 0 →  is key number 1 (default);  
 1 →  is key number 0;
- CONTROL Register 14* Softkey numbering on ITF keyboard (see above register description).
- STATUS Register 15* Currently in 98203 keyboard compatibility mode:  
 0→OFF (default)  
 1→ON
- CONTROL Register 15* Turns “98203 keyboard compatibility mode” on ( $\neq 0$ ) and off ( $=0$ ). (See the *HP BASIC Programming Guide* for further information about using this mode.) Note that instead of using the CONTROL register 15 statement you can use the KBD CMODE statement to turn the “98203 keyboard compatibility mode” ON and OFF.
- STATUS Register 16* Returns the enabled/disabled status of the up and down arrow keys, , , and  (both shifted and un-shifted for all of these keys). If the status value is 1 it means these keys are deactivated. Note that the default value is 0.
- CONTROL Register 16* Allows you to disable or re-enable the display scrolling keys mentioned for STATUS Register 16. This prevents accidental scrolling of the display screen. Executing a 1 with the CONTROL statement deactivates the print scrolling keys and a 0 activates them.
- STATUS Register 17* Automatic menu switching:  
 1 → enabled (default)  
 0 → disabled

*CONTROL Register 17*

Automatic menu switching:

<>0 → enable

0 → disable

This register controls whether a system with an ITF keyboard will switch to (from) the User 2 Menu automatically on entering (leaving) EDIT mode.



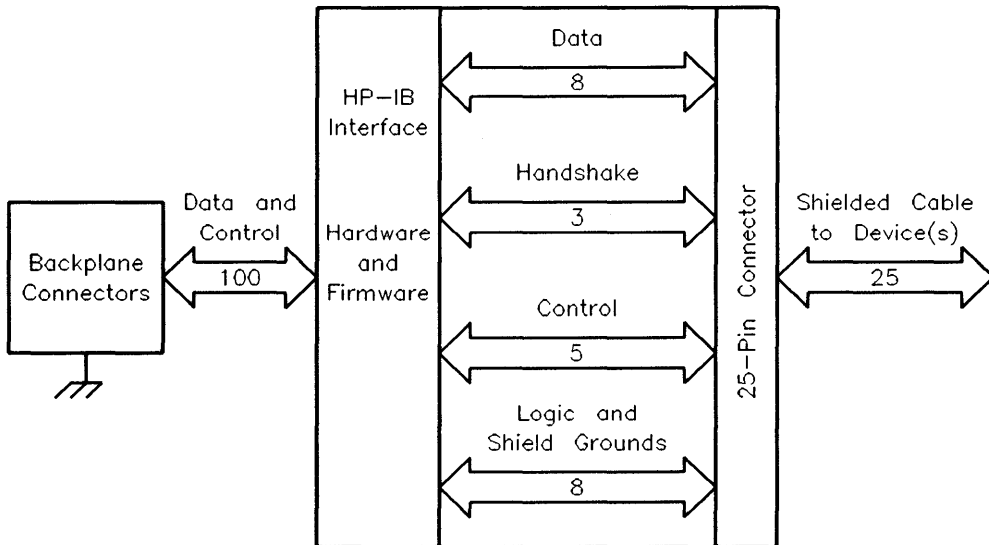


## The HP-IB Interface

### Introduction

This chapter describes the techniques necessary for programming the HP-IB interface. Many of the elementary concepts have been discussed in previous chapters; this chapter describes the specific details of how this interface works and how it is used to communicate with and control systems consisting of various HP-IB devices.

The HP-IB (Hewlett-Packard Interface Bus), commonly called the **bus**, provides compatibility between the computer and external devices conforming to the IEEE 488-1978 standard. Electrical, mechanical, and timing compatibility requirements are all satisfied by this interface.



**HP-IB Interface Block Diagram**

The HP-IB Interface is both easy to use and allows great flexibility in communicating data and control information between the computer and external devices. It is one of the easiest methods to connect more than one device to the same interface.

---

## Initial Installation and Verification

Refer to the HP-IB Installation Note for information about setting the switches and installing an external HP-IB interface. Once the interface has been properly installed, you can verify that the switch settings are what you intended by running the following program. The defaults of the internal HP-IB interface can also be checked with the program. The results are displayed on the CRT.

```
100  PRINTER IS CRT
110  PRINT CHR$(12) ! Clear screen w/ FF.
120  !
130  Ask: INPUT "Enter HP-IB interface select code",Isc
140  IF Isc<7 OR Isc>30 THEN GOTO Ask
150  !
160  STATUS Isc;Card_id
170  IF Card_id<>1 THEN
180    PRINT "Interface at select code";Isc;
190    PRINT "is not an HP-IB"
200    PRINT
210    STOP
220  END IF
230  !
240  PRINT "HP-IB interface present"
250  PRINT " at select code";Isc
260  PRINT
270  !
280  STATUS Isc,1;Intr_dma
290  Level=3+(BINAND(32+16,Intr_dma) DIV 16)
300  PRINT "Hardware interrupt level =";Level
310  !
320  STATUS Isc,3;Addr_ctrlr
330  Address=Addr_ctrlr MOD 32
340  PRINT "Primary address =";Address
350  !
360  Sys_ctrlr=BIT(Addr_ctrlr,7)
```

```
370  IF Sys_ctrl THEN
380    PRINT "System Controller"
390  ELSE
400    PRINT "Non-system Controller"
410  END IF
420  !
430  END
```

The hardware interrupt level is described in Chapter 7. Hardware interrupt level is set to 3 on built-in HP-IB interface, but can range from 3 to 6 on optional interfaces. Primary address is further described in “HP-IB Device Selectors” in the next section.

The term “System Controller” is also further described later in this chapter in “General Structure of the HP-IB”. The internal HP-IB has a jumper or switch that is set at the factory to make it a system controller. To find out the location of this jumper or switch, refer to the documentation that comes with your computer. Note that the location varies with different Models of computers. External HP-IB interfaces have a switch that controls this interface state.

---

## Communicating with Devices

This section describes programming techniques used to output data to and enter data from HP-IB devices. General bus operation is also briefly described in this chapter. Later chapters will describe: further details of specific bus commands, handling interrupts, and advanced programming techniques.

### HP-IB Device Selectors

Since the HP-IB allows the interconnection of several devices, each device must have a means of being uniquely accessed. Specifying just the interface select code of the HP-IB interface through which a device is connected to the computer is not sufficient to uniquely identify a specific device on the bus.

Each device “on the bus” has an **primary address** by which it can be identified; this address must be unique to allow individual access of each device. Each HP-IB device has a set of switches that are used to set its address. Thus, when

a particular HP-IB device is to be accessed, it must be identified with both its interface select code and its bus address.

The interface select code is the first part of an HP-IB device selector. The interface select code of the internal HP-IB is 7; external interfaces can range from 8 to 31. The second part of an HP-IB device selector is the device's primary address, which are in the range of 0 through 30. For example, to specify the device:

3

the interface at select code 7            use device selector = 722  
the device at primary address 22

the interface at select code 10        use device selector = 1002  
the device at primary address 2

Remember that each device's address must be unique. The procedure for setting the address of an HP-IB device is given in the installation manual for each device. The HP-IB interface also has an address. The default address of the internal HP-IB is 21 or 20, depending on whether or not it is a System Controller, respectively. The addresses of external HP-IB interfaces are set by configuring the address switches on each interface card. Each HP-IB interface's address can be determined by reading STATUS register 3 of the appropriate interface select code, and each interface's address can be changed by writing to CONTROL register 3. See "Determining Controller Status and Address" and "Changing the Controller's Address" for further details.

### Moving Data Through the HP-IB

Data is output from and entered into the computer through the HP-IB with the OUTPUT and ENTER statements, respectively; all of the techniques described in Chapters 4 and 5 are completely applicable with the HP-IB. The only difference between the OUTPUT and ENTER statements for the HP-IB and those for other interfaces is the addressing information within HP-IB device selectors.

*Examples*

```

100  Hpib=7
110  Device_addr=22
120  Device_selector=Hpib*100+Device_addr
130  !
140  OUTPUT Device_selector;"F1R7T2T3"
150  ENTER Device_selector;Reading

```

```

320  ASSIGN @Hpib_device TO 702
330  OUTPUT @Hpib_device;"Data message"
340  ENTER @Hpib_device;Number

```

```

440  OUTPUT 822;"F1R7T2T3"

```

```

380  ENTER 724;Readings(*)

```

All of the IMAGE specifiers described in Chapters 4 and 5 can also be used by OUTPUT and ENTER statements that access the HP-IB interface, and the definitions of all specifiers remain exactly as stated in those chapters.

*Examples*

```

100  ASSIGN @Printer TO 701
110  OUTPUT @Printer USING "6A,3X,2D.D";Item$,Quantity

```

```

860  ASSIGN @Device TO 825
870  OUTPUT @Device USING "#,B";65,66,67,13,10
870  ENTER @Device USING "#,K";Data$

```

## Using an Interface in the HP-UX Environment

This section explains the interface locking and burst I/O, which are useful when using an interface in the HP-UX environment, and applies to BASIC/UX only.

## Locking an Interface to a Process

In a multi-user environment, interface cards are usually accessible to several users. BASIC/UX supports this sharing by making no attempt to guarantee exclusive access to an interface *unless it is directed to do so*. This allows you to access instruments, for instance, on an HP-IB bus that is shared with other peripherals. Although this is not a recommended configuration, it is allowed.

3

BASIC/UX provides interface locking to support exclusive access to an interface. When an interface is locked to a process, all other processes are prevented from using that interface. For instance, this feature can prevent the loss of important data while a process is taking measurements from an instrument by keeping other users or processes from using the same interface.

Interface locking is enabled and disabled by using pseudo-register 255 and the interface's select code. For example:

```
CONTROL 7,255;1  Enables HP-IB interface locking.
CONTROL 7,255;0  Disables HP-IB interface locking.
```

In order to be a “good citizen” on a multi-user system, you should unlock an interface after you no longer need to have it locked.

Note that attempting to lock an HP-IB connected to a system disc will result in an error.

In addition, attempting to lock an interface that is already locked to another process will cause a program to suspend execution until:

- The interface is unlocked (by the other process to which it is currently locked).
- A timeout occurs.
- You press **Reset** or **Clr I/O**.

## Using the Burst I/O Mode

The default mode of HP-UX I/O transactions requires many time consuming HP-UX system calls to send data to the destination.

Another method, “burst I/O”, maps the interface into your “user address space”, thereby bypassing the memory buffer. This direct-write method decreases the number of calls to HP-UX I/O system routines, which establishes

## 3-6 The HP-IB Interface

a short, highly tuned path for performing I/O operations. The interface is also implicitly locked when burst mode is enabled (see above explanation of interface locking).

Burst I/O provides the fastest I/O performance available with BASIC/UX for the “smaller” I/O transactions that are typical of many instruments. For instance, an 8-byte ENTER operation is over an order of magnitude faster when burst mode is enabled. For larger I/O operations, of more than 4 000 bytes for example, burst mode becomes increasingly slower than the default (buffered or DMA) I/O modes.

Burst I/O is enabled and disabled by using register 255 and the interface’s select code. For example:

**CONTROL 7,255;3** *Enables HP-IB interface burst I/O.*  
**CONTROL 7,255;0** *Disables HP-IB interface burst I/O.*

In order to be a “good citizen” on a multi-user system, you should unlock an interface after you no longer need to have it locked.

In addition, attempting to use burst mode with an interface that is already locked to another process will cause a program to suspend execution until:

- The interface is unlocked (by the other process to which it is currently locked).
- A timeout occurs.
- You press **Reset** or **Clr I/O**.

Note also that you cannot set up an ON TIMEOUT for an interface when using burst mode.

## General Structure of the HP-IB

Communications through the HP-IB are made according to a precisely defined set of rules. These rules help to ensure that only orderly communication may take place on the bus. For conceptual purposes, the organization of the HP-IB can be compared to that of a committee. A committee has certain “rules of order” that govern the manner in which business is to be conducted. For the HP-IB, these rules of order are the IEEE 488-1978 standard.

One member, designated the “committee chairman,” is set apart for the purpose of conducting communications between members during the meetings. This chairman is responsible for overseeing the actions of the committee and generally enforces the rules of order to ensure the proper conduct of business. If the committee chairman cannot attend a meeting, he designates some other member to be “acting chairman.”

On the HP-IB, the **System Controller** corresponds to the committee chairman. The system controller is generally designated by setting a switch on the interface and cannot be changed under program control. However, it is possible to designate an “acting chairman” on the HP-IB. On the HP-IB, this device is called the **Active Controller**, and may be any device capable of directing HP-IB activities, such as a desktop computer.

When the System Controller is first turned on or reset, it assumes the role of Active Controller. Thus, only one device can be designated System Controller. These responsibilities may be subsequently passed to another device while the System Controller tends to other business. This ability to pass control allows more than one computer to be connected to the HP-IB at the same time.

In a committee, only one person at a time may speak. It is the chairman’s responsibility to “recognize” which one member is to speak. Usually, all committee members present always listen; however, this is not always the case on the HP-IB. One of the most powerful features of the bus is the ability to selectively send data to individual (or groups of) devices.

Imagine slow note takers and a fast note takers on the committee. Suppose that the speaker is allowed to talk no faster than the slowest note taker can write. This would guarantee that everybody gets the full set of notes and that no one misses any information. However, requiring all presentations to go at that slow pace certainly imposes a restriction on our committee, especially if the slow note takers do not need the information. Now, if the chairman knows which presentations are not important to the slow note takers, he can direct them to put away their notes for those presentations. That way, the speaker and the fast note taker(s) can cover more items in less time.

A similar situation may exist on the HP-IB. Suppose that a printer and a flexible disc are connected to the bus. Both devices do not need to listen to all data messages sent through the bus. Also, if all the data transfers must be slow enough for the printer to keep up, saving a program on the disc would take as long as listing the program on the printer. That would certainly not be a very



effective use of the speed of the disc drive if it was the only device to receive the data. Instead, by “unlistening” the printer whenever it does not need to receive a data message, the computer can save a program as fast as the disc can accept it.

During a committee meeting, the current chairman is responsible for telling the committee which member is to be the talker and which is (are) to be the listener(s). Before these assignments are given, he must get the *attention* of all members. The talker and listener(s) are then designated, and the next data message is presented to the listener(s) by the talker. When the talker has finished the message, the designation process may be repeated.

On the HP-IB, the Active Controller takes similar action. When talker and listener(s) are to be designated, the **attention signal line** (ATN) is asserted while the talker and listener(s) are being addressed. ATN is then cleared, signaling that those devices not addressed to listen may ignore all subsequent data messages. Thus, *the ATN line separates data from commands*; commands are accompanied by the ATN line being true, while data messages are sent with the ATN line false.

On the HP-IB, devices are *addressed to talk* and *addressed to listen* in the following orderly manner. The Active Controller first sends a single command which causes all devices to *unlisten*. The talker's address is then sent, followed by the address(es) of the listener(s). After all listeners have been addressed, the data can be sent from the talker to the listener(s). Only device(s) addressed to listen accept any data that is sent through the bus (until the bus is reconfigured by subsequent addressing commands).

The data transfer, or **data message**, allows for the exchange of information between devices on the HP-IB. Our committee conducts business by exchanging ideas and information between the speaker and those listening to his presentation. On the HP-IB, *data is transferred from the active talker to the active listener(s) at a rate determined by the slowest active listener on the bus*. This restriction on the transfer rate is necessary to ensure that no data is lost by any device addressed to listen. The **handshake** used to transfer each data byte ensures that all data output by the talker is received by all active listeners.

### *Examples of Bus Sequences*

Most data transfers through the HP-IB involve a talker and only one listener. For instance, when an OUTPUT statement is used (by the Active Controller) to send data to an HP-IB device, the following sequence of commands and data is sent through the bus.

3

```
OUTPUT 701;"Data"
```

1. The unlisten command is sent.
2. The talker's address is sent (here, the address of the computer; "My Talk Address"), which is also a command.
3. The listener's address (01) is sent, which is also a command.
4. The data bytes "D", "a", "t", "a", CR, and LF are sent; all bytes are sent using the HP-IB's interlocking handshake to ensure that the listener has received each byte.

Similarly, most ENTER statements involve transferring data from a talker to only one listener. For instance, the following ENTER statement invokes the following sequence of commands and data-transfer operations.

```
ENTER 722;Voltage
```

1. The unlisten command is sent.
2. The talker's address (22) is sent, which is a command.
3. The listener's address is sent (here, the computer's address; "My Listen Address"), also a command.
4. The data is sent by device 22 to the computer using the HP-IB handshake.

Bus sequences, hardware signal lines, and more specific HP-IB operations are discussed in the "HP-IB Control Lines" and "Advanced Bus Management" sections.

## Addressing Multiple Listeners

HP-IB allows more than one device to listen simultaneously to data sent through the bus (even though the data may be accepted at differing rates). The following examples show how the Active Controller can address multiple listeners on the bus.

```
100  ASSIGN @Listeners TO 701,702,703
110  OUTPUT @Listeners;String$
120  OUTPUT @Listeners USING Image_1;Array$(*)
```

This capability allows a single OUTPUT statement to send data to several devices simultaneously. It is however, necessary for all the devices to be on the same interface. When the preceding OUTPUT statement is executed, the unlisten command is sent first, followed by the Active Controller's talk address and then listen addresses 01, 02, and 03. Data is then sent by the controller and accepted by devices at addresses 1, 2, and 3.

If an ENTER statement that uses the same I/O path name is executed by the Active Controller, the first device is addressed as the talker (the source of data) and all the rest of the devices, including the Active Controller, are addressed as listeners. The data is then sent from the device at address 01 to the devices at addresses 02 and 03 and to the Active Controller.

```
130  ENTER @Listeners;String$
140  ENTER @Listeners USING Image_2;Array$(*)
```

## Secondary Addressing

Many devices have operating modes which are accessed through the extended addressing capabilities defined in the bus standard. Extended addressing provides for a second address parameter in addition to the primary address. Examples of statements that use extended addressing are as follows.

```
100  ASSIGN @Device TO 72205 ! 22=primary, 05=secondary.
110  OUTPUT @Device;Message$

200  OUTPUT 72205;Message$

150  ASSIGN @Device TO 7220529 ! Additional secondary
160                                     ! address of 29.
170  OUTPUT @Device;Message$

120  OUTPUT 7220529;Message$
```

The range of secondary addresses is 00-31; up to six secondary addresses may be specified (a total of 15 digits including interface select code and primary address). Refer to the device's operating manual for programming information associated with the extended addressing capability. The HP-IB interface also has a mechanism for detecting secondary commands. For further details, see the discussion of interrupts.

---

## General Bus Management

The HP-IB standard provides several mechanisms that allow managing the bus and the devices on the bus. Here is a summary of the statements that invoke these control mechanisms.

ABORT	is used to abruptly terminate all bus activity and reset all devices to power-on states.
CLEAR	is used to set all (or only selected) devices to a pre-defined, device-dependent state.
LOCAL	is used to return all (or selected) devices to local (front-panel) control.

LOCAL LOCKOUT	is used to disable all devices' front-panel controls.
PPOLL	is used to perform a parallel poll on all devices (which are configured and capable of responding).
PPOLL CONFIGURE	is used to setup the parallel poll response of a particular device.
PPOLL UNCONFIGURE	is used to disable the parallel poll response of a device (or all devices on an interface).
REMOTE	is used to put all (or selected) devices into their device-dependent, remote modes.
SEND	is used to manage the bus by sending explicit command or data messages.
SPOLL	is used to perform a serial poll of the specified device (which must be capable of responding).
TRIGGER	is used to send the trigger message to a device (or selected group of devices).

These statements (and functions) are described in the following discussion. However, the actions that a device takes upon receiving each of the above commands are, in general, different for each device. Refer to a particular device's manuals to determine how it will respond. Detailed descriptions of the actual sequence of bus messages invoked by these statements are contained in "Advanced Bus Management" later in this chapter.

## Remote Control of Devices

Most HP-IB devices can be controlled either from the front panel or from the bus. If the device's front-panel controls are currently functional, it is in the Local state. If it is being controlled through the HP-IB, it is in the Remote state. Pressing the front-panel "Local" key will return the device to Local (front-panel) control, unless the device is in the Local Lockout state (described in a subsequent discussion).

The Remote message is automatically sent to all devices whenever the System Controller is powered on, reset, or sends the Abort message. A device also

enters the Remote state automatically whenever it is addressed. The REMOTE statement also outputs the Remote message, which causes all (or specified) devices on the bus to change from local control to remote control. The computer must be the System Controller to execute the REMOTE statement.

### *Examples*

3        REMOTE 7

         ASSIGN @Device TO 700

         REMOTE @Device

         REMOTE 700

## **Locking Out Local Control**

The Local Lockout message effectively locks out the “local” switch present on most HP-IB device front panels, preventing a device’s user from interfering with system operations by pressing buttons and thereby maintaining system integrity. As long as Local Lockout is in effect, no bus device can be returned to local control from its front panel.

The Local Lockout message is sent by executing the LOCAL LOCKOUT statement. This message is sent to all device on the specified HP-IB interface, and it can only be sent by the computer when it is the Active Controller.

### *Examples*

         ASSIGN @Hpib TO 7

         LOCAL LOCKOUT @Hpib

         LOCAL LOCKOUT 7

The Local Lockout message is cleared when the Local message is sent by executing the LOCAL statement. However, executing the ABORT statement does not cancel the Local Lockout message.

## Enabling Local Control

During system operation, it may be necessary for an operator to interact with one or more devices. For instance, an operator might need to work from the front panel to make special tests or to troubleshoot. And, in general, it is good systems practice to return all devices to local control upon conclusion of remote-control operations. Executing the LOCAL statement returns the specified devices to local (front-panel) control. The computer must be the Active Controller to send the LOCAL message.

### *Examples*

```
ASSIGN @HpiB TO 7  
LOCAL @HpiB
```

```
ASSIGN @Device TO 700  
LOCAL @Device
```

If primary addressing is specified, the Go-to-Local message is sent only to the specified device(s). However, if only the interface select code is specified, the Local message is sent to all devices on the specified HP-IB interface and any previous Local Lockout message (which is still in effect) is automatically cleared. The computer must be the System Controller to send the Local message (by specifying only the interface select code).

## Triggering HP-IB Devices

The TRIGGER statement sends a Trigger message to a selected device or group of devices. The purpose of the Trigger message is to initiate some device-dependent action; for example, it can be used to trigger a digital voltmeter to perform its measurement cycle. Because the response of a device to a Trigger Message is strictly device-dependent, neither the Trigger message nor the interface indicates what action is initiated by the device.

### *Examples*

```
ASSIGN @HpiB TO 7  
TRIGGER @HpiB
```

```
ASSIGN @Device TO 707  
TRIGGER @Device
```

Specifying only the interface select code outputs a Trigger message to all devices currently addressed to listen on the bus. Including device addresses in the statement triggers only those devices addressed by the statement. The computer can also respond to a trigger from another controller on the bus. See “Interrupts While Non-Active Controller” for details.

3

## Clearing HP-IB Devices

The CLEAR statement provides a means of “initializing” a device to its predefined, device-dependent state. When the CLEAR statement is executed, the Clear message is sent either to all devices or to the specified device(s), depending on the information contained within the device selector. If only the interface select code is specified, all devices on the specified HP-IB interface are cleared. If primary-address information is specified, the Clear message is sent only to the specified device. Only the Active Controller can send the Clear message.

### Examples

```
ASSIGN @Hpib TO 7  
CLEAR @Hpib
```

```
ASSIGN @Device TO 700  
CLEAR @Device
```

## Aborting Bus Activity

This statement may be used to terminate *all* activity on the bus and return all the HP-IB interfaces of all devices to a reset (or power-on) condition. Whether this affects other modes of the device depends on the device itself. The computer must be either the active or the system controller to perform this function. If the System Controller (which is not the current Active Controller) executes this statement, it regains active control of the bus. *Only the interface select code may be specified*; device selectors which contain primary-addressing information (such as 724) may not be used.

### Examples

```
ASSIGN @Hpib TO 7  
ABORT @Hpib
```

```
ABORT 7
```



## HP-IB Service Requests

Most HP-IB devices, such as voltmeters, frequency counters, and spectrum analyzers, are capable of generating a “service request” when they require the Active Controller to take action. Service requests are generally made after the device has completed a task (such as making a measurement) or when an error condition exists (such as a printer being out of paper). The operating and/or programming manuals for each device describe the device’s capability to request service and conditions under which the device will request service.

To request service, the device sends a Service Request message (SRQ) to the Active Controller. The mechanism by which the Active Controller detects these requests is the SRQ interrupt. Interrupts allow an efficient use of system resources, because the system may be executing a program until interrupted by an event’s occurrence. If enabled, the external event initiates a program branch to a routine which “services” the event (executes remedial action).

Chapter 7 described interrupt events in general. This chapter describes the two types of interrupts that can occur on an HP-IB Interface: SRQ interrupts from external devices (that can occur while the computer is an Active Controller), and interrupts that can occur while the computer is a non-Active Controller. The first type of interrupts are described in this section. The second type are described in the section called “The Computer as a Non-Active Controller.”

### Setting Up and Enabling SRQ Interrupts

In order for an HP-IB device to be able to initiate a service routine in the Active Controller, two prerequisites must be met: the SRQ interrupt event must have a service routine defined, and the SRQ interrupt must be enabled to initiate the branch to the service routine. The following program segment shows an example of setting up and enabling an SRQ interrupt.

```

100 Hpib=7
110 ON INTR Hpib GOSUB Service_routine
120 !
130 Mask=2
140 ENABLE INTR Hpib;Mask

```

The value of the mask in the ENABLE INTR statement determines which type(s) of interrupts are to be enabled. The value of the mask is automatically written into the HP-IB interfaces’s interrupt-enable register (CONTROL register 4) when this statement is executed. Bit 1 is set in the preceding

example, enabling SRQ interrupts to initiate a program branch. Reading STATUS register 4 at this point would return a value of 2.

3

When an SRQ interrupt is generated by any device on the bus, the program branches to the service routine when the current line is exited (either when the line's execution is finished or when the line is exited by a call to a user-defined function). The service routine, in general, must perform the following operations:

- determine which device(s) are requesting service (parallel poll)
- determine what action is requested (serial poll)
- clear the SRQ line
- perform the requested action
- re-enable interrupts
- return to the former task (if applicable)

### Servicing SRQ Interrupts

The SRQ is a level-sensitive interrupt; in other words, if an SRQ is present momentarily but does not remain long enough to be sensed by the computer, an interrupt will not be generated. The level-sensitive nature of the SRQ line also has further implications, which are described in the following paragraphs.

#### *Example*

Assume only one device is currently on the bus. The following service routine first serially polls the device requesting service, thereby clearing the interrupt request. In this case, the computer did not have to determine which device was requesting service because only one device is on the bus. It is also assumed that only service request interrupts have been enabled; therefore, the type of interrupt need not be determined either. The service is then performed, and the SRQ event is re-enabled to generate subsequent interrupts.

```
500  Serv_rtn:  Ser_poll=SPOLL(@Device)
510              !
520              ! Additional service routine code
530              ! can be included here.
540              !
550              ENABLE INTR 7 ! Use previous mask.
560              RETURN
```

The IEEE standard has defined that when an interrupting device is serially polled, it is to stop interrupting until a new condition arises (or the same condition arises again). In order to “clear” the SRQ line, it is necessary to perform a serial poll on the device. This poll is an acknowledgement from the controller to the device that it has seen the request for service and is responding. The device then removes its request for service (by releasing SRQ).

Had the SRQ line not been released, the computer would have branched to the service routine immediately upon re-enabling interrupts on this interface. This is another implication of the level-sensitive nature of the SRQ interrupt.

It is also important to note that once an interrupt is sensed and logged, the interface cannot generate another interrupt until the initial interrupt is serviced. The computer disables all subsequent interrupts from an interface until a pending interrupt is serviced. For this reason, it was necessary to allow for subsequent branching.

## **Polling HP-IB Devices**

The Parallel Poll is the fastest means of gathering device status when several devices are connected to the bus. Each device (with this capability) can be programmed to respond with one bit of status when Parallel Polled, making it possible to obtain the status of several devices in one operation. If a device responds affirmatively (“I need service”) to a Parallel Poll, then more information as to its specific status can be obtained by conducting a Serial Poll of the device.

## **Configuring Parallel Poll Responses**

Certain devices can be remotely programmed by the Active Controller to respond to a Parallel Poll. A device which is currently configured for a Parallel Poll responds to the poll by placing its current status on one of the bus data lines. The logic sense of the response and the data-bit number can be programmed by the PPOLL CONFIGURE statement. No multiple listeners can be specified in the statement; if more than one device is to respond on a single bit, each device must be configured with a separate PPOLL CONFIGURE statement.

### *Example*

```
ASSIGN @Device TO 701
PPOLL CONFIGURE @Device;Configure_code
```

3 The value of `Configure_code` (any numeric expression can be specified) is first rounded to an integer and then used to configure the device's Parallel Poll Response. The least-significant 3 bits (2 thru 0) of the expression are used to determine which data line the device is to respond on (place its status on). Bit 3 specifies the logic sense of the Parallel Poll Response bit of the device. For instance, a value of 0 implies that the device's response is 0 when its Status Bit message is "I need service."

### *Example*

The following statement configures the device at address 01 on the HP-IB interface at select code 7 to respond by placing a 0 on bit 4 (DIO5) when its Status Bit response is affirmative.

```
$PPOLL CONFIGURE 701; 4
```

### **Conducting a Parallel Poll**

The PPOLL function returns a single byte containing up to 8 status bit messages of the devices on the bus (which are capable of responding to the Parallel Poll. Each bit returned by the function corresponds to the status bit of the device(s) configured to respond to the Parallel Poll. (Recall that one or more devices can respond on a single line.) The PPOLL function can only be executed by the Active Controller.

```
Response=PPOLL(7)
```

### **Disabling Parallel Poll Responses**

The PPOLL UNCONFIGURE statement gives the Active Controller the capability of disabling the Parallel Poll responses of one or more devices on the bus.

```
PPOLL UNCONFIGURE 705
```

The following statement disables all devices on the HP-IB interface at select code 8 from responding to a Parallel Poll.

```
PPOLL UNCONFIGURE 8
```

If no primary addressing is specified, all bus devices are disabled from responding to a Parallel Poll. If primary addressing is specified, only the specified devices (which have the Parallel Poll Configure capability) are disabled.

### Conducting a Serial Poll

A sequential poll of individual devices on the bus is known as a Serial Poll. One entire byte of device-specific status is returned in response to a Serial Poll. This byte is called the “Status Byte” message and, depending on the device, may indicate an overload, a request for service, or a printer being out of paper. The particular response of each device depends on the device.

The SPOLL function performs a Serial Poll of the specified device; the computer must currently be the Active Controller in order to execute this function.

#### *Examples*

```
ASSIGN @Device TO 700
Status_byte=SPOLL(700)
```

```
Spoll_724=SPOLL(724)
```

Just as the Parallel Poll is not defined for individual devices, the Serial Poll is meaningless for an interface; therefore, *primary addressing must be used* with the SPOLL function.

### Special Case: Serial Polling a Non-Active Controller

If you wish to perform a serial poll on a non-active BASIC controller that will cause it to generate a Serial Poll Addressed State (SPAS) interrupt, you must follow a specific procedure. You *cannot* use an ordinary SPOLL to obtain this behavior, you must construct a custom HP-IB message using SEND. You must send Serial Poll Enable (SPE) *before* sending the talk address of the non-active BASIC controller. Refer to the SPOLL simulation example in the following section, “Explicit Bus Messages”/“Examples of Sending Commands” for details and example code.

---

## Advanced Bus Management

3

Bus communication involves both sending data to devices and sending commands to devices and the interface itself. “General Structure of the HP-IB” stated that this communication must be made in an orderly fashion and presented a brief sketch of the differences between data and commands. However, most of the bus operations described so far in this chapter involve sequences of commands and/or data which are sent automatically by the computer when HP-IB statements are executed. This section describes both the commands and data sent by HP-IB statements and how to construct your own, custom bus sequences.

### The Message Concept

The main purpose of the bus is to send information between two (or more) devices. These quantities of information sent from talker to listener(s) can be thought of as messages. However, before data can be sent through the bus, it must be properly configured. A sequence of commands is generally sent before the data to inform bus devices which is to send and which is (or are) to listen to the subsequent message(s). These commands can also be thought of as messages.

Most bus messages are transmitted by sending a byte (or sequence of bytes) with numeric values of 0 through 255 through the bus data lines. When the Attention line (ATN) is true, these bytes are considered commands; when ATN is false, they are interpreted as data. Bus command groups and their ASCII characters and codes are shown in “Bus Commands and Codes”.

### Types of Bus Messages

The messages can be classified into twelve types. This computer is capable of implementing all twelve types of interface messages. The following list describes each type of message.

1. A Data message consists of information which is sent from the talker to the listener(s) through the bus data lines.
2. The Trigger message causes the listening device(s) to initiate device-dependent action(s).

3. The Clear message causes either the listening device(s) or all of the devices on the bus to return to their device-dependent “clear” states.
4. The Remote message causes listening devices to change to remote program control when addressed to listen.
5. The Local message clears the Remote message from the listening device(s) and returns the device(s) to local front-panel control.
6. The Local Lockout message disables a device’s front-panel controls, preventing a device’s operator from manually interfering with remote program control.
7. The Clear Lockout/Local message causes all devices on the bus to be removed from Local Lockout and to revert to the Local state. This message also clears the Remote message from all devices on the bus.
8. The Service Request message can be sent by a device at any time to signify that the device needs to interact with the the Active Controller. This message is cleared by sending the device’s Status Byte message, if the device no longer requires service.
9. A Status Byte message is a byte that represents the status of a single device on the bus. This byte is sent in response to a serial poll performed by the Active Controller. Bit 6 indicates whether the device is sending the Service Request message, and the remaining bits indicate other operational conditions of the device.
10. A Status Bit message is a single bit of device-dependent status. Since more than one device can respond on the same line, this Status Bit may be logically combined and/or concatenated with Status Bit messages from many devices. Status Bit messages are returned in response to a Parallel Poll conducted by the Active Controller.
11. The Pass Control message transfers the bus management responsibilities from the Active Controller to another controller.
12. The Abort message is sent by the System Controller to assume control of the bus unconditionally from the Active Controller. This message terminates all bus communications, but is not the same as the Clear message.

These messages represent the full implementation of all HP-IB system capabilities; all of these messages can be sent by this computer. However, each device in a system may be designed to use only the messages that are applicable to its purpose in the system. It is important for you to be aware of the HP-IB functions implemented on each device in your HP-IB system to ensure its operational compatibility with your system.

3

### **Bus Commands and Codes**

The following table shows the decimal values of IEEE-488 command messages. Remember that *ATN is true* during all of these commands. Notice also that these commands are separated into four general categories: Primary Command Group, Listen Address Group, Talk Address Group, and Secondary Command Group. Subsequent discussions further describe these commands.



### HP-IB Commands and Codes

Decimal Value	ASCII Character	Interface Message	Description
		<i>PCG</i>	<i>Primary Command Group</i>
1	SOH	GTL	Go to Local
4	EOT	SDC	Selected Device Clear
5	ENQ	PPC	Parallel Poll Configure
8	BS	GET	Group Execute Trigger
9	HT	TCT	Take Control
17	DC1	LLO	Local Lockout
20	DC4	DCI	Device Clear
21	NAK	PPU	Parallel Poll Unconfigure
24	CAN	SPE	Serial Poll Enable
25	EM	SPD	Serial Poll Disable
		<i>LAG</i>	<i>Listen Address Group</i>
32-62	Space through > (Numbers/special chars.)		Listen Addresses 0-30
63	?	UNL	Unlisten
		<i>TAG</i>	<i>Talk Address Group</i>
64-94	@ through ↑ (Uppercase letters)		Talk Addresses 0-30
95	—(underscore)	UNT	Untalk
		<i>SCG</i>	<i>Secondary Command Group</i>
96-126	‘ through ~ (Lowercase letters)		Secondary Commands 0-30
127	DEL		Ignored

## Address Commands and Codes

The following table shows the ASCII characters and corresponding codes of the Listen Address Group and Talk Address Group commands. The next section describes how to send these commands.

3

**HP-IB Listen and Talk Address Commands**

Listen Address Character	Talk Address Character	Address Code	Address Switch Settings
Space	@	0	0 0 0 0 0
!	A	1	0 0 0 0 1
"	B	2	0 0 0 1 0
#	C	3	0 0 0 1 1
\$	D	4	0 0 1 0 0
%	E	5	0 0 1 0 1
&	F	6	0 0 1 1 0
'	G	7	0 0 1 1 1
(	H	8	0 1 0 0 0
)	I	9	0 1 0 0 1
*	J	10	0 1 0 1 0
+	K	11	0 1 0 1 1
,	L	12	0 1 1 0 0
-	M	13	0 1 1 0 1
.	N	14	0 1 1 1 0
/	O	15	0 1 1 1 1
0	P	16	1 0 0 0 0
1	Q	17	1 0 0 0 1
2	R	18	1 0 0 1 0

### HP-IB Listen and Talk Address Commands (continued)

Listen Address Character	Talk Address Character	Address Code	Address Switch Settings
3	S	19	1 0 0 1 1
4	T	20	1 0 1 0 0
5	U	21	1 0 1 0 1
6	V	22	1 0 1 1 0
7	W	23	1 0 1 1 1
8	X	24	1 1 0 0 0
9	Y	25	1 1 0 0 1
:	Z	26	1 1 0 1 0
;	[	27	1 1 0 1 1
<	/	28	1 1 1 0 0
=	]	29	1 1 1 0 1
>	^	30	1 1 1 1 0

The preceding table implicitly shows that:

- Listen address commands can be calculated from the primary address by using one of the following equations:

$$\text{Listen\_address} = 32 + \text{Primary\_address}$$

OR

$$\text{Listen\_address} = \text{CHR}\$(32 + \text{Primary\_address})$$

- Similarly, talk address commands can be calculated from the primary address by using one of the following equations

$$\text{Talk\_address} = 64 + \text{Primary\_address}$$

OR

$$\text{Talk\_address} = \text{CHR}\$(64 + \text{Primary\_address})$$

However, the table does not show that:

- the Unlisten command is "?", CHR\$(63)
- the Untalk command is "\_", CHR\$(95)
- therefore, primary address 31 is an unusable device address, but can be used to send the Unlisten and Untalk commands.

## Explicit Bus Messages

It is often desirable (or necessary) to manage the bus by sending explicit sequences of bus messages. The SEND statement is the vehicle by which explicit commands and data can be sent through the bus. The SEND statement is also a method of sending data with odd parity through the bus (instead of using the PARITY attribute discussed in the "I/O Path Attributes" chapter). This section shows several uses of this statement.

### *Examples of Sending Commands*

As a simple example, suppose the following statement is executed by the Active Controller to configure the bus (i.e., to address the talker and listener).

```
OUTPUT 701 USING "#,K"
```

The SEND statement can be used to send the same sequence of commands, as shown in the following statement.

```
SEND 7;CMD "?U!"
```

This statement configures the bus explicitly by sending the following commands:

- the unlisten command (ASCII character “?”; decimal code 63)
- talk address 21 (ASCII character “U”; decimal code 85)
- listen address 1 (ASCII character “!”; decimal code 33)

The same sequence of commands and data is sent with any of the following statements.

```
SEND 7;CMD UNL MTA LISTEN 1
```

```
SEND 7;CMD UNL TALK 21 LISTEN 1
```

```
SEND 7;CMD 32+31,64+21,32+1
```

Commands can be sent by specifying the secondary keyword CMD. The list of commands (following CMD) can be any numeric or string expressions. If more than one expression is listed, they must be separated by commas. A numeric expression will be evaluated, rounded to an integer (MOD 256), and sent as one byte. Each character of a string expression will be sent individually. *All bytes are sent with ATN true.* The computer must be the current Active Controller to send commands.

```
SEND Isc;CMD 8                ! Group Execute Trigger
SEND Isc;TALK New_controller CMD 9 ! Pass Control
SEND 8;CMD 1                  ! Go to Local
```

If SEC is used, the specified secondary commands will be sent. An extended talker may be addressed by using SEC after the talk address; extended listener(s) may be addressed by using SEC after the listen address(es).

```
SEND 7;MTA UNL LISTEN 1 CMD 5 SEC 16 ! SEND PPD.
```

The computer must be the Active Controller to send CMD, LISTEN, UNL, MLA, TALK, UNT, MTA, and SEC. If a non-Active Controller attempts to send any of these messages, an error is reported.

Simulate the following S POLL function with SEND and ENTER statements.

```
A=SPOLL(724)
```

When an S POLL is performed, the resulting bus activity is:

- Unlisten command
- 3 ■ My Listen Address (the computer's listen address; MLA)
- device's talk address (one of the TAG commands)
- Serial Poll Enable command (SPE; decimal code 24)
- one data byte is read (the Status Byte message)
- Serial Poll Disable (SPD; decimal code 25)
- Untalk command

This is accomplished by either of the following sequences:

```
SEND 7;CMD "?5"&CHR$(24)&"X" ! Configure the bus; send SPE.  
ENTER 7 USING "#,B";A ! Read Status Byte.  
SEND 7;CMD CHR$(25)&"_" ! Send SPD and Untalk.
```

```
SEND 7;UNL MLA CMD 24 TALK 24 ! Configure the bus; send SPE.  
ENTER 7 USING "#,B";A ! Read Status Byte.  
SEND 7;CMD 25 UNT ! Send SPD and Untalk.
```

The preceding secondary keywords provide the capability of sending various command messages through the bus. The activity that results on the bus when several other high-level commands are issued is summarized in "HP-IB Message Mnemonics".

### *Examples of Sending Data*

Data messages can be sent by specifying the secondary keyword DATA. If the computer is the Active Controller, the data is sent immediately. However, if the computer is not the Active Controller, it waits to be addressed to talk before sending the data.

```
SEND 7;DATA "Message",13,10 ! Send with CR/LF.
```

```
SEND Bus;DATA "Data" END ! Send with EOI.
```

The data list may contain any mixture of numeric or string expressions; if more than one expression is specified, they must be separated by commas.

Each numeric expression is evaluated as an integer (MOD 256) and sent as a single byte. Each string item is evaluated and all resultant characters are sent serially. Each byte is *sent with ATN false* (sent as a data message). The last expression may be followed by the secondary keyword END, which causes the EOI terminator to be sent concurrently with the last data byte.

As another example, simulate this ENTER statement with a SEND statement.

```
ENTER 724;Number,String$
```

Any of the following pairs of statements can be used to accomplish the same operation.

```
SEND 7;UNL TALK 24 MLA
ENTER 7;Number,String$
```

```
SEND 7;UNL TALK 24 LISTEN 21
ENTER 7;Number,String$
```

```
SEND 7;CMD "?X5"
ENTER 7;Number, String$
```

## HP-IB Message Mnemonics

This section contains the descriptions of several bus messages described by the IEEE 488-1978 standard. The following table describes message mnemonics, their meanings, and the secondary keywords used with the SEND statement. The HP-IB messages that require primary keywords are noted in the table.

All BASIC statements which send HP-IB messages (except SEND) always set ATN-true (command) messages with the most-significant bit set to zero. Using CMD (with SEND) allows you to send ATN-true messages with the most-significant bit set to one. This may be useful for non-standard IEEE-488 devices which require the most-significant bit to have a particular value.

The CMD and DATA secondary keywords of SEND statements allow string expressions as well as numeric expressions (e.g., CMD "?" is the same as CMD 63). All other secondary keywords which need data require *numeric* expressions. Keep this in mind while reading through this table.

### HP-IB Messages and Mnemonics

Message Mnemonic	Message Description	SEND Clause Required (numeric values are decimal)
DAB	Data Byte	DATA 0 through 255
DCL	Device Clear	CMD 20 (or 148)
EOI	End or Identify	DATA data list END
GET	Group Execute Trigger	CMD 8 (or 136)
GTL	Go To Local	CMD 1 (or 129)
IFC	Interface Clear	Not possible with SEND; use the ABORT statement.
LAG	Listen Address	LISTEN 0 through 30; or CMD 32 through 62; or CMD 160 through 190
LLO	Local Lockout	CMD 17
MLA	My Listen Address	MLA
MTA	My Talk Address	MTA
PPC	Parallel Poll Configure	CMD 5 (or 133)



### HP-IB Messages and Mnemonics (continued)

Message Mnemonic	Message Description	SEND Clause Required (numeric values are decimal)
PPD	Parallel Poll Disable	SEC 16; or CMD 112 (or 240) (Must be preceded by PPC.)
PPE	Parallel Poll Enable	SEC 0+Mask: SEC 0 through 15; or CMD 96 through 111; or CMD 224 through 239 (Must be preceded by PPC.)
PPU	Parallel Poll Unconfig.	CMD 21 (or 149)
PPOLL	Parallel Poll	Not possible with SEND; use the PPOLL function.
REN	Remote Enable	Not possible with SEND; use the REMOTE statement.
SDC	Selected Device Clear	CMD 4 (or 132)
SPD	Serial Poll Disable	CMD 25 (or 153)
SPE	Serial Poll Enable	CMD 24 (or 152)
TAD	Talk Address	TALK 0 through 30; or CMD 64 through 94; or CMD 192 through 222

### HP-IB Messages and Mnemonics (continued)

Message Mnemonic	Message Description	SEND Clause Required (numeric values are decimal)
TCT	Take Control	CMD 9 (or 137)
UNL	Unlisten	UNL; or LISTEN 31; or CMD 63 (or 191)
UNT	Untalk	UNT; or TALK 31; or CMD 95 (or 223)

3

---

## The Computer As a Non-Active Controller

The section called “General Structure of the HP-IB” described how communications take place through HP-IB Interfaces. The functions of the System Controller and Active Controller were likened to a “committee chairman” and “acting chairman,” respectively, and the functions of each were described. This section describes how the Active Controller may “pass control” to another controller and assume the role of a non-Active Controller. This action is analogous to designating another committee member to take the responsibility of acting chairman and then becoming a committee member who listens to the acting chairman and speaks when given the floor. The following topics will be discussed:

- Determining whether the computer is currently the Active Controller and/or System Controller
- Determining the computer’s HP-IB primary address, and changing it, if necessary
- Passing control to another HP-IB controller
- Requesting service from the Active Controller
- Responsibilities of being a non-Active Controller
- Responding to interrupts that occur while non-Active Controller

## Determining Controller Status and Address

It is often necessary to determine if an interface is the System Controller and to determine whether or not it is the current Active Controller. It is also often necessary to determine or change the interface's primary address. The example program shown in the beginning of this chapter interrogated interface STATUS registers and printed the resultant System-Controller status and primary address. Those operations are explained in the following paragraphs.

### *Example*

Executing the following statement reads STATUS register 3 (of the internal HP-IB) and places the current value into the variable `Stat_and_addr`. Remember that if the statement is executed from the keyboard, the variable `Stat_and_addr` must be defined in the current context.

```
STATUS 7,3;Stat_and_addr
```

*STATUS Register 3*      Controller Status and Address

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary address of HP-IB interface				
value=128	value=64	value=0	value=16	value=8	value=4	value=2	value=1

If *bit 7* is set (1), it signifies that the interface is the System Controller; if clear (0), it is not the System Controller. Only one controller on each HP-IB interface should be configured as the System Controller.

If *bit 6* is set (1), it signifies that the interface is currently the Active Controller; if it is clear (0), another controller is currently the Active Controller.

*Bits 4 through 0* represent the current value of the interface's primary address, which is in the range of 0 through 30. The power-on default value for the internal HP-IB is 21 (if it is the System Controller) and 20 (if not the System Controller). For external HP-IB interfaces, the default address is set to 21 at the factory but may be changed by setting the address switches on the card itself.

### *Example*

Calculate the primary address of the interface from the value previously read from STATUS register 3.

```
Intf_addr=Stat_and_addr MOD 32
```

3

This numerical value corresponds to the talk (or listen) address sent by the computer when an OUTPUT (or ENTER) statement containing primary-address information is executed. Talk and listen addresses are further described in “Advanced Bus Management”.

## **Changing the Controller's Address**

It is possible to use the CONTROL statement to change an HP-IB interface's address.

### *Example*

```
CONTROL 7,3;Intf_addr
```

The value of Intf\_addr is used to set the address of the HP-IB interface (in this case, the internal HP-IB). The valid range of addresses is 0 through 30; *address 31 is not used*. Thus, if a value greater than 30 is specified, the value MOD 32 is used (for example: 32 MOD 32 equals 0, 33 MOD 32 equals 1, 62 MOD 32 equals 30, and so forth).

## **Passing Control**

The current Active Controller can pass this capability to another computer by sending the Take Control message (TCT). The Active Controller must first address the prospective new Active Controller to talk, after which the TCT message is sent. If the other controller accepts the message, it then assumes the role of Active Controller; this computer then assumes the role of a non-Active Controller.

Passing control can be accomplished in one of two ways: it can be handled by the system, or it can be handled by the program. To handle it programmatically, use the PASS CONTROL statement. For example, the following statements first define the HP-IB Interface's select code and new Active Controller's primary address and then pass control to that controller.

```
100 Hp_ib=7
110 New_ac_addr=20
120 PASS CONTROL 100*Hp_ib+New_ac_addr
```

The following statements perform the same functions.

```
100 Hp_ib=7
110 New_ac_addr=20
120 SEND Hp_ib;UNL TALK New_ac_addr CMD 9
```

Once the new Active Controller has accepted the TCT command, the controller passing control assumes the role of a non-Active Controller (or "HP-IB device") on the specified HP-IB Interface. The next section describes the responsibilities of the computer while it is a non-Active Controller.

### Restrictions to Passing Control with BASIC/UX

On BASIC/UX if the HP-IB device contains swap space or a mounted file system, you cannot pass control to that device.

### Interrupts While Non-Active Controller

When the computer is not an Active Controller, it must be able to detect and respond to many types of bus messages and events.

The computer (as a non-Active Controller) needs to keep track of the following information.

- It must keep track of itself being addressed as a listener so that it can enter data from the current active talker.
- It must keep track of itself being addressed as a talker so that it can transmit the information desired by the active controller.
- It must keep track of being sent a Clear, Trigger, Local, or Local Lockout message so that it can take appropriate action.
- It must keep track of control being passed from another controller.

One way to do this is to continually monitor the HP-IB interface by executing the STATUS statement and then taking action when the values returned match the values desired. This is obviously a great waste of computer time if the computer could be performing other tasks. Instead, the interface hardware can be enabled to monitor bus activity and then generate interrupts when certain events take place.

The computer has the ability to keep track of the occurrences of all of the preceding events. In fact, it can monitor up to 16 different interrupt conditions. STATUS registers 4, 5 and 6 provide access to the interface state and interrupt information necessary to design very powerful systems with a great degree of flexibility.

Each individual bit of STATUS register 4 corresponds to the same bit of STATUS register 5. Register 4 provides information as to which condition *caused* an interrupt, while register 5 keeps track of which interrupt conditions are *currently enabled*. To enable a combination of conditions, add the decimal values for each bit that you want set in the interrupt-enable register. This total is then used as the mask parameter in an ENABLE INTR statement.

Note that non-active controller interrupts that occur during an ENTER or OUTPUT operation are lost.

*STATUS Register 5*      Interrupt Enable Mask

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel poll configuration change	My Talk address received	My Listen address received	EOI received	SPAS	Remote/local change	Talker/listener address change
value= -32 768	value= 16 384	value= 8 192	value= 4 096	value= 2 048	value= 1 024	value= 512	value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger received	Handshake error	Unrecognized universal command	Secondary command while addressed	Clear received	Unrecognized addressed command	SRQ received	IFC received
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 15* enables an interrupt upon becoming the Active Controller. The computer then has the ability to manage bus activities.

*Bit 14* enables an interrupt upon detecting a change in Parallel Poll Configuration. This condition requires accepting data from the bus and then explicitly releasing the bus. Refer to the “Advanced Bus Management” section for further details.

*Bit 13* enables an interrupt upon being addressed as an active talker by the Active Controller.

*Bit 12* enables an interrupt upon being addressed as an active listener by the Active Controller.

*Bit 11* enables an interrupt when an EOI is received during an ENTER operation (the EOI signal line is also described in “HP-IB Control Lines”).

*Bit 10* enables an interrupt when the Active Controller performs a Serial Poll on the computer (in response to its service request).

*Bit 9* enables an interrupt upon receiving either the Remote or the Local message from the active controller, if addressed to listen. The action taken by the computer is, of course, dependent on the user-programmed service routine.

*Bit 8* enables an interrupt upon a change in talk or listen address. An interrupt will be generated if the computer is addressed to listen or talk or "idled" by an Unlisten or Untalk command.

*Bit 7* enables an interrupt upon receiving a Trigger message, if the computer is currently addressed to listen. This interrupt can be used in situations where the computer may be "armed and waiting" to initiate action; the active controller sends the Trigger message to the computer to cause it to begin its task.

*Bit 6* enables an interrupt if a bus error occurs during an OUTPUT statement. Particularly, the error occurs if none of the devices on the bus respond to the HP-IB's interlocking handshake (see "HP-IB Control Lines"). The error typically indicates that either a device is not connected or that its power is off.

*Bit 5* enables an interrupt upon receiving an unrecognized Universal Command. This interrupt condition provides the computer with the capability of responding to new definitions that may be adopted by the IEEE standards committee. This condition also requires accepting data from the bus and then explicitly releasing the bus. Refer to the "Advanced Bus Management" section for further details.

*Bit 4* enables an interrupt upon receiving a Secondary Command (extended addressing) after the interface receives either its primary talk address or primary listen address. Again, this interrupt provides the computer with a way to detect and respond to special messages from another controller. This condition requires accepting data from the bus and then explicitly releasing the bus. Refer to the "Advanced Bus Management" section for further details.

*Bit 3* enables an interrupt on receiving a Clear message. Reception of either a Device Clear message (to all devices) or a Selected Device Clear message (addressed to the computer) will cause this type of interrupt. The computer is free to take any "device-dependent" action; such as, setting up all default values again, or even restarting the program, if that is defined by the programmer to be the "cleared" state of the machine.



*Bit 2* enables an interrupt upon receiving an unrecognized Addressed Command, if the computer is currently addressed to listen. This interrupt is used to intercept and respond to bus commands which are not defined by the standard. This condition requires accepting data from the bus and then explicitly releasing the bus. Refer to the "Advanced Bus Management" section for further details.

*Bit 1* enables an interrupt upon detecting a Service Request.

*Bit 0* enables an interrupt upon detecting an Interface Clear (IFC). The interrupt is generated only when the computer is not the System Controller, as only a System Controller is allowed to set the Interface Clear signal line. The service routine typically is used to recover from the abrupt termination of an I/O operation caused by another controller sending the IFC message.

Note that most of the conditions are state- or event-sensitive; the exception is the SRQ event, which is level-sensitive. State- or event-sensitive events can never go unnoticed by the computer as can service requests; the event's occurrence is "remembered" by the computer until serviced.

For instance, if the computer is enabled to generate an interrupt on becoming addressed as a talker, it would interrupt the first time it received its own talk address. After having responded to the service request (most likely with some sort of OUTPUT operation), it would not generate another interrupt, even if it was still left assigned as a talker by the Active Controller. Thus, it would not generate another interrupt until the event occurred a second time.

An oversimplified example of a service routine that is to respond to multiple conditions might be as follows.

```
100  ON INTR Hpib GOSUB Service
110  Mask=INT(2^13)+INT(2^12)
120  ENABLE INTR Hpib;Mask ! Interrupt on receiving
130                                ! talk or listen addr.
140 Idle: GOTO Idle
150      !
160 Service: STATUS Hpib,4;Status,Mask
170      IF BIT(Status,13) THEN Talker
180      IF BIT(Status,12) THEN Listener
190      RETURN! Ignore other interrupts.
200 Talker: ! Take action for talker.
210      GOTO Exit_point
220      !
230 Listener: ! Take action for listener.
240      !
250 Exit_point: ENABLE INTR Hpib;Mask
260      RETURN
270  END
```

Register 4, the interrupt status register, is a “read-destructive” register; reading the register with a STATUS statement returns its contents and then *clears the register* (to a value of 0). If the service routine’s action depends on the contents of STATUS register 4, the variable in which it is stored must not be used for any other purposes before all of the information that it contains has been used by the service routine.

The computer is automatically addressed to talk (by the Active Controller) whenever it is Serially Polled. If interrupts are concurrently enabled for My Address Change and/or Talker Active, the ON INTR branch will be initiated due to the reception of the computer’s talk address. However, since the Serial Poll is automatically finished with the Untalk Command, the computer may no longer be addressed to talk by the time the interrupt service routine begins execution. See “Responding to Serial Polls” for further details.

## Addressing a Non-Active Controller

The bus standard states that *a non-Active Controller cannot perform any bus addressing*. When *only the interface select code* is specified in an ENTER or OUTPUT statement that uses an HP-IB interface, *no bus addressing is performed*.

If the computer currently is *not the Active Controller*, it can still act as either talker or listener, provided it has been *previously addressed* as such. Thus, if an ENTER or OUTPUT statement is executed while the computer is not an Active Controller, the computer first determines whether or not it is an active talker or listener. If not addressed to talk or listen, the computer waits until it is properly addressed and then finishes executing the statement. It relies on the Active Controller (another computer or device) to perform the bus addressing, and then simply participates as a device in the exchange of the data. Example statements which send and receive data while the computer is not an Active Controller are as follows.

```

100 OUTPUT 7;"Data" ! If not talker, then wait until
110                ! addressed as talker to send data.

200 ENTER 7;Data$  ! If not listener, then wait until
210                ! addressed as listener to accept data.

```

If the computer is the *Active Controller*, it proceeds with the data transfer without addressing which devices are talker and listener(s). However, if the bus has not been configured previously, an error is reported (Error 170 I/O operation not allowed). The following program does not require the "overhead" of addressing talker and listeners each time the OUTPUT statement in the FOR..NEXT loop is executed, because the bus is not reconfigured each time.

```

100 OUTPUT 701 USING "#,K" ! Configure the bus:
110                        ! This interface = talker, and
120                        ! printer (701) = listener.
130                        !
140 FOR Iteration=1 TO 25
150     OUTPUT 7;"Data message"
160 NEXT Iteration
170 !
180 END

```

This type of HP-IB addressing should be used with the understanding that if an event initiates a branch between the time that the initial addressing was made (line 100) and the time that any of the OUTPUT statements are executed (line 150), the event's service routine may reconfigure the bus differently than the initial configuration. If so, the data will be directed to the device(s) addressed to listen by the last I/O statement executed in the service routine. Events may need to be disabled if this method of addressing is used.

In general, most applications do not require this type of bus-overhead minimization; the computer's I/O language has already been optimized to provide excellent performance. Advanced methods of explicit bus management will be described in the section called "Advanced Bus Management".

---

### Note



This type of HP-IB addressing is not allowed for TRANSFER in BASIC/UX. If only a select code is specified, and that select code is Active Controller, an error is reported (Error 170 I/O operation not allowed).

---

## Requesting Service

When the computer is a non-Active Controller, it has the capability of sending an SRQ to the current Active Controller. The following statement is an example of requesting service from the Active Controller of the HP-IB Interface on select code 7.

```
CONTROL 7,1;64
```

The REQUEST statement can be used to perform the same function.

```
REQUEST 7;64
```

Both of the preceding example place a logic True on the SRQ line. (Note that the line may already be set True by another device.) Other bits may be set in the Status Byte message, indicating that other device-dependent conditions exist.

The SRQ line is held True until the Active Controller executes a Serial Poll or this computer executes a REQUEST with bit 6 equal to 0. (Note also that the line may still be held True by another device.)

When the Active Controller detects an SRQ message, it usually polls device(s) on the bus to determine which need(s) service and what kind of service is needed. To determine *which* device(s) are requesting service, the Active Controller conducts a Parallel Poll. If there are not more than one device currently capable of requesting service, the Parallel Poll is not necessary.

The Parallel Poll is conducted by sending an Identify (ATN & EOI). This non-Active Controller's response to a Parallel Poll performed by the Active Controller depends on the current Parallel Poll Response set up for this controller. Setting up this controller's Parallel Poll Response is described in the next section.

If the Active Controller needs to determine what service action is required for a particular device, it performs a Serial Poll on the device(s) that responded to the Parallel Poll with an "I need service." As each device is Serially Polled, it responds by placing its Status Byte on the bus.

This non-Active Controller's response to a Serial Poll performed by the Active Controller is handled automatically by the system. The Status Byte is the byte sent to the Serial Poll Response Byte Register (with CONTROL or REQUEST, as shown above). A subsequent section further describes this non-Active Controller's responses to Serial Polls.

## Responding to Parallel Polls

Before performing a Parallel Poll of bus devices, the Active Controller configures selected device(s) to respond on one of the eight data lines. Each device is directed to respond on a particular data line with a logic True or False; the logic sense of the response informs the Active Controller either "I do need service" or "I don't need service." The logic sense of the response is also specified by the Active Controller. This response to the Parallel Poll is known as the Status Bit message.

After the desired devices have been told how to respond, the Active Controller can send the Identify message and read the Status Bits placed on the data lines to determine which device(s) need service. Identify is sent by placing ATN and EOI in the logic True state. All devices which are currently configured for the poll respond as configured.

To configure its own Parallel Poll Response, the computer must receive a Parallel Poll Configure (PPC) command followed by a Parallel Poll Enable

(PPE) command from the Active Controller. Receiving this “Parallel Poll Configuration Change” generates an interrupt (this type of interrupt is enabled by setting bit 14 of the Interrupt Enable Register). The service routine takes care of configuring this controller’s response by first accepting the encoded “configure byte” (the PPE command from the Active Controller) and then setting up a corresponding response.

The desired Status Bit message can be configured and sent by one of two methods. The first, and simplest, method is to define an automatic response by using the PPOLL RESPONSE statement. With this method, the computer reads the configure byte from the data lines (HP-IB STATUS Register 7) and then writes the byte’s numeric value into HP-IB CONTROL Register 5. The following statements show an example of configuring this controller’s Parallel Poll Response.

```

100 STATUS 7,7;Configure_code
110 CONTROL 7,5;Configure_code
120 I_need_service=0
130 PPOLL RESPONSE 7;I_need_service

```

When the computer receives a subsequent Identify from the Active Controller, the specified response (“I do/don’t need service”) is automatically sent to the Active Controller. The computer will probably need to respond to a Serial Poll, which is described in the next section.

The second method requires that the service routine decode the configure byte and set up the corresponding response. The configure byte read from HP-IB STATUS Register 7 contains 5 bits of data encoded with the following information:

*CONTROL Register 5 Parallel Poll Response Mask*

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used			Uncon- figure	Logic sense	Data bit used for response		
value=128	value=64	value=0	value=16	value=8	value=4	value=2	value=1

*Bit 4* determines whether a response will or will not be configured. A 1 tells this controller *not* to configure a response, and a 0 tells the controller to configure a response.

*Bit 3* determines the logic sense of the Status Bit. If this bit is 0, then the “I need service” message is a 0; if this bit is 1, the “I need service” message is 1.

*Bits 2 through 0* determine the data line on which the Status Bit is to be placed. For instance, if these bits are “000”, then the Status Bit is to be placed on DIO1. If these bits are “111”, then the response is to be placed on DIO8.

The service routine calculates the desired response and places the appropriate bit pattern in HP-IB CONTROL Register 2. For instance, if the configure byte has a value of 12 (positive-true logic on DIO5 for “I need service”), the value sent to CONTROL Register 2 is 16 for “I need service.” The appropriate statement might be:

```
CONTROL 7,2;16
```

When the Identify is received from the Active Controller, the specified response is made automatically.

As another example, suppose that the configure byte has a value of 7. The Status Bit to be written into DIO8 would be a 0 for “I need service.” The corresponding statement might be:

```
CONTROL 7,2;0
```

The following general routine calculates the value to be sent to CONTROL Register 2:

```
790 STATUS 7,7;Config_code ! Read data lines.
800 Config_code=Config_code MOD 256 ! Strip 8 MSBs.
810 Unconfig=BIT(Config_code,4)
820 Sense=BIT(Config_code,3)
830 IF Unconfig=1 OR Sense=0 THEN ! Unconfigure.
840   Ppoll_response=0
850 ELSE ! Configure.
860   Status_bit=Config_code MOD 8 ! Get bits 2-0.
870   Ppoll_response=2^Status_bit ! Set proper bit.
880 END IF
890 CONTROL 7,2;Ppoll_response
```

## Responding to Serial Polls

As a non-Active Controller, the response to Serial Polls is automatically handled by the system. The desired Serial Poll Response Byte is sent to HP-IB CONTROL Register 1. If bit 6 is set (bit 6 has a value of 64), an SRQ is indicated from this controller. All other bits can be considered to be “device-dependent,” and can thus be set according to the program’s needs.

The following statement sets up a response with SRQ and bits 1 and 0 set to 1’s.

```
CONTROL 7,1;64+2+1
```

When the Active Controller performs a Serial Poll on this non-Active Controller, the specified byte is automatically sent to the Active Controller by the system.

This non-Active Controller is automatically addressed to talk by the Active Controller during a Serial Poll. If interrupts are concurrently enabled for My Address Change and/or Talker Active interrupts, the ON INTR branch will be initiated due to the reception of this controller’s talk address. However, since the Serial Poll Response is terminated with the Untalk command, this controller may no longer be addressed to talk when the service routine begins its execution. In such a case, the SPAS interrupt (if enabled) will also be indicated. If desired, the interrupt may be ignored.

## Interface-State Information

It is often necessary to determine which state the interface is in. STATUS register 6 contains interface-state information in its upper byte; it also contains the same information as STATUS register 3 in its lower byte. In advanced applications, it may be necessary to detect and act on the interface’s current state. Register 6’s definition is shown below.



*STATUS Register 6*      Interface State Information

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
REM	LLO	ATN true	LPAS	TPAS	LADS	TADS	*
value= -32 768	value= 16 384	value= 8 192	value= 4 096	value= 2 048	value= 1 024	value= 512	value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary address of HP-IB interface				
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

\* Least-significant bit of last address recognized.

*Bit 15* set indicates that the interface is in the Remote state.

*Bit 14* set indicates that the interface is in the Local Lockout state.

*Bit 13* set indicates that the ATN line is currently set (true).

*Bit 12* set indicates that the interface is in the Listener Primary Addressed State (has received its primary listen address).

*Bit 11* set indicates that the interface is in the Talker Primary Addressed State (has received its primary talk address).

*Bit 10* set indicates that the interface is in the Listener Addressed State and is currently an active listener. If Bit 4 of the Interrupt Enable register is set (Secondary Command While Addressed), two additional conditions are required to enter this state: the interface must have first received its own primary address followed by a secondary command, and it must have *accepted* the secondary command (by writing a non-zero value to CONTROL register 4 to release the NDAC Holdoff).

*Bit 9* set indicates that the interface is in the Talker Addressed State and is currently an active talker. This state is entered in a manner analogous to the Listener Addressed State (see Bit 10 above).

*Bit 8* contains the least-significant bit of the last address recognized by this interface.

*Bits 7 through 0* have the same definitions as STATUS register 3.

3

### Servicing Interrupts that Require Data Transfers

During the discussion on interrupts, three special types of interrupt conditions were described (which are enabled by setting bits in CONTROL register 4). These interrupts occur upon receiving: an unrecognized Universal Command, an unrecognized Addressed Command, or a Secondary Command. These situations all require the computer to read a byte of information from the bus and respond as desired by the programmer.

*STATUS Register 4*    Interrupt Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel poll configuration change	My Talk address received	My Listen address received	EOI received	SPAS	Remote/local change	Talker/listener address change
value= -32 768	value= 16 384	value= 8 192	value= 4 096	value= 2 048	value= 1 024	value= 512	value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger received	Handshake error	Unrecognized universal command	Secondary command while addressed	Clear received	Unrecognized addressed command	SRQ received	IFC received
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

As a reminder, these interrupt conditions occur under the following circumstances.

*Bit 14* enables an interrupt on any change in Parallel Poll configuration. If a Parallel Poll Configure command is received, the computer must set up its own Parallel Poll Response designated by the Active Controller. The response itself is set up by writing to CONTROL register 2 of the HP-IB interface.

*Bit 5* enables an interrupt upon receiving an unrecognized Universal Command. This interrupt condition provides the computer with the ability to respond to new definitions that may be adopted by the IEEE standards committee.

*Bit 4* enables an interrupt upon receiving a Secondary Command, if addressed to either talk or listen during the command mode. Again, this allows the computer to detect and respond to special information from another controller.

*Bit 2* enables an interrupt upon receiving an unrecognized Addressed Command, if addressed to listen. This interrupt is used to detect and respond to commands that are undefined by the standard (but which may be recognized by the computer).

Whenever any of the above interrupt conditions are enabled and occur, the computer logs the interrupt and then sets a **bus holdoff**. In other words, all bus activity is “frozen” until the program has released this holdoff. The holdoff is established to allow the program time to determine the current state of the bus.

The bus state is determined by reading HP-IB STATUS register 7, which returns the current logic state of the data and control lines as a 16-bit integer.

STATUS 7,7;Bus\_lines

After reading the state of the lines, it is necessary to release the bus holdoff by writing any value into HP-IB CONTROL register 4.

CONTROL 7,4;Any\_value

### CONTROL Register 4 Release NDAC Holdoff

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 = Don't accept secondary command All non-zero values accept secondary (Writing anything to this register releases NDAC holdoff)							
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

As a reminder, these interrupt conditions occur under the following circumstances.

*Bit 14* enables an interrupt on any change in Parallel Poll configuration. If a Parallel Poll Configure command is received, the computer must set up its own Parallel Poll Response designated

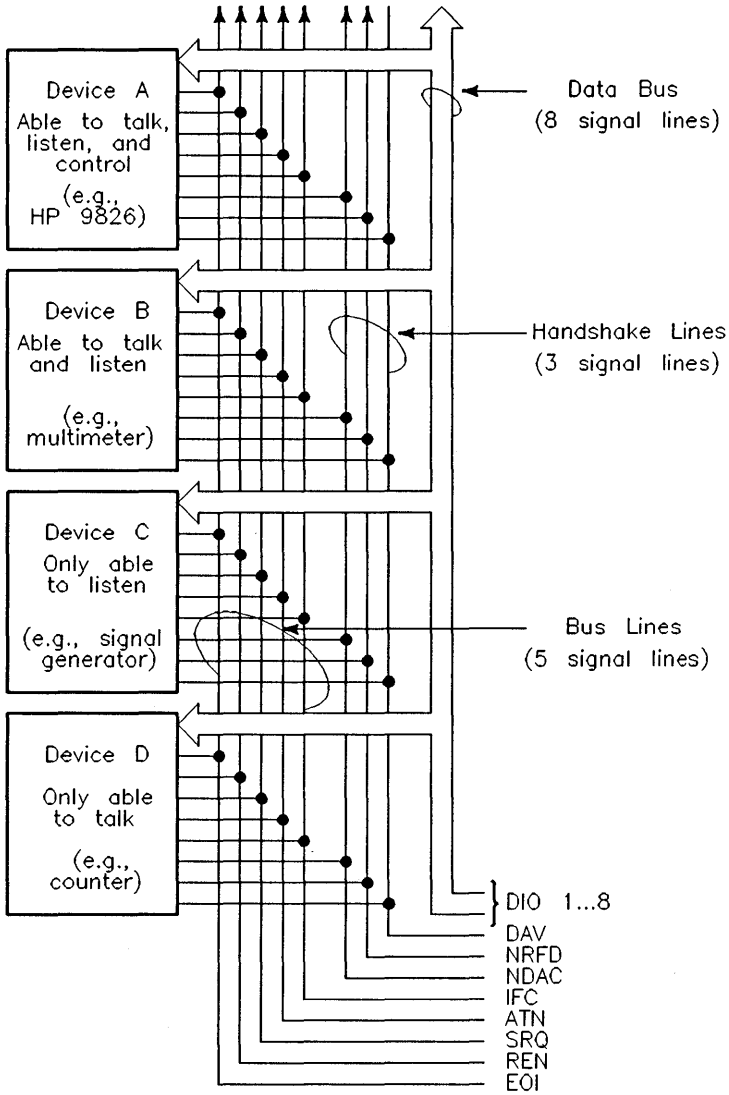
When a Secondary Command is received, two computer responses are possible.

- The first is to accept the address as a valid secondary address and consequently become an Extended Talker or Listener.
- The second is not to accept the address as valid and consequently remain in the primary addressed state.

If Secondary Command interrupts are enabled (while the computer is a non-Active Controller), the computer will not respond to its primary address alone; a valid secondary address is also required. Statements such as ENTER 7, OUTPUT 7, and LIST #7 should only be executed in the interrupt service routine after CONTROL has been used to indicate that a valid secondary address has been received but before interrupts are re-enabled.

When you no longer want the computer to respond as an Extended Talker/Listener, execute an ENABLE INTR with a mask which has bit 4 equal to zero.

# HP-IB Control Lines



**HP-IB Control Lines**

## Handshake Lines

3 The preceding figure shows the names given to the eight control lines that make up the HP-IB. Three of these lines are designated as the “handshake” lines and are used to control the timing of data byte exchanges so that the talker does not get ahead of the listener(s). The three handshake lines are as follows.

DAV	Data Valid
NRFD	Not Ready for Data
NDAC	Not Data Accepted

The HP-IB interlocking handshake uses the lines as follows. All devices currently designated as active listeners would indicate when they are ready for data by using the NRFD line. A device not ready would pull this line low (true) to signal that it is not ready for data, while any device that is ready would let the line float high. Since an active low overrides a passive high, this line will stay low until all active listeners are ready for data.

When the talker senses that all devices are ready, it places the next data byte on the data lines and then pulls DAV low (true). This tells the listeners that the information on the data lines is valid and that they may read it. Each listener then accepts the data and lets the NDAC line float high (false). As with NRFD, only when all listeners have let NDAC go high will the talker sense that all listeners have read the data. It can then float DAV (let it go high) and start the entire sequence over again for the next byte of data.

## The Attention Line (ATN)

Command messages are encoded on the data lines as 7-bit ASCII characters, and are distinguished from normal data characters by the logic state of the attention line (ATN). That is, when ATN is *false*, the states of the data lines are interpreted as *data*. When ATN is *true*, the data lines are interpreted as *commands*. The set of 128 ASCII characters that can be placed on the data lines during this ATN-true mode are divided into four classes by the states of data lines DIO6 and DIO7. These classes of commands are shown in a table in the section called “Advanced Bus Management”. Only the Active Controller can set ATN true.

## The Interface Clear Line (IFC)

Only the System Controller can set the IFC line true. By asserting IFC, all bus activity is unconditionally terminated, the System Controller regains the capability of Active Controller (if it has been passed to another device), and any current talker and listeners become unaddressed. Normally, this line is only used to terminate all current operations, or to allow the System Controller to regain control of the bus. It overrides any other activity that is currently taking place on the bus.

## The Remote Enable Line (REN)

This line is used to allow instruments on the bus to be programmed remotely by the Active Controller. Any device that is addressed to listen while REN is true is placed in the Remote mode of operation.

## The End or Identify Line (EOI)

Normally, data messages sent over the HP-IB are sent using the standard ASCII code and are terminated by the ASCII line-feed character, CHR\$(10). However, certain devices may wish to send blocks of information that contain data bytes which have the bit pattern of the line-feed character but which are actually part of the data message. Thus, no bit pattern can be designated as a terminating character, since it could occur anywhere in the data stream. For this reason, the EOI line is used to mark the end of the data message.

The EOI line is used as an END indication (ATN false) during ENTER statements and as the Identify message (ATN true) during an identify sequence (the response to parallel poll). During data messages, the EOI line is set true by the talker to signal that the current data byte is the last one of the data transmission. Generally, when a listener detects that the EOI line is true, it assumes that the data message is concluded. However, EOI may either be used or ignored by the computer when entering data with an ENTER statement that uses an image. Chapter 5 fully describes the definitions of EOI during all ENTER statements and shows how to use the image specifiers that modify the statement-termination conditions.

ENTER statements can use images to re-define the meaning of EOI to provide a very great degree of flexibility. Using the “#” or “%” specifier in an ENTER statement affects the definition of the EOI signal as shown in the following table.

**Definition of EOI During ENTER Statements**

	<b>Free-Field ENTER Statements</b>	<b>ENTER USING without # or %</b>	<b>ENTER USING with #</b>	<b>ENTER USING with %</b>
<b>Definition of EOI</b>	Immediate statement terminator	Item terminator or statement terminator	Item terminator or statement terminator	Immediate statement terminator
<b>Statement Terminator Required?</b>	Yes	Yes	No	No
<b>Early Termination Allowed?</b>	No	No	No	Yes

### The Service Request Line (SRQ)

The Active Controller is always in charge of the order of events that occur on the HP-IB. If a device on the bus needs the Active Controller’s help, it can set the Service Request line true. This line sends a request, not a demand, and it is up to the Active Controller to choose when and how it will service that device. However, the device will continue to assert SRQ until it has been “satisfied”. Exactly what will satisfy a service request depends on the requesting device, which is explained in the device’s operating manual.

### Determining Bus-Line States

STATUS register 7 contains the current states of all bus hardware lines. Reading this register returns the states of these lines in the specified numeric variable.

`STATUS Hpib,7;Bus_lines`



*STATUS Register 7*    Bus Control and Data Lines


Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN true	DAV true	NDAC <sup>1</sup> true	NRFD <sup>1</sup> true	EOI true	SRQ <sup>2</sup> true	IFC true	REN true
value= -32 768	value= 16 384	value= 8 192	value= 4 096	value= 2 048	value= 1 024	value= 512	value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

<sup>1</sup>Only if currently addressed to Talk, else not valid.

<sup>2</sup>Only if currently Active Controller, else not valid.

---

**Note**  Due to the way the bi-directional buffers work, NDAC and NRFD are not accurately read by this STATUS statement unless the interface is currently addressed to talk. Also, SRQ is not accurately shown unless the interface is currently the active controller.

---

3

## Summary of HP-IB STATUS and CONTROL Registers

*STATUS Register 0* Card identification = 1

*CONTROL Register 0* Reset interface if non-zero

*STATUS Register 1* Interrupt and DMA Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts enabled	Interrupt requested	Hardware level	Interrupt switches	0	0	DMA channel 1 enabled	DMA channel 0 enabled
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*CONTROL Register 1* Poll Response Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device dependent status	SRQ 1=I did it 0=I didn't	Device dependent status					
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

## HP-IB Status and Control Registers (continued)

*STATUS Register 2* Busy Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Reserved for future use	Handshake in progress	Interrupts enabled	TRANS- FER in progress
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*CONTROL Register 2* Parallel Poll Response Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8 1=true	DIO7 1=true	DIO6 1=true	DIO5 1=true	DIO4 1=true	DIO3 1=true	DIO2 1=true	DIO1 1=true
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*STATUS Register 3* Controller Status and Address

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary address of HP-IB interface				
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

## HP-IB Status and Control Registers (continued)

*CONTROL Register 3* Set My Address

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used			Primary address				
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*STATUS Register 4* Interrupt Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel poll configuration change	My Talk address received	My Listen address received	EOI Received	SPAS	Remote/local change	Talker/listener address change
value=-32 768	value=16 384	value=8 192	value=4 096	value=2 048	value=1 024	value=512	value=256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger received	Handshake error	Unrecognized universal command	Secondary command while addressed	Clear received	Unrecognized addressed command	SRQ received	IFC received
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

## HP-IB Status and Control Registers (continued)

*CONTROL Register 4* Writing anything to this register releases NDAC holdoff. If non-zero, accept last secondary address as valid. If zero, don't accept last secondary address (stay in LPAS or TPAS state).

*STATUS Register 5* Interrupt Enable Mask

3

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel poll configuration change	My Talk address received	My Listen address received	EOI received	SPAS	Remote/local change	Talker/listener address change
value= -32 768	value= 16 384	value= 8 192	value= 4 096	value= 2 048	value= 1 024	value= 512	value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger received	Handshake error	Unrecognized universal command	Secondary command while addressed	Clear received	Unrecognized addressed command	SRQ received	IFC received
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

## HP-IB Status and Control Registers (continued)

### *CONTROL Register 5* Parallel Poll Response Mask

3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used			Uncon- figure	Logic sense	Data bit used for response		
value=128	value=64	value=0	value=16	value=8	value=4	value=2	value=1

### *STATUS Register 6* Interface Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
REM	LLO	ATN true	LPAS	TPAS	LADS	TADS	*
value= -32 768	value= 16 384	value= 8 192	value= 4 096	value= 2 048	value= 1 024	value= 512	value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary address of interface				
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

\* Least-significant bit of last address recognized

---

## HP-IB Status and Control Registers (continued)

*STATUS Register 7*    Bus Control and Data Lines

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN true	DAV true	NDAC <sup>1</sup> true	NRFD <sup>1</sup> true	EOI true	SRQ <sup>2</sup> true	IFC true	REN true
value= -32 768	value= 16 384	value= 8 192	value= 4 096	value= 2 048	value= 1 024	value= 512	value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

<sup>1</sup>Only if currently addressed to Talk, else not valid.

<sup>2</sup>Only if currently Active Controller, else not valid.

## HP-IB Status and Control Registers (continued)

### *Interrupt Enable Register (ENABLE INTR)*

3

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel poll configuration change	My Talk address received	My Listen address received	EOI received	SPAS	Remote/local change	Talker/listener address change
value= -32 768	value= 16 384	value= 8 192	value= 4 096	value= 2 048	value= 1 024	value= 512	value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger received	Handshake error	Unrecognized universal command	Secondary command while addressed	Clear received	Unrecognized addressed command	SRQ received	IFC received
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

- STATUS Register 255*
- 0: HP-IB interface unlocked and HP-IB interface burst I/O disabled. (BASIC/WS and BASIC/DOS accept this command but always return the value "3".)
  - 1: HP-IB interface locked.
  - 3: HP-IB interface burst I/O enabled.

- CONTROL Register 255*
- 0: disables HP-IB interface locking and HP-IB interface burst I/O. (BASIC/WS and BASIC/DOS accept this command but always set the value "3".)
  - 1: enables HP-IB interface locking.
  - 3: enables HP-IB interface burst I/O.



---

## Summary of HP-IB READIO and WRITEIO Registers

### READIO Registers

Register 1	Card Identification
Register 3	Interrupt and DMA Status
Register 5	Controller Status and Address
Register 17	Interrupt Status 0 (READIO operation will change the state of the interface.)
Register 19	Interrupt Status 1 (READIO operation will change the state of the interface.)
Register 21	Interface Status
Register 23	Control-Line Status
Register 29	Command Pass-Through
Register 31	Data-Line Status (READIO operation will change the state of the interface.)
<i>HP-IB READIO Register 1</i>	Card Identification

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Future use jumper installed	0	0	0	0	0	0	1
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* is set (1) if the “future use” jumper is installed and clear (0) if not.

*Bits 6 through 0* constitute a card identification code (=1 for all HP-IB cards).

**Note**

This register is only implemented on external HP-IB cards. The internal HP-IB, at interface select code 7, “floats” this register (i.e., the states of all bits are indeterminate).

3

**HP-IB READIO Register 3** Interrupt and DMA Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupt enabled	Interrupt requested	Interrupt level		X	X	DMA1	DMA0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* is set (1) if interrupts are currently enabled. (On BASIC/UX, this bit is not cleared (0) by DISABLE INTR.)

*Bit 6* is set (1) when the card is currently requesting service.

*Bits 5 and 4* constitute the card’s hardware interrupt level (a switch setting on all external cards, but fixed at level 3 on the internal HP-IB).

Bit 5	Bit 4	Hardware Interrupt Level
0	0	3
0	1	4
1	0	5
1	1	6

*Bits 3 and 2* are not used (indeterminate).

*Bit 1* is set (1) if DMA channel one is currently enabled.

*Bit 0* is set (1) if DMA channel zero is currently enabled.

**Note**

Bits 7, 5, 4, 3, 2, and 1 are not implemented on the internal HP-IB (interface select code 7).

---

**HP-IB READIO Register 5     Controller Status and Address**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Not Active Controller	X	HP-IB primary address of interface				
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* is set (1) if the interface is the System Controller.

*Bit 6* is set (1) if the interface is *not* the current Active Controller and clear (0) if it *is* the Active Controller.

*Bit 5* is not used.

*Bits 4 through 0* contain the card's Primary Address switch setting. The following bit patterns indicate the specified addresses.

Bit 4 3 2 1 0	Primary Address
0 0 0 0 0	0
0 0 0 0 1	1
:	:
:	:
1 1 1 0 1	29
1 1 1 1 0	30
1 1 1 1 1	(not allowed)

---

**Note**

Bits 5 through 0 are not implemented on the internal HP-IB.

**3****HP-IB READIO Register 17** MSB of Interrupt Status

This READIO Register is not supported on BASIC/UX.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MSB interrupt	LSB interrupt	Byte received	Ready for next byte	End detected	SPAS	Remote/local change	My address change
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* set (1) indicates that an interrupt has occurred whose cause can be determined by reading the contents of this register.

*Bit 6* set (1) indicates that an interrupt has occurred whose cause can be determined by reading Interrupt STATUS Register 1 (READIO Register 19).

*Bit 5* set (1) indicates that a data byte has been received.

*Bit 4* set (1) indicates that this interface is ready to accept the next data byte.

*Bit 3* set (1) indicates that an End (EOI with ATN=0) has been detected.

*Bit 2* set (1) indicates that the computer is in the Serial Poll Addressed State (SPAS).

*Bit 1* set (1) indicates that a Remove/Local State change has occurred.

*Bit 0* set (1) indicates that a change in My Address has occurred.

HP-IB READIO Register 19 LSB of Interrupt Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger received	Handshake error	Unrecognized command group	Secondary command while addressed	Clear received	My address received (MLA or MTA)	SRQ received	IFC received
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* set (1) indicates that a Group Execute Trigger command has been received.

*Bit 6* set (1) indicates that an Incomplete-Source-Handshake error has occurred.

*Bit 5* set (1) indicates that an unidentified command has been received.

*Bit 4* set (1) indicates that a Secondary Address has been sent in while in the extended-addressing mode.

*Bit 3* set (1) indicates that the interface has entered the Device-Clear-Active State.

*Bit 2* set (1) indicates that My Address has been received.

*Bit 1* set (1) indicates that a Service Request has been received.

*Bit 0* set (1) indicates that the Interface Clear message has been received.

*HP-IB READIO Register 21* Interface Status

3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
REM	LLO	ATN true	LPAS	TPAS	LADS	TADS	LSB of last address
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* set (1) indicates that this interface is in the Remote State.

*Bit 6* set (1) indicates that this interface is in the Local Lockout State.

*Bit 5* set (1) indicates that the ATN signal line is true.

*Bit 4* set (1) indicates that this interface is in the Listener-Primary-Addressed State.

*Bit 3* set (1) indicates that this interface is in the Talker-Primary-Addressed State.

*Bit 2* set (1) indicates that this interface is in the Listener-Addressed State.

*Bit 1* set (1) indicates that this interface is in the Talker-Addressed State.

*Bit 0* set (1) indicates that this is the least-significant bit of the last address recognized by this interface.

*HP-IB READIO Register 23* Control-Line Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ATN true	DAV true	NDAC <sup>1</sup> true	NRFD <sup>1</sup> true	EOI true	SRQ <sup>2</sup> true	IFC true	REN true
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

<sup>1</sup> Only if addressed to TALK, else not valid.

<sup>2</sup> Only if Active Controller, else not valid.

**3-70 The HP-IB Interface**

A set bit (1) indicates that the corresponding line is currently true; a 0 indicates that the line is currently false.

*HP-IB READIO Register 29* Command Pass-Through

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

This register can be read during a bus holdoff to determine which Secondary Command has been detected.

*HP-IB READIO Register 31* Bus Data Lines

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

A set bit (1) indicates that the corresponding HP-IB data line is currently true; a 0 indicates the line is currently false.

## HP-IB WRITEIO Registers

- Register 3—Interrupt Enable
- Register 17—MSB of Interrupt Mask
- Register 19—LSB of Interrupt Mask
- Register 23—Auxiliary Command Register
- Register 25—Address Register
- Register 27—Serial Poll Response
- Register 29—Parallel Poll Response
- Register 31—Data Out Register

*HP-IB WRITEIO Register 3* Interrupt and DMA Enable

3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable interrupt	X	X	X	X	X	Enable channel 1	Enable channel 0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* enables interrupts from this interface if set (1) and disables interrupts if clear (0).

*Bits 6 through 2* are “don’t cares” (i.e., their values have no effect on the interface’s operation).

*Bit 1* enables DMA channel 1 if set (1) and disables if clear (0).

*Bit 0* enables DMA channel 0 if set (1) and disables if clear (0).

**Note**



Bits 7 through 1 are not implemented on the internal HP-IB interface and thus have no effect on the interface’s operation.

*HP-IB WRITEIO Register 17* MSB of Interrupt Mask

Setting a bit of this register enables an interrupt for the specified condition. The bit assignments are the same as for the MSB of Interrupt Status Register (READIO Register 17), except that bits 7 and 6 are not used.

*HP-IB WRITEIO Register 19* LSB of Interrupt Mask

Setting a bit of this register enables an interrupt for the specified condition. The bit assignments are the same as for the LSB of Interrupt Status Register (READIO Register 19).



## HP-IB WRITEIO Register 23 Auxiliary Command Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Set	X	X	Auxiliary command function				
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* is set (1) for a Set operation and clear (0) for a Clear operation.

*Bits 6 and 5* are “don’t cares.”

*Bits 4 through 0* are Auxiliary-Command-Function-Select bits. The following commands can be sent to the interface by sending the specified numeric values.

### Auxiliary Commands

Decimal Value	Description of Auxiliary Command
0	Clear Chip Reset
128	Set Chip Reset
1	Release ACDS holdoff. If Address Pass Through is set, it indicates an invalid secondary has been received.
129	Release ACDS holdoff. If Address Pass Through is set, indicates a valid secondary has been received.
2	Release RFD holdoff.
130	Same command as decimal 2 (above).
3	Clear holdoff on all data.
131	Set holdoff on all data.
4	Clear holdoff on EOI only.
132	Set holdoff on EOI only.
5	Set New Byte Available (nba) false.
133	Same command as decimal 5 (above).
6	Pulse the Group Execute Trigger line, or clear the line if it was set by decimal command 134.
134	Set Group Execute Trigger line.
7	Clear Return To Local (rtl).
135	Set Return To Local (must be cleared before the device is able to enter the Remote state).
8	Causes EOI to be sent with the next data byte.
136	Same command as decimal 8 (above).

3

### Auxiliary Commands (continued)

Decimal Value	Description of Auxiliary Command
9	Clear Listener State (also cleared by decimal 138).
137	Set Listener State.
10	Clear Talker State (also cleared by decimal 137).
138	Set Talker State.
11	Go To Standby (gts; controller sets ATN false).
139	Same command as decimal 11 (above).
12	Take Control Asynchronously (tca; ATN true).
140	Same command as decimal 12 (above).
13	Take Control Synchronously (tcs; ATN true).
141	Same command as decimal 13 (above).
14	Clear Parallel Poll
142	Set Parallel Poll (read Command-Pass-Through register before clearing).
15	Clear the Interface Clear line (IFC).
143	Set Interface Clear (IFC maintained > 100 $\mu$ s).
16	Clear the Remote Enable (REN) line.
144	Set Remote Enable.
17	Request control (after TCT is decoded, issue this to wait for ATN to drop and receive control).
145	Same command as decimal 17 (above).
18	Release control (issued after sending TCT to complete a Pass Control and set ATN false).
146	Same command as decimal 18 (above).

### Auxiliary Commands (continued)

Decimal Value	Description of Auxiliary Command
19	Enable all interrupts.
147	Disable all interrupts.
20	Pass Through next Secondary Command.
148	Same command as decimal 20 (above).
21	Set TI delay to 10 clock cycles (2 $\mu$ s at 5 MHz).
149	Set TI delay to 6 clock cycles (1.2 $\mu$ s at 5 MHz).
22	Clear Shadow Handshake
150	Set Shadow Handshake.

#### *HP-IB WRITEIO Register 25* Address Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable dual addressing	Disable listen	Disable talker	Primary address				
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* set (1) enables the Dual-Primary-Addressing Mode.

*Bit 6* set (1) invokes the Disable-Listen function.

*Bit 5* set (1) invokes the Disable-Talker function.

*Bits 4 through 0* set the device's Primary Address (same address bit definitions as READIO Register 5).

*HP-IB WRITEIO Register 27 Serial Poll Response Byte*


Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device dependent status	Request service	Device dependent status					
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

3

*Bits 7 and 5—0* specify the Device Dependent Status.

*Bit 6* sends an SRQ if set (1).

---

**Note**  Given an unknown state of the Serial Poll Response Byte, it is necessary to write the byte with bit 6 set to zero followed by a write of the byte with bit 6 set to the desired final value. This will insure that a SRQ will be generated if one was desired.

---

*HP-IB WRITEIO Register 29 Parallel Poll Response*

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

A 1 sets the appropriate bit true during a Parallel Poll; a 0 sets the corresponding bit false. Initially, and when Parallel Poll is not configured, this register must be set to all zeros.

## HP-IB WRITEIO Register 31 Data-Out Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

---

### Summary of Bus Sequences

The following tables show the bus activity invoked by executing HP-IB statements and functions. The mnemonics used in these tables were defined in the previous sections of this chapter.

Note that bus messages are sent by using single lines (such as the ATN line) and multiple lines (such as DCL). The information shows the state of and changes in the state of the ATN line during these bus sequences. The tables implicitly show that *these changes in the state of ATN remain in effect unless another change is explicitly shown in the table*. For example, if a statement sets ATN (true) with a particular command, then it remains true unless the table explicitly shows that it is set false. The ATN line is implemented in this manner to avoid unnecessary transitions in this signal whenever possible. It should not cause any dilemmas in most cases.

## ABORT

### Summary of Bus Actions

	System Controller	Not System Controller
Active Controller	IFC (duration $\geq 100 \mu\text{sec}$ ) $\frac{\text{REN}}{\text{ATN}}$	ATN MTA $\frac{\text{UNL}}{\text{ATN}}$
Not Active Controller	IFC (duration $\geq 100 \mu\text{sec}$ ) <sup>1</sup> $\frac{\text{REN}}{\text{ATN}}$	No Action

<sup>1</sup> The IFC message allows a non-active controller (which is the system controller) to become the active controller.

## CLEAR

The computer *must* be active controller to execute this statement.

### Summary of Bus Actions

Interface Select Code Only	Primary Address Specified
ATN DCL	ATN MTA UNL LAG SDC

## LOCAL

### Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Address Specified	Interface Select Code Only	Primary Address Specified
Active Controller	$\overline{\text{REN}}$	ATN MTA UNL LAG GTL	ATN GTL	ATN MTA UNL LAG GTL
Not Active Controller	$\overline{\text{REN}}$	Error	Error	Error

## LOCAL LOCKOUT

The computer sending LOCAL LOCKOUT *must* be the active controller and only an interface select code may be specified, not a primary address.

### Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Address Specified	Interface Select Code Only	Primary Address Specified
Active Controller	ATN LLO	Error	ATN LLO	Error
Not Active Controller	Error	Error	Error	Error



## PASS CONTROL

The computer *must* currently be active controller to execute this statement, and primary addressing (but not multiple listeners) *must* be specified. The controller may be either a System or Non-system controller.

**Summary of Bus Actions**

3

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Address Specified	Interface Select Code Only	Primary Address Specified
Active Controller	Error	ATN UNL TAD TCT <u>ATN</u>	Error	ATN UNL TAD TCT <u>ATN</u>
Not Active Controller	Error	Error	Error	Error

## PPOLL

The computer must be the active controller to execute this function.

**Summary of Bus Actions**

Interface Select Code Only	Primary Address Specified
ATN & EOI (duration $\geq 25\mu s$ ) Read byte $\overline{EOI}$ Restore ATN to previous state	Error

## PPOLL CONFIGURE

This statement assumes that the device's response is bus-programmable. The computer *must* be the active controller to execute this statement.

### Summary of Bus Actions

Interface Select Code Only	Primary Address Specified
Error	ATN
	MTA
	UNL
	LAG
	PPC
	PPE

## PPOLL UNCONFIGURE

The computer *must* be the active controller to execute this statement. The computer may be either a System or Non-System Controller.

### Summary of Bus Actions

Interface Select Code Only	Primary Address Specified
ATN	ATN
PPU	MTA
	UNL
	LAG
	PPC
	PPD

## REMOTE

The computer *must* be the system controller to execute this statement, and it must be the active controller to place individual devices in the remote state.

### Summary of Bus Actions

	Interface Select Code Only	Primary Address Specified
Active Controller	REN ATN	REN ATN MTA UNL LAG
Not Active Controller	REN	Error

3

## SPOLL

The computer *must* be the active controller to execute this statement, and multiple listeners are not allowed. One secondary address may be specified to get status from an extended talker.

3

**Summary of Bus Actions**

Interface Select Code Only	Primary Address Specified
Error	ATN
	UNL
	MLA
	TAD
	SPE
	$\overline{\text{ATN}}$
	Read data
	ATN
	SPD
	UNT

## TRIGGER

The computer *must* currently be active controller to execute this statement.

### Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Address Specified	Interface Select Code Only	Primary Address Specified
Active Controller	ATN GET	ATN UNL LAG GET	ATN GET	ATN MTA UNL LAG GET
Not Active Controller	Error	Error	Error	Error

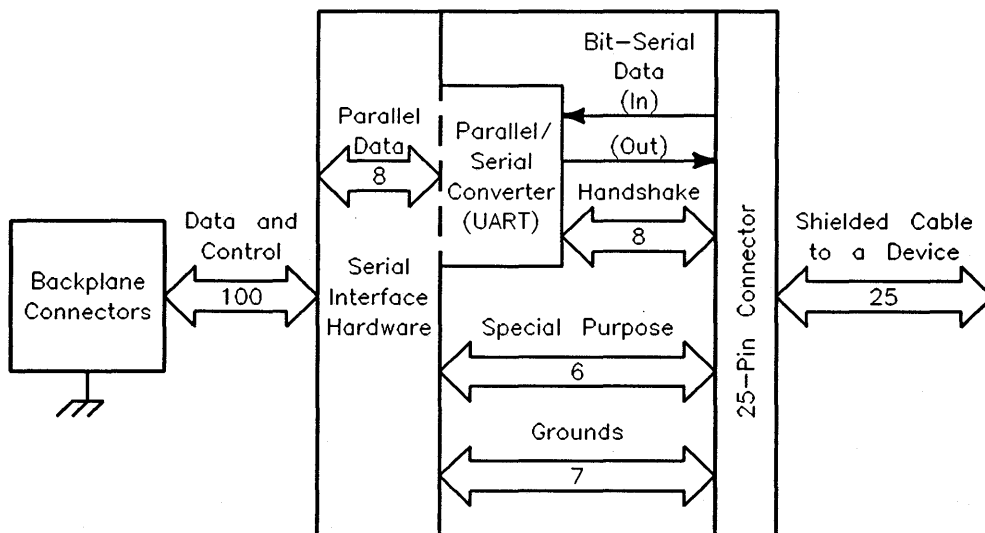
3



## RS-232C Serial Interfaces

### Overview

The HP 98626 and HP 98644 serial interfaces are RS-232C compatible interfaces used for simple asynchronous I/O applications such as driving line printers, terminals, or other peripherals where the more sophisticated capabilities of the HP 98628 and HP 98642 data communications interfaces are not justified. (See the "Datacomm Interfaces" chapter for more information on data communication interfaces.) Because the HP 98626 and HP 98644 Serial Interface cards have only a few differences, this chapter will deal mainly with the HP 98626 interface card. Information on differences between these two serial interface cards can be found in the sections "HP 98644 Interface Differences" and "Series 300 Built-In 98644 Interface Differences."



Block Diagram of the Serial Interface

The BASIC system must provide most control functions because these cards do *not* have their own microprocessor (as do the HP 98628 and HP 98642 cards). Consequently, there is more interaction between these cards and computer than when you use a more intelligent interface except for relatively simple applications.

The RS-232C interface standard establishes electrical and mechanical interface requirements, but does not define the exact function of all the signals that are used by various manufacturers of data communications equipment and serial I/O devices. Consequently, when you plug your serial interface into an RS-232 connector, there is no guarantee the devices can communicate unless you have configured optional parameters to match the requirements of the device you are connecting to.

---

**Note**

The RS-232C data communication standard is established and published by the Electronic Industries Association (EIA). Copies of the standard are available from the association at 2001 Eye Street N. W., Washington D. C. 20006. Its equivalent for European applications is CCITT V.24.

---

## Asynchronous Data Communication

The terms Asynchronous (Async for short) data communication and Serial I/O refer to a technique of transferring information between two communicating devices by means of bit-serial data transmission. This means that data is sent, one bit at a time, and that characters are not synchronized with preceding or subsequent data characters; that is, each character is sent as a complete entity without relationship to other events, before or after. Characters may be sent in close succession, or they may be sent sporadically as data becomes available. Start and stop bits are used to identify the beginning and end of each character, with the character data placed between them.

### Character Format

Each character frame consists of the following elements:

**Start Bit**            The start bit signals the receiver that a new character is being sent. Since the receiver knows how many bits per second are being transmitted (specified by the baud rate), it can

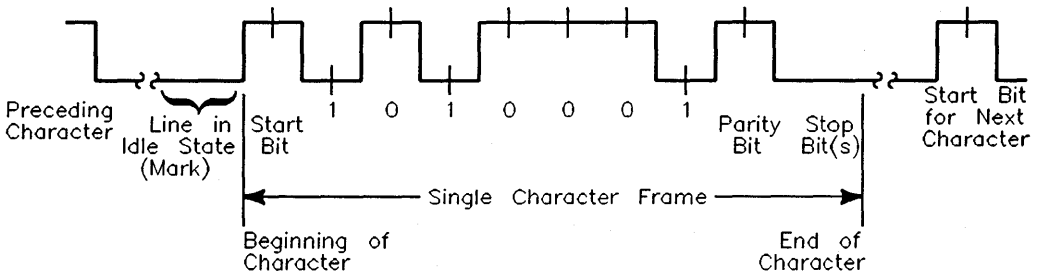


determine the expected arrival time for all subsequent bits in that character frame. All other bits in a given frame are synchronized to the start bit.

- 5-8 Character Data Bits** The next bits are the binary code of the character being transmitted, consisting of 5, 6, 7, or 8 bits; depending on the application. The parity bit is not included in the character data bits.
- Parity Bit** The parity bit is optional, included only when parity is enabled.
- Stop Bit(s)** One or more stop bits identify the end of each character. The serial interface has no provision for inserting time gaps between characters.

4

Here is a simple diagram showing the structure of an asynchronous character and its relationship to other characters in the data stream:



### Asynchronous Format

#### Parity

The parity bit is used to detect errors as incoming characters are received. If the parity bit does not match the expected sense, the character is assumed to be incorrectly received. The action taken when an error is detected depends upon how the interface and the BASIC program are configured.

System requirements govern parity sense, which is determined by counting the number of ones in the character *including* the parity bit. Consequently, the parity sense is reversed from the number of ones in a character without the parity bit.

The parity bit may be included or omitted from each character by enabling or disabling the parity function. If the parity bit is enabled, four options are available. Parity is checked by the receiver for all parity options including ONE and ZERO. (The HP 98628 and HP 98642 Datacomm Interface cards do not check parity when parity is set to ONE or ZERO.)

Parity options include:

NONE	Parity function is DISABLED, and the parity bit is omitted from each character frame.
ODD	Parity bit is SET if there is an EVEN number of ones in the data character. The receiver performs parity checks on incoming characters.
EVEN	Parity bit is SET if there is an ODD number of ones in the data character. The receiver performs parity checks on incoming characters.

4

### Error Detection

Two types of incoming data errors can be detected by serial receivers:

- **Parity errors** are signaled when the parity bit does not match the number of ones, including the parity bit, even or odd as defined by interface configuration. When parity is disabled, no parity check is made.
- **Framing errors** are signaled when start and stop bits are not properly received during the expected time frame. They can be caused by a missing start bit, noise errors near the end of the character, or by improperly specified character length at the transmitter or receiver.

Two additional error types are detected by the receiver section of the serial interface:

- **Overrun** errors result when the desktop computer does not consume characters as fast as they arrive. The card provides only one character of buffer space, so the current character must be consumed by an ENTER before the next character arrives. Otherwise, the character is lost when the next character replaces it, and an error is sent to BASIC.
- **Received BREAKs** are detected as a special type of framing error. They generate the same type of BASIC error as framing errors.

## Data Transfers Between Computer and Peripheral

Five statements are used to transfer information between your desktop computer and the interface card:

- The **OUTPUT** statement sends data to the interface which, in turn, sends the information to the peripheral device.
- The **ENTER** statement inputs data from the interface card after the interface has received it from the peripheral device.
- The **STATUS** statement is used to monitor the interface and obtain information about interface operation such as buffer status, detected errors, and interrupt enable status.
- The **CONTROL** statement is used to control interface operation and defines such parameters as baud rate, character format, or parity.
- The **TRANSFER** statement is used to input or output data from/to the interface and, in turn, from/to the peripheral device.

Since the interface has no on-board processor, **ENTER** and **OUTPUT** statements cause the computer to wait until the **ENTER** or **OUTPUT** operation is complete before continuing to the next line. For **OUTPUT** statements, this means that the computer waits until the last bit of the last character has been sent over the serial line before continuing with the next program statement.

---

## Overview of Serial Interface Programming

Serial interface programming techniques are similar to most general I/O applications. The interface card is initialized by use of **CONTROL** statements; **STATUS** statements evaluate its readiness for use. Data is transferred between the desktop computer and a peripheral device by **OUTPUT** and **ENTER** statements.

Due to the asynchronous nature of serial I/O operations, special care must be exercised to ensure that data is not lost by sending to another device before the device is ready to receive. Modem line handshaking can be used to help solve

this problem. These and other topics are discussed in greater detail elsewhere in this chapter.

## Determining Operating Parameters

Before you can successfully transfer information to a device, you must match the operating characteristics of the interface to the corresponding characteristics of the peripheral device. This includes matching signal lines and their functions as well as matching the character format for both devices.

### Handshake and Baud Rate

To determine hardware operating parameters, you need to know the answer for each of the following questions about the peripheral device:

- Which of the following signal and control lines are actively used during communication with the peripheral?

- Data Set Ready (DSR)
- Data Carrier Detect (DCD or CD)
- Clear to Send (CTS)
- Ring Indicator (RI)

- What baud rate (line speed) is expected by the peripheral?

### Character Format Parameters

To define the character format, you must know the requirements of the peripheral device for the following parameters:

Character Length	How many data bits are used for each character, excluding start, stop, and parity bits?
Parity Enable	Is parity enabled (included) or disabled (absent) for each character?
Parity Sense	Is the parity bit, if enabled, ODD, EVEN, always ONE, or always ZERO?
Stop Bits	How many stop bits are included with each character: 1, 1.5, or 2?

## Using BASIC/WS Interface Defaults to Simplify Programming

The serial interface includes three default configuration switch clusters in addition to the select code and interrupt level switches. Their functions are described in the following paragraphs.

### Modem-Line Disconnect Switches

The Modem Line Disconnect switches are used to connect or disconnect the following modem lines from the interface cable:

- Data Set Ready (DSR), RS-232C
- Data Carrier Detect (DCD or CD), RS-232C
- Clear to Send (CTS), RS-232C
- Ring Indicator (RI), RS-232C

4

When a given switch is in the CONNECT position, the corresponding modem line is connected from the peripheral device to the interface circuitry. When it is in the disconnected position, the modem line is disconnected, and the interface receiver input for that line is held HIGH (true). Any modem lines that are not actively used while communicating with the peripheral should be disconnected to minimize errors due to electrical noise in the cable. *Modem line disconnect switch settings cannot be altered under program control.* To reconfigure the switches, the interface must be removed from the computer, and the settings changed by hand.

---

### Note



The built-in HP 98626 serial interface in Series 200 Model 216 and 217 computers has no “modem-line disconnect” switches. Because switch settings can vary, cable connections between the computer and an external device can require some cross-wiring. Use of a breakout box can be helpful.

---

## Baud Rate Select Switches

The rate at which data bits are transferred between the interface and the peripheral is called the baud rate. The interface card must be set to transmit and receive at the same rate as the peripheral, or data cannot be successfully transferred. (These switches are not implemented on the 98644 interface. See the description of register 13, which allows you to set a SCRATCH A default value for the baud rate.) To preset the baud rate, the Baud Rate Select switches can be set to any one of the following values:

**Baud Rate Select Switch Settings**

Baud Rate	Switch Settings 3 2 1 0	Baud Rate	Switch Settings 3 2 1 0
50	0 0 0 0	1200	1 0 0 0
75	0 0 0 1	1800	1 0 0 1
110	0 0 1 0	2400	1 0 1 0
134.5	0 0 1 1	3600	1 0 1 1
150	0 1 0 0	4800	1 1 0 0
200	0 1 0 1	7200	1 1 0 1
300	0 1 1 0	9600	1 1 1 0
600	0 1 1 1	19200	1 1 1 1

## Line-Control Switches

The Line Control switches are used to preset character format and parity options. Functions are as shown in the following table. (These switches are not implemented on the 98644 interface. See the description of register 14, which allows you to set a SCRATCH A default value for the character format.)

## Line Control Switch Settings

Parity Sense (Switches 5&4)	Parity Enable (Switch 3)	Stop Bits (Switch 2)	Character Length (Switches 1&10)
00 ODD parity	0 Disabled	0 1 stop bit	00 5 bits/char
01 EVEN parity	1 Enabled	1 1.5 stop bits	01 6 bits/char
10 Always ONE		(if 5 bits/char),	10 7 bits/char
11 Always ZERO		or 2 stop bits	11 8 bits/char
		(if 6, 7, or 8 bits/char)	

Bits 6 and 7 are reserved for future use.

4

### Serial Configuration for BASIC/UX

There is no capability in BASIC/UX for reading the hardware bit settings on either the HP 98626 or HP 98644 Serial Interface cards. Therefore, BASIC/UX provides two methods for configuring modem control options:

- The `stty` command from the HP-UX environment.
- The keyword `CONTROL` and registers directly related to the modem control options.

Of the two methods mentioned above the best one to use is the `stty` command while in the HP-UX environment. The reason for this is any modem control options set by using the keyword `CONTROL` are lost when you leave BASIC/UX. However, if you prefer to change these options while in the BASIC/UX environment, then read the subsequent section "Using Program Control to Override Defaults."

This section deals with the first method mentioned above which is the use of the `stty` command from the HP-UX environment.

### Defaults for the Serial Interface

When HP-UX is being booted up, the defaults for all serial interfaces are:

```

Baud rate      300
Bits per character  7
Parity        Odd
Stop bits     2
    
```

The above values are used by BASIC/UX as defaults, unless configured as explained in the next section.

Some common serial interface configuration settings are:

Baud rate to	9600
Bits per character to	8
Parity to	Odd and disabled
Stop bits to	1

### Configuring a Serial Interface for BASIC/UX

4 To configure your serial interface with the values mentioned in the previous section, you can execute the following HP-UX command before entering BASIC/UX:

```
/bin/stty 9600 cs8 -parenb parodd -cstopb < /dev/rmb/serialnn
```

where:

9600 is the baud rate. The following are baud rates you can use with the **stty** command:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19200

cs8 is the number of bits per character. In the case of this example, the number of bits per character is 8. Other character lengths can be set using **cs5**, **cs6**, or **cs7** for 5, 6, or 7 bits per character respectively.

-parenb disables parity generation and detection. Removing the minus sign that is prefixed to this **stty** option causes parity generation and detection to be enabled.

parodd selects odd parity. Prefixing a minus sign to this **stty** option selects even parity.

-cstopb causes one stop bit per character to be used. Removing the minus sign that is prefixed to this



`stty` option causes two stop bits per character to be used.

`< /dev/rmb/serialnn` assigns the `stty` options to the serial interface located at select code number `nn`.

For more information on `stty` options, see the *HP-UX Language Reference*.

## Using Program Control to Override Defaults

You can override some of the interface default configuration options by use of `CONTROL` statements. This not only enables you to guarantee certain parameters, but also provides a means for changing selected parameters in the course of a running program. `CONTROL` Register tables are listed at the end of this chapter as well as in the *HP BASIC 6.2 Language Reference*. Refer to them as needed during the discussion which follows.

4

### Interface Reset

Whenever an interface is connected to a modem that may still be connected to a telecommunications link from a previous session, it is good programming practice to reset the interface to force the modem to disconnect, unless the status of the link and remote connection are known. When the interface is connected to a line printer or similar peripheral, resetting the interface is usually unnecessary unless an error condition requires it.

```
100 CONTROL Sc,0;1 ! Reset interface.
```

When the interface is reset by use of a `CONTROL` statement to `CONTROL` Register 0 with a non-zero value, the interface is restored to its power-up condition, except that the current character format for `BASIC/WS` is not altered whether or not it is the same as the current default switch configuration. If you are not sure of the present settings, or if your application requires changing the configuration during program operation, you can use `CONTROL` statements to configure the interface. An example of where this may be necessary is when several peripherals share a single interface through a manually operated `RS-232` switch such as those used to connect multiple terminals to a single computer port, or a single terminal to multiple computers.

## Selecting the Baud Rate

In order to successfully transfer information between the interface card and a peripheral, the interface and peripheral must be set to the same baud rate. A `CONTROL` statement to register 3 (or 13 with 98644 interfaces) can be used to set the interface baud rate to any one of the following values:

50	75	110	134
150	200	300	600
1200	1800	2400	3600
4800	7200	9600	19200

4 For example, to select a baud rate of 3600, the following program statement is used:

```
CONTROL Sc,3;3600
```

Use of values other than those shown may result in incorrect operation.

To verify the current baud rate setting, use a `STATUS` statement addressed to register 3. All rates are in baud (bits/second).

## Setting Character Format and Parity

CONTROL Register 4 overrides the Line Control switches that control parity and character format. To determine the value sent to the register, add the appropriate values selected from the following table:

**Character Format and Parity Settings**

Handshake (Bits <sup>2</sup> 7&6)	Parity Sense <sup>1</sup> (Bits <sup>2</sup> 5&4)	Par. Enable (Bit <sup>2</sup> 3)	Stop Bits (Bit <sup>2</sup> 2)	Char. Length (Bits <sup>2</sup> 1&0)
00 no-op	00 ODD parity	0 Disabled	0 1 stop bit	00 5 bits/char
01 Xon/Xoff	01 EVEN parity	1 Enabled	1 2 stop bits	01 6 bits/char
Bidirectional	10 Unsupported			10 7 bits/char
10 Unsupported	11 Unsupported			11 8 bits/char
11 Handshake Disabled				

4

<sup>1</sup> Parity sense valid only if parity is enabled (bit 3=1). If parity is disabled, parity sense is meaningless.

<sup>2</sup> These bits correspond to equivalent switch settings on the HP 98626 and HP 98644 serial interface cards. A 1 is the same as set.

### Note



With HP 98644 interfaces, there are no Line Control switches. You can simulate their effect by writing to CONTROL Register 14. Note that individual bits of this register are the same as for register 4.

For example, to configure a character format of eight bits per character, two stop bits, and EVEN parity, use the following CONTROL statement:

```
CONTROL Sc,4;IVAL("11111",2)
```

OR

```
CONTROL Sc,4;31
```

To configure a 5-bit character length with 1 stop bit and no parity bit, use the following:

```
CONTROL Sc,4;0
```

---

## Transferring Data

The serial interface card is designed for relatively simple serial I/O operations. It is not intended for sophisticated applications that use ON INTR statements to service the interface. If your situation for BASIC/WS requires full interrupt capability such as in terminal emulator applications, use the HP 98628 Datacomm Interface instead. Limited ON INTR capabilities are provided by the serial interface for error trapping and other simple tasks.

## Entering and Outputting Data

4

When the interface is properly configured, either by use of default switches or CONTROL statements, you are ready to begin data transfers. OUTPUT statements are used to send information to the peripheral; ENTER statements to input information from the external device.

```
OUTPUT 20;"String data",Numeric_var,Etc
```

```
ENTER 20;String_var$,Numeric_var,Etc
```

Any valid OUTPUT or ENTER statement and variable(s) list may be used, but you must be sure that the data format is compatible with the peripheral device. For example, non-ASCII data sent to an ASCII line printer may result in unexpected behavior.

Various other I/O statements can be used in addition to OUTPUT and ENTER, depending on the situation. For example, the LIST statement can be used to list programs to an RS-232 line printer—provided the interface is properly configured *before* the operation begins.

### Outputting Data

To send data to a peripheral, use OUTPUT, OUTPUT USING, or any other similar or equivalent construct. Suppression of end-of-line delimiters and other formatting capabilities are identical to normal operation in general I/O applications. The OUTPUT statement hangs the computer until the last bit of the last character in the statement variable list is transmitted by the interface. When the output operation is complete, the computer then continues to the next line in the program. See the "Outputting Data" chapter for details of the OUTPUT statement.

## Entering Data

To input data from a peripheral, use `ENTER`, `ENTER USING`, or an equivalent statement. Inclusion or elimination of end-of-line delimiters and other information is determined by the formatting specified in the `ENTER` statement. The `ENTER` statement hangs the computer until the input variables list is satisfied. To minimize the risk of waiting for another variable that isn't coming, you may prefer to specify only one variable for each `ENTER` statement, and analyze the result before starting the next input operation. See the "Entering Data" chapter for details of the `ENTER` statement.

Be sure that the peripheral is not transmitting data to the interface while no `ENTER` is in progress. Otherwise, data may be lost because the card provides buffering for only one character. Also, interrupts from other I/O devices, or operator inputs to the computer keyboard can cause delays in computer service to the interface that result in buffer overrun at higher baud rates.

4

## Modem Line Handshaking (HP BASIC/WS only)

Modem line handshaking, when used, is performed automatically by the computer as part of the `OUTPUT` or `ENTER` operation. If the modem line states have not been latched in a fixed state by Control Register 5, the following sequence of events is executed automatically during each `OUTPUT` or `ENTER` operation.

For `OUTPUT` operations:

1. Set Data Terminal Ready and Request-to-Send modem lines to active state.
2. Check Data Set Ready and Clear-to-Send modem lines to be sure they are active.
3. Send information to the interface and thence to the peripheral.
4. After data transfer is complete, clear Data Terminal Ready and Request-to-Send signals.

For ENTER operations:

1. Set Data Terminal Ready line to active state. Leave Request-to-Send inactive.
2. Check Data Set Ready and Data Carrier Detect modem lines to be sure they are active.
3. Input information from the interface as it is received from the peripheral.
4. After the input operation is complete, clear the Data Terminal Ready signal.

After a given OUTPUT or ENTER operation is completed, the program continues execution on the next line.

4

Control Register 5 can be used to force selected modem control lines to their active state(s). The Data Rate Select line is set or cleared by bit 2. For BASIC/WS, the Secondary Request-to-send line is set or cleared by bit 3. Request-to-send and Data Terminal Ready are held in their active states when bits 1 and 0 are true, respectively. If bits 1 and/or 0 are false, the corresponding modem line is toggled during OUTPUT or ENTER as explained previously.

## BASIC/UX Modem Line Handshaking

BASIC/UX requires additional system administration before modem line handshaking can be used with the HP98626/HP98644 card. The minor numbers of any serial device files in the /dev/rmb directory must be changed to 0xSS0009 where SS is the select code of the serial interface. For example,

```
crw-rw-rw-  1 root    other    1 0x090004 Feb 12 12:44 /dev/rmb/serial9
```

would be changed to

```
crw-rw-rw-  1 root    other    1 0x090009 Feb 12 12:44 /dev/rmb/serial9
```

The mechanism used for modem line handshaking is limited by the features provided by the HP-UX operating system. In particular, BASIC/UX uses the simple mode of modem line handshaking with a call-out device file. A full discussion of HP-UX facilities is beyond the scope of this manual. Refer to modem(7) and termio(7) for more details.

HP-UX allows for three different types of opens on RS232 interfaces: call-in, call-out, and direct connect. There are anomalies associated with each type

## 4-16 RS-232C Serial Interfaces

of open, and thus existing applications or workarounds which execute outside the BASIC/UX environment may not work correctly as a result of changing the device file used by BASIC/UX. Because of these anomalies, BASIC/UX continues to use a direct connect device file for backward compatibility. See MODEM(7) for more details.

HP-UX simple mode of modem line handshaking uses the following algorithm for modem line handshaking. DTR is asserted by the interface, and DCD and CTS must be asserted by the device before data transfers can take place. If DCD is lowered, DTR will also be lowered until DCD is asserted again. See MODEM(7) for a further description of the simple mode of modem line handshaking.

## Incoming Data Error Detection and Handling (BASIC/WS only)

The serial interface card can generate several errors that are caused when certain conditions are encountered while receiving data from the peripheral device. The UART detects a given error condition. The card then generates a pending error to BASIC. Errors can be generated by any of the following conditions:

- 4 Parity error     The parity bit on an incoming character does not match the parity expected by the receiver. This condition is most commonly caused by line noise. When this error occurs on BASIC/WS, bit 2 of Status Register 10 is set.
- Framing error     Start and stop bit(s) do not match the timing expectations of the receiver. This can occur when line noise causes the receiver to miss the start bit or obscures the stop bits. When this error occurs on BASIC/WS, bit 3 of Status Register 10 is set.
- Overrun error     Incoming data buffer overrun caused a loss of one or more data characters. This is usually caused when data is received by the interface, but no ENTER statement has been activated to input the information. When this error occurs on BASIC/WS, bit 1 of Status Register 10 is set.
- Break received     A BREAK was sent to the interface by the peripheral device. The desktop computer program must be able to properly interpret the meaning of a break and take appropriate action. When this condition occurs on BASIC/WS, bit 4 of Status Register 10 is set. Since a BREAK is detected as a special type of framing error, the framing error indicator, bit 3, is also set.



All UART status errors are generated by *incoming* data, never by outbound data. When a UART error occurs, the corresponding bit of Status Register 10 is set, and a pending error (ERROR 167: Interface status error) is sent to BASIC. BASIC processes the error according to the following rules:

- If an ENTER is in progress, the error is handled immediately as part of the ENTER process. An active ON ERROR causes the error trap to be executed. If no ON ERROR is active, the error is fatal and causes the program to terminate.
- If an OUTPUT is in progress, or if there is no current activity between the computer and interface, the error is flagged, but nothing is done by BASIC until an ENTER statement is encountered. When the computer begins execution of the ENTER statement, if an ON ERROR is active, the error trap is executed. If there is no active ON ERROR for that select code, the fatal ERROR 167 causes the BASIC program to terminate.
- If a STATUS statement is executed to Status Register 10 before an ENTER statement is encountered for that select code, the pending BASIC error is cleared, and the program continues as if no error had been generated. For BASIC/WS, whenever a STATUS statement is executed to Status Register 10, bits 1 through 4 of the register are cleared and the data is destroyed. If you need to perform multiple operations (such as IF BIT tests) on the register contents, be sure to store the information in a variable before you use it.

Note that the above UART status errors cannot be detected using BASIC/UX.

### Trapping Serial Interface Errors on BASIC/WS

Pending BASIC errors can be trapped by using an ON ERROR statement in conjunction with an error trapping service routine to evaluate the error condition. Here is an example technique:

```

1200 Sc=9                ! Set serial interface select code.
1210 ON ERROR GOTO Error ! Set up error trap routine.
    .
    .
1400 ENTER Sc;A$        ! Input line of data from interface.
    .
    .
1530 Error:             ! Error trap routine:
1535   IF ERRN167 THEN Other_error
1540   STATUS Sc,10;Uart_error ! Get UART error information.
1550   IF BIT (Uart_error,1) THEN Overrun ! Overrun error.
1560   IF BIT (Uart_error,2) THEN Parity ! Parity error.
1570   IF BIT (Uart_error,4) THEN Break ! BREAK received.
1580   IF BIT (Uart_error,3) THEN Framing ! Framing error.
1590 Other:             ! Other error type.
    .
    .
1650 Overrun:          ! Overrun error routine:
    .
    .
1700 Parity:           ! Parity error routine:
    .
    .
1750 Framing:          ! Framing error routine:
    .
    .
1800 Break:            ! BREAK received routine:
    .
    .
1850 Other_error: ! Not error 167. Process accordingly.|A
    .
    .

```

This example is not intended to show a specific application, but only to illustrate the technique for trapping interface errors. Only UART errors are shown in this example, but the technique is valid for other errors related to a given interface.

Note that in this example, the UART error information is checked for a BREAK before looking at the framing error bit. When a break is received, both the BREAK and framing error bits are set. Consequently, if the error check sequence were reversed, it would be necessary to check for a BREAK whenever a framing error is processed. Reversing the order eliminates an extra step by making it unnecessary to check for framing errors when a BREAK occurs. That is because whenever the BREAK is processed, the framing error is also cleared, making it unnecessary to perform any operations related to framing errors that are handled by the BREAK routine.

---

## Advanced Programming Information

This section provides advanced programming information for applications requiring special techniques.

### RS-232 Software Portability

The status/control register sets of the serial and datacomm interfaces are different (i.e., register numbers, functionality, etc). Unfortunately, this makes it difficult to write programs which can be run on either interface, or future interfaces which may not present the same status/control interface. Since RS-232 interfaces support a set of common primitives, portability can be enhanced by calling subprograms to perform these primitives rather than accessing status/control registers directly.

For example, all RS-232 interfaces should provide a mechanism for changing baud rate, number of stop bits, etc. When writing RS-232 programs in BASIC, use subprograms which determine the interface type, and access the appropriate status/control registers based on the interface type determined. Doing so will allow you to develop your code in a hardware independent fashion, with the details of how to communicate with a particular interface isolated to a few lines of code.

In addition to using subprograms, avoid the use of interface dependent features. For example, many of the status/control registers on the HP98628 implement functionality which does not exist on other RS-232 interfaces. For example, the

HP9828 CONTROL register 24 (character filter) is specific to the HP98628. Such functionality is normally not present on other RS-232 interfaces.

If you use subprograms to improve portability, unportable functionality should be apparent when you are unable to support a particular subprogram for all interfaces. In some cases it is reasonable to call a subprogram which does nothing for a particular interface (i.e., a nop), as long as a program does not depend on the behavior. For example, a subprogram to put an interface into asynchronous mode would do nothing for interfaces which support asynchronous mode only.

4 Below are two examples of subprograms which isolate the details of controlling a particular RS-232 interface. A BASIC program could use these subroutines without dependencies on the type of RS-232 interface actually in use.

```
!
! RESET_RS232
!
! Syntax:
!   Reset_rs232(Scd)
!   Scd - Select code
!
! Description:
!   This is used to reset the RS232 cards. It also set
!   some of the registers to reasonable defaults.
!
SUB Reset_rs232(Scd)
STATUS Scd,0;Id
IF Id=52 THEN
  RESET Scd
  CONTROL Scd,0;1
  CONTROL Scd,16;0           ! Connect timeout
  CONTROL Scd,17;0           ! No activity timeout
  CONTROL Scd,18;0           ! Lost carrier timeout
  CONTROL Scd,19;0           ! Transmit timeout
  CONTROL Scd,22;0           ! SW Handshaking
  CONTROL Scd,23;0           ! HW Handskaking
END IF
IF Id=2 OR Id=66 THEN
  RESET Scd
  CONTROL Scd,0;1
  CONTROL Scd,12;176         ! Turn off H/W Handshaking
END IF
SUBEND
```

```

!
! SET_PAR
!
! Syntax:
!   Set_par(Scd,Par)
!   Scd - Select code
!   Par - Parity 0 - none, 1 - odd, 2 - even
!
! Description:
!   This is used to set the parity for the RS232 cards.
!
SUB Set_par(Scd,Par)
STATUS Scd,0;Id
Found=0
IF Id=52 THEN
    Found=1
    CONTROL Scd,36;Par
END IF
IF Id=2 OR Id=66 THEN
    Found=1
    STATUS Scd,4;Stat
    SELECT Par
    CASE 0
        Reg_4=BINAND(Stat,7)    ! None 000XXX unset bits 5,4, and 3
    CASE 1
        Stat=BINAND(Stat,7)
        Reg_4=BINIOR(Stat,8)    ! Odd 001XXX set bit 3
    CASE 2
        Stat=BINAND(Stat,7)
        Reg_4=BINIOR(Stat,24)   ! Even 011XXX set bits 4 & 3
    CASE ELSE
        PRINT "ERROR: Invalid parity sent to Set_par()"
        STOP
    END SELECT
    CONTROL Scd,4;Reg_4
END IF
IF Found=0 THEN
    PRINT "ERROR: Unrecognized ID for select code."
END IF
SUBEND

```

## Sending BREAK Messages

A BREAK is a special character transmission that usually indicates a change in operating conditions. Interpretation of break messages varies with the application. To send a break message, send a non-zero value to Control Register 1 as follows (Sc is the interface select code):

```
CONTROL Sc,1;1      ! Send a BREAK to peripheral.
```

## Using the Modem Control Register

Control Register 5 controls various functions related to modem operation. Bits 0 thru 3 control modem lines, and bit 4 enables a self-test loopback configuration.

### Modem Handshake Lines (RTS and DTR)

As explained earlier in this chapter, Request-to-send and Data Terminal Ready lines are set or cleared at the beginning and end of each OUTPUT or ENTER operation. In some cases, it may be advantageous or necessary to maintain either or both in an active state. This is done by setting bit 1 or 0 respectively in Control Register 5 as follows:

```
1650 CONTROL Sc,5;2      ! Set RTS line only and hold active.
1660 CONTROL Sc,5;1      ! Set DTR line only and hold active.
1670 CONTROL Sc,5;3      ! Set both RTS and DTR lines active.
1680 CONTROL Sc,5;0      ! Return to normal modem line handshake.
```

When RTS and/or DTR are set by Control Register 5, they are *not* toggled during OUTPUT or ENTER operations, but remain constantly in an active state until the CONTROL register is cleared by:

- writing a different value to CONTROL Register 5
- an interface reset to CONTROL Register 0
- an interface reset ((Reset)) from the keyboard ((Shift) Break) on an ITF keyboard, or (SHIFT)-(PAUSE) on a 98203 keyboard).

### **Programming the DRS Modem Line for BASIC/UX**

Bit 2 of Control Register 5 controls the present state of the Data Rate Select (DRS). When bit 2 is set, the modem line is activated. When bit 2 is cleared, the modem line is cleared. To set the DRS line, the following statement or its equivalent can be used:

```
1690 CONTROL Sc,5;4 ! Sets the DRS line.
```

This line is also cleared by a CONTROL statement to Control Register 5 with bit 2 cleared, or by an interface reset.

### **Programming the DRS and SRTS Modem Lines for BASIC/WS**

4

Bits 3 and 2 of Control Register 5 control the present state of the Data Rate Select (DRS) and Secondary Request-to-send (SRTS) lines, respectively. When either bit is set, the corresponding modem line is activated. When the bit is cleared, so is the modem line. To set both lines, the following statement or its equivalent can be used:

```
1690 CONTROL Sc,5;8+4 ! Set DRS and SRTS lines.
```

These lines are also cleared by a CONTROL statement to Control Register 5 with bits 2 and 3 cleared, or by an interface reset.

### **Configuring the Interface for BASIC/WS Self-test Operations**

Self-test programs can be written for the serial interface. Prior to testing the interface, it must be properly configured. Using bit 4 of Control Register 5, you can rearrange the interconnections between input and output lines on the interface, enabling the interface to feed outbound data to the inbound circuitry.

When LOOPBACK is enabled (bit 4 is set), the UART output is set to its MARK state and sent to the Transmitted Data (TxD) line. The output of the transmitter shift register is then connected to the input of the receiver shift register, causing outbound data to be looped back to the receiver. In addition, the following modem control lines are connected to the indicated modem status lines:

#### Modem Control Line/Modem Status Line Connections

Modem Control Line	Modem Status Line
DTR Data Terminal Ready	DSR Data Set Ready
RTS Request-to-send	CTS Clear-to-send
SRTS Secondary RTS	DCD Data Carrier Detect
DRS Data Rate Select	RI Ring Indicator

4

When loopback is active, receiver and transmitter interrupts are fully operational. Modem control interrupts are then generated by the modem control outputs instead of the modem status inputs. Refer to serial interface hardware documentation for information about card hardware operation.

---

## READIO and WRITEIO Registers for BASIC/WS

For those cases where you need to write special interface driver routines, the interface card provides registers that can be accessed by use of READIO and WRITEIO statements. These capabilities are intended for use by experienced programmers who understand the inherent programming complexities that accompany this versatility.

Some registers are read/write; that is, both READIO and WRITEIO operations can be performed on a given register. Writing places a new value in the register; a read operation returns the current value. All registers have 8 bits available, and accept values from 0 through 255 unless noted otherwise. When the value of a given bit is 1, the bit is set; otherwise, it is zero (cleared or inactive).



---

**Note**

Some READIO and WRITEIO registers are similar in *structure and function* to Status and Control Registers. However, their interaction with the BASIC operating system is *considerably different*. To prevent incorrect program operation, do *not* intermix the use of STATUS/CONTROL registers and READIO/WRITEIO registers in a given program.

---

## Interface Hardware Registers

READIO and WRITEIO registers 1, 3, 5, and 7 access interface registers. Their functions are as follows:

*READIO Register 1*      Interface ID

This register returns the interface ID value: 2 for the HP 98626 Serial Interface; 66 for the HP 98644 interface.

*WRITEIO Register 1*      Interface Reset

Writing any value to this register, 1 thru 255, resets the interface as when using a CONTROL statement to Control Register 0.

*READIO Register 3*      Interrupt Status

Only the upper four bits of Register 3 are used. Bit 7 returns the current interrupt enable value. Bit 6 is set when an interrupt request is originated by the UART. (No interrupt can occur unless bit 7, Interrupt Enable, is set by a WRITEIO statement.)

Bits 5 and 4 return the setting of the Interrupt Level switches on the interface. With 98644 interfaces (which have no interrupt level switches), this register always indicates an interrupt level of 5. Their values are as follows:

- 00 Interrupt Level 3
- 01 Interrupt Level 4
- 10 Interrupt Level 5
- 11 Interrupt Level 6

*WRITEIO Register 3*

Interrupt Enable

Only bit 7 can be affected by WRITEIO statements. Writing a 1 into this bit enables interrupts, while a 0 disables them.

*READIO Register 5*

Optional Circuit and Baud Rate Status

READIO returns current states of the optional circuit drivers, plus the following:

- Bit 5 Optional Circuit Receiver 2 state.
- Bit 4 Optional Circuit Receiver 3 state.
- Bits 3-0 Current Baud Rate switch setting (not necessarily the current UART baud rate) as shown in the following table.

**Baud Rate Switch Settings**

Baud Rate	Switch Settings	Baud Rate	Switch Settings
	3 2 1 0		3 2 1 0
50	0 0 0 0	1200	1 0 0 0
75	0 0 0 1	1800	1 0 0 1
110	0 0 1 0	2400	1 0 1 0
134.5	0 0 1 1	3600	1 0 1 1
150	0 1 0 0	4800	1 1 0 0
200	0 1 0 1	7200	1 1 0 1
300	0 1 1 0	9600	1 1 1 0
600	0 1 1 1	19200	1 1 1 1

*WRITEIO Register 5*

**Optional Circuit and Baud Rate Control**

WRITEIO to bits 7 and 6 control the state of optional circuit drivers 3 and 4, respectively.

Bits 3-0      WRITEIO to this register *cannot* be used to set the baud rate. (Use Register 23, bit 7 and Registers 17 and 19 instead.)

*READIO Register 7*

**Line Control Switch Monitor**

READIO to this register enables you to input the present settings of the Line Control switches that preset default character format and parity. Bit functions are included in the table earlier in this chapter under Using Interface Defaults to simplify programming. Bits 7 thru 0 correspond to switches 7 thru 0, respectively. Since the 98644 interface does not have these switches, READIO of this register will be meaningless.

*WRITEIO Register 7*

WRITEIO operations to this register are meaningless.

**UART Registers**

Registers 17 through 29 access UART registers. They are used to directly control certain UART functions. The function of Registers 17 and 19 are determined by the state of bit 7 of Register 23.

*READIO Register 17*

Receive Buffer/Transmitter Holding Register

When bit 7 of Register 23 is clear (0), this register accesses the single-character receiver buffer by use of READIO.

The receiver and transmitter are doubly buffered. When the transmitter shift register becomes empty, a character is transferred from the holding register to the shift register. You can then place a new character in the holding register while the preceding character is being transmitted. Incoming characters are transferred to the receiver buffer when the receiver shift register becomes full. You can then input the character (READIO) while the next character is being constructed in the shift register.

*WRITEIO Register 17*

Receive Buffer/Transmitter Holding Register

A WRITEIO statement places a character in the transmitter holding register.

*READIO/WRITEIO  
Registers 17 and 19*

Baud Rate Divisor Latch

When bit 7 of Register 23 is set (1), Registers 17 and 19 access the 16-bit divisor latch used by the UART to set the baud rate. Register 17 forms the lower byte; Register 19 the upper. The baud rate is determined by the following relationship:

$$\text{Baud Rate} = 153\,600 / \text{Baud Rate Divisor}$$

To access the Baud Rate Divisor latch, set bit 7 of Register 23. This disables access to the normal functions of Registers 17 and 19, but preserves access to the other registers. When the proper value has been placed in the latch, be sure to clear bit 7 of Register 23 to return to normal operation.

*READIO Register 19*

Interrupt Enable Register

When bit 7 of Register 23 is clear (0), this register enables the UART to interrupt when specified

conditions occur. Only bits 0 thru 3 are used. Interrupt enable conditions are as follows:

- Bit 3      **Enable Modem Status Change Interrupts**, when set, enables an interrupt whenever a modem status line changes state as indicated by Register 29, bits 0 thru 3.
- Bit 2      **Enable Receiver Line Status Interrupts**, when set, enables interrupts by errors, or received BREAKs as indicated by Register 27, bits 1 thru 4.
- Bit 1      **Enable Transmitter Holding Register Empty Interrupt**, when set, allows interrupts when bit 5 of Register 27 is also set.
- Bit 0      **Enable Receiver Buffer Full Interrupts**, when set, enables interrupts when bit 0 of Register 27 is also set.

4

*WRITEIO Register 19*      Interrupt Enable Register

When bit 7 of Register 23 is clear (0), this register enables the UART to interrupt when specified conditions occur. Only bits 0 thru 3 are used. WRITEIO establishes a new value for each bit. Interrupt enable conditions are described in the preceding explanation of READIO register 19.

*READIO Register 21*      Interrupt Identification Register

This register identifies the cause of the highest-priority, currently-pending interrupt. Only bits 2, 1, and 0 are used. Bit 0, if set, indicates no interrupt pending. Otherwise an interrupt is pending as defined by bits 2 and 1. Causes of pending interrupts in order of priority are as follows:

**Bits 2&1    Interrupt Cause**

- 11        **Receiver Line Status interrupt** (highest priority) is caused when bit 2 of Register 19 is set and a framing, parity, or overrun error, or a BREAK is detected by the receiver (indicated by bits 1 thru 4 of Register 27). The interrupt is cleared by reading Register 27.
- 10        **Receive Buffer Register Full interrupt** is generated when bit 0 of Register 19 is set and the Data Ready bit (bit 0) of Register 27 is active. To clear the interrupt, read the receiver buffer, or write a zero to bit 0 of Register 27.
- 01        **Transmitter Holding Register Empty interrupt** occurs when bit 1 of Register 19 is set and bit 5 of Register 27 is set. The interrupt is cleared by writing data into the transmitter holding register (Register 17 with bit 7 of Register 23 clear) with a WRITEIO statement, or by reading this register (Interrupt Identification).
- 00        **Modem Line Status Change interrupt** occurs when bit 3 of Register 19 is set and a modem line change is indicated by one or more of bits 0 thru 3 of Register 29. To clear the interrupt, read Register 29 which clears the status change bits.

*READIO/WRITEIO  
Register 23*

**Character Format Control Register**

This register is functionally equivalent to Control and Status Register 4 except for bits 6 and 7. WRITEIO sets a new character format; READIO returns the current character format setting. Since the 98644 interface does not have these switches, READIO of bits 5 through 0 of this register will be meaningless.

**Bit 7**      **Divisor Latch Access Bit**, when set, enables you to access the divisor latches of the Baud Rate generator during read/write operations to registers 17 and 19.

**Bit 6**      **Set BREAK**, when set, holds the serial line in a BREAK state (always zero), independent of other transmitter activity. *This bit must be cleared to disable the break and resume normal activity.*

**Bits 5,4**    **Parity Sense** is determined by both bits 5 and 4. When bit 5 is set, parity is always ONE or ZERO. If bit 5 is not set, parity is ODD or EVEN as defined by bit 4. The combinations of bits 5 and 4 are as follows:

- 00    ODD parity
- 01    EVEN parity
- 10    Always ONE
- 11    Always ZERO

**Bit 3**      **Parity Enable**, when set, sends a parity bit with each outbound character, and checks all incoming characters for parity errors. Parity is defined by bits 4 and 5.

**Bits 2,1&0** **Stop Bit(s)** are defined by a combination of bit 2 and bits 1 & 0.

Bit 2	Character Length	Stop Bits
0	5, 6, 7, or 8	1
1	5	1.5
1	6, 7, or 8	2

Bits 1&0 **Character Length** is defined as follows:

Bits 1&0	Character Length
00	5 bits
01	6 bits
10	7 bits
11	8 bits

4

*READIO/WRITEIO*  
*Register 25*

**Modem Control Register**

This is a READ/WRITE register. READIO returns current control register value. WRITEIO sets a new value in the register. This register is equivalent to interface Control Register 5.

Bits 7, 6, and 5 Not used.

Bit 4 **Loopback**, when set, enables a loopback feature for diagnostic testing. Serial line is set to MARK state, UART receiver is disconnected, and transmitter output shift register is connected to receiver input shift register. Modem line outputs and inputs are connected as follows: DTR to CTS, RTS to DSR, DRS to DCD, and SRTS to RI. Interrupts are enabled, with interrupts caused by modem control outputs instead of inputs from modem.

Bit 3 **Data Rate Select** controls the OCD1 driver output. 1=Active, 0=Disabled.

Bit 2 **Secondary Request-to-Send** controls the OCD2 driver output. 1=Active, 0=Disabled.



- Bit 1      **Request-to-Send** controls the RTS modem control line state. When bit 1=1, RTS is always active. When bit 1=0, RTS is toggled by the OUTPUT statement as described earlier in this chapter.
- Bit 0      **Data Terminal Ready** holds the DTR modem control line active when the bit is set. If not set, DTR is controlled by the OUTPUT or ENTER statement as described earlier.

*READIO Register 27*

Line Status Register

- Bit 7      Not used.
- Bit 6      **Transmitter Shift Register Empty** indicates no data present in transmitter shift register.
- Bit 5      **Transmitter Holding Register Empty** indicates no data present in transmitter holding register. The bit is cleared whenever a new character is placed in the register.
- Bit 4      **Break Indicator** indicates that the received data input remained in the spacing (line idle) state for longer than the transmission time of a full character frame. This bit is cleared when the line status register is read.
- Bit 3      **Framing Error** indicates that a character was received with improper framing; that is, the start and stop bits did not conform with expected timing boundaries.
- Bit 2      **Parity Error** indicates that the received character did not have the expected parity sense. This bit is cleared when the register is read.

*READIO Register 29*

- Bit 1      **Overrun Error** indicates that a character was destroyed because it was not read from the receiver buffer before the next character arrived. This bit is cleared by reading the line status register.
- Bit 0      **Data Ready** indicates that a character has been placed in the receiver buffer register. This bit is cleared by reading the receiver buffer register, or by writing a zero to this bit of the line status register.
- Modem Status Register
- Bit 7      **Data Carrier Detect**, when set, indicates DCD modem line is active.
- Bit 6      **Ring Indicator**, if set, indicates that the RI modem line is active.
- Bit 5      **Data Set Ready**, if set, indicates that the DSR modem line is active.
- Bit 4      **Clear-to-send**, if set, indicates that CTS is active.
- Bit 3      **Change in Carrier Detect**, when set, indicates that the DCD modem line has changed state since the last time the modem status register was read.
- Bit 2      **Trailing Edge of Ring Indicator** is set when the RI modem line changes from active to inactive state.
- Bit 1      **Delayed Data Set Ready** is set when the DSR line has changed state since the last time the modem status register was read.
- Bit 0      **Change in Clear-to-send**, if set, indicates that the CTS modem line has changed

state since the last time the register was read.

---

## Cable Options and Signal Functions

The HP 98626A Serial Interface is available with RS-232C DTE and DCE cable configurations. The DTE cable option consists of a male RS-232C connector and cable designed to function as Data Terminal Equipment (DTE) when used with the serial interface. This means that the cable and connector are wired so that signal paths are correctly routed when the cable is connected to a peripheral device wired as Data Communication Equipment (DCE), such as a modem. The cables are designed so that you can write programs that work for both DCE and DTE connections without requiring modifications to accommodate equipment changes.

The DCE cable option includes a female connector and cable wired so that the interface and cable behave like normal DCE. This means that signals are routed correctly when the female cable connector is connected to a male DTE connector.

Line printers and other peripheral devices that use RS-232C interfacing are frequently wired as DTE with a female RS-232C chassis connector. This means that if you use a male (DTE) cable option to connect to the female DTE device connector, no communication can take place because the signal paths are incompatible. To eliminate the problem, use an adapter cable to convert the female RS-232C chassis connector to a cable connector that is compatible with the male or female interface cable connector. The HP 13242 adapter cable is available in various configurations to fit most common applications. Consult cable documentation to determine which adapter cable to use.

### The DTE Cable

The signals and functions supported by the DTE cable are shown in the signal identification table which follows. The table includes RS-232C signal identification codes, CCITT V.24 equivalents, the pin number on the interface card rear panel connector, the RS-232C connector pin number, the signal

mnemonic used in this manual, whether the signal is an input or output signal, and its function.

**RS-232C DTE (Male) Cable Signal Identification Table**

RS-232C Signal	V.24 Signal	Interface Pin#	RS-232C Pin#	Mnemonic	I/O	Function
AA	101	24	1	—	—	Safety Ground
BA	103	12	2	Out		Transmitted Data
BB	104	42	3	In		Received Data
CA	105	13	4	RTS	Out	Request to Send
CB	108	44	5	CTS	In	Clear to Send
CC	107	45	6	DSR	In	Data Set Ready
AB	102	48	7	—	—	Signal Ground
CF	109	46	8	DCD	In	Data Carrier Detect
SCF (OCR2)	122	47	12	SDCD	In	Secondary DCD
SCA (OCD2)	120	15	19	SRTS	Out	Secondary RTS
CD	108.1	14	20	DTR	Out	Data Terminal Ready
CE (OCR1)	125	9	22	RI	In	Ring Indicator
CH (OCD1)	111	40	23	DRS	Out	Data Rate Select

4

## Optional Circuit Driver/Receiver Functions

Not all signals from the interface card are included in the cable wiring. RS-232C provides for four optional circuit drivers and two receivers. Only two drivers and two receivers are supported by the DCE and DTE cable options. They are as follows:

### Drivers

Name	Function
OCD1	Data Rate Select
OCD2	Secondary Request-to-send
OCD3	Not used
OCD4	Not used

4

### Receivers

Name	Function
OCR1	Ring Indicator
OCR2	Secondary Data Carrier Detect

If your application requires use of OCD3 or OCD4, you must provide your own interface cable to fit the situation.

## The DCE Cable

The DCE cable option is designed to adapt a DTE cable and serial or data communications interface to an identical interface on another desktop computer. It is also used with the serial interface to simulate DCE operation when driving a peripheral wired for DTE operation. The DCE cable is equipped with a female connector. Since most DTE peripherals are also equipped with female connectors (pin numbering is the same as the standard male DTE connector), an adapter (such as the HP 13242M) is used to connect the two female connectors as explained earlier.

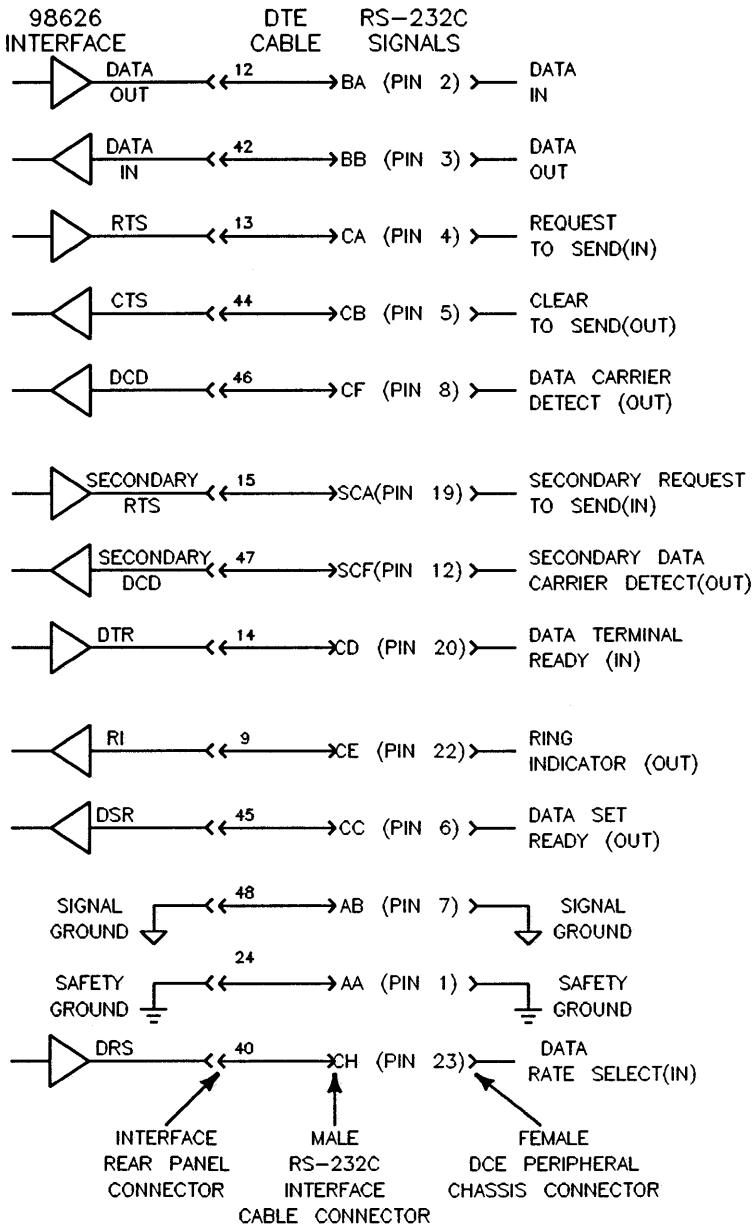
---

**Note**

Not all RS-232C devices are wired the same. To ensure proper operation, you must know whether the peripheral device is wired as DTE or DCE. The interface cable option and associated adapter cable, if needed, must be configured to properly mate with the female DTE chassis connector.

---

The following schematic diagram shows the input and output signals for the serial interface and how they are connected to a DCE peripheral.



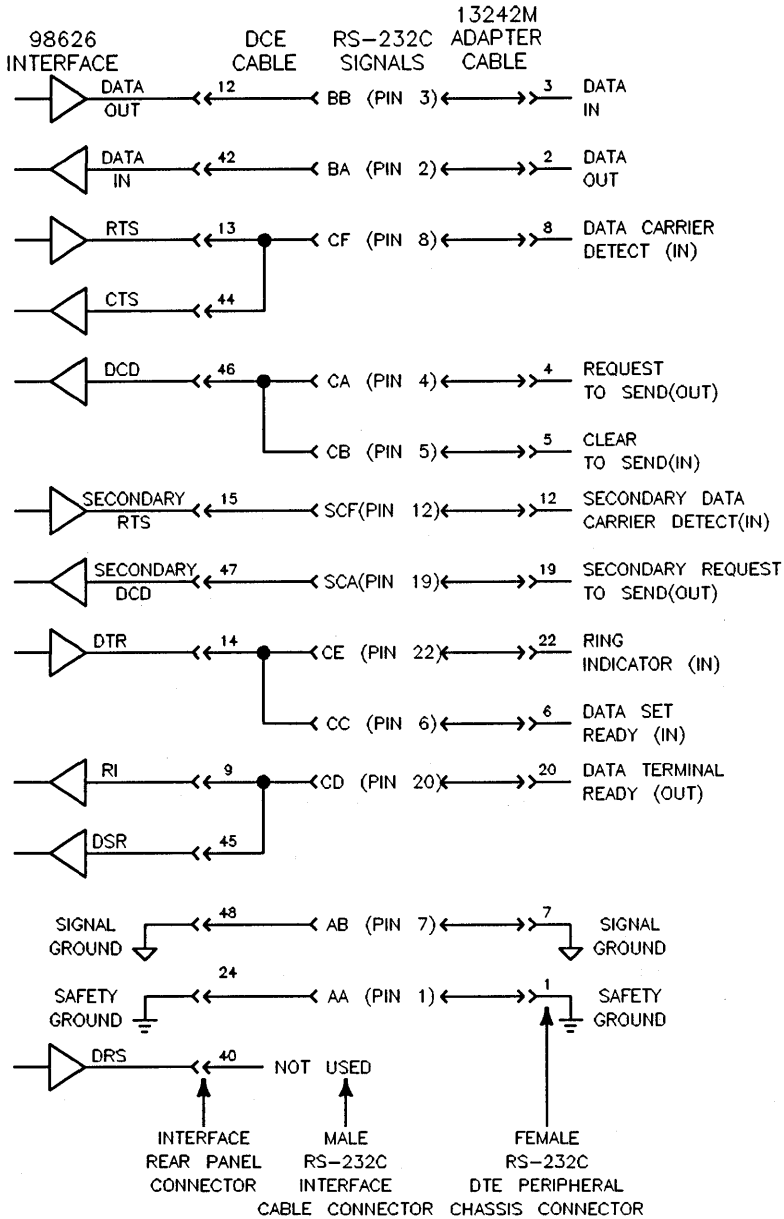
DCE Interface  
Signals to and  
from Peripheral

NOTE: Some DCE  
peripherals may not  
provide for all the  
signal lines shown.

**DTE Cable Interconnection Diagram**

This diagram shows an HP 13242M adapter cable connected to a DCE interface cable and a DTE peripheral. Note that RTS is connected to CTS in the DCE cable. If your peripheral uses RTS/CTS handshaking, a different adapter cable must be used with the appropriate DTE or DCE interface cable option.





DCE Interface  
Signals to and  
from Peripheral

NOTE: Some DTE  
peripherals may not  
provide for all the  
signal lines shown.

**DCE Cable Interconnection Diagram**

## RS-232C / CCITT V.24

The following table provides information about each data communications interface function. The pin assignments are also shown. Not all functions provided by RS-232C standard are implemented. The functions denoted with a \* are implemented.

### RS-232C/CCITT V.24<sup>1</sup>

RS-232C	CCITT V.24	Signal Name
*Pin 1	101	<i>Protective Ground.</i> Electrical equipment frame and ac power ground.
*Pin 2 <sup>2</sup>	103	<i>Transmitted Data.</i> Data originated by the terminal to be transmitted via the sending modem.
*Pin 3 <sup>2</sup>	104	<i>Received Data.</i> Data from the receiving modem in response to analog signals transmitted from the sending modem.
*Pin 4	105	<i>Request to Send.</i> Indicates to the sending modem that the terminal is ready to transmit data.
*Pin 5	106	<i>Clear to Send.</i> Indicates to the terminal that its modem is ready to transmit data.
*Pin 6	107	<i>Data Set Ready.</i> Indicates to the terminal that its modem is not in a test mode and that modem power is ON.
*Pin 7 <sup>2</sup>	102	<i>Signal Ground.</i> Establishes common reference between the modem and the terminal.
*Pin 8	109	<i>Data Carrier Detect.</i> Indicates to the terminal that its modem is receiving carrier signals from the sending modem.
Pin 9		Reserved for test.
Pin 10		Reserved for test.

<sup>1</sup> International Telephone and Telegraph Consultative Committee European standard.

<sup>2</sup> Signal on this pin is commonly used for three-wire (no modem) links.

### RS-232C/CCITT V.24 (continued)

RS-232C	CCITT V.24	Signal Name
Pin 11		Unassigned.
*Pin 12	122	<i>Secondary Data Carrier Detect.</i> Indicates to the terminal that its modem is receiving secondary carrier signals from the sending modem.
Pin 13	121	<i>Secondary Clear to Send.</i> Indicates to the terminal that its modem is ready to transmit signals via the secondary channel.
Pin 14	118	<i>Secondary Transmitted Data.</i> Data from the terminal to be transmitted by the sending modem's channel.
*Pin 15	114	<i>Transmitter Signal Element Timing.</i> Signal from the modem to the transmitting terminal to provide signal element timing information.
Pin 16	119	<i>Secondary Received Data.</i> Data from the modem's secondary channel in response to analog signals transmitted from the sending modem.
*Pin 17	115	<i>Receiver Signal Element Timing.</i> Signal to the receiving terminal to provide signal element timing information.
Pin 18		Unassigned.
*Pin 19	120	<i>Secondary Request to Send.</i> Indicates to the modem that the sending terminal is ready to transmit data via the secondary channel.
*Pin 20	108.2	<i>Data Terminal Ready.</i> Indicates to the modem that the associated terminal is ready to receive and transmit data.
Pin 21	110	<i>Signal Quality Detector.</i> Signal from the modem telling whether a defined error rate in the received data has been exceeded.

### RS-232C/CCITT V.24 (continued)

RS-232C	CCITT V.24	Signal Name
*Pin 22	125	<i>Ring Indicator.</i> Signal from the modem indicating that a ringing signal is being received over the line.
*Pin 23	111	<i>Data Signal Rate Selector.</i> Selects one of two signaling rates in modems having two rates.
*Pin 24	113	<i>Transmit Signal Element Timing.</i> Transmit clock provided by the terminal.
Pin 25		Unassigned.

4

---

## HP 98626 and HP 98644 Serial Interface STATUS and CONTROL Registers

Most Control registers accept values in the range of zero through 255. Some registers accept only specified values as indicated, or higher values for baud rate settings. Values less than zero are not accepted. Higher-order bits not needed by the interface are discarded if the specified value exceeds the valid range.

Reset value is the default value used by the interface after a reset or power-up until the value is overridden by a CONTROL statement.

The STATUS and CONTROL register information contained in this section applies only to BASIC/UX. BASIC/WS and BASIC/DOS support additional STATUS and CONTROL registers not covered in this section.

### *STATUS Register 0*

#### Card Identification

Value returned: 2 indicates a 98626 (if 130 is returned, the Remote jumper wire has been removed from the interface card); 66 indicates a 98644 (194 if the Remote jumper has been removed).

*CONTROL Register 0*

Interface Reset

Any value from 1 thru 255 resets the card. Execution is immediate; any data transfers in process are aborted and any buffered data is destroyed. A value of 0 causes no action.

*STATUS Register 1*

Interrupt Status

Bit 7 set: Interface hardware interrupt to CPU enabled.

Bit 6 set: Card is requesting interrupt service.

Bits 5&4:

00      Interrupt Level 3

01      Interrupt Level 4

10      Interrupt Level 5

11      Interrupt Level 6

Bits 3 through 0 not used.

*CONTROL Register 1*

Send BREAK

Any non-zero value causes a BREAK to be sent.

*STATUS Register 2*

Interface Activity Status

Bit 7 thru 4 are not used.

Bit 3 set      Error condition. Handshake ended with an escape.

Bit 2 set      Handshake in progress. This occurs only during multi-line function calls.

Bit 1 set      Firmware interrupts enabled (ENABLE INTR active for this select code).

Bit 0 set      TRANSFER in progress.

*STATUS Register 3*

Current Baud Rate

Returns one of the values listed under CONTROL Register 3.

*CONTROL Register 3*

Set New Baud Rate

Use any one of the following values:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19200

From 25 to 28800, the value will be rounded. Any value outside this range gives an error.

*STATUS Register 4*

Current Character Format

See CONTROL Register 4 for function of individual bits.

*CONTROL Register 4*

Set New Character Format

**Character Format and Parity Settings for BASIC/UX**

Handshake (Bits 7&6)	Parity Sense <sup>1</sup> (Bits <sup>2</sup> 5&4)	Par. Enable (Bit <sup>2</sup> 3)	Stop Bits (Bit <sup>2</sup> 2)	Char. Length (Bits <sup>2</sup> 1&0)
00 no-op	00 ODD parity	0 Disabled	0 1 stop bit	00 5 bits/char
01 Xon/Xoff	01 EVEN parity	1 Enabled	1 2 stop bits	01 6 bits/char
Bidirectional	10 Unsupported			10 7 bits/char
10 Unsupported	11 Unsupported			11 8 bits/char
11 Handshake Disabled				

<sup>1</sup> Parity sense valid only if parity is enabled (bit 3=1). If parity is disabled, parity sense is meaningless.

<sup>2</sup> These bits correspond to equivalent switch settings on the HP 98626 and HP 98644 serial interface cards. A 1 is the same as set.

### Character Format and Parity Settings for BASIC/WS

Parity Sense <sup>1</sup> (Switches 5&4)	Parity Enable (Switch 3)	Stop Bits (Switch 2)	Character Length (Switches 1&10)
00 ODD parity	0 Disabled	0 1 stop bit	00 5 bits/char
01 EVEN parity	1 Enabled	1 1.5 stop bits (if 5 bits/char),	01 6 bits/char
10 Always ONE		or 2 stop bits	10 7 bits/char
11 Always ZERO		(if 6, 7, or 8 bits/char)	11 8 bits/char

Bits 6 and 7 are reserved for future use.

<sup>1</sup> Parity sense valid only if parity is enabled (bit 3=1). If parity is disabled, parity sense is meaningless.

*STATUS Register 5*      Current Status of Modem Control Lines  
Returns CURRENT line state values. See CONTROL Register 5 for function of each bit.

*CONTROL Register 5*      Set Modem Control Line States  
Sets Modem Control lines or interface state as follows:

- Bit 2 set      Set Data Rate Select modem line to active state.
- Bit 1 set      Force Request-to-Send modem line to fixed active state.
- Bit 1 clear    Toggle RTS line as in normal OUTPUT operations.
- Bit 0 set      Force Data Terminal Ready modem line to fixed active state.
- Bit 0 clear    Toggle DTR line as in normal OUTPUT and ENTER operations.

*STATUS Register 6*

Data In

Reads character from input buffer. Buffer contents is not destroyed, but bit 0 of STATUS Register 10 is cleared.

*CONTROL Register 6*

Data Out

Sends character to transmitter holding register. This register is sometimes used to transmit protocol control characters or other characters without using OUTPUT statements. Modem control lines are not affected.

4

*STATUS Register 7*

Optional Receiver/Driver Status

Returns current value of optional circuit drivers or receivers as follows:

Bit 3 Optional Circuit Driver 3 (OCD3).

Bit 2 Optional Circuit Driver 4 (OCD4).

Bit 1 Optional Circuit Receiver 2 (OCR2).

Bit 0 Optional Circuit Receiver 3 (OCR3).

Other bits are not used (always 0).

*CONTROL Register 7*

Set New Optional Driver States

Sets (bit=1) or clears (bit=0) optional circuit drivers as follows:

Bit 3 Optional Circuit Driver 3 (OCD3),

Bit 2 Optional Circuit Driver 4 (OCD4).

Other bits are not used.

*STATUS Register 8*

Current Interrupt Enable Mask

Returns value of interrupt mask associated with most recent ENABLE INTR statement. Bit functions are as follows:



- Bit 3      Enable interrupt on modem line change. STATUS Register 11 shows which modem line has changed.
- Bit 2      Enable interrupt on UART status error. This bit is used to trap ERROR 167 caused by UART error conditions. STATUS Register 10, bits 4 thru 1, show cause of error.
- Bit 1      Enable interrupt when Transmitter Holding Register is empty (supported only on BASIC/WS).
- Bit 0      Enable interrupt when Receiver Buffer is full (supported only on BASIC/WS).

4

*STATUS Register 9*

Cause of Current Interrupt

Returns cause of interrupt as follows:

Bits 2&1 Return cause of interrupt

- 11    UART error (BREAK, parity, framing, or overrun error). See STATUS Register 10.
- 10    Receiver Buffer full. Cleared by STATUS to Register 6. (Supported only on BASIC/WS.)
- 01    Transmitter Holding Register empty. Cleared by CONTROL Register 6 or STATUS to Register 9. (Supported only on BASIC/WS.)
- 00    Interrupt caused by change in modem status line(s). See STATUS Register 11.

- Bit 0      Set when no active interrupt requests from UART are pending. Clear until all pending interrupts have been serviced.

*STATUS Register 10*

UART Status

Bit set indicates UART status or detected error as follows:

- Bit 7 Not used.
- Bit 6 Transmit Shift Register empty.
- Bit 5 Transmit Holding Register empty.
- Bit 4 Break received.
- Bit 3 Framing error detected.
- Bit 2 Parity error detected.
- Bit 1 Receive Buffer Overrun error.
- Bit 0 Receiver Buffer full.

*STATUS Register 11*

Modem Status

Bit set indicates that the specified modem line or condition is active, and bit clear indicates that the specified modem line is not active. The default settings are given with each bit.

- Bit 7 Data Carrier Detect (DCD) modem line active.
- Bit 6 Ring Indicator (RI) modem line active.
- Bit 5 Data Set Ready (DSR) modem line active.
- Bit 4 Clear-to-Send (CTS) modem line active.
- Bit 3 Change in DCD line state detected.
- Bit 2 RI modem line changed from true to false.
- Bit 1 Change in DSR line state detected.
- Bit 0 Change in CTS line state detected.

*STATUS Register 12*

Modem handshake status.

### Modem Handshake Control for BASIC/WS

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect Disable <sup>1</sup>	0	Data Set Ready Disable <sup>2</sup>	Clear to Send Disable <sup>3</sup>	0	0	0	0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*CONTROL Register 12* Modem handshake control.

### Modem Handshake Control for BASIC/WS

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect Disable <sup>1</sup>	Not Used	Data Set Ready Disable <sup>2</sup>	Clear to Send Disable <sup>3</sup>	Not Used	Not Used	Not Used	Not Used
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

<sup>1</sup> Wait for Carrier Detect on Enter Operations; 1=Don't wait. BASIC/UX supports bit 7 and bit 4 in combination only. See the "BASIC/UX Hardware Handshaking" section for more details.

<sup>2</sup> Wait for Data Set Ready on Enter and Output Operations; 1=Don't wait. BASIC/UX does not support bit 5.

<sup>3</sup> Wait for Clear to Send on Output Operations; 1=Don't wait. BASIC/UX supports bit 7 and bit 4 in combination only. See the "BASIC/UX Hardware Handshaking" section for more details.

### Interrupt Enable Register (ENABLE INTR)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Not Used	Not Used	Not Used	Modem Status Change	Receiver Line Status	Transmitter Holding Register Empty	Receiver Buffer Full
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

4

<sup>1</sup> Supported only on BASIC/WS.

*STATUS Register 13*

Read 98644 “SCRATCH A default” baud rate

Returns the baud rate that will be restored whenever SCRATCH A is executed (same bit-definitions as STATUS register 3).

*CONTROL Register 13*

Set 98644 “SCRATCH A default” baud rate

Sets both the “current” and the “default” baud rate that will be restored whenever SCRATCH A is executed (same bit-definitions as CONTROL register 3). Default value in this register is 9600 baud for BASIC/WS.

*STATUS Register 14*

Read 98644 “SCRATCH A default” character format

Returns the character format parameters that will be restored whenever SCRATCH A is executed (same bit-definitions as STATUS register 4).

*CONTROL Register 14*

Set 98644 “SCRATCH A default” character format

Sets both the “current” and the “default” character format parameters that will be restored whenever SCRATCH A is executed (same bit-definitions as CONTROL register 4). BASIC/WS default value in this register specifies a character format of 8 bits/character, 1 stop bit, and parity disabled.

---

## Model 216 and 217 Built-In Interface Differences for BASIC/WS

This section describes the differences between the HP 98626 Serial interface and the built-in Serial interface in the Model 216 (HP 9816) and 217 (HP 9817) Computers.

The hardware differences between the built-in serial interfaces and the 98626 interface occur in the following areas:

- There are no “Select Code” switches (the select code is hard-wired to 9).
- There are no “Interrupt Level” switches (the interrupt level is hard-wired to 3).
- There are no “Status Line Disconnect” switches (the modem status lines are always monitored; you *cannot* throw switches to make them “ALWAYS ON” like you can with the 98626 interface).

There are no differences between programming these two interfaces with the BASIC system.

---

## HP 98644 Interface Differences

The HP 98644 RS-232 Serial Interface is nearly identical to the HP 98626 RS-232 Serial Interface. This section describes the few differences between them.

## Hardware Differences

The differences in the hardware of the two cards occur in the following areas:

- Card ID register contains 66 (rather than 2) or 194 if the Remote jumper on the HP 98644 interface card has been removed.
- There are no optional driver and receiver lines.
- There are fewer configuration switches (there are no Baud Rate or Line Control switches).
- There is a 25-pin coverplate connector (instead of 50).
- There are different cables available.

### Card ID Register

The default card ID for the HP 98644 interface is 66, and the default card ID for the HP 98626 is 2.

---

#### Note



HP 98644 cards are logged as HP 98626 interfaces while booting machines with Boot ROM 3.0 (and earlier versions). This is not a problem, because the BASIC recognizes the 98644 card properly.

You can also change the card ID to 2 (to make it look like a 98626) by cutting a jumper on the card. See the 98644's installation manual for details.

---

See the following “BASIC Differences” section for details of how to read this register with software.

### Optional Driver Receiver Circuits

On the 98626 interface, there are two optional driver lines (OCD3 and OCD4) and two optional receiver lines (OCR2 and OCR3). These lines are *not* implemented on the 98644 interface.

## Configuration Switches

The HP 98644 card does not implement the following configuration switches on the card:

- Baud Rate
- Line Control (character length, parity, etc.)

These operating parameters are set in the same manner as the HP 98626 interface card. See the previous section “Serial Configuration for HP-UX” for details.

## Coverplate Connector

The connector on the HP 98644 interface’s coverplate is set up for DTE (Data Terminal Equipment) applications; it has a 25-pin, female, D-series connector (the connector on the HP 98626 is a 50-pin connector). The pin designators for the connector follow.

### Coverplate Connector Pin Designators

Pin	Signal Description
1	Safety Ground
2	Transmitted Data
3	Received Data
4	Request to Send
5	Clear to Send
6	Data Set Ready
7	Signal Ground
8	Carrier Detect
9	not used
10	not used
11	not used
12	not used
13	not used
14	not used
15	not used
16	not used
17	not used
18	not used
19	not used
20	Data Terminal Ready
21	not used
22	Ring Indicator
23	Data Rate Select
24	not used
25	not used



## Cables

You can use standard RS-232C compatible cables, as long as the signal lines are connected properly. Here are cables available from HP Computer Supplies Operation.

### Available RS-232C-Compatible Cables

HP Product Number	Description
13242N	Modem cable (male to male)
13242G	DTE cable (male to male, with pins 2 and 3 reversed)
13242H	DCE cable (male to female, with pins 2 and 3 reversed)

4

## BASIC Differences

The only differences between programming these two interfaces with the BASIC system are in the register definitions given in this section. See the "Summary of RS-232 Serial STATUS and CONTROL Registers" section for further details.

### Card ID Register

The card ID register is Status register 0. It will contain a value of 66 if the interface is a 98644. (It will contain 2 if the card ID jumper has been cut.) If the REMOTE jumper has been removed, then the value returned will be 194 (=128+66) or 130 (=128+2). For BASIC/WS, the card ID can also be determined by reading READIO Register 1.

### Optional Driver/Receiver Registers for BASIC/WS

Since there are no optional driver or receiver lines on the 98644 interface, Status and Control register 7 are meaningless for this card. (Status register 7 always contains 0, and Control register 7 is a no-op.)

The hardware register bits that are *not* defined because of this difference are as follows: bits 7 and 6 of WRITEIO register 5 (for writing OCD3 and OCD4, respectively); bits 7 and 6 of READIO register 5 (for reading OCD3 and

OCD4, respectively); bits 5 and 4 of READIO register 5 (for reading OCR2 and OCR3, respectively).

### **Baud-Rate and Line-Control Registers for BASIC/WS**

Since there are no switches to set the default baud rate and line control parameters, the BASIC system sets them to its own default values, which are as follows:

**Baud Rate and Line Control Default Values**

<b>Parameter</b>	<b>Default Value</b>
Baud rate	9600 baud
Character length	8 bits/character
Stop bits	1 stop bit
Parity	Parity disabled
Parity type	Odd parity

Status registers 3 (baud rate) and 4 (line control) are still implemented for the 98644 interface and retain their original definitions. However, the hardware registers no longer contain any baud rate and line control information (since there are no switches to read). The hardware registers affected are READIO register 5 (bits 3 thru 0) and READIO register 7 (bits 7 thru 0), respectively.

You can still program the baud rate and line control parameters by writing to Control register 3 (baud rate) and Control register 4 (character format). These registers correspond to WRITEIO register 5 (bits 3 thru 0) and register 23 (bits 5 thru 0), respectively.

---

## Series 300

### Built-In 98644 Interface Differences

The differences between the separate HP 98644 RS-232C serial interface and the built-in 98644-like interface of Series 300 computers are as follows:

- There are no “Select Code” switches (the select code is hard-wired to 9).
- There are no “Interrupt Level” switches (the interrupt level is hard-wired to 5).

There are no differences in programming these interfaces with the BASIC system.



## **Datacomm Interfaces**

---

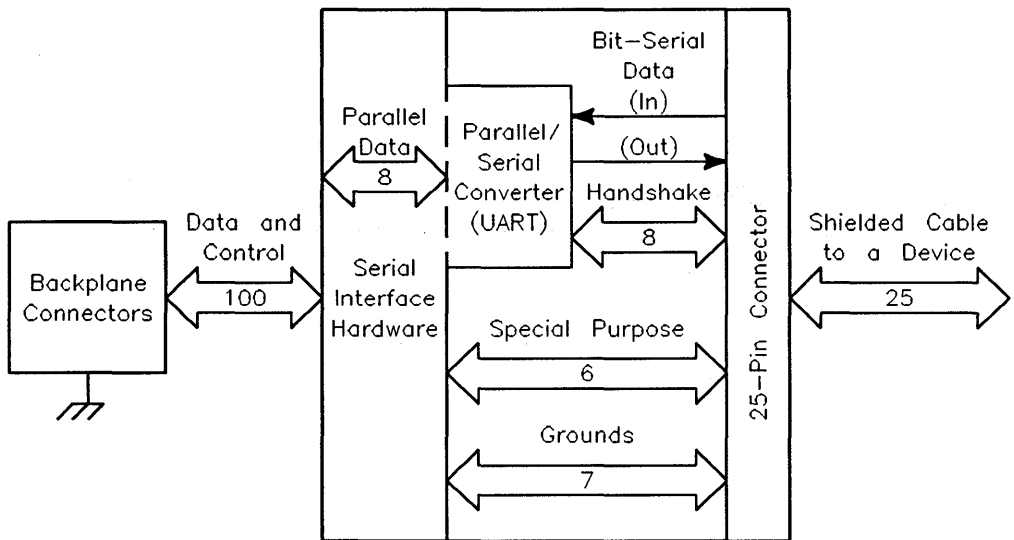
The HP 98628 and HP 98642 Data Communications Interfaces enable your desktop computer to communicate with any device that is compatible with standard asynchronous data communication protocols. Devices can include modems or equipment with standard RS-232C links. Because the HP 98628 and HP 98642 Data Communications Interface cards have only a few differences, this chapter will deal mainly with the HP 98628 interface card. Information on differences between these two data communications cards can be found in the section “The HP 98642 4-Channel Multiplexer.”

---

**Note**

The HP 98642 4-channel multiplexer is supported on BASIC/UX only.

---



**Block Diagram of the Datacomm Interface**

## Prerequisites

It is assumed that you are familiar with the information presented in Data Communication Basics, and that you understand data communication hardware well enough to determine your needs when configuring the datacomm link. Configuration parameters include such items as half/full duplex, handshake, and timeout requirements. If you have any questions concerning equipment installation or interconnection, consult the appropriate interface or adapter installation manuals.

The datacomm interface supports several cable and adapter options. They include:

- RS-232C Interface cable and connector wired for operation with data communication equipment (male cable connector) or with data terminal equipment (female cable connector).
- HP 13264A Data Link Adapter for use in HP 1000- or HP 3000-based Data Link network applications for BASIC/WS.

## 5-2 Datacomm Interfaces

- HP 13265A Modem for asynchronous connections up to 300 baud, including built-in autodial capability.

The HP 13265A modem is compatible with Bell 103 and Bell 113 Modems, and is approved for use in the USA and Canada. Most other countries do not allow use of user-owned modems. Contact your local HP Sales and Service office for information about local regulations.

- HP 13266A Current Loop Adapter for use with current loop links or devices.

Some of the information contained in this chapter pertains directly to certain of these devices in specific applications.

Before you begin datacomm operation, be sure all interfaces, cables, connectors, and equipment have been properly plugged in. Power must be on for all devices that are to be used. Consult applicable installation manuals if necessary.

---

## Protocol

Two protocols are switch selectable on the datacomm interface. They are also software selectable during normal program operation. The switch setting on the interface determines the default protocol when the computer is first powered up. Protocol is changed between Async and Data Link during program operation by selecting the new protocol, waiting for the message to reach the card, then resetting the card. The exact procedure is explained in **Protocol Selection**.

### Asynchronous Communication Protocol

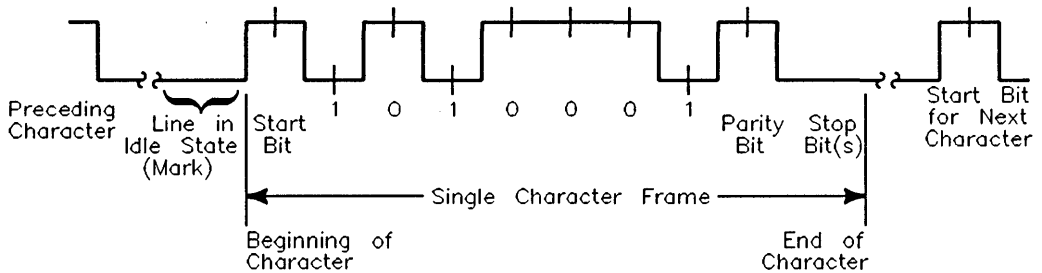
Asynchronous protocol is the only protocol supported by BASIC/UX. Asynchronous data communication is the most widely used protocol, especially in applications where high data integrity is not mandatory. Data is transmitted, one character at a time, with each character being treated as an individual message. Start and stop bits are used to maintain timing coordination between the receiver and transmitter. A parity bit is sometimes included to detect character transmission errors. Asynchronous character format is as follows: Each character consists of a start bit, 5 to 8 data bits, an

optional parity bit, and 1, 1.5, or 2 stop bits, with an optional time gap before the beginning of the next character. The total time from the beginning of one start bit to the beginning of the next is called a character frame.

Parity options include:

- NONE            No parity bit is included.
- ODD            Parity set if EVEN number of "1"s in character bits.
- EVEN           Parity set if ODD number of "1"s in character bits.
- ONE            Not supported.
- ZERO           Not supported.

Here is a simple diagram showing the structure of an asynchronous character and its relationship to previous and succeeding characters:



**Structure of Asynchronous Character**

## **Data Link Communication Protocol (BASIC/WS Only)**

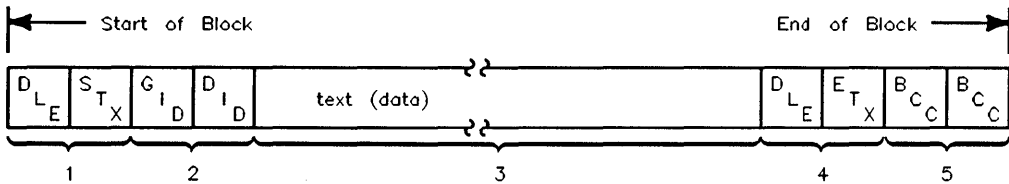
Data Link protocol overcomes the data integrity limitations of Async by handling data in blocks. Each block is transmitted as a stream of individual asynchronous characters, but protocol control characters and block check characters are also transmitted with the data. The receiver uses the protocol control characters to determine block boundaries and data format. Block check characters are used to detect transmission errors. If an error occurs, the block is usually retransmitted until it is successfully received. Block protocol and format is similar to Binary Synchronous Communication (BSC or Bisync, for short).



Data Link protocol provides for two transmission modes: transparent, and normal. In transparent mode, any data format can be transferred because datacomm control characters are preceded by a DLE character. If a control character is sent without an accompanying DLE, it is treated as data. When normal mode is used, only ASCII data can be sent, and datacomm control characters are not allowed in the data stream.

The HP 1000 and HP 3000 computers usually transmit in transparent mode. All transmissions from your desktop computer are sent as transparent data. If your application involves non-ASCII data transfers (discussed later in this chapter), be sure the HP 1000 or HP 3000 network host is using transparent mode for all transmissions to your computer.

Each data block sent to the network host by the datacomm interface is structured as follows:



**Structure of Data Block Sent by Datacomm Interface**

1. The “start transmission” control characters identify the beginning of valid data. If a DLE is present, the data is transparent; If absent, data is normal. All data from your desktop computer is transparent.
2. The terminal identification characters are included in blocks sent to the network host. Blocks received from the network host do not contain these two characters.
3. Data characters are transmitted in succession with no time lapse between characters.
4. The “end transmission” control characters identify the end of data. DLE ETX or DLE ETB indicate transparent data. ETX or ETB indicates normal data.
5. Block check characters (usually two characters) are used to verify data integrity. If the value received does not match the value calculated by

the receiver, the entire block is rejected by the receiver. Block check includes Group Identifier (GID) and Device Identifier (DID) characters in transmissions to the network host.

Protocol control characters are stripped from the data transfer, and are not passed from the interface to the computer. For information about network polling, terminal selection and other Data Link operations, consult the Data Link network manuals supplied with the HP 1000 or HP 3000 network host computer.

---

## Data Transfers Between Computer and Interface

5 Data transfers between your desktop computer and its datacomm interface involve two message types: control blocks and data. Control blocks contain information sent to and received from the interface regarding its operation. Data is sent to and received from a remote device through the interface. Control blocks are not sent to or received from remote devices. Both types are encountered in both output and input operations as follows:

- Outbound control blocks are created by CONTROL statements.
- Outbound data messages are created by OUTPUT statements.
- Inbound control blocks are created by certain protocol operations such as Data Link block boundaries, or Async prompt, end-of-line, parity/framing error, or break detection.
- Inbound data messages are created by the interface as messages are received from the remote. They are transferred to BASIC by ENTER statements.

### Outbound Control Blocks

Outbound control blocks are messages from your computer to the datacomm interface that contain interface control information. They are usually generated by CONTROL statements, although OUTPUT ... END creates a control block that terminates a given Async transmission or forces a block to be sent on the Data Link. Outbound control blocks are serially queued with data, and executed by the interface in the same order as created by BASIC. The single

exception to the queued control block rule is when a non-zero value is output to Control Register 0 (Interface Reset) which is executed immediately.

---

**Note**

When an interface card reset is executed by use of a CONTROL statement, the control block that results is transmitted directly to the interface. It is not queued up, so any previously queued data and control blocks are destroyed. To prevent loss of data, be sure that all queued messages have been sent before resetting the datacomm interface. Status Register 38 returns a value of 1 when the outbound queue is empty. Otherwise, its value is 0. To prevent loss of inbound data, Status Register 5 must return a value of zero prior to reset.

---

### **Inbound Control Blocks for BASIC/UX**

Inbound control blocks are messages from the interface to the computer that identify protocol control information. Refer to the *HP BASIC 6.2 Language Reference* for details about register contents for various control block types.

For Async applications, terminal emulator programs usually use prompt and end-of-line control blocks. Use of other functions such as break or error detection depend on the requirements of the individual application.

### **Inbound Control Blocks for BASIC/WS**

Inbound control blocks are messages from the interface to the computer that identify protocol control information. Which item(s) are allowed to create a control block is determined by the contents of Control Register 14, Status Registers 9 and 10 identify the contents of the block, and Control Register 24 defines what protocol characters are also included with inbound Async data messages. Refer to the *HP BASIC 6.2 Language Reference* for details about register contents for various control block types.

For Async applications, terminal emulator programs usually use prompt and end-of-line control blocks. Use of other functions such as break or error detection depend on the requirements of the individual application.

Two types of information are contained in each control block: type and mode. The type is contained in STATUS register 9; the mode in STATUS register 10. Type and Mode values can be used to interpret datacomm operation as follows:

### Async Protocol Control Blocks

Type	Mode	Interpretation
250	1	Break received (channel A).
251	1 <sup>1</sup>	Framing error in the following character.
251	2 <sup>1</sup>	Parity error in the following character.
251	3 <sup>1</sup>	Both Framing and Parity error in the following character.
252	1	End-of-line terminator detected.
253	1	Prompt received from remote.

5

<sup>1</sup> Parity/framing error control blocks are not generated when characters with parity and/or framing errors are replaced by an underscore (-) character.

### Data Link Protocol Control Blocks

Type	Mode	Interpretation
254	1	Preceding block terminated by ETB character.
254	2	Preceding block terminated by ETX character.
253 <sup>1</sup>		(See following table for Mode interpretation.)

Mode Bit(s)	Interpretation
0	1=Transparent data in following block. 0=Normal data in following block.
2,1	00=Device Select (most common). 01=Group Select 10=Line Select
3	1=Command Channel 0=Data Channel

<sup>1</sup> This type is used mainly in specialized applications. In most cases, you can expect a Mode value of zero or one for Type 253 Data Link control blocks. For most Data Link applications, control blocks are not used by programmers.

For Data Link applications, control blocks are normally set up for end-of-block (ETB or ETX). Control blocks are then used to terminate ENTER operations. Control block contents are not important for most applications unless you are doing sophisticated protocol-control programming.

For Async applications, terminal emulator programs usually use prompt and end-of-line control blocks. Use of other functions such as break or error detection depend on the requirements of the individual application.

## Outbound Data Messages

Outbound data messages are created when an OUTPUT statement is executed. Here is a short summary of how OUTPUT parameters can affect datacomm operation.

- Async protocol: Data is transmitted directly from the outbound queue. When operating in half-duplex, OUTPUT ... END causes the interface to turn the line around and allow the remote device to send information back (line turn-around is initiated when the interface sets the Request-to-send line low). OUTPUT ... END has no effect when operating in full duplex.
- Data Link protocol (BASIC/WS only): Data messages are concatenated until at least 512 characters are available, then a block of 512 characters is sent. Block boundaries may or may not coincide with the end of a given OUTPUT message.

You can force transmission of shorter blocks by using the `OUTPUT ... END` statement. The interface then transmits the last pending block regardless of its length. This technique is useful for ensuring that block boundaries coincide with message boundaries, or for sending one message string per block when you are transmitting short records.

- Unless a semicolon or `END` appears at the end of a free-field `OUTPUT` statement, an EOL sequence is automatically sent at the end of the data. The EOL sequence is also suppressed by using the appropriate `IMAGE` specifier in an `OUTPUT` statement. For further information, see the chapter called "Outputting Data."

## Inbound Data Messages

Inbound data messages are created by the datacomm interface as information is received from the remote. `ENTER` statements are terminated when a control block is encountered or the input variable is filled. For `BASIC/WS`, whether control characters are included in the data stream depends on the configuration of Control Register 24 (Async operation only). Control information is never included in inbound data messages when using Data Link protocol.

With this brief introduction to the data communications capabilities of the HP 98628 Datacomm Interface, you are ready to begin programming your desktop computer for datacomm operation. The next section of this chapter introduces `BASIC` datacomm programming techniques using simple terminal emulator examples that can be readily expanded into much more sophisticated datacomm programs.

---

## Overview of Datacomm Programming

Your desktop computer uses four BASIC statements for data communication with remote computers, terminals, and other peripheral devices. Datacomm programs include part or all of the following elements:

- CONTROL statements to configure the datacomm link and establish the connection.
- OUTPUT and ENTER statements to transfer information.
- STATUS statements to monitor operation.
- CONTROL statements to alter link parameters during the session, if needed for unusual applications.
- OUTPUT and ENTER statements to transfer additional information.
- A CONTROL statement to disconnect at the end of the session.

Here is a simple BASIC/WS example of an Async terminal emulator that uses default parameters. The user must disconnect at the end of a session by executing the command CONTROL Sc,12;0 from the keyboard.

```
1000      Sc=27                ! Datacomm on Select Code 27.
1010      CONTROL Sc,14;6      ! Set Control Block Mask.
1020      OUTPUT Sc;CHR$(13);  ! Datacomm interface uses defaults
1025                                ! and automatically connects to line.
1030      Check_reader:DIM A$[700] ! Up to 700 characters per line.
1040      STATUS Sc,5;Rx_avail_bits ! Get Rx queue status.
1050      IF Rx_avail_bits>1 THEN
1060          ENTER Sc USING "#,K";A$ ! Get data from queue.
1070          PRINT USING "#,K";A$    ! Print data.
1080          STATUS Sc,9;R          ! Get Control Block TYPE field.
1090          IF R=253 THEN
1100              LINPUT "Enter line to send to remote.";A$
1110              OUTPUT Sc;A$;CHR$(13);
1120          END IF
1130      END IF
1140      GOTO Check_reader
1150      END
```

While this program shows the relative simplicity of using your computer for data communication, most applications require more sophisticated techniques.

The following pages show more elaborate structures to illustrate some of the concepts used in creating programs for datacomm applications.

Two sample terminal emulator programs, one for Async and one for Data Link, are used in this chapter to show you how to write datacomm programs with a minimum of difficulty and complexity. Both versions are very similar; differences are explained fully. The emulators are explained in logical sequence, with complete program listings included at the end. The examples can be used as written, or expanded to include other features. They are designed to demonstrate program structures and programming techniques that are used in many data communication applications.

## RS-232 Software Portability

The status/control register sets of the serial and datacomm interfaces are different (i.e., register numbers, functionality, etc). Unfortunately, this makes it difficult to write programs which can be run on either interface, or future interfaces which may not present the same status/control interface. Since RS-232 interfaces support a set of common primitives, portability can be enhanced by calling subprograms to perform these primitives rather than accessing status/control registers directly.

For example, all RS-232 interfaces should provide a mechanism for changing baud rate, number of stop bits, etc. When writing RS-232 programs in BASIC, use subprograms which determine the interface type, and access the appropriate status/control registers based on the interface type determined. Doing so will allow you to develop your code in a hardware independent fashion, with the details of how to communicate with a particular interface isolated to a few lines of code.

In addition to using subprograms, avoid the use of interface dependent features. For example, many of the status/control registers on the HP98628 implement functionality which does not exist on other RS-232 interfaces. For example, the HP9828 CONTROL register 24 (character filter) is specific to the HP98628. Such functionality is normally not present on other RS-232 interfaces.

If you use subprograms to improve portability, unportable functionality should be apparent when you are unable to support a particular subprogram for all interfaces. In some cases it is reasonable to call a subprogram which does nothing for a particular interface (i.e., a nop), as long as a program does



not depend on the behavior. For example, a subprogram to put an interface into asynchronous mode would do nothing for interfaces which support asynchronous mode only.

Below are two examples of subprograms which isolate the details of controlling a particular RS-232 interface. A BASIC program could use these subroutines without dependencies on the type of RS-232 interface actually in use.

```
!  
! RESET_RS232  
!  
! Syntax:  
!   Reset_rs232(Scd)  
!     Scd - Select code  
!  
! Description:  
!   This is used to reset the RS232 cards. It also set  
!   some of the registers to reasonable defaults.  
!  
SUB Reset_rs232(Scd)  
STATUS Scd,0;Id  
IF Id=52 THEN  
  RESET Scd  
  CONTROL Scd,0;1  
  CONTROL Scd,16;0      ! Connect timeout  
  CONTROL Scd,17;0      ! No activity timeout  
  CONTROL Scd,18;0      ! Lost carrier timeout  
  CONTROL Scd,19;0      ! Transmit timeout  
  CONTROL Scd,22;0      ! SW Handshaking  
  CONTROL Scd,23;0      ! HW Handshaking  
END IF  
IF Id=2 OR Id=66 THEN  
  RESET Scd  
  CONTROL Scd,0;1  
  CONTROL Scd,12;176    ! Turn off H/W Handshaking  
END IF  
SUBEND
```

```

!
! SET_PAR
!
! Syntax:
!   Set_par(Scd,Par)
!   Scd - Select code
!   Par - Parity 0 - none, 1 - odd, 2 - even
!
! Description:
!   This is used to set the parity for the RS232 cards.
!
SUB Set_par(Scd,Par)
STATUS Scd,0;Id
Found=0
IF Id=52 THEN
    Found=1
    CONTROL Scd,36;Par
END IF
IF Id=2 OR Id=66 THEN
    Found=1
    STATUS Scd,4;Stat
    SELECT Par
    CASE 0
        Reg_4=BINAND(Stat,7)    ! None 000XXX unset bits 5,4, and 3
    CASE 1
        Stat=BINAND(Stat,7)
        Reg_4=BINIOR(Stat,8)    ! Odd 001XXX set bit 3
    CASE 2
        Stat=BINAND(Stat,7)
        Reg_4=BINIOR(Stat,24)   ! Even 011XXX set bits 4 & 3
    CASE ELSE
        PRINT "ERROR: Invalid parity sent to Set_par()"
        STOP
    END SELECT
    CONTROL Scd,4;Reg_4
END IF
IF Found=0 THEN
    PRINT "ERROR: Unrecognized ID for select code."
END IF
SUBEND

```

---

## Establishing the Connection

### Determining Protocol and Link Operating Parameters

Before information can be successfully transferred between two devices, a communication link must be established. You must include the necessary protocol parameters to ensure compatibility between the communicating machines. To determine the proper parameters for your application, select Async or Data Link protocol, then answer the following questions:

*For BOTH Async and Data Link Operation:*

- Is a modem connection being used? What handshake provisions are required? (Data Link does not use modems, but multi-point Async modem connections use a protocol compatible with Data Link.)
- Is half-duplex or full-duplex line protocol being used?

*For Async Operation ONLY:*

- What line speed (baud rate) is being used for transmitting?
- What line speed is being used for receiving?
- How many bits (excluding start, stop, and parity bits) are included in each character?
- What parity is being used: none, odd, even, always zero, or always one?
- How many stop bits are required on each character you transmit?
- What line terminator should you use on each outgoing line?
- How much time gap is required between characters (usually 0)?
- What prompt, if any, is received from the remote device when it is ready for more data?
- What line terminator, if any, is sent at the end of each incoming line?

*For Data Link Operation ONLY:*

- What line speed (baud rate) is being used? (Data Link uses the same speed in both directions.)

- What parity is being used: none (HP 1000 network host), or odd (HP 3000 network host)?
- What is the device Group IDentifier (GID) and Device IDentifier (DID) for your terminal?
- What is the maximum block length (in bytes) the network host can accept from your terminal?

All these parameters are configured under program control by use of CONTROL statements. Alternately, default values for line speed, modem handshake, parity, and Async or Data Link protocol selection can be set using the datacomm interface configuration switches. Other default parameters are preset by the datacomm interface to accommodate common configurations. You can use the defaults, or you can override them with CONTROL statements for program clarity and immunity to card settings. Default Control Register values are shown in the *HP BASIC 6.2 Language Reference*. The *HP 98628 Datacomm Interface Installation* manual explains how to set the default switches on the interface.

5

## Datacomm Configuration for BASIC/UX

There is no capability in BASIC/UX for reading the hardware switches on either the HP 98628 Datacomm Interface card or the HP 98642 4-Channel Multiplexer card. Therefore, BASIC/UX provides two methods for configuring modem control options:

- The `stty` command from the HP-UX environment.
- The keyword CONTROL and registers directly related to the modem control options.

Of the two methods mentioned above the best one to use is the `stty` command while in the HP-UX environment. The reason for this is any modem control options set by using the keyword CONTROL are lost when you leave BASIC/UX. However, if you prefer to change these options while in the BASIC/UX environment, then read the subsequent section “Datacomm Options for Async Communications.”

This section deals with the first method mentioned above which is the use of the `stty` command from the HP-UX environment.

## Defaults for the Serial Interface

When HP-UX is being booted, the defaults for all Datacomm Interfaces are:

Baud rate	300
Bits per character	7
Parity	Odd
Stop bits	2

The above values are used by BASIC/UX as defaults, unless configured as explained in the next section.

Some common datacomm interface configuration settings are:

Baud rate to	9600
Bits per character to	8
Parity to	Odd and disabled
Stop bits to	1

5

## Configuring a Datacomm Interface for BASIC/UX

To configure your datacomm interface with the values mentioned in the previous section, you can execute the following HP-UX command before entering BASIC/UX:

```
/bin/stty 9600 cs8 -parenb parodd -cstopb < /dev/rmb/dcommnn
```

where:

**9600**

is the baud rate. The following are baud rates you can use with the `stty` command:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134	600	3600	19200

**cs8**

is the number of bits per character. In the case of this example, the number of bits per character is 8. Other character lengths can be set using `cs5`, `cs6`, or `cs7` for 5, 6, or 7 bits per character respectively.

<b>-parenb</b>	disables parity generation and detection. Removing the minus sign that is prefixed to this <b>stty</b> option causes parity generation and detection to be enabled.
<b>parodd</b>	selects odd parity. Prefixing the minus sign to this <b>stty</b> option selects even parity.
<b>-cstopb</b>	causes one stop bit per character to be used. Removing the minus sign that is prefixed to this <b>stty</b> option causes two stop bits per character to be used.
<b>&lt; /dev/rmb/dcommnn</b>	assigns the <b>stty</b> options to the serial interface located at select code number <i>nn</i> .

For more information on **stty** options, see the *HP-UX Language Reference*.

## 5 Resetting the Datacomm Interface

Before you establish a connection, the datacomm interface must be in a known state. *The datacomm interface does not automatically disconnect from the datacomm link when the computer reaches the end of a program.* To prevent potential problems caused by unknown link conditions left over from a previous session, it is a good practice to reset the interface card at the beginning of your program before you start configuring the datacomm connection. Resetting the card causes it to disconnect from the line and return to a known set of initial conditions (see the previous section “Datacomm Configuration for BASIC/UX”).

In the following example, a numeric variable is used to define the select code. The second statement resets the card after the select code has been defined.

```
1110   Sc=20           ! Set select code to 20.
1160   CONTROL Sc,0;1 ! Reset the card to disconnect from line.
```

## Protocol Selection for BASIC/WS

During power-up and reset, the card uses the default switches to preset the card to a known state. The protocol select switch defines which protocol the card uses at power-up only. If the default protocol is the same as you are using, you can skip the protocol selection statements. However, if the switch might be set to the wrong protocol, or if you want to change protocol in the middle of a program, you can use a CONTROL statement to select the protocol. After the protocol is selected, reset the card again to make the change. Here is how to do it:

Select the protocol to be used:

```
1170    CONTROL Sc,3;1    ! Select Async Protocol
```

or

```
1170    CONTROL Sc,3;2    ! Select Data Link Protocol
```

Wait until the protocol select message has been sent to the card, (lines 1180-1200) then reset the card. The Reset command restarts the interface microcomputer using the selected protocol.

```
1180    Wait:STATUS Sc,38;All_sent ! Get transmit queue status.
1190        IF NOT All_sent THEN Wait ! If not done, wait.
1200        CONTROL Sc,0;1          ! Reset interface card.
```

---

### Note



Be careful when resetting the interface card during normal program operation. Data and Control information are sent to the card in the same sequence as the statements originating the information are executed. When a card reset is initiated by a CONTROL statement, the reset is not placed in the queue with outbound data, but is executed immediately. Therefore, if there is other information in the output queue waiting to be sent, a reset can cause the data to be lost. To prevent loss of data, use STATUS statements (register 38) to verify that all data transfers have run to completion before you reset the interface.

---

You are now ready to program datacomm options that are related to the selected protocol. In applications where defaults are used, the options are very simple. The following pair of examples shows how to set up datacomm options for each protocol.

## Datacomm Options for Async Communications

This section explains how to configure the datacomm interface for asynchronous data communication. The example used shows how to set up all configurable options without considering default values. Some statements in the example are redundant because they override interface defaults having the same value. Others may or may not be redundant because they override default configurations. The remaining statements are necessary because they override the default values, replacing them with non-default values required for proper operation of the example program. If you are not familiar with Asynchronous protocol, consult the section on protocol for the needed background information.

The following BASIC/WS program lines set up all the CONTROL register options (a 300-baud connection to an HP 1000 is assumed). The \* marks program lines that may be redundant because they are the same as the interface default. The → marks program lines that may be redundant because they override the configuration switch option.

5

```
1250 CONTROL Sc,14;3      ! Set control block mask for EOL & Prompt.
* 1260 CONTROL Sc,15;0    ! No modem line-change notification.
1270 CONTROL Sc,16;0      ! Infinite connection timeout.
→ 1280 CONTROL Sc,17;0    ! Disable No Activity timeout.
* 1290 CONTROL Sc,18;40   ! Lost Carrier 400 ms. *
* 1300 CONTROL Sc,19;10   ! Transmit timeout 10 s.
→ 1310 CONTROL Sc,20;7    ! Transmit speed = 300 baud.
→ 1320 CONTROL Sc,21;7    ! Receive speed = 300 baud.
1330 CONTROL Sc,22;2      ! EQ/AK (as terminal) handshake.
→ 1340 CONTROL Sc,23;1    ! Full Duplex connection.
1350 CONTROL Sc,24;66     ! Remove protocol characters except
1360                       ! EOL. Change errors to Underscore.
1370 CONTROL Sc,26;6      ! Assign AK character for EQ/AK.
1380 CONTROL Sc,27;5      ! Assign EQ character for EQ/AK.
* 1390 CONTROL Sc,28;2,13,10 ! Set EOL sequence to be CR-LF.
* 1400 CONTROL Sc,31;1,17 ! Set prompt to be DC1. (33 not used).
→ 1410 CONTROL Sc,34;2    ! Seven bits per character.
→ 1420 CONTROL Sc,35;0    ! One stop bit.
→ 1430 CONTROL Sc,36;1    ! Odd parity.
* 1440 CONTROL Sc,37;0    ! No inter-character time gap.
* 1450 CONTROL Sc,39;4    ! Set BREAK to four character times.
```



The following BASIC/WS program lines set up all the CONTROL register options (a 300-baud connection to an HP 1000 is assumed). The → marks program lines that may be redundant because they override the configuration switch option.

```
→ 1310 CONTROL Sc,20;7      ! Transmit speed = 300 baud.
    1330 CONTROL Sc,22;5      ! DC1/DC3 (as terminal & host) handshake.
→ 1340 CONTROL Sc,34;2      ! Seven bits per character.
→ 1350 CONTROL Sc,35;0      ! One stop bit.
    1360 CONTROL Sc,36;1      ! Odd parity.
```

Refer to the Control Register tables in the *HP BASIC 6.2 Language Reference* as you examine the CONTROL statements. The paragraphs which follow explain register functions and how to configure them.

### Control Block Contents for BASIC/WS

Configuration of the link begins with register 14 which determines what information is placed in the control blocks that appear in the input (receive) queue. In this example, only the end-of-line position and prompts are identified. Parity or framing errors in received data, and received breaks are not identified in the queue. This register interacts with Control registers 28 thru 33.

5

### Modem-initiated ON INTR Branching Conditions for BASIC/WS

Register 15 is rarely used in most applications because the interface usually manages all interaction with the modem. Modem interrupts are helpful when you are simulating your own line protocol. This register determines what changes in one or more modem lines can cause a program branch to occur when an ON INTR statement is active for that select code. Values from 0 thru 31 can be used, where a “1” in a bit position enables branching whenever the corresponding signal line changes state. Lines correspond to bits 0 thru 4 of STATUS register 7. In this example, modem functions are handled by the interface; no interaction with BASIC is necessary. If this register is given a non-zero value, bit 3 of the ENABLE INTR mask should be set. (ENABLE INTR statement is line 1820 of the example terminal emulator program.)

## Datacomm Line Timeouts

Registers 16-19 set timeout values to force an automatic disconnect from the datacomm link when certain time limits are exceeded. For most applications, the default values are adequate. A value of zero disables the timeout for any register where it is used. Each register accepts values of 0 thru 255; units vary with the register function.

- Register 16 (Connection timeout) sets the time limit (in seconds) allowed for connecting to the remote device. It is useful for aborting unsuccessful attempts to dial up a remote computer using public telephone networks.
- Register 17 (No Activity timeout) sets an automatic disconnect caused by no datacomm activity for the specified number of minutes. Default value is determined by default handshake switch setting. Default is not affected by CONTROL statements to Control Register 23 (hardware handshake).

- Register 18 (Lost Carrier timeout) disconnects when:

- Full Duplex: Data Set Ready (Data Mode) or Data Carrier Detect go false,

or

- Half Duplex: Data Set Ready goes false,

indicating that the carrier from the remote modem has disappeared from the line. Value is in multiples of 10 milliseconds.

- Register 19 (Transmit timeout) disconnects when a loss-of-clock occurs or a clear-to-send (CTS) is not returned by the modem within the specified number of seconds.

## Line Speed (Baud Rate)

The transmit and receive line speed(s) are set by Control Registers 20 and 21, respectively. Each is independent of the other, and they are not required to have identical values. The following baud rates are available for Async communication:

### Async Baud Rates

Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate
0	0 <sup>1</sup>	4	134	8	600 <sup>2</sup>	12	3600
1	50	5	150 <sup>2</sup>	9	1200 <sup>2</sup>	13	4800 <sup>2</sup>
2	75	6	200	10	1800	14	9600 <sup>2</sup>
3	110 <sup>2</sup>	7	300 <sup>2</sup>	11	2400 <sup>2</sup>	15	19 200

<sup>1</sup> An external clock must be provided for this option.

<sup>2</sup> These speeds can be programmed using the default switches on the interface card. Other speeds are accessed by CONTROL statements. (The HP 13265A Modem can be operated up to 300 baud.)

All configurable line speeds are available to CONTROL Registers 20 and 21. Only the eight speeds indicated can be selected using the default switches. When the configuration switch defaults are used, transmit and receive speeds are identical. The selected line speed must not exceed the capabilities of the modem or link.

## Handshake

Registers 22 and 23 configure handshake parameters. There are two types of handshake:

- **Software or protocol handshake** specifies which of the participants is allowed to transmit while the other agrees to receive until the exchange is reversed. Options include:
  - **No handshake**, commonly used with connections to non-interactive devices such as printers.
  - **DC1/DC3 handshake**, with the desktop computer configured either as a host *or* a terminal. Handshake characters are defined by registers 26 and 27 for BASIC/WS.
  - **DC1/DC3 handshake** with the desktop computer as both a host *and* a terminal. Handshake characters are defined by registers 26 and 27 for BASIC/WS. This option simplifies communication between two desktop computers.
- **Hardware or modem handshake** that establishes the communicating relationship between the interface and the associated datacomm hardware such as a modem or other link device for BASIC/WS only. The four available options are:
  - **Handshake Off, non-modem** connection—most commonly used for 3-wire direct connections to a remote device.
  - **Full Duplex modem** connection—used with full-duplex modems or equivalent connections.
  - **Half Duplex modem** connection (BASIC/WS only)—used with half-duplex modems or equivalent connections.
  - **Handshake On, non-modem** connection (BASIC/WS only)—used with printers and other similar devices that use the Data Carrier Detect (DCD) and Clear-to-send (CTS) lines to signal the interface card. When DCD is held down by the peripheral, the interface ignores incoming data. When CTS is held down, the interface does not transmit data to the device until CTS is raised.

Options 2 and 3 are usually associated with modems or similar devices, but may be used occasionally with direct connections when the remote device provides the proper signals. Refer to the table at the end of this chapter for a list of handshake signals and how they are handled for each cable or adapter option.

## BASIC/UX Modem Line Handshaking

BASIC/UX requires additional system administration before modem line handshaking can be used with the HP98628 card. The minor numbers of any serial device files in the `/dev/rmb` directory must be changed to `0xSS0009` where SS is the select code of the serial interface. For example,

```
crw-rw-rw-  1 root    other    1 0x090004 Feb 12 12:44 /dev/rmb/serial9
```

would be changed to

```
crw-rw-rw-  1 root    other    1 0x090009 Feb 12 12:44 /dev/rmb/serial9
```

The mechanism used for modem line handshaking is limited by the features provided by the HP-UX operating system. In particular, BASIC/UX uses the simple mode of modem line handshaking with a call-out device file. A full discussion of HP-UX facilities is beyond the scope of this manual. Refer to `modem(7)` and `termio(7)` for more details.

HP-UX allows for three different types of opens on RS232 interfaces: call-in, call-out, and direct connect. There are anomalies associated with each type of open, and thus existing applications or workarounds which execute outside the BASIC/UX environment may not work correctly as a result of changing the device file used by BASIC/UX. Because of these anomalies, BASIC/UX continues to use a direct connect device file for backward compatibility. See `modem(7)` for more details.

HP-UX simple mode of modem line handshaking uses the following algorithm for modem line handshaking. DTR is asserted by the interface, and DCD and CTS must be asserted by the device before data transfers can take place. If DCD is lowered, DTR will also be lowered until DCD is asserted again. See `MODEM(7)` for a further description of the simple mode of modem line handshaking.

## Handling of Non-Data Characters for BASIC/WS

Register 24 specifies what non-data characters are to be included in the input queue. For each bit that is set, the corresponding information is passed along with the incoming data. If the bit is not set, the information is discarded, and is not included in the inbound data stream that is passed to the desktop computer by the interface.

- Bit 0 Include handshake characters in data stream. They are defined by Control Registers 26 and 27.
- Bit 1 Include incoming end-of-line character(s). EOL characters are defined by Control Registers 28-30.
- Bit 2 Include incoming prompt character(s). Prompt is defined by Control Registers 31-33.
- Bit 3 Include any null characters encountered.
- Bit 4 Include any DEL (rubout) characters in data.
- Bit 5 Include any CHR\$(255) encountered. This character is encountered ONLY when 8-bit characters are received.
- Bit 6 Change any characters received with parity or framing errors to an underscore. If this bit is not set, all inbound characters are transferred exactly as received, with or without errors.

5

Register 25 is not used.

### **Protocol Handshake Character Assignment for BASIC/WS**

Registers 26 and 27 establish what characters are to be used for handshaking between communicating machines. You can select the values of 6 (AK) or 17 (DC1) for register 26, and 5 (EQ) or 19 (DC3) for register 27. Any ASCII value from 0 thru 255 can be used, but non-standard values should be reserved for exceptional situations.

### **End-Of-Line Recognition for BASIC/WS**

Registers 28, 29, and 30 operate in conjunction with registers 14 (control block mask) and 24 (non-data character stripping) and defines the end-of-line sequence used to identify boundaries between incoming records. Register 28 (value of 0, 1 or 2) defines the number of characters in the sequence, while registers 29 and 30 contain the decimal equivalent of the ASCII characters. If register 28 is set for one character, register 30 is not used. Register 29 contains the first EOL character, and register 30, if used, contains the second. If register 28 is zero, registers 29 and 30 are ignored and the interface cannot recognize line separators.

## **Prompt Recognition for BASIC/WS**

Registers 31, 32, and 33 operate in conjunction with registers 14 and 24 and define the prompt sequence that identifies a request for data by the remote device. As with end-of-line recognition, the first register defines the number of characters (0, 1, or 2), while the second and third registers contain the decimal equivalents of the prompt character(s). Register 33 is not used with single-character prompts. If register 31 is zero, registers 32 and 33 are ignored and the interface is unable to recognize any incoming prompts.

## **Character Format Definition**

Registers 34 through 37 are used to define the character format for transmitted and incoming data.

- Register 34 sets the character length to 5, 6, 7, or 8 bits. The value used is the number of bits per character minus five (0=5 bits, 3=8 bits). When 8-bit format is specified, parity must be Odd, Even, or None (parity "1" or "0" cannot be used).
- Register 35 specifies the number of stop bits sent with each character. Values of 0 or 2 are used to select 1 or 2 stop bits, respectively.

- Register 36 specifies the parity to be used. Options include:

### Parity Options

Register Value	Parity	Result
0	None	Characters are sent with no parity bit. No parity checks are made on incoming data.
1	Odd <sup>1</sup>	Parity bit is set if there is an EVEN number of ones in the character code. Incoming characters are also checked for odd parity.
2	Even <sup>1</sup>	Parity bit is set if there is an ODD number of ones in the character code.
3	0	Unsupported on BASIC/UX.
4	1	Unsupported on BASIC/UX.

5

<sup>1</sup> Parity sense is based on the number of ones in the character including the parity bit. An EVEN number of ones in the character, plus the parity bit set produces an ODD parity. An ODD number of ones in the character plus the parity bit set produces an EVEN parity.

- Register 37 (BASIC/WS only) sets the time gap (in character times, including start, stop, and parity bits) between one character and the next in a transmission. It is usually included to allow a peripheral, such as a teleprinter, to recover at the end of each character and get ready for the next one. A value of zero causes the start bit of a new character to immediately follow the last stop bit of the preceding character.

### Break Timing for BASIC/WS

Register 39 sets the break time (2-255 character times). A Break is a time gap sent to the remote device to signify a change in operating conditions. It is commonly used for various interrupt functions. The interface does not accept values less than 2. Register 6 is used to transmit a break to the remote computer or device.



## Datacomm Options for Data Link Communication for BASIC/WS

This section explains how to configure the datacomm interface for Data Link operation. The example used shows how to set up configuration options without considering default values. Some statements in the example are redundant because they override interface defaults having the same value. Others may or may not be redundant because they override configuration switch options. The remaining statements are necessary because they override the default values, replacing them with non-default values required for proper operation of the example program. If you are not familiar with Data Link protocol and terminology, consult the section called "Protocol."

The following program lines set up all the CONTROL register options (a 9600-baud connection to an HP 1000 network host is assumed). The \* marks program lines that may be redundant because they are the same as the interface default. The → marks program lines that may be redundant because they override the configuration switch option.

5

```
* 1250 CONTROL Sc,14;6      ! Set Control Block Mask for ETB/ETX.
* 1260 CONTROL Sc,15;0      ! No modem line-change notification.
  1270 CONTROL Sc,16;0      ! Disable Connection timeout.
→ 1280 CONTROL Sc,17;0      ! Disable No Activity timeout.
* 1290 CONTROL Sc,18;40     ! Set Lost Carrier to 400 ms.
  1300 CONTROL Sc,19;10     ! Set Transmit Timeout=10 s.
→ 1310 CONTROL Sc,20;14     ! Set Line Speed to 9600 baud.
* 1320 CONTROL Sc,21;1      ! Set GID character to "A".
→ 1330 CONTROL Sc,22;1      ! Set DID character to "A".
→ 1340 CONTROL Sc,23;0      ! Hardware Handshake Off for HP 13264A.
* 1350 CONTROL Sc,24;0      ! Set transmit block size to 512.
* 1360 CONTROL Sc,36;0      ! Parity not used with HP 1000.
```

If your application requires a different GID/DID pair, you can use either of the following two techniques (assume: GID="C" and DID="@"):

```
1320 CONTROL Sc,21;3      ! Set GID character to "C".
1330 CONTROL Sc,22;0      ! Set DID character to "@".
```

OR

```
1320 CONTROL Sc,21;3,0    ! Set GID/DID to "C@".
```

Here is an alternative method using string operations:

```
1320 CONTROL Sc,21;NUM("C")-64
1330 CONTROL Sc,22;NUM("@")-64
```

or

```
1320 CONTROL Sc,21;NUM("C")-64,NUM("@")-64
```

Refer to the Control Register tables in the *HP BASIC 6.2 Language Reference* as you examine the CONTROL statements. The paragraphs which follow explain register functions and how to configure them. When the register function is identical for both Async and Data Link, you are referred to the previous explanation in the Async section.

### **Control Block Contents for BASIC/WS**

Data Link configuration begins with Control Register 14. This register determines what information is to be placed in control blocks and included with inbound data transferred from the interface to the desktop computer.

- ETX (Bit 1) identifies the end of a transmission block that contains one or more complete records.
- ETB (Bit 2) identifies the end of a transmission block where the last record is continued in the next block of data.
- Bit 0 causes a control block to be inserted that identifies the beginning of a new block of data.

### **ON INTR Branching Conditions and Line Speed for BASIC/WS**

Registers 15 through 19 are functionally identical for both Async and Data Link. Refer to the preceding Async section for more information. Register 20 sets the line speed for both transmitting and receiving (Data Link does not accommodate split-speed operation). The following line speed options are available:

### Data-Link Baud Rates

Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate	Register Value	Baud Rate
0	External Clock <sup>1</sup>	9	1200 <sup>2</sup>	12	3600	15	19 200 <sup>2</sup>
7	300 <sup>2</sup>	10	1800	13	4800		
8	600	11	2400	14	9600 <sup>2</sup>		

<sup>1</sup> An external clock must be provided for this option.

<sup>2</sup> These speeds can be programmed using the default switches on the interface card. Other speeds are accessed by CONTROL statements.

### Terminal Identification for BASIC/WS

Registers 21 and 22 specify the terminal identifier characters for the datacomm interface. Register 21 contains the GID (Group Identifier), and register 22 contains the DID (Device Identifier). Values of 0-26 correspond to the characters @, A, B, . . . , Z. These registers must be configured to match the terminal identification pair assigned to your device by the Data Link Network Manager. In the example, Line 1320 is redundant because it duplicates the default GID value. Line 1330 overrides the DID default switch on the interface card, and may or may not be necessary. Alternate methods for assigning different GID/DIDs are shown following the group of configuration CONTROL statements.

5

### Handshake for BASIC/WS

Register 23 establishes the hardware handshake type. There is no formal software handshake with Data Link because the network host controls all data transfers. Hardware or modem handshake options are identical to Asynchronous operation. Handshake should be OFF (register set to 0) when using the HP 13264A Data Link Adapter. When you are using non-standard interconnections such as direct or modem links to the network host, select the handshake option that fits your application. Refer to the table at the end of this chapter for a list of handshake signals and how they are handled for each cable or adapter option.

## Transmitted Block Size for BASIC/WS

Register 24 defines the maximum transmitted block length. When transmitting blocks of data to the network host, the block length must not exceed the available buffer space on the receiving device. Block size can be specified for increments of two from 2 to 512 characters per block. A value of zero forces the block length to a maximum of 512 bytes. For other values, the block length limit is twice the value sent to the register. For example, a register value of 130 produces a transmitted block length not exceeding 260 characters (bytes).

## Parity for BASIC/WS

Register 36 defines the parity to be used. Unlike Async, Data Link has only two parity options: None, or Odd. Odd parity is:

**Data-Link Parity Options**

Register Value	Parity	Application
0	NONE	Required for operation with HP 1000 network host
1	ODD	Required for operation with HP 3000 network host

Registers 25 through 35, and 37 and above are not used.

## Connecting to the Line

Interface configuration is now complete. You are ready to begin connecting to the datacomm line. The exact procedure used to connect to the line varies slightly, depending on the type of link being used. Before you connect, you must know what the link requirements are, including dialing procedures, if any.

## **Switched (Public) Telephone Links**

When you are using a public or switched telecommunications link, the modem connection between computers must be established. The HP 13265A Modem can be used in any Async application that requires a Bell 103- or Bell 113-compatible modem operating at up to 300 baud line speed. However, the HP 13265A Modem is not suitable for data rates exceeding 300 baud. For higher baud rates, use a modem that is compatible with the one at the remote computer site.

## **Private Telecommunications Links**

Private (leased) links require modems unless the link is short enough for direct connection (up to 50 feet, depending on line speed). The HP 13265A Modem can be used at data rates up to 300 baud. For higher speeds, a different modem must be used.

## **Direct Connection Links**

For short distances, a direct connection may be used without modems or adapters, provided both machines use compatible interfaces. Async connections normally use RS-232C interfaces.

## **Connection Procedure for BASIC/WS**

This section describes procedures for modem connections using telephone telecommunications circuits. If you *are not* using a switched, modem link, skip to the next section: Initiating the Connection.

## **Dialing Procedure for Switched (Public) Modem Links**

Except for dialing, connection procedures do not usually vary between switched and dedicated links. Dialing procedures depend on whether the modem is designed for manual or automatic dialing. Automatic dialing can be used with the HP 13265A Modem, but other modems must be operated with manual dialing unless you design your own interface to an Automatic Calling Unit. For manual dialing procedures, consult the operating manual for the modem you are using.

## Automatic Dialing with the HP 13265A Modem

The automatic dialer in the HP 13265A Modem is accessed by Control Register 12. The CONTROL statement is followed by an OUTPUT statement that contains the telephone number string, including dial rate and timing characters. The two statements set up the automatic dialer, but dialing is not started until a “start connection” command is sent to Control Register 12. Here is an example sequence:

```
1500 CONTROL Sc,12;2      ! Enable the Automatic Dialer.
1510 OUTPUT Sc;"> 9 000 (303)-555-1234";
```

The OUTPUT statement contains several essential elements.

- The first character (“>”), if included, specifies a fast dialing rate. If it is omitted, the default slow dialing rate is used.
- A time delay character “@” may be inserted anywhere in the string. A one-second time delay is executed in the dialing sequence each time a delay character is encountered.
- Numeric character sequences define the telephone number. Multiple dial-tone sequences, such as when calling out from a PBX (Private Branch Exchange), can be used by inserting a suitable delay to wait for the next dial tone.
- Unrecognized characters such as parentheses, hyphens, and spaces can be included for clarity. They are ignored by the automatic dialer.
- Up to 500 characters can be included in the telephone number string.

Here is how an autodial connection is executed:

- The CONTROL Sc,12;2 statement places a “start dialing” control block in the outbound queue to the interface. The OUTPUT statement places the telephone number string (including spaces and other characters) in the queue after the control block. When the interface encounters the control block, it transfers the string to the HP 13265A Modem’s autodial circuit. No other action is taken at this time.
- When a CONTROL Sc,12;1 statement (line 1600 in the example) is executed, another control block is queued up. When the interface encounters the block, it sends a “start connection” command to the modem. The modem then disconnects from the line, waits two seconds, then reconnects. The autodialer waits 500 milliseconds, then starts executing the telephone number string.

The string is executed character-by-character in the same sequence as sent by the OUTPUT statement.

- If your application requires more than 500 milliseconds to guarantee a dial tone is present, you can increase the delay by adding delay characters (“@”) where needed, one second per character. Be sure to provide adequate delays in multiple dial tone sequences, such as when calling through a private branch exchange (PBX) to a public telephone network.
- When dialing is complete, the modem is connected to the line, and you are ready to start communication. The next section explains how to determine when connection is complete.

Two dialing rates are available: slow (default) and fast. To select the fast rate, you must include the fast rate character (“>”) as the FIRST character in the telephone number string. Here is a summary of differences between the two options:

**Dialing Options**

Parameter	Slow Dialing	Fast Dialing
Click Length	60 milliseconds	32.5 milliseconds
Click Gap	40 milliseconds	17.5 milliseconds
Number Gap	700 milliseconds	300 milliseconds

One to ten dial pulses (clicks) are sent for each digit 1 through 0, respectively. The number gap is the time lag between the end of the last click of one number and the beginning of the first click of the next number.

Most Bell System facilities can handle both fast and slow dialing rates, but private or independent telephone systems or companies may require slow dialing.

## Initiating the Connection

After you have executed the necessary dialing procedures, if any, you are ready to initiate the connection. The following statement is used to start the connection:

```
1600 CONTROL Sc,12;1 ! Start Connection.
```

This statement sends a control block to the interface telling it to connect to the datacomm line. If the HP 13265A Modem is being used, and the autodialer is enabled, it starts dialing the number. Otherwise, the interface executes a direct connection to the line, or tells the modem or data link adapter to connect.

The status of the connection process can be monitored by using the STATUS statement. The following lines hold the computer in a continuous loop until the connection is complete:

```
1650 Conn:STATUS Sc,12;Line_state ! Get datacomm line status.
1660     IF Line_state=2 THEN DISP "Dialing"
1670     IF Line_state=1 THEN DISP "Trying to Connect"
1680     IF Line_state=3 THEN Conn
1690     DISP "Connected"
```

5

Refer to the *HP BASIC 6.2 Language Reference* for interpretation of the values in Status Register 12. Only values of 1, 2, or 3 are usually encountered at this stage of the program.

As soon as Status Register 12 indicates that connection is complete, you are ready to continue into the main body of the terminal emulator or other program you are writing. This completes the datacomm initialization and connection phase of the program.

## Connection Procedure for Hayes-Compatible Modems

Now that the CONTROL registers are set up for the Datacomm card (refer back to the section "Datacomm Options for Async Communications"), you are ready to initiate the connection. By OUTPUTting a dialing command to your modem, the connection will automatically be made. Here is an example of a simple dialing command:

```
OUTPUT Sc;"ATDT 555-1234"
```



The OUTPUT statement contains several essential elements.

- AT is sent to get the attention of the modem and to tell it that you are going to be sending a command. AT must precede all commands sent to the modem.
- DT informs the modem that you want to dial using touch-tone dialing. Use "ATD" for rotary dialing.
- Numeric character sequences define the telephone number.

Once the connection is made, any data which you OUTPUT/ENTER to/from the Datacomm select code will be transmitted/received by the modem over the data line. To return to command mode, you need to OUTPUT an escape code. This escape code is usually "+++", but may vary with different modems. Once the escape code is sent, all data sent to the Datacomm select code will be treated by the modem as a command. Some of the commands are as follows:

AT	Attention
H	Hang-up
EO	Turn echo off
D	Dial
,	Pause
A/	Re-dial
O	Return on-line (Get back to data transmit/receive mode)

Note that all of the above commands must be preceded by "AT".

Refer to the user manual for your modem for a complete list of commands.

## Example Modem Session

A simple modem session may be as follows:

```
70   CONTROL Sc,0;1           ! Reset card.
80   CONTROL Sc,8;2           ! Set DTR line.
90   CONTROL Sc,20;11         ! Set rate to 2400.
100  CONTROL Sc,22;5          ! DC1/DC3 (as terminal & host) handshake.
110  CONTROL Sc,34;2          ! Seven bits per character.
120  CONTROL Sc,35;0          ! One stop bit.
130  CONTROL Sc,36;1          ! Odd parity.
140  OUTPUT Sc;"ATDT 555-1234" ! Establish the connection.
    .
    .
    .
1990 ENTER Sc;Data$           ! Receive/Transmit.
2000 OUTPUT Sc;"Data"         ! Data over the connection.
    .
    .
    .
3000 OUTPUT Sc;"++++";        ! Return to command mode (Send without
3010                                ! CR/LF).
3020 OUTPUT Sc;"ATH"          ! Hang-up!
3030  END
```

5

---

## Setting up the Interrupt System for BASIC/WS

Most datacomm programs, especially complex ones, use interrupt branching extensively to maintain efficient, orderly program operation. Branching is usually set up for:

- I/O interrupts from peripheral devices by use of ON INTR and ENABLE INTR statements.
- Datacomm interrupts from the datacomm interface. Statements used are the same as for other I/O interrupts.
- Operator interrupts using softkeys for program control. A separate ON KEY statement is used to set up the branch for each key used.

- Operator interrupts using ASCII keys for program input. The ON KBD statement is used to set up the branch, and KBD\$ is the keyboard-entry string holding the data.

Each interrupt branch must be provided with a corresponding interrupt service routine, with priority levels assigned when appropriate. General I/O interrupt techniques are explained in the chapter "Interface Events." This section explains the interrupt structures commonly encountered in datacomm applications.

## Setting up Softkey Interrupts

Softkeys are usually set up for repetitively executed functions to improve operator convenience and efficiency. Labels can have up to eight or 14 characters for each key, depending on CRT screen width. The following statements add a disconnect and break capability to the emulator example we are using:

```
1750 ON KEY 0 LABEL " Disconnect" GOTO Disconnect
1760 ON KEY 1 LABEL " Break" GOSUB Break
```

Other keys can be set up and labelled as needed, but remember a service routine is required for each label specified by a GOTO, GOSUB, CALL, or RECOVER.

## Setting Up Program Operator Inputs

Two methods are commonly used to input information from the operator through the computer keyboard. The first method uses the LINPUT (or INPUT) statement for data entry. An example program using the LINPUT statement is shown in the overview of datacomm programming earlier in this chapter. When the LINPUT statement requests a data entry, type the information, use the keyboard editor to make any necessary corrections, then press CONTINUE to transfer the information to the running program. This is the simplest method for programming keyboard entry. The second method is used in our ongoing example. It uses the ON KBD statement in conjunction with an interrupt service routine that is responsible for all data manipulation, including display, editing, and transfer to the program. The following statement sets up the keyboard interrupt. The interrupt service routine is discussed later.

## Setting Up Datacomm Interrupts

The ON INTR and ENABLE INTR statements are used to set up program branching for the datacomm interface. STATUS Register 4 contains information that shows the cause(s) of the most recent interrupt. The interrupt mask specified in the ENABLE INTR statement determines the events that are allowed to cause an interrupt branch. Bits 0 thru 5 of the interrupt mask and STATUS register are identical for both Async and Data Link protocols. Bits 6 and 7 are used for Async only.

The following statements set up the interrupt structure for datacomm:

```
1810 ON INTR Sc GOSUB Datacomm
1820 ENABLE INTR Sc;1 ! Interrupt when data received.
```

In more elaborate applications, you may want to enable additional interrupt causes by changing the interrupt mask. Here are the available interrupt bits and their functions:

### Interrupt Mask Bits for Async Operation

Bit	Value	Function	Bit	Value	Function
0	1	Data in Receive Queue	4	16	No Activity Timeout
1	2	Prompt Received	5	32	Lost Carrier Timeout
2	4	Framing/Parity Error	6	64	End-of-line Received
3	8	Modem Line Change	7	128	Break Received

### Interrupt Mask Bits for Data Link Operation

Bit	Value	Function	Bit	Value	Function
0	1	Data in Receive Queue	3	8	Modem Line Change
1	2	Block Successfully Sent	4	16	No Activity Timeout
2	4	Transmit or Receive Error	5	32	Lost Carrier Timeout

Interrupt mask bits 6 and 7 are not used with Data Link protocol.

To construct the interrupt mask value, add the bit values for each function that is to cause an interrupt. For example, to interrupt when there is data in the receive queue (bit value=1), or a modem line change (bit value=8) or a Lost Carrier timeout (bit value=32), the interrupt mask becomes:  $1 + 8 + 32 = 41$ . The ENABLE INTR statement becomes:

```
1820 ENABLE INTR Sc;41
```

---

## Background Program Routines for BASIC/WS

After the interrupt structures have been established by the running program, the program begins executing a “background” routine while it waits for interrupts. Background routines vary according to application, and can consist of anything from a simple idle loop to a very complex program. They are called background programs or background routines because their execution is generally suspended whenever interrupts from previously defined sources are received. See the chapter “Interface Events” for more discussion of interrupt and software priority.

Background program operations can affect interrupt handling under certain conditions. For example, if the background program contains a subprogram call, the interrupt service routines are temporarily suspended until subprogram execution is complete if the ON INTR statements use GOSUB, or GOTO. Incoming data is held in the receive queue during subprogram execution, and the remote is held off by the interface when the queue is full, if handshaking between devices is active. If handshaking is not being used in Async operation, buffer overflow can occur. When handshake is being used, be sure that the remote computer does not disable the link when extended hold-offs occur.

When interrupt service routines are subprograms accessed by an ON INTR ... CALL statement, background subprograms may be temporarily suspended to allow interrupt processing. Be careful when using subprograms to be sure that variables are properly used for orderly flow of information between contexts.

Most BASIC programmers, to maintain clarity in program flow, place interrupt service routines after the background routines. This technique simplifies documentation and makes it easier for others to understand program operation.

The location of subroutines or program labels in BASIC programs does not affect efficiency or speed of execution by the desktop computer.

A detailed discussion of background programs is beyond the scope of this chapter because they are dependent upon the individual application. In the example shown in this chapter, a simple idle loop is sufficient. A typical idle loop resembles the following statement:

```
1880 Background: GOTO Background ! Background program idle loop.
```

The next topics addressed are interrupt service routines for datacomm and keyboard operations.

---

## Interrupt Service Routines

5

Interrupt service routines are required to service any peripheral device or interface that uses interrupt to access the computer. In the example we are using, interrupt service routines are required for the datacomm interface, computer keyboard, and softkeys. Each routine is treated separately in this section.

### Servicing Datacomm Interrupts

Whenever the datacomm interface interrupts a running BASIC program, the interrupt request is first logged and then `DISABLE INTR` is automatically executed by the system. The cause of interrupt is then placed in `STATUS Register 4`. The interrupt service routine must do several things to guarantee that: (1) the interrupt is properly handled, (2) the interrupt structure is restored after the current interrupt is acknowledged, and (3) no data is left in the receive queue after the last interrupt request is processed. The following items outline the basic elements of the datacomm interrupt service routine (similar techniques are used for other interfaces).

- Read `STATUS Register 4` to clear the interrupt request and determine the cause of the interrupt. If you do not clear the interrupt request, it remains active and a new interrupt is generated as soon as you exit the service routine, whether or not there is any information to process.

- Use `ENABLE INTR` (usually without specifying a new interrupt mask) to reactivate the datacomm interrupt system. It is usually unnecessary to redefine the interrupt mask when this is done.
- Take appropriate action based on what caused the interrupt.
- Exit the interrupt service routine with a `RETURN` (or equivalent statement as appropriate) taking care to maintain proper program structure.

In most applications, interrupts are generated when data is available for transfer between the interface and your desktop computer. The interrupt service routine then processes the transfer using the `ENTER` statement. Here is an example of a typical datacomm interrupt service routine where `A$` is dimensioned to a length of one character (`DIM A$[1]`). The calling sequence might be:

```
ON INTR Sc GOSUB Datacomm ENABLE INTR Sc;Mask

2090 Datacomm:STATUS Sc,4;Interrupt_cause
2100     ENABLE INTR Sc
2110 Dc:  STATUS Sc,5;Rx_queue_status
2120     IF Rx_queue_status=0 THEN RETURN
2130     ENTER Sc USING "#,-K";A$
2140     PRINT USING "#,K";A$
2150     GOTO Dc
```

5

While this **interrupt service routine** (ISR) looks deceptively simple, its structure performs several important functions:

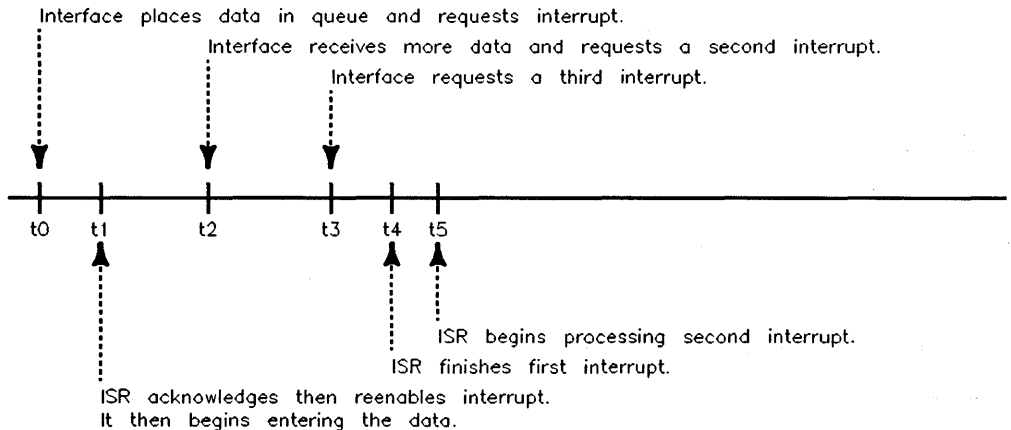
- Line 2090 acknowledges the interrupt and places the cause-of-interrupt information in `Interrupt_cause`.
- Line 2100 reenables the interrupt without changing the mask.
- Line 2110 gets the receive queue status. Four values are possible:
  - `Rx_queue_status=0`: Receive queue is empty.
  - `Rx_queue_status=1`: Receive queue contains data.
  - `Rx_queue_status=2`: Receive queue contains at least one control block.
  - `Rx_queue_status=3`: Receive queue contains both data and at least one control block.

- Line 2120 checks to make sure there is data or control information available before continuing. This prevents attempts to enter data that does not exist. The placement of this statement is explained under Exit Conditions.
- Line 2130 enters the data. The format used guarantees that no data is lost during searches for end-of-line delimiters. The “#” IMAGE specifier prevents search for end-of-line (EOL) delimiters. Use of “-K” places CR, LF, and CR-LF end-of-line delimiters in the string variable when they are encountered. BASIC can then locate the delimiters by using separate operations.
- Line 2140 prints the data on the current PRINTER IS device. The “#” specifier suppresses the EOL sequence because terminators are already contained in the string variable.
- Line 2150 goes back to check for more data before exiting. This guarantees that no data is missed in the event that additional data arrives during interrupt service. Otherwise, some interrupt requests may be missed.

5

To understand why the interrupt is handled as shown, consider the following sequence of events:

Interface places data in queue and requests interrupt.



### How BASIC Handles Datacomm Interrupts



At time  $t_0$ , the interface places data in the receive queue and requests interrupt service. At  $t_1$ , the ISR responds and acknowledges the interrupt. The interrupt is reenabled, but subsequent interrupt service requests are logged but not serviced until the routine is finished. While the ISR is processing the first interrupt request, a second and third request are made at  $t_2$  and  $t_3$ . (The already active interrupt request line is reactivated by the third request. From the computer's point of view, nothing happened because the second interrupt request was already active). When the ISR completes the first interrupt process ( $t_4$ ), it exits, then acknowledges, the second interrupt ( $t_5$ ).

Here is what really happens when the example routine is executed: Since the routine checks for no more data in the queue before it processes the interrupt, and remains in the ENTER/PRINT loop until the queue is empty, all available information is processed before exit occurs. Therefore, data placed in the queue at the time of the second and third interrupt requests is processed before the exit at  $t_4$ , guaranteeing that nothing is left. When the second entry is made to the routine ( $t_5$ ) in response to the second interrupt request, no data is in the queue unless it was placed there between exit and reentry. In this case, the queue is empty, so exit is immediate. The third interrupt request cannot be recognized, because the second was still pending when it occurred.

If the routine were written differently, and only one ENTER statement was executed for each interrupt request, the example sequence would result in only two interrupts being acknowledged. The third interrupt request and its corresponding data would not be processed until a fourth request caused the third data entry to be executed. Such a structure presents a risk of data loss.

### Exit Conditions

In the preceding example, line 2120 exits or continues the interrupt service routine, depending on the status of the receive queue. The example shown assumes that A\$ can hold only a single ASCII character or data byte. The ENTER statement is terminated as soon as A\$ is filled, so data transfer is one byte at a time. By checking for Status Register 5=0, you are guaranteed that no data messages remain in the receive queue. Control blocks are immaterial in this case.

When using Data Link protocol, most programmers specify data transfer formats of one record per block. This eliminates the need to search data for delimiters. (The HP 3000 packs multiple records per block when transferring

ASCII text files, so you must decode delimiters to find record boundaries. Consult the appropriate HP 3000 Data Link manuals for more information.) Since the datacomm interface can receive Data Link transmission blocks up to 1000 characters, it is wise to dimension A\$ to a length exceeding the maximum expected block length; for example, DIM A\$[1050]. In such cases, it is necessary to modify line 2120 to provide exit if a full block is not available for A\$. Instead of examining for the presence of data, a test is made to look for a control block in the queue, indicating the presence of a full block of data. (Control Register 14 must be set so that only ETB/ETX terminators are allowed to create a control block.) If a control block is present, a full block of data is also available. When the ENTER statement is executed, the input operation terminates when the control block is encountered, and the resulting length of A\$ matches the received block length. To operate in “block mode” instead of “character mode” as earlier, change line 2120 to:

```
2120      IF Rx_avail_bits<2 THEN RETURN
```

- 5 Only the dimension of A\$ is affected by this change. Other interrupt service routine statements remain unchanged.

---

**Note**

It is good programming practice to be sure the receive queue or input buffer is completely empty before exiting an interrupt service routine, and make sure there is data present before trying to process it.

---

This example datacomm interrupt service routine is adequate for most applications where data is not sent with a known, fixed format, and where prevention of data loss is important. In other situations, where loss of data between the end of the input variables list and the delimiter in incoming data is unimportant, or a fixed format is used, other formats can be specified. It is usually wise to avoid using multiple variables with the ENTER statement when using the formats shown in this example. Here's why:

A control block indicates End-of-data, not End-of-information. Consequently, an ENTER statement is terminated whenever a control block is encountered (variables are terminated by EOI, not EOD). If more than one variable is included in the statement, and EOD (control block) occurs before the list is filled, the unfilled variables retain their previous values which can lead to improper results.

## Data Formats for Datacomm Transfers

All datacomm data transfers use the OUTPUT and ENTER statements. Consequently, any formatting techniques that are compatible with these statements can also be used. However, since most computers send and expect to receive a limited variety of data formats, most data transfers use a limited assortment of formats.

**ASCII Data Transfers**—In asynchronous data communications applications, information is usually transferred as lines of ASCII text. In most cases, lines are terminated by a carriage-return followed by a line-feed (CR-LF), or by a carriage-return only. Other methods may be used occasionally to recognize record boundaries in special applications.

Most Data Link applications consist of ASCII text records transferred between the network host computer and other terminals and/or computers in the network. Records are transmitted in blocks, one or more records per block. When multiple-record blocks are transferred, delimiters between records are included as part of the text, and individual records must be unpacked by the receiver.

**Non-ASCII Data Transfers**—Non-ASCII data includes non-text or non-ASCII text data that must be transmitted over the datacomm link, but may contain characters that could be interpreted as datacomm control characters. Examples of non-ASCII data includes encoded data files, non-text program files, or specially formatted data. To provide a means of transferring non-ASCII data formats requires non-standard techniques in Async, and transparent transmission when using Data Link.

To transfer non-ASCII data using asynchronous protocol, use an eight-bit character format with or without parity as dictated by your application. End-of-line and prompt recognition, and any character stripping functions must be disabled to allow passage of arbitrary character patterns. Use of Async for such applications is uncommon, primarily because of the limited reliability of parity checks as a means for error detection.

Transfer of non-ASCII data using Data Link protocol is much easier because all data transmitted by the desktop computer through the datacomm interface is sent as transparent data; i. e., data that could be mistaken for control characters is transferred intact. Data Link transfers from the network host are also sent as transparent data. In order to transfer non-ASCII data from the

network host, a cooperating program on the host must originate the data, and suppress end-of-line and other unwanted character sequences.

## Servicing Keyboard Interrupts

The keyboard interrupt service routine has several functions. In the case of a terminal emulator or similar application, it inputs keystrokes, interprets them, then transmits the results to the datacomm interface. In addition, it may be required to display the keystroke(s) or perform backspace and editing operations (such as in line-mode terminal emulators). Certain keys may also be reserved to perform program command functions while others are used to transmit information to the host.

Here is a simple example of a keyboard interrupt service routine that sends ASCII keystrokes to the datacomm interface as each key is pressed, then sends an end-of-line (CR) if Async, or end-of-block if Data Link. The example shown is for Async protocol; Line 2410 is changed for Data Link. The calling sequence might be ON KBD GOSUB Keyboard. An explanation follows the example.

5

```
2290 Keyboard:K$=KBD$
2300 K:  IF NOT LEN(K$) THEN RETURN
2310     Key=NUM(K$)
2320     K$=K$[2]
2330     IF Key=255 THEN
2340         Key=NUM(K$)
2350         K$=K[2]
2360         IF Key=255 THEN
2370             Key=NUM(K$)
2380             K$=K$[2]
2390         END IF
2400         IF Key=NUM("E") THEN
2410             OUTPUT Sc;CHR$(13);END
2420         ELSE
2430             BEEP
2440         END IF
2450     ELSE
2460         OUTPUT Sc;CHR$(Key);
2470     END IF
2480     GOTO K
```

To change the example for Data Link, eliminate the carriage return in line 2410 as follows:

2410      OUTPUT Sc;END

This Async example assumes that the host echoes any data sent to it; that is, when a character is sent to the host, the host sends the same character back to the terminal where it is displayed. Consequently, keystrokes are displayed AFTER they are returned by the host. Data Link protocol does not provide this feature (called echo-plex). To print each keystroke on the CRT as it is keyed in, add the following line to the Data Link example:

```
2465      PRINT CHR$(Key);
```

This keyboard routine is a good illustration of how to use an IF ... THEN ... ELSE structure to decode a keystroke, and decide whether it is ASCII, end-of-line, or an unrecognized character. If ASCII, it is transmitted. If the ENTER key is pressed, it sends an EOL. Any other key is ignored, but the computer beeps to acknowledge the keystroke.

To understand the routine, you must be aware that several data formats are found in KBD\$. ASCII keystrokes are stored, one byte per stroke, as key codes equivalent in value to the NUM value of the corresponding ASCII character code. Non-ASCII keys are stored as two bytes; the first byte is CHR\$(255), the second byte is the keycode. If the CONTROL key is pressed simultaneously with a non-ASCII key, a three-byte entry is made in KBD\$. The first is CHR\$(255) representing a non-ASCII key, the second is also CHR\$(255) representing the CONTROL key, and the third byte is the keystroke. Keycode values for non-ASCII keys are listed in the Keyboard Output Codes table in the back of the BASIC Language Reference for your computer. The following table shows the KBD\$ data format for each keystroke:

**KBD\$ Data Formats**

Keystroke(s)	First Byte	Second Byte	Third Byte
ASCII or CONTROL-ASCII	ASCII keycode	None	None
Non-ASCII Key	CHR\$(255)	Non-ASCII keycode	None
CONTROL-Non-ASCII Key	CHR\$(255)	CHR\$(255)	Non-ASCII keycode

The contents of KBD\$ is destroyed when you transfer it to another string or perform any other operation on KBD\$. Since only one read from KBD\$ is possible, K\$ is used as a temporary storage and work area for the contents of KBD\$, permitting additional string operations.

The first IF ... THEN ... ELSE looks for a CHR\$(255) indicating a non-ASCII key. If none is found, the ASCII key is sent to the datacomm interface. The second IF ... THEN ... ELSE is entered ONLY if the first character indicates a non-ASCII key. It looks for a second CHR\$(255), which is discarded, if found. (Both ENTER and CTRL-ENTER are accepted as end-of-line.) The keystroke data byte is then checked to see if it is the ENTER key. If the value is not equivalent to NUM("E"), the key is rejected. Otherwise, an end-of-line/end-of-block is sent to the datacomm interface.

In more elaborate applications, other keys such as backspace or other cursor control characters could be interpreted, and the CRT display and other program parameters varied accordingly.

5

Note that the interrupt service routine remains active until the entire contents of KBD\$ as it existed at time of interrupt is processed. If, in the meantime, more keystrokes are placed in KBD\$, a new interrupt occurs as soon as the service routine is finished.

## Service Routines for ON KEY Interrupts

ON KEY interrupt service routines are usually simpler than ON KBD service routines. The tasks are usually well-defined and relatively simple. In this example, KEY 0 disconnects the datacomm line, and KEY 1 sends a BREAK. The routines are implemented as follows:

To send a BREAK on either Async or Data Link, set bit zero of Control Register 6. Here is how:

```
2520 Break:CONTROL Sc,6;1
2530 RETURN
```

To disconnect from the datacomm line, clear Control Register 12 as follows:

```
2570 Disco:CONTROL Sc,12;0
2580 DISP "Disconnected"
2590 END
```

You now have a working terminal emulator.

## 5-50 Datacomm Interfaces

---

## Cooperating Programs for BASIC/WS

Some applications, while similar in some respects to terminal emulators, require unattended operation of the desktop computer and network host. In such cases, cooperating programs on the host and terminal computer are used. Applications can include such things as the desktop computer controlling a local data gathering system, making preliminary calculations, and sending the results to the network host. Since data integrity is important in such cases, Data Link is frequently used because of its ability to detect transmission errors.

Here is an example of cooperating programs you can run on your desktop computer and an HP 1000 Data Link network host computer. The FORTRAN program COOP runs on the HP 1000, and is responsible for opening and transferring the specified file(s) from the HP 1000 to the Data Link. A cooperating BASIC program on the desktop computer acts as an interface between the operator and the HP 1000. The specified file is transferred from the Data Link to local mass storage as it is received from the HP 1000. Assuming the file is an ASCII program file containing valid BASIC statements, it can then be attached to the cooperating program and run. Note that variables used by both the original BASIC program and the downloaded program must be specified as COM variables to prevent destroying their values during pre-RUN initialization of the downloaded program. The program listings are as follows:

5

### FORTRAN Program COOP for the HP 1000:

```
FTN4,L
  PROGRAM COOP
C This is a FORTRAN program that runs on the HP 1000 and cooperates
C with a compatible program running simultaneously on a Series 200/300
C computer.
C
C This program waits in I/O suspend until the Series 200/300 computer returns
C a file name. When the name is received, it is parsed, and the
C success status of the parse is sent to the Series 200/300 computer. If the
C file name parses successfully, this program tries to open the file.
C The status of the OPEN is also sent to the Series 200/300 computer.
C
  INTEGER DCB(144),IDBUF(10),IBUF(80)
  INTEGER NAME(3),SCODE,CRN
```

```
INTEGER DTC,ERROR,OK
EQUIVALENCE (NAME,IDBUF),(SCODE,IDBUF(5)),(CRN,IDBUF(6))
```

```
C ***INITIALIZE DTC TO BE THE LU# OF THE SERIES 200/300 COMPUTER***
```

```
DTC=21
```

```
C ***Send the ASCII string "SYNCHRONIZE" to the Series 200/300 computer***
```

```
C This signals the Series 200/300 computer to begin executing the sister
C program to this one.
```

```
CALL EXEC(2,DTC,11HSYNCHRONIZE,-11)
```

```
C ***Now wait in I/O suspend until the Series 200/300 computer sends the***
```

```
C name of the program file that is to be downloaded to the
C Series 200/300 computer.
```

```
CALL EXEC(1,DTC,IBUF,-40)
```

```
CALL ABREG(IA,LEN)
```

```
IP=1
```

```
IF ( NAMR (IDBUF, IBUF, LEN, IP) ) 9200,100
```

```
100 CALL EXEC (2,DTC,2HOK,-2)
```

```
C ***OPEN THE FILE AND SEND THE CONTENTS TO THE SERIES 200/300 COMPUTER***
```

```
IF ( OPEN (DCB,ERROR,NAME,0,SCODE,CRN) ) 9100,200
```

```
200 CALL EXEC (2,DTC,2HOK,-2)
```

```
250 CALL READF(DCB,ERROR,IBUF,80,LENGTH)
```

```
IF (LENGTH,EQ,-1) GOTO 300
```

```
CALL EXEC (2,DTC,IBUF,LENGTH)
```

```
GOTO 250
```

```
C ***TELL THE SERIES 200/300 COMPUTER THAT THE END OF FILE HAS BEEN***
```

```
C REACHED, THEN STOP.
```

```
300 CALL EXEC(2,DTC,11H*ENDOFFILE*,-11)
```

```
STOP
```

```
C *****
```

```
C ERROR HANDLING ROUTINES
```

```
C *****
```

```
C *****THIS ROUTINE HANDLES DISC ERRORS*****
```

5



C BY SENDING THE FMP ERROR AND CLOSING THE FILE.

```
9100 WRITE(DTC,9101)ERROR
9101 FORMAT ("THE OPEN FMP ERROR CODE WAS "I6)
      CALL CLOSE(DCB)
      STOP
```

C \*\*\*\*\*THIS ROUTINE HANDLES PARSING ERRORS\*\*\*\*\*

```
9200 WRITE(DTC,9201)
9201 FORMAT ("THE FILE NAME RECEIVED DID NOT PARSE CORRECTLY")
      STOP
      END
```

### Cooperating BASIC Program for the Desktop Computer:

```
1000 ! *****
1010 ! This BASIC program cooperates with the FORTRAN program "COOP" and
1020 ! downloads a BASIC program file from the HP 1000 for execution on
1030 ! the Series 200/300 computer. While the program is not elegant, it
1040 ! illustrates the basic concepts involved in downloading files to
1050 ! local mass storage, then loading them into memory for execution.
1060 ! The same technique is useful for transferring data files.
1070 !
1080 ! *****
1090 !
1100 COM Sc,Insep$(4),Prompt$(2) ! The values of these variables must be
1110                               ! preserved between programs.
1120 Sc=20                          ! Set select code.
1130 DIM Rx$(1050),Tx$(1050)       ! Set up data transfer strings.
1140 Insep$=CHR$(13)&CHR$(10)&CHR$(27)&"_" ! HP 1000 EOL string.
1150 Esc_u_score$=CHR$(27)&"_"     ! Escape-Underscore.
1160 INTEGER A
1170 !
1180 ! *****
1190 ! Set up DATA LINK protocol
1200 !
1210 CONTROL Sc,0;1                ! Reset the interface.
1220 CONTROL Sc,3;2                ! Set Data Link protocol
1230 Wait: STATUS Sc,38;All_sent
1240 IF NOT All_sent THEN Wait    ! Wait for control block sent.
1250 CONTROL Sc,0;1                ! Reset interface to start new protocol.
1260 !
1270 ! *****
```

```

1280 ! Set up the datacomm configuration.
1290 !
1300         CONTROL Sc,16;0      ! Disable Connect timeout.
1310         CONTROL Sc,17;0      ! Disable No Activity timeout.
1320         CONTROL Sc,20;14     ! Set baud rate to 9600.
1330         CONTROL Sc,21;1      ! GID="A".
1340         CONTROL Sc,22;1      ! DID="A".
1350         CONTROL Sc,23;0      ! Override default switches and set
1360                 ! Hardware Handshake OFF, non-modem connection.
1370         CONTROL Sc,24;0      ! Transmit block length maximum: 512 bytes.
1380         CONTROL Sc,36;0      ! Set parity: NONE (HP 1000 connection).
1390 !
1400 ! *****
1410 ! Connect to the Data Link
1420 !
1430         CONTROL Sc,12;1      ! Send connection command to the interface.
1440         DISP "Trying to connect"
1450 Conn:    STATUS Sc,12;Line_state
1460                 IF line_state3 THEN Conn ! Wait for connection complete.
1470                 DISP "Connected"
1480 !
1490 !*****
1500 !This is a MINIMAL Terminal Emulator.
1510 !
1520 Prompt:  LINPUT Tx$          ! Get line to send to network host.
1530         PRINT USING "#,K";Tx$ ! Print line on CRT.
1540         OUTPUT Sc;Tx$ END    ! Send line to host.
1550 !
1560 Idle:    STATUS Sc,5;Receive ! Look for reply from host.
1570         IF NOT Receive THEN Idle ! If nothing, try again.
1580 !
1590         ENTER Sc USING "#,-K";Rx$ ! Get reply message.
1600         PRINT USING "#,K";Rx$[1,POS(Rx$,Esc_u_score$)-1] ! Print reply.
1610 !
1620 ! Trap messages from HP-1000:
1630 !
1640         IF POS(Rx$,"UNABLE TO COMPLETE LOG-ON") THEN Prompt ! If error,
1650         IF POS(Rx$,"END OF SESSION") THEN Prompt ! try again.
1660         IF POS(Rx$,"SYNCHRONIZE") THEN Coop ! When synchronized, start.
1670 !
1680         STATUS Sc,5;Receive ! Look for line with EOL characters missing.
1690                 ! If not CrLfEsc_, it is a system or sub-
1700                 ! system prompt from the HP 1000. Otherwise,
1710                 ! go to idle loop.
1720         IF NOT Receive AND (POS(Rx$,Insep$)=0) THEN Prompt! Prompt?

```

5

```

1730          GOTO Idle
1740 !
1750 ! *****
1760 ! This section starts the cooperating program.
1770 !
1780 Coop:    LINPUT "TYPE IN A FILE NAME",Tx$    ! Get file name for transfer.
1790 T1:      STATUS Sc,4;Transmit                ! Get transmit queue status.
1800          IF NOT BIT(1,Transmit) THEN T1      ! If not empty, wait.
1810          OUTPUT Sc;Tx$;END                  ! Send file name.
1820 !
1830 R1:      STATUS Sc,5;Receive                 ! Get receive queue status.
1840          IF NOT Receive THEN R1              ! If empty, wait for data.
1850          ENTER Sc USING "#,-K";Rx$          ! Get data. Keep CR-LF.
1860          IF POS(Rx$,"OK") THEN R2           ! If OK, continue.
1870          PRINT Rx$                          ! Not OK. Print error message.
1880          STOP                               ! Error. STOP.
1890 !
1900 R2:      STATUS Sc,5;Receive                 ! Look for another OK from
1910          IF NOT Receive THEN R2              ! the HP 1000.
1920          ENTER Sc USING "#,-K";Rx$          !
1930          IF POS(Rx$,"OK") THEN Rd_prog      ! If OK, start download.
1940          PRINT Rx$                          ! Not OK. Print error message.
1950          STOP                               ! Error. STOP.
1960 !
1970 ! *****
1980 ! For this section to work, the HP 1000 must send the 4-character
1990 ! end-of-line sequence: CR-LF followed by escape-code, underscore.
2000 ! Auto-answer must be disabled, and the data being sent from the
2010 ! HP 1000 MUST consist of valid BASIC program lines, each including a
2020 ! valid line number.
2030 !
2040 Rd_prog:  ASSIGN @File TO "DOWNLOAD"         ! Assign destination file for
2050                                         ! file transfer.
2060 R3:      STATUS Sc,5;Receive                 ! Look for data record.
2070          IF NOT Receive THEN R3              ! If nothing, wait for record.
2080          ENTER Sc USING "#,-K";Rx$          ! Get record. Keep CR-LF.
2090          PRINT Rx$                          ! Print record on printer.
2100          IF POS(Rx$,"*ENDOFFILE*") THEN get_prog !Check for end-of-file.
2110          OUTPUT @File;Rx$[1,POS(Rx$,Esc_u_score$)-1 ! Store record on
2120          GOTO R3          ! Mass Storage file and repeat for next record.
2130 !
2140 Get_prog: ! File has been downloaded to local mass storage.
2150          ASSIGN @File TO *                   ! Close the file.
2160          GET "DOWNLOAD",2200,2200           ! Get the downloaded program.
2170 !

```

2200                    END    ! This statement is destroyed by GET.

### Program File to be Downloaded from the HP 1000:

```
1000 !            This program is downloaded to the desktop computer for execution.
1010 !
1020            DIM A$(20)
1040            PRINT "Now I'll count to 10."
1050            FOR I=1 TO 10
1060            NEXT I
1070            PRINT "That's the end of the demo!!"
1090            PRINT "Nice to meet you, ";A$
1100            GOTO Idle
1110            END
```

### Modified Cooperating BASIC Program After Loading:

5

```
2080            ENTER Sc USING "#,-K";Rx$ ! Get record. Keep CR-LF.
2090            PRINT Rx$                    ! Print record on printer.
2100            IF POS(Rx$,"*ENDOFFILE*") THEN Get_prog !Check for end-of-file.
2120            GOTO R3                    !Mass Storage file and repeat for next record.
2130 !
2140 Get_prog:            ! File has been downloaded to local mass storage. Get it.
2150            ASSIGN @File TO *            ! Close the downloaded file first.
2160            GET "DOWNLOAD",2200,2200    ! Get the downloaded program.
2170 !
2200 !            This program is downloaded to the desktop computer for execution.
2210 !
2220            DIM A$(20)
2230            INPUT "HJ. J'm the downloaded program. What is your name?";A$
2240            FOR I=1 TO 10
2260            PRINT "
2270            NEXT I
2280            PRINT "That's the end of the demo!!"
2290            PRINT "Nice to meet you, ";A$
2300            GOTO Idle
2310            END
```

### Results:

Assuming you have logged onto the HP 1000, the printed output that is displayed on the CRT screen or current PRINTER IS device should look something like this:

```
:
RU<COOP
SYNCHRONIZE
TYPE IN A FILE NAME
FAB2::10
HI, I'm the downloaded program.  What is your name?
SUE
Now I'll count to 10
      :1
      :3
      :4
      :5
      :6
      :7
      :8
      :9
      :10
```

That's the end of the demo!!

Nice to meet you SUE

COOP: STOP

EX

\$END FMGR

FMG21 REMOVED

SESSION 21 OFF 1:26 PM FRI., 11 SEP., 1981

CONNECT TIME: 00 HRS., 08 MIN., 28 SEC.

CPU USAGE 00 HRS., 00 MIN., 00 SEC., 470 MS.

CUMULATIVE CONNECT TIME 01 HRS., 09 MIN., 02 SEC.

END OF SESSION

5

---

## Terminal Emulator Example Programs for BASIC/WS

The following pages contain complete listings of two terminal emulator programs based on the preceding discussion. The first program is for asynchronous data communication with an HP 1000. It can be easily adapted for other remote computers and different operating parameters. The second program uses Data Link to communicate with an HP 1000 network host. It can be used with the HP 3000, but the parity specifier must be changed, and other changes made as appropriate.

Both programs can be enhanced and expanded to include many additional features. The examples shown illustrate the general structure of terminal emulator programs, and are recommended as a basis for developing your own.

Other example programs are also included for your convenience and to further illustrate some of the concepts discussed in this chapter. If you have an HP 46020/21A keyboard, you need to adjust the ON KEY 0 LABEL statement in *line 1750* (and any other affected lines).

5

```

1000 ! *****
1010 ! *
1020 ! *          ****Example Async Terminal Emulator****
1030 ! *
1040 ! *****
1050 ! * This sample terminal emulator program is a simple example of the
1060 ! * program structure of general-purpose emulators. It is not elegant,
1070 ! * but contains the essential elements and illustrates commonly used
1080 ! * programming techniques.
1090 ! *****
1100 !
1110          Sc=20                      ! Select code of datacomm interface.
1120          DIM A$[1],K$[100]          ! Set up string variables.
1130 !
1140 ! Reset datacomm interface and enable Async protocol.
1150 !
1160          CONTROL Sc,0;1              ! Reset card to disconnect from line.
1170          CONTROL Sc,3;1              ! Select Async protocol.
1180 Wait:   STATUS Sc,38;All_sent        ! Wait until Control Block is sent to
1190          IF NOT All_sent THEN Wait   ! interface before resetting again.
1200          CONTROL Sc,0;1              ! Reset card to start new protocol.
1210 !
1220 ! Set up datacomm options. Normally Just a few are included in the
1230 ! program. This group overrides ALL defaults including switches.
1240 !
1250          CONTROL Sc,14;3             ! Set Control Block mask for EOL and Prompt.
1260          CONTROL Sc,15;0             ! No modem line-charge notification.
1270          CONTROL Sc,16;0             ! Disable connection timeout.
1280          CONTROL Sc,17;0             ! Disable No Activity timeout.
1290          CONTROL Sc,18;40            ! Lost Carrier 400ms (default).
1300          CONTROL Sc,19;10           ! Transmit timeout 10 s (default).
1310          CONTROL Sc,20;7             ! Transmit Speed: 300 baud.
1320          CONTROL Sc,21;7            ! Receive Speed: 300 baud.
1330          CONTROL Sc,22;2            ! EQ/AK (as terminal) handshake.
1340          CONTROL Sc,23;1            ! Full Duplex Modem connection.

```

```

1350      CONTROL Sc,24;66      ! Remove protocol characters except
1360                                     ! EOL. Change errors to underscores.
1370      CONTROL Sc,26;6      ! Assign AK character for EQ/AK.
1380      CONTROL Sc,27;5      ! Assign EQ character for EQ/AK.
1390      CONTROL Sc,28;2,13,10 ! Set EOL sequence to CR/LF (default).
1400      CONTROL Sc,31;1,17   ! Set prompt to be DC1 (default).
1405                                     ! Register 33 is not used.
1410      CONTROL Sc,34;2      ! Seven bits per character.
1420      CONTROL Sc,35;0      ! One stop bit per character.
1430      CONTROL Sc,36;1      ! Odd parity.
1440      CONTROL Sc,37;0      ! No inter-character time gap (default).
1450      CONTROL Sc,39;4      ! Set BREAK to 4 character times (default)
1460 !
1470 ! You are now ready to connect to the remote computer.  Optionally, this
1480 ! may include autodialing with the HP 13265A Modem.
1490 !
1500      CONTROL Sc,12;2      ! Start Autodial.
1510      OUTPUT Sc;"> 9 @ (303) 555-1234" ! Send telephone number string.
1520 !
1530 !           | |           Unrecognized characters are ignored.
1540 !           |           Insert 1-second pause (used with PBX to wait for
1550 !           Select FAST dialing rate.                               dial tone).
1560 !
1570 ! Autodialing is not started until Start Connection is initiated by the
1580 ! following CONTROL statement:
1590 !
1600      CONTROL Sc,12;1      ! Start the connection.
1610 !
1620 ! If desired, this is the proper place to monitor STATUS Register 12 to
1630 ! see if the connection is actually made.
1640 !
1650 Conn: STATUS Sc,12;Line_state ! Get Line State from STATUS Register.
1660      IF Line_State=2 THEN DISP "Dialing" ! State=2
1670      IF Line_State=L THEN DISP "Waiting to Connect" ! State=1.
1680      IF Line_State<>3 THEN Conn ! Wait for connection.
1690      DISP "Connected" ! Connection is now complete.
1700 !
1710 ! Softkey 0 is set up so you can disconnect easily.
1720 ! Softkey 1 sends a break to the remote computer.
1730 ! Most other keys are trapped by the ON KBD interrupt service routine.
1740 !
1750      ON KEY 0 LABEL " Disconn" GOTO Disconnect ! Set up Softkey 0.
1760      ON KEY 1 LABEL " Break" GOTO Break ! Set up Softkey 1.
1770      ON KBD GOSUB Keyboard ! Set up keyboard interrupt.
1780 !

```





```

2240 !
2250 ! The keyboard routine loops until the keyboard string has been
2260 ! completely serviced. Notice the similarities between the keyboard and
2270 ! datacomm interrupt service routines.
2280 !
2290 Keyboard:  K$=KBD$
2300 K:      IF NOT LEN(K$) THEN RETURN      ! Stay in routine until K$ is empty
2310      Key=NUM(K$)                       ! Get key or prefix (255=non-ASCII)
2320      K$=K$[2]                          ! Strip first character from string
2330      IF Key=255 THEN                   ! If not 255, transmit character
2340      Key=NUM(K$)                       ! 255. Get value of next character.
2350      K$=K$[2]                          ! Strip second character.
2360      IF Key=255 THEN                   ! If 255 (CONTROL),
2370      Key=NUM(K$)                       ! get third character value.
2380      K$=K$[2]                          ! Strip third character and check
2390      END IF                            ! for ENTER>
2400      IF Key=NUM("E") THEN              ! Check non-ASCII to see if ENTER.
2410      OUTPUT Sc;CHR$(L3);END           ! Send CR then turn line around.
2420      ELSE                              ! Illegal character.Beep and return
2430      BEEP                              ! for next character(s).
2440      END IF
2450      ELSE                              ! ASCII key. Send it to the remote
2460      OUTPUT Sc;CHR$(Key);             ! computer.
2470      END IF                            ! End of character check routine.
2480      GOTO K                            ! Go get next keystroke, if any.
2490 !
2500 ! Key 1 sends a BREAK indication to the datacomm interface card.
2510 !
2520 Break: CONTROL Sc,6;1                 ! Tell card to send a BREAK.
2530      RETURN                            ! End of routine.
2540 !
2550 ! Key 0 disconnects the card and stops the program.
2560 !
2570 Disconnect: CONTROL Sc,12;0          ! Disconnect gracefully.
2580      DISP "Disconnected-"
2590      END

```

5

If you have an HP 46020/21A keyboard, adjust the ON KEY statements to reflect available keys.

```

1000 ! *****
1010 ! *
1020 ! *      *****Example Data Link Terminal Emulator*****
1030 ! *
1040 ! *****

```

```

1050 ! * This sample terminal emulator program is a simple example of the *
1060 ! * program structure of general-purpose emulators. It is not elegant, *
1070 ! * but contains the essential elements and illustrates commonly used *
1080 ! * programming techniques. Line numbers are matched to the Async *
1081 ! * example for your convenience in comparing the two versions. *
1090 ! *****
1100 Sc=20 ! Select code of datacomm interface.
1120 DIM A$[1050],K$[100] ! *****->-> A$ now handles 1000 characters.
1130 !
1140 ! Reset datacomm interface and enable Async protocol.
1150 !
1160 ! CONTROL Sc,0;1 ! Reset card to disconnect from line.
1170 ! CONTROL Sc,3;2 ! Select Data Link protocol.
1180 Wait: STATUS Sc,38;All_sent ! Wait until Control Block is sent to
1190 ! IF NOT All_sent THEN Wait ! interface before resetting again.
1200 ! CONTROL Sc,0;1 ! Reset card to start new protocol.
1210 !
1220 ! Set up datacomm options. Normally just a few are included in the
1230 ! program. This group overrides ALL defaults including switches.
1240 !
1250 ! CONTROL Sc,14;6 ! Set Control Block Mask for ETB/ETX.
1260 ! CONTROL Sc,15;0 ! Set ON INTR mask for data in receive queue.
1270 ! CONTROL Sc,16;0 ! Disable Connection timeout.
1280 ! CONTROL Sc,17;0 ! Disable Lost Carrier timeout.
1290 ! CONTROL Sc,18;40 ! Set Lost Carrier to 400 ms (default).
1300 ! CONTROL Sc,19;10 ! Set Transmit Timeout=10 s (default).
1310 ! CONTROL Sc,20;14 ! Set Line Speed to 9600 baud.
1320 ! CONTROL Sc,21;1 ! Set GID character to "A" (default).
1330 ! CONTROL Sc,22;1 ! Set DID character to "A".
1340 ! CONTROL Sc,23;0 ! Hardware Handshake OFF for HP 13264A.
1350 ! CONTROL Sc,24;0 ! Set transmit block size to 512 (default).
1360 ! CONTROL Sc,36;0 ! Parity not used with HP 1000 (default).
1460 !
1570 ! Now we can initiate Start Connection.
1590 !
1600 ! CONTROL Sc,12;1 ! Start the connection.
1610 !
1620 ! If desired, this is the proper place to monitor STATUS Register 12 to
1630 ! see if the connection is actually made.
1640 !
1645 ! DISP "Trying to connect"
1650 Conn: STATUS Sc,12;Line_state ! Get Line State from STATUS Register.
1680 ! IF Line_state<>3 THEN Conn ! Wait for connection.
1690 ! DISP "Connected" ! Connection is now complete.
1700 !

```



```

2160 !
2170 ! This keyboard routine is not very exotic, but it CAN handle a fast
2180 ! typist. Some of the nested IF ... THENs are used to decode the 255-
2190 ! and 255-255 notations for special and CONTROL-special keys. The only
2200 ! special key allowed by this routine is ENTER (code is NUM("E")). It
2210 ! is connected to an end-of-block (;END) indication. All ASCII keys are
2220 ! transmitted to the card without alteration.
2240 !
2250 ! The keyboard routine loops until the keyboard string has been
2260 ! completely serviced. Notice the similarities between the keyboard and
2270 ! datacomm interrupt service routines.
2280 !
2290 Keyboard: K$=KBD$
2300 K:   IF NOT LEN(K$) THEN RETURN ! Stay in routine until K$ is empty.
2310     Key=NUM(K$)                 ! Get key or prefix (255=non-ASCII).
2320     K$=K$[2]                   ! Strip first character from string.
2330     IF Key=255 THEN             ! If not 255, transmit character
2340         Key=NUM(K$)            ! 255. Get value of next character.
2350     K$=K$[2]                   ! Strip second character.
2360     IF Key=255 THEN             ! If 255 (CONTROL).
2370         Key=NUM(K$)            ! get third character value.
2380         K$=K$[2]              ! Strip third character and check
2390     END IF                      ! for ENTER.
2400     IF Key=NUM("E") THEN        ! Check non-ASCII to see if ENTER.
2410         OUTPUT Sc;END          ! Send end-of-block.
2420     ELSE                        ! Illegal character. Beep and return
2430         BEEP                   ! for next character(s).
2440     END IF
2450     ELSE                        ! ASCII key. Send it to the remote
2460         OUTPUT Sc;CHR$(Key);    ! computer.
2465         PRINT USING "#,A";CHR$(Key) ! Print character not echoed by DL.
2470     END IF                      ! End of character check routine.
2480     GOTO K                      ! Go get next keystroke, if any.
2490 !
2500 ! Key 1 sends a BREAK indication to the datacomm interface card.
2510 !
2520 Break:   CONTROL Sc,6;1        ! Tell card to send a BREAK.
2530         RETURN                 ! End of routine.
2540 !
2550 ! Key 0 disconnects the card and stops the program.
2560 !
2570 Disconnects: CONTROL Sc,12;0  ! Disconnect gracefully.
2580         DISP "Disconnected"
2590         END

```

---

## **Datacomm Programming Helps for BASIC/WS**

This section is designed to assist you in writing datacomm programs for special applications by discussing selected techniques and characteristics that can present obstacles to the beginning programmer.

### **Terminal Prompt Messages**

Care must be exercised to ensure that messages are never transmitted to the network host if the host is not prepared to properly handle the message. Receipt of a poll from the host does not necessarily mean that the host can handle the message properly when it is received. Therefore, prompts or interpretation of messages from the host are used to determine the status of the host operating system.

Prompts are message strings sent to the terminal by a cooperating program. They are well-defined and predictable, and are usually tailored to specific applications. When the terminal interacts directly with RTE or one or more subsystems, the process becomes less straightforward. Each subsystem usually has its own prompt which is not identical to other subsystem prompts. To maintain orderly communication with subsystems, you must interpret each message string from the host to determine whether it is to be treated as a prompt.

### **Prevention of Data Loss on the HP 1000**

On the HP 1000, the RTE Operating System manages information transfer between programs or subsystems and system I/O devices, including DSN/DL. Terminals are continually polled by the host's data link interface (unless auto-poll has been disabled by use of an HP 1000 File Manager CN command). Since there is no relationship between automatic polling and HP 1000 program and subsystems execution, it is possible to poll a terminal when there is no need for information from that terminal. If the terminal sends a message in response to a poll when no data is being requested, the HP 1000 discards the message, causing the data to be lost, and treats it as an asynchronous interrupt. A break-mode prompt is then sent to the terminal by the host.

The terminal must determine that the host is ready to receive a message in order to ensure that messages are properly handled by the host. This is done by checking all messages from the host (ENTER until queue is empty) and

not transmitting (OUTPUT) until a prompt message or its equivalent has been received (unless you want to enter break-mode operation). Since the HP 1000 does not generate a consistent prompt message for all programs and subsystems, it is easiest to use cooperating programs to generate a predictable prompt. If your application requires interaction with other subsystems, prompts can usually be most easily identified by the ABSENCE of the sequence: CrLfEc\_ at the end of a message. When a proper sequence has been identified, you are reasonably certain that the host is ready for your next message block.

Here is an example of host messages where a prompt is sent by the File Manager (FMGR) and answered by a RUN, EDITR command. Note that the prompt from the interactive editor fits the description of a prompt because a line-feed is not included after the carriage-return in the sequence.

5	:Ec	Prompt is sent by FMGR to terminal.
	RU,EDITR	EDITR Run command is sent to host.
	SOURCE FILE	File name message is sent by the host,
	NAME?CrLfEc_Cr/BlEc_	followed by a prompt sequence which has no line-feed. Sequence is different from FMGR prompt.

Whenever an unexpected message from a terminal is received by RTE, it is treated as an asynchronous interrupt which terminates normal communication with that terminal. A break-mode prompt is sent to the terminal by RTE, and the next message is expected to be a valid break-mode command. If the message is not a valid command (such as data in a file being transferred), the data is discarded, and an error message is sent to the terminal. If, in the meantime, the cooperating program or subsystem generates an input request, the next data block is sent to the proper destination, but is out of sequence because at least one block has been lost. You can prevent such data losses and the mass confusion that usually ensues (especially during high-speed file transfers to the host), by disabling auto-poll on the HP 1000 data link interface. With auto-poll OFF, no polls are sent to your terminal unless the host is prepared to receive data.

## Disabling Auto-poll on the HP 1000

To operate with auto-poll OFF, log on to the network host, disable auto-poll, perform all datacomm activities and file transfers, enable auto-poll, then log off. *If you don't enable auto-poll at the end of a session, polling is suspended to you terminal after log-off, and you cannot reestablish communication with the host unless polling is restored from another terminal or the network host System Console.*

The auto-poll ON/OFF commands are:

```
CN,LU#,23B,101401B  Auto-poll OFF1
CN,LU#,23B,001401B  Auto-poll ON1
```

where LU# is the logical unit number assigned to your terminal.

<sup>1</sup> The File Manager CN (Control) command parameters for the multipoint interface are described in more detail in the *91730A Multipoint Terminal Interface Subsystem User's Guide*.

When auto-poll is disabled, no polls are sent to your terminal unless an input request is initiated by the cooperating program or subsystem on the network host. When the request is made, a poll is scheduled, and polling continues until a reply is received from the terminal. When the reply is received, and acknowledged, polling is suspended until the next input is scheduled. Operating with auto-poll OFF is especially useful when transferring files to the HP 1000. Otherwise, in most applications, it is practical to leave auto-poll ON.

## Prevention of Data Loss on the HP 1000

Neither the HP 1000 nor the HP 3000 provide a DC1 poll character when they are ready for data inputs from DSN/DL. The HP 3000, like the HP 1000, also discards data if it has not requested the transfer. Since the HP 3000 does not provide an auto-poll disable command, you must interpret messages from the HP 3000 to determine that it is ready for the next data block before you transmit the block.

## Secondary Channel, Half-duplex Communication

Half-duplex telecommunications links frequently use secondary channel communication to control data transmission and provide for proper line turn-around. This is done by using Secondary Request-to-send (SRTS) and Secondary Data Carrier Detect (SDCD) modem signals.

Consider two devices communicating with each other: Each connects to the datacomm link, then waits for SDCC to become active (true). As each device connects to the line, Secondary Request-to-send is enabled, causing each modem to activate its secondary carrier output. The Secondary Data Carrier Detect is, in turn, activated by each modem as it receives the secondary data carrier from the other end.

When communication begins, the first device to transmit (assumed to be your computer, in this case) clears its Secondary Request-to-send modem line. This removes the secondary data carrier from the line, causing the other modem to clear SDCC to its terminal or computer, telling it that you have the line. (The modems also maintain proper line switching and prevent timing conflicts so both ends don't try to get the line simultaneously.) The other device receives data, and must not attempt to transmit until you relinquish control of the line as indicated by SDCC true. After you finish transmitting, you must again activate SRTS so that SDCC can be activated to the other device, allowing it to use the line if it has a message.

The following example is a simple terminal emulator that uses secondary channel communication to control data flow on a half-duplex link:

```
1000 ! *****
1010 ! *
1020 ! *   HALF-DUPLEX TERMINAL EMULATOR FOR SECONDARY CHANNEL OPERATION   *
1030 ! *
1040 ! * This program uses secondary channel modem lines to indicate which *
1050 ! * end is in control of the line. BASIC is used to assemble data     *
1060 ! * for transmission to the other end. This example is compatible     *
1070 ! * with the Option 001 (male) cable only.                             *
1080 ! *
1090 ! *****
1100 !
1110     Sc=20                ! Select code of HP 98628 datacomm interface.
1120     DIM A$[1],K$[100] ! Size of datacomm and keyboard strings.
1130 !
```



```

1140 !   Reset the card to disconnect, then select Async protocol.
1150 !
1160     CONTROL Sc, 0;1
1170     CONTROL Sc, 3;1
1180 Wait: STATUS Sc,38; All_sent
1190     IF NOT ALL_sent THEN Wait
1200     CONTROL Sc, 0;1
1210 !
1220 !   Set up all the interface configuration options for Async protocol.
1230 !
1240     CONTROL Sc, 14;0     ! Set Control Block mask off.
1250     CONTROL Sc,15;16   ! Interrupt when Secondary Carrier Detect
1255                             !   modem line changes state.
1260     CONTROL Sc,16;0     ! Disable connection timeout.
1270     CONTROL Sc,17;0     ! Disable No Activity timeout.
1280     CONTROL Sc,18;40   ! Lost Carrier 400 ms (default)
1290     CONTROL Sc,19;10   ! Transmit timeout 10 ds (default).
1300     CONTROL Sc,20;7,7  ! Line speed: 300 baud in both directions.
1310     CONTROL Sc,22;0     ! Disable protocol handshake.
1320     CONTROL Sc,23;2     ! Half duplex modem connection.
1330     CONTROL Sc,24;255  ! Do not remove protocol characters.
1340     CONTROL Sc,28;2,13,10! EOL sequence CR/LF (default).
1350     CONTROL Sc,31;1,17 ! Prompt DC1 (default).
1360     CONTROL Sc,34;2     ! 7 bits per character.
1370     CONTROL Sc,35;0     ! 1 stop bit.
1380     CONTROL Sc,36;1     ! odd parity.
1390     CONTROL Sc,37;0     ! No inter-character gap (default).
1400     CONTROL Sc,39;4     ! Set Break to 4 character times (default)
1410 !
1420 !   Initiate connection to the telecommunications line.
1430 !
1440     CONTROL Sc,12;1
1450 !
1460 !   Tell the operator what is happening, then wait for connection to finish.
1470 !
1480     DISP "Waiting to connect"
1490 Conn: STATUS Sc,L2;Line_state
1500     IF Line_state=L THEN Conn
1510     DISP "Waiting for SDCD to become active"
1520 !
1530 !   Get the SDCD handshake started properly by waiting for the other end to
1540 !   relinquish control of the line by activating SDCD.
1550 !
1560 Statck:STATUS Sc,7;Modem_lines
1570     IF NOT BINAND(Modem_lines,16) THEN Statck

```

```

1580     DISP "Connected"
1590     !
1600     ! Set up a key to gracefully disconnect the datacomm connection.
1610     !
1620         ON KEY 0 LABEL " Disconn" GOTO Disconnect
1630     !
1640     ! Interrupt on data received or modem line change (change in SDCD)
1650     !
1660         ON INTR Sc GOSUB Datacomm
1670         ENABLE INTR Sc;1+8
1680     !
1690     ! Send a "READY" message to the remote to get things started. This is
1700     ! optional.
1710     !
1720         CONTROL Sc,8;7 ! Put down SRTS
1730         OUTPUT Sc;"READY";CHR$(L3);END
1740         CONTROL Sc,8;15 ! Put up SRTS
1750     !
1760     ! The background idle loop simply waits for interrupts to happen.
1770     !
1780 Background: GOTO Background
1790     !
1800     ! *****
1810     !             DATACOMM INTERRUPT SERVICE ROUTINE
1820     !
1830     ! First, acknowledge interrupt by reading STATUS register 4.
1840     !
1850     ! Read all existing data in the buffer.
1860     !
1870     ! When SDCD becomes true, it indicates that the remote is through
1880     ! transmitting. A LINPUT statement is provided to let the user enter a
1890     ! line of data. The line is then sent to both the screen and the
1900     ! datacomm card. To maintain control of the line, we disable SRTS
1910     ! (Control Register 8), then reactivate it when we are through sending.
1920     !
1930     ! Finally, re-enable interrupts and exit the interrupt routine.
1940     !
1950 Datacomm:STATUS Sc,4;Interrupt_bits
1960 Read: STATUS Sc,5;Rx_avail_bits
1970         IF Rx_avail_bits=0 THEN Chkmdm
1980             ENTER Sc USING "#,-K";A$
1990             PRINT USING "#,K";A$
2000         GOTO Read
2010 Chkmdm:STATUS Sc,7;Modem_lines
2020         IF BINAND(Modem_lines,L6) THEN

```

5

## 5-70 Datacomm Interfaces

```

2030          CONTROL Sc,8;7! Pup down SRTS
2040          LINPUT "Line to send ... ?",K$
2050          PRINT K$
2060          OUTPUT Sc;K$;CHR$(13);END
2070          CONTROL Sc,8;15 ! Put up SRTS
2080          END IF
2090          ENABLE INTR Sc
2100          RETURN
2110 ! *****
2120 ! Key 0 was set up to disconnect from the datacomm line.
2130 !
2140 Disconnect:CONTROL Sc,12;0
2150          DISP "Disconnected"
2160          END

```

## Automatic Answering Applications

Desktop computers are sometimes used in applications where they may have to be able to automatically answer incoming calls from other computers by means of public (switched) telephone lines. For instance, a desktop computer may be located at an unattended remote site in a data gathering network where the network host computer periodically calls the remote site for data updates. In other situations, the desktop computer may be the host for several computers or terminals that originate the calls. Other applications may require that two (or more) desktop computers be able to call each other in either direction at will.

In automatic answering applications, the Ring Indicator (RI) modem line is used by the desktop computer to recognize incoming calls from the host. This enables the desktop computer to answer the call by connecting to the datacomm line. Usually, a continuously running program on the unattended computer contains an ON INTR statement set up to monitor the RI modem signal. When RI is activated by the incoming call, normal program flow is interrupted, and the connection is initiated. The desktop computer then sets up the necessary datacomm and other program interrupts, and proceeds to the program segment responsible for transferring data to the remote computer. The following example illustrates the general technique and how it fits into overall program structure:

```

1000 ! *****
1010 ! *

```

```

1020 ! *      TERMINAL EMULATOR WITH AUTOMATIC ANSWERING CAPABILITY      *
1030 ! *
1040 ! * This program waits for the ring-indicator modem line to change  *
1050 ! * (indicating an incoming datacomm call), then connects to the    *
1060 ! * datacomm line. Use with Option 001 (male) modem cable.          *
1070 ! *
1080 ! *****
1090 !
1100     Sc=20                      ! Select code of HP 98628 datacomm interface.
1110     DIM A$[1],K$[100]         ! Size of datacomm and keyboard strings.
1120 !
1130 ! Reset the card to disconnect, then select Async protocol.
1140 !
1150     CONTROL Sc,0;1
1160     CONTROL Sc,3;1
1170 Wait: STATUS Sc,38;All_sent
1180     IF NOT All_sent THEN Wait
1190     CONTROL Sc,0;1
1200 !
1210 ! Set up all the interface configuration options for Async protocol.
1220 !
1230     CONTROL Sc,14;0           ! Set Control Block mask off.
1240     CONTROL Sc,15;8           ! Interrupt when Ring Indicator line changes.
1250     CONTROL Sc,16;0           ! Disable connection timeout.
1260     CONTROL Sc,17;0           ! Disable No Activity timeout.
1270     CONTROL Sc,18;40          ! Lost Carrier400ms (default).
1280     CONTROL Sc,19;10          ! Transmit timeout 10 s (default).
1290     CONTROL Sc,20;7,7         ! Line speed: 300baud in both directions.
1300     CONTROL Sc,22;0           ! Disable protocol handshake.
1310     CONTROL Sc,23;1           ! Full duplex modem connection.
1320     CONTROL Sc,24;255        ! Remove no protocol characters.
1330     CONTROL Sc,28;2,13,10! EOL sequence CR/LF (default).
1340     CONTROL Sc,31;1,17!      ! Prompt DC1 (default).
1350     CONTROL Sc,34;2           ! 7 bits per character.
1360     CONTROL Sc,35;0           ! 1 stop bit.
1370     CONTROL Sc,36;1           ! Odd parity.
1380     CONTROL Sc,37;0           ! No inter-character gap (default).
1390     CONTROL Sc,39;4           ! Set Break to 4 character times (default).
1400 !
1410 ! Wait for Ring Indicator modem line to change.
1420 !
1430     ON INTR Sc GOTO Ri_int
1440     ENABLE INTR Sc;8
1450     DISP "Waiting for ring to come in"
1460 Waitri:GOTO Waitri

```

5

```

1470 !
1480 ! When interrupt occurs, initiate connection to the datacomm line.
1490 !
1500 Ri_int:CONTROL Sc,12;1
1510 !
1520 ! Tell the operator what is happening, then wait for connection to finish.
1530 !
1540     DISP "Waiting to connect"
1550 Conn: STATUS Sc,L2;Line_state
1560     IF Line_state=1 THEN Conn
1570     DISP "Connected"
1580 !
1590 ! Set up key 0 to gracefully disconnect from the datacomm line, then
1600 ! set up key 1 to send a break.
1610 !
1620     ON KEY 0 LABEL " Disconn" GOTO Disconnect
1630     ON KEY 1 LABEL "   Break" GOSUB Break
1640 !
1650 ! Interrupt on data received. Also set up keyboard interrupts.
1660 !
1670     ON INTR Sc GOSUB Datacomm
1680     ENABLE INTR Sc;L
1690     ON KBD GOSUB Keyboard
1700 !
1710 ! The background idle loop simply waits for interrupts to happen.
1720 !
1730 Background: GOTO Background
1740 ! *****
1750 !             DATACOMM INTERRUPT SERVICE ROUTINE
1760 !
1770 ! First, acknowledge interrupt by reading STATUS register 4.
1780 !
1790 ! Re-enable interrupts, then read all existing data in the buffer.
1800 !
1810 ! When the buffer is empty, exit the service routine.
1820 !
1830 Datacomm:STATUS Sc,4;Interrupt_bits
1840     ENABLE INTR Sc
1850 Read: STATUS Sc,5;Rx_avail_bits
1860     IF Rx_avail_bits=0 THEN RETURN
1870     ENTER Sc USING "#,-K";A$
1880     PRINT USING "#,K";A$
1890     GOTO Read
1900 ! *****
1910 ! This keyboard interrupt service routine is similar to the other

```

```

1920 ! examples in this chapter. It sends ASCII keys to the remote, and
1930 ! accepts ENTER as a Carriage-Return. Other keys cause a BEEP.
1940 !
1950 Keyboard:K$=KBD$
1960 K: IF NOT LEN(K$) THEN RETURN ! Repeat until K$ is empty:
1970 Key=NUM(K$) ! Get key or prefix
1980 K$=K$[2]
1990 IF Key=255 THEN
2000 Key=NUM(K$) ! If prefix, get next character
2010 K$=K$[2]
2020 IF Key=255 THEN ! If control-key prefix, get
2030 Key=NUM(K$) ! the third character
2040 K$=K$[2]
2050 END IF
2060 IF Key=NUM("E") THEN ! Check for ENTER key
2070 OUTPUT Sc;CHR$(L3);END ! If so, send carriage return
2080 ELSE
2090 BEEP
2100 END IF
2110 ELSE
2120 OUTPUT Sc;CHR$(Key); ! ASCII key: just send it
2130 END IF
2140 GOTO K ! Repeat until K$ is empty
2150 ! *****
2160 ! Key 1 was set up to send a break.
2170 !
2180 Break: CONTROL Sc,6;1
2190 RETURN
2200 !
2210 ! Key 0 was set up to disconnect the interface from the datacomm line.
2220 !
2230 Disconnect:CONTROL Sc,12;0
2240 DISP "Disconnected"
2250 END

```

5

## Communication Between Desktop Computers

Two desktop computers can be connected, directly, or by use of modems. DC1/DC3 handshake protocol can be used conveniently to enable each computer to transmit at will without risk of buffer or queue overruns. To ensure proper operation, the following guidelines apply:

- Set up Control Register 22 with a value of 5. This allows both computers to act either as host or terminal in any given situation, depending on which one initiates the action.
- Set up Control Registers 26 and 27 for DC1 and DC3 respectively, or use two other characters if necessary.
- Data to be transmitted must NOT contain any characters matching the contents of Control Register 26 or 27. This prevents the receiving interface from confusing data with control characters.
- If both computers attempt to transmit large amounts of data at the same time, a lock-up condition may result where each side is waiting for the other to empty its buffers.

---

## Datacomm Error Recovery

5

When a forced disconnect terminates the connection, the interface is placed in a **SUSPENDED** state. The interface cannot be reconnected to the datacomm line when it is **SUSPENDED**. **CLEAR**, **ABORT**, and **RESET** are used to recover from the suspended state and resume normal card operation. Executing **OUTPUT** and **CONTROL** statements while the card is suspended places corresponding data and control block(s) in the transmit (outbound) queue and can continue to do so until the queue is filled, at which time the desktop computer operating system hangs. **ENTER** statements can be executed to retrieve data that was there prior to **SUSPEND** until the receive (inbound) queue is empty. Subsequent **ENTER** statements, if executed while the card is suspended, hang the computer.

To recover from a **SUSPENDED** interface, three programmable options are available, all of which destroy any existing data in the transmit and receive queues. They are:

- The **CLEAR** statement clears the receive and transmit queues. In addition, if the interface card is suspended, it disconnects the card from the datacomm line. If the card is not suspended, its connection state is not changed, but the queues are cleared.

- The ABORT statement is identical to the CLEAR statement, except that the interface card is unconditionally disconnected from the datacomm line.
- RESET interface (Control Register 0) clears all buffers and queues, and resets all CONTROL options to their power-up state.

A fourth (keyboard only) option is available. **SHIFT PAUSE** (or **RESET**) causes a hardware reset to be sent to ALL peripherals. This completely resets the datacomm interface to its power-up state.

---

## Datacomm Error Detection and Program Recovery

5

When a timeout or datacomm error occurs, an interrupt is generated by the interface card to BASIC. If an ON ERROR is active for that select code, the error is trapped and handled by the error routine specified by the ON ERROR statement. If no ON ERROR is active for that select code, the program is stopped at the end of the current line by the BASIC operating system, and an error message is sent to the PRINTER IS device.

When a datacomm error is trapped by an error routine, the routine must decide what to do about the problem. Since datacomm interface errors are not related to a specific program line, the ERRL function is always false, and ERRN returns the error number generated by the interface card. ERRL and ERRN are discussed in greater detail in the BASIC Programming Techniques manual for your desktop computer.

---

## Cable and Adapter Options and Functions

The HP 98628A Datacomm Interface is available with RS-232C DTE and DCE cable configurations, or it can be connected to various modems or adapters for other applications.



## DTE and DCE Cable Options

DTE and DCE cable options are designed to simplify connecting two desktop computers without the use of modems. The DTE cable (male RS-232 connector) is configured to make the datacomm interface look like standard data terminal equipment when it is connected to an RS-232C modem. The DCE cable (female RS-232 connector) is configured so that it eliminates the need for modems in a direct connection. When you connect two computers to each other in a direct non-modem connection, both datacomm interfaces are functionally identical. The DCE cable acts as an adapter so that both interfaces behave exactly as they would if they were connected to a pair of modems by means of DTE cables.

Several signal lines are rerouted in the DCE cable so that, in direct connections, outputs from one interface are connected to the corresponding inputs on the other interface. Certain outputs on each interface are also connected to inputs on the same card by “loop-back” connections in the DCE cable.

The schematic diagram in this section shows two datacomm interfaces directly connected through a DTE-DCE cable pair. Note that the DCE cable wiring complements the DTE cable so that output signals are properly routed to their respective destinations. Signal names at the RS-232C connector interface are the same as the signal names for the DTE interface. However, because the DCE cable adapts signal paths, the signal name at the RS-232C connector does not necessarily match the signal name at the DCE interface. Connector pin numbers are included in the diagram for your convenience.

**RS-232C DTE (male) Cable Signal Identification Tables**

Signal RS-232C	Signal V.24	Interface Pin#	RS-232C Pin#	Mnemonic	I/O	Function
AA	101	24	1	—	—	Safety Ground
BA	103	12	2	Out		Transmitted Data
BB	104	42	3	In		Received Data
CA	105	13	4	RTS	Out	Request to Send
CB	108	44	5	CTS	In	Clear to Send
CC	107	45	6	DSR	In	Data Set Ready
AB	102	48	7	—	—	Signal Ground
CF	109	46	8	DCD	In	Data Carrier Detect
SCF (OCR2)	122	47	12	SDCD	In	Secondary DCD
DB	114	41	15	In		DCE Transmit Timing
DD	115	43	17	In		DCE Receive Timing
SCA (OCD2)	120	15	19	SRTS	Out	Secondary RTS
CD	108.1	14	20	DTR	Out	Data Terminal Ready
CE (OCR1)	125	9	22	RI	In	Ring Indicator
CH (OCD1)	111	40	23	DRS	Out	Data Rate Select
DA	113	7	24	Out		Terminal Transmit Timing

5

## Optional Circuit Driver/Receiver Functions

Two optional drivers and receivers are used with the RS-232C cable options. Their functions are as follows:

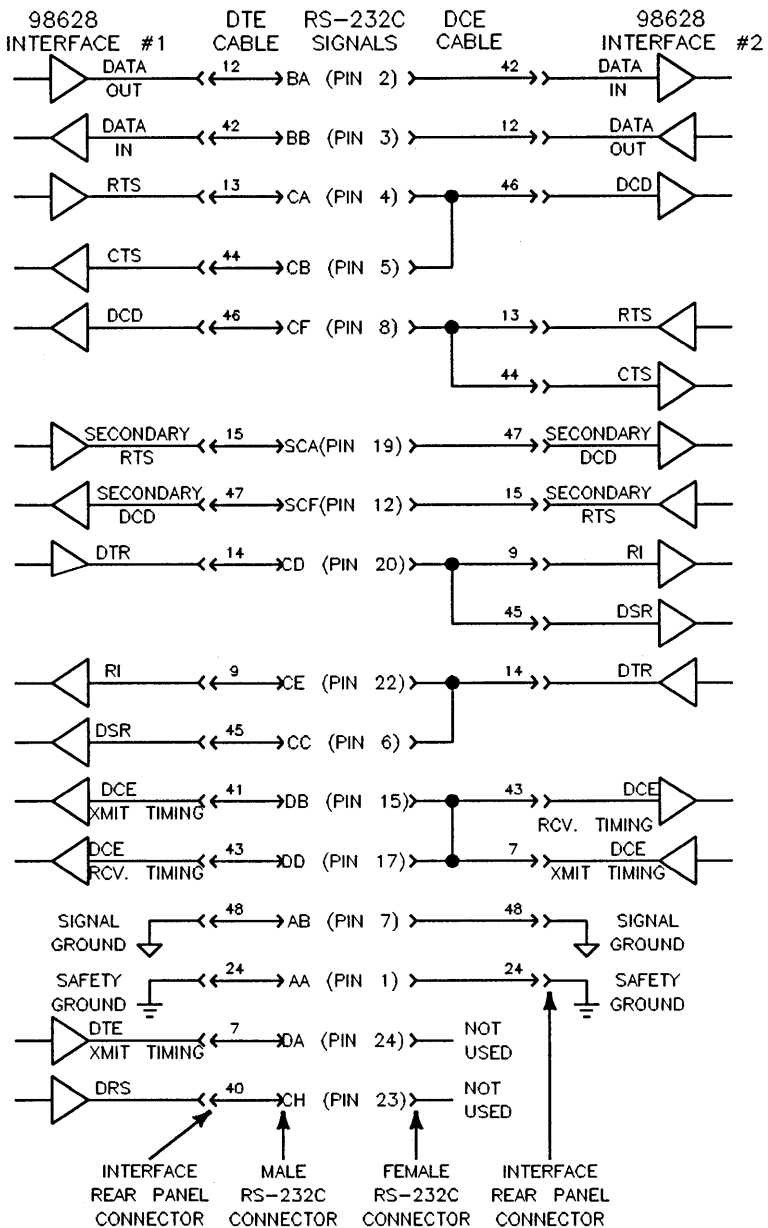
### Drivers

Name	Function
OCD1	Data Rate Select
OCD2	Secondary Request-to-send
OCD3	Not used
OCD4	Not used

### Receivers

Name	Function
OCR1	Ring Indicator
OCR2	Secondary Data Carrier Detect

OCD2 is used for autodial pulsing in the HP 13265A Modem. None of the optional drivers and receivers are used for Data Link and Current Loop Adapters.



**DTE/DCE Interface Cable Wiring**

## RS-232C/CCITT V24

The following table provides information about each data communications interface function. The pin assignments are also shown. Not all of the functions provided by RS-232C are implemented. The functions denoted with an \* are implemented.

### RS-232C/CCITT V.24<sup>1</sup>

RS-232C	CCITT V.24	Signal Name
*Pin 1	101	<i>Protective Ground.</i> Electrical equipment frame and ac power ground.
*Pin 2 <sup>2</sup>	103	<i>Transmitted Data.</i> Data originated by the terminal to be transmitted via the sending modem.
*Pin 3 <sup>2</sup>	104	<i>Received Data.</i> Data from the receiving modem in response to analog signals transmitted from the sending modem.
*Pin 4	105	<i>Request to Send.</i> Indicates to the sending modem that the terminal is ready to transmit data.
*Pin 5	106	<i>Clear to Send.</i> Indicates to the terminal that its modem is ready to transmit data.
*Pin 6	107	<i>Data Set Ready.</i> Indicates to the terminal that its modem is not in a test mode and that modem power is ON.
*Pin 7 <sup>2</sup>	102	<i>Signal Ground.</i> Establishes common reference between the modem and the terminal.
*Pin 8	109	<i>Data Carrier Detect.</i> Indicates to the terminal that its modem is receiving carrier signals from the sending modem.
Pin 9		Reserved for test.
Pin 10		Reserved for test.

<sup>1</sup> International Telephone and Telegraph Consultative Committee European standard.

<sup>2</sup> Signal on this pin is commonly used for three-wire (no modem) links.

**RS-232C/CCITT V.24 (continued)**

RS-232C	CCITT V.24	Signal Name
Pin 11		Unassigned.
*Pin 12	122	<i>Secondary Data Carrier Detect.</i> Indicates to the terminal that its modem is receiving secondary carrier signals from the sending modem.
Pin 13	121	<i>Secondary Clear to Send.</i> Indicates to the terminal that its modem is ready to transmit signals via the secondary channel.
Pin 14	118	<i>Secondary Transmitted Data.</i> Data from the terminal to be transmitted by the sending modem's channel.
*Pin 15	114	<i>Transmitter Signal Element Timing.</i> Signal from the modem to the transmitting terminal to provide signal element timing information.
Pin 16	119	<i>Secondary Received Data.</i> Data from the modem's secondary channel in response to analog signals transmitted from the sending modem.
*Pin 17	115	<i>Receiver Signal Element Timing.</i> Signal to the receiving terminal to provide signal element timing information.
Pin 18		Unassigned.
*Pin 19	120	<i>Secondary Request to Send.</i> Indicates to the modem that the sending terminal is ready to transmit data via the secondary channel.
*Pin 20	108.2	<i>Data Terminal Ready.</i> Indicates to the modem that the associated terminal is ready to receive and transmit data.
Pin 21	110	<i>Signal Quality Detector.</i> Signal from the modem telling whether a defined error rate in the received data has been exceeded.

### RS-232C/CCITT V.24 (continued)

RS-232C	CCITT V.24	Signal Name
*Pin 22	125	<i>Ring Indicator.</i> Signal from the modem indicating that a ringing signal is being received over the line.
*Pin 23	111	<i>Data Signal Rate Selector.</i> Selects one of two signaling rates in modems having two rates.
*Pin 24	113	<i>Transmit Signal Element Timing.</i> Transmit clock provided by the terminal.
Pin 25		Unassigned.

---

## The HP 98642 4-Channel Multiplexer

This interface is supported by BASIC/UX for data communications. The topics covered in this section are:

- Specifics on the HP 98642 4-channel multiplexer
- Using the HP 98642 4-channel multiplexer
- Keywords used by the HP 98642 4-channel Multiplexer.

### Specifics on the HP 98642 4-Channel Multiplexer

The HP 98642 4-channel multiplexer has one port that functions the same as an HP 98628 Data Communication interface card and the remaining ports function the same as an HP 98628 interface card without the hardware handshaking or modem control. Each port has its own set of STATUS and CONTROL registers that are the same as those for an HP 98628 interface card. The datacomm interface multiplexer allows your computer to communicate with any device (such as a modem) that is compatible with standard asynchronous data communication protocols.

## Using the HP 98642 4-Channel Multiplexer

To communicate with another device using the multiplexer, you need to know the multiplexer's select code (for example, select code 16) and the primary address of each port on the multiplexer. The primary addresses associated with these ports are:

- 00                    This port functions the same as an HP 98628 interface card.
- 01, 02, and        These ports function the same as an HP 98628 interface card
- 03                    without hardware handshaking or modem control.

The select code and primary address together form the device selector. If the HP 98642's select code is 16, then it will have four ports with device selectors: 1600, 1601, 1602, and 1603. The following example shows you how to use device selector 1600 to check for the current transmit timeout limit for a device connected to an HP 98642 multiplexer at port 1.

5                    STATUS 1600,19;Tran\_stat

## Keywords Used by the HP 98642 4-Channel Multiplexer

The following table contains a list of keyword examples used by the HP 98642 4-Channel Multiplexer and a description of the examples. Note that these same keywords can be used by the HP 98628 Data Communications interface.



## Keywords Used by the HP 98642 Interface Card

Keyword Examples	Example Description
ABORTIO @Source	Terminates any active transfer associated with the I/O path name. Assume that the I/O path name called @Source was assigned device selector 1600 (see the keyword ASSIGN given below).
ASSIGN 1600 TO @Source ASSIGN @Buffer TO BUFFER Real_buf(*)	Assigns the I/O path name called @Source to device selector 1600 and the I/O path name called @Buffer to a named buffer called Real_buf(*).
BREAK 1601	Directs the datacomm interface located at device selector 1601 to send a break sequence.
ENTER 1601 USING "K";Str_val\$	Reads character values from device selector 1601 into the string called Str_val\$.
CONTROL 1602,6;1	Causes a BREAK to be sent to device selector 1602.
OFF TIMEOUT 1603	Cancels event-initiated branches, from the interface at device selector 1603, previously defined and enabled by an ON TIMEOUT statement. An OFF TIMEOUT without any device selector disables all of the channels of the HP 98642 4-Channel Multiplexer.
ON TIMEOUT 1603,10 GOSUB Time_out	Defines and enables an event-initiated branch to be taken when an I/O timeout occurs on the specified interface located at device selector 1603.
OUTPUT 1602 USING "DD";22	Writes the integer value 22 to device selector 1602.

### Keywords Used by the HP 98642 Interface Card (continued)

Keyword Examples	Example Description
RESET 1603	Resets the interface located at device selector 1603.
STATUS 1602,0;Ret_val	Returns the card identification status of device selector 1602.
TRANSFER @Source TO @Buffer	Transfers the contents of the I/O name path called @Source to the I/O path name called @Buffer. Note that @Source was created using the keyword ASSIGN and device selector 1600, and @Buffer was created using the secondary keyword BUFFER with the keyword ASSIGN and the array called Real_buf(*) (see the keyword ASSIGN given above).

5

## HP 98628 and HP 98642 Datacomm Interface Status and Control Registers

Control registers accept values in the range of zero through 255. Some registers require specified values, as indicated. Illegal values or values less than zero or greater than 255, cause error 327.

Reset value, shown for various control registers, is the default value used by the interface after a reset or power-up until the value is overridden by a CONTROL statement.

*STATUS 0*            Card Identification  
 Value returned: 52 (if 180 is returned, check select code switch cluster and make sure switch R is ON).

*CONTROL 0*         Card Reset  
 Any value, 1 thru 255, resets the card. Immediate execution. Data in queues is destroyed.

*STATUS 1* Hardware Interrupt Status (not used in most applications)  
1=Enabled 0=Disabled

*STATUS 2* Datacomm Activity (BASIC/UX)

Bit 0 set: TRANSFER in progress.

Bit 1 set: Firmware interrupts enabled (ENABLE INTR active for this select code).

Bit 2 set: Handshake in progress. Only during multi-line function calls.

Bit 3 set: Handshake ended with an escape.

Datacomm Activity (BASIC/WS)

0 = No activity pending on this select code.

Bit 0 set: ENTER in process.

Bit 1 set: OUTPUT in process.

Non-zero ONLY during multi-line function calls.)

*STATUS 3* Current Protocol Identification: 1 = Async, 2 = Data Link Protocol

*CONTROL 3* Protocol to be used after next card reset: CONTROL Sc,0;1  
1 = Async Protocol, 2 = Data Link Protocol. (Data Link Protocol only supported on BASIC/WS.) This register overrides default switch configuration.

*STATUS 4* Cause of ON INTR program branch (supported on BASIC/WS only).

Bit	Function: Async Protocol	Function: Data Link Protocol
0	Data and/or Control Block available	Data Block Available
1	Prompt received	Space available for a new transmission block
2	Framing and/or parity error	Receive or transmit error
3	Modem line change	Modem line change
4 <sup>1</sup>	No Activity timeout (forces a disconnect)	No Activity timeout (forces a disconnect)
5 <sup>1</sup>	Lost carrier or connection timeout (forces a disconnect)	Lost carrier or connection timeout (forces a disconnect)
6	End-of-line received	Not used
7	Break received	Not used

5

<sup>1</sup> Supported only on BASIC/WS.

Contents of the register are cleared when a STATUS statement is executed to it.

*STATUS 5* Inbound queue status.

Value	Interpretation
0	Queue is empty
1	Queue contains data but no control blocks
2 <sup>1</sup>	Queue contains one or more control blocks but no data
3 <sup>1</sup>	Queue contains both data and one or more control blocks

<sup>1</sup> Supported only on BASIC/WS.

*CONTROL 5* Terminate Transmission (supported on BASIC/WS only).

OUTPUT S,5;0 is equivalent to OUTPUT S;END

Data Link: Sends previous data as a single block with an ETX terminator, then idles the line with an EOT.

Async: Tells card to turn half-duplex line around. Does nothing when line is full-duplex. The next data OUTPUT automatically regains control of the line by raising the RTS (request-to-send) modem line.

### *STATUS 6*

Break status.

1 = BREAK transmission pending, 0 = no BREAK pending.

### *CONTROL 6*

Send Break; causes a Break to be sent.

Data Link Protocol (BASIC/WS): Send Reverse Interrupt (RVI) reply to inbound block, or CN character instead of data in next outbound block.

Async Protocol: Transmit Break. Length is defined by Control Register 39.

Note that the value sent to the register is arbitrary.

### *STATUS 7*

Modem receiver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Data Mode (Data Set Ready) line

Bit 1: Receive ready (Data Carrier Detect line)

Bit 2: Clear-to-send (CTS) line

Bit 3: Incoming call (Ring Indicator line)

Bit 4: Depends on cable option or adapter used

### *STATUS 8*

Returns modem driver line states.

*CONTROL 8*

Sets modem driver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Request-to-send (RS or RTS) line 1 = line set (active)

Bit 1: Data Terminal Ready (DTR) line 0 = line clear (inactive)

Bit 2: Data Rate Select (DRS) line

Bit 3: Driver 2: Depends on cable option or adapter used

Bit 4: Driver 3: Depends on cable option or adapter used

Bit 5: Driver 4: Depends on cable option or adapter used

Bit 6: Sets the SIO Transmitter clock, 1 = internal, 0 = external

Bit 7: Sets the SIO Receiver clock, 1 = internal, 0 = external

**Reset value=0** prior to connect. Post-connect value is handshake dependent. Note that RTS line cannot be altered (except by OUTPUT or OUTPUT ... END) for half-duplex modem connections.

*STATUS 9*

Returns control block TYPE if last ENTER terminated on a control block (supported on BASIC/WS only). See Status Register 10 for values.

*STATUS 10*

Returns control block MODE if last ENTER terminated on a control block (supported on BASIC/WS only).

### Async Protocol Control Blocks

Type	Mode	Interpretation
250	1	Break received (Channel A)
251	1 <sup>1</sup>	Framing error in the following character
251	2 <sup>1</sup>	Parity error in the following character
251	3 <sup>1</sup>	Parity and framing errors in the following character
252	1	End-of-line terminator detected
253	1	Prompt received from remote
0	0	No Control Block encountered

<sup>1</sup> Parity/framing error control blocks are not generated when characters with parity and/or framing errors are replaced by an underscore (-) character.

5

### Data Link Protocol Control Blocks

Type	Mode	Interpretation
254	1	Preceding block terminated by ETB character
254	2	Preceding block terminated by ETX character
253 <sup>1</sup>	—	(See following table for Mode interpretation)
0	0	No Control Block encountered

<sup>1</sup> This type is used primarily in specialized applications.

### Data Link Protocol Control Blocks

Mode Bit(s)	Interpretation
0	1 = Transparent data in following block 0 = Normal data in following block
2,1	00 = Device select 01 = Group select 10 = Line select
3	1 = Command channel 2 = Data channel

*STATUS 11* Returns available outbound queue space (in bytes), provided there is sufficient space for at least three control blocks. If not, value is zero.

*STATUS 12* Datacomm Line connection status (supported for BASIC/WS only).

Value	Interpretation
0	Disconnected
1	Attempting Connection
2	Dialing
3	Connected <sup>1</sup>
4	Suspended
5	Currently receiving data (Data Link only)
6	Currently transmitting data (Data Link only)

<sup>1</sup> When using Data Link: Connected - datacomm idle

*CONTROL 12* Connects, initiates auto-dial sequence, and disconnects interface from datacomm line (supported on BASIC/WS only).



Value	Interpretation
0	Disconnected
1	Connected to datacomm line (set DTR & RTS)
2	Start auto dial. (Followed by OUTPUT of telephone numbers)

*STATUS 13* Returns current ON INTR mask.

*CONTROL 13* Sets ON INTR mask.<sup>1</sup>

#### Data Link Protocol (BASIC/WS only)

Bit	Value	Enables interrupt when:
0	1	A full block is available in receive queue
1	2	Transmit queue is empty
2	4	Receive or transmit error detected
3	8	A modem line changed
4	16 <sup>2</sup>	No Activity timeout forced a disconnection
5	32 <sup>2</sup>	Lost Carrier or Connection timeout caused a disconnection

## Async Protocol

Bit	Value	Enables interrupt when:
0 <sup>3</sup>	1	Data or control block available in receive queue
1	2	Prompt received from remote device
2	4	Framing or parity error detected in incoming data
3	8	A modem line changed
4 <sup>3</sup>	16 <sup>2</sup>	No Activity timeout forced a disconnection
5 <sup>3</sup>	32 <sup>2</sup>	Lost Carrier or Connection timeout caused a disconnection
6	64	End-of-line received
7	128	Break received

5

<sup>1</sup> If a CONTROL statement is used to access this register, the control block is placed in the outbound queue. If the ENABLE INTR ... statement is used with a mask, the mask value is placed directly in the control register, bypassing any queue delays.

<sup>2</sup> If bits 4 and 5 are not set, the corresponding errors can be trapped by using an ON ERROR statement.

<sup>3</sup> Supported only on BASIC/WS.

Reset value = 0

*STATUS 14*

Returns current Control Block mask (supported on BASIC/WS only).

*CONTROL 14*

Sets Control Block mask (supported on BASIC/WS only). Control block information is queued sequentially with incoming data as follows:

Bit	Value	Async Control Block Passed	Data Link Control Block Passed
0	1	Prompt position	Transparent/Normal Mode <sup>1</sup>
1	2	End-of-line position	ETX Block Terminator <sup>2</sup>
2	4	Framing and/or Parity error <sup>3</sup>	ETB Block Terminator <sup>2</sup>
3	8	Break received	

<sup>1</sup> Transparent/Normal format identification control block occurs at the *beginning* of a given block of data in the receive queue.

<sup>2</sup> ETX and ETB Block Termination identification control blocks occur at the END of a given block of data in the receive queue.

<sup>3</sup> This control block precedes each character containing a parity or framing error.

Reset Value: 0 (Control Blocks disabled) 6 (ETX/ETB Enabled)

Bits 4, 5, 6, and 7 are not used.

*STATUS 15*

Returns current modem line interrupt mask

*CONTROL 15*

Sets modem line interrupt mask. Enable an interrupt to ON INTR when Bit 3 of Control Register 13 is set as follows:

### Data Link Protocol

Bit	Value	Modem Line to Cause Interrupt
0	1	Data Mode (Data Set Ready)
1	2	Receive Ready (Data Carrier Detect)
2	4	Clear-to-send
3	8	OCR1, Incoming Call (Ring Indicator)
4	16	OCR2, Cable or adapter dependent

Reset Value = 0

*STATUS 16* Returns current connection timeout limit (supported on BASIC/WS only).

*CONTROL 16* Sets Attempted Connection timeout limit (supported on BASIC/WS only). Acceptable values: 1 thru 255 seconds. 0=timeout disabled. *Reset Value=25 seconds*

*STATUS 17* Returns current No Activity timeout limit (supported on BASIC/WS only).

*CONTROL 17* Sets No Activity timeout limit (supported on BASIC/WS only). Acceptable values: 1 thru 255 minutes. 0=timeout disabled. *Reset Value=10 minutes* (disabled if Async, non-modem handshake).

*STATUS 18* Returns current Lost Carrier timeout limit (supported on BASIC/WS only).

*CONTROL 18* Sets Lost Carrier timeout limit in units of 10 ms (supported on BASIC/WS only). Acceptable values: 1 thru 255. 0=timeout disabled. *Reset Value=40* (400 milliseconds)

*STATUS 19* Returns current Transmit timeout limit (supported on BASIC/WS only).

*CONTROL 19* Sets Transmit timeout limit (supported on BASIC/WS only). Note that loss of clock or CTS not returned by

modem when transmission is attempted. Acceptable values: 1 thru 255.0=timeout disabled. *Reset Value=10 seconds*

**STATUS 20**

Returns current transmission/receive speed (baud rate). See table for values.

**CONTROL 20**

Sets transmission speed (baud rate) as follows:

Register Value	Baud Rate	Register Value	Baud Rate
0	External Clock	8	600
1*	50	9	1200
2*	75	10	1800
3*	110	11	2400
4*	134	12	3600
5*	150	13	4800
6*	200	14	9600
7	300	15	19200

\* Async only for BASIC/WS. These values cannot be used with Data Link. These values set transmit speed *only* for Async; transmit *and* receive speed for Data Link. Default value is defined by the interface card configuration switches.

**STATUS 21**

For BASIC/WS, protocol dependent. Returns receive speed (Async) or GID address (Data Link) as specified by Control Register 21.

For BASIC/UX, same as Register 20.

*CONTROL 21*

For BASIC/WS, protocol dependent. Functions are as follows:

**Data Link:** Sets Group Identifier (GID) for terminal. Values 0 through 26 correspond to identifiers @, A, B, . . . , Y, Z, respectively. Other values cause an error. Default value is 1 ("A").

**Async:** Sets datacomm receiver speed (baud rate). Values and defaults are the same as for Control Register 20.

For BASIC/UX, same as Register 20.

*STATUS 22*

Protocol dependent. Returns DID (Data Link) or protocol handshake type (Async) as specified by CONTROL Register 22.

*CONTROL 22*

Protocol dependent. Functions are as follows:

**Data Link:** For BASIC/WS only. Sets Device Identifier (DID) for terminal. Values are the same as for Control Register 21. Default is determined by interface card configuration switches.

**Async:** Defines the asynchronous protocol handshake type that is to be used.

Value	Handshake type
0	Protocol handshake disabled
1 <sup>1</sup>	ENQ/ACK with desktop computer as the host (ignored)
2 <sup>1</sup>	ENQ/ACK with desktop computer as a terminal (ignored)
3	DC1/DC3, desktop computer as host
4	DC1/DC3, desktop computer as a terminal
5	DC1/DC3, desktop computer as both host and terminal

<sup>1</sup> Supported only on BASIC/WS.

*STATUS 23* Returns current hardware handshake type.

*CONTROL 23* Sets hardware handshake.

0=Handshake OFF, non-modem connection.

1=FULL-DUPLEX modem connection.

2=HALF-DUPLEX modem connection (supported only on BASIC/WS).

3=Handshake ON, non-modem connection (supported only on BASIC/WS).

Reset Value is determined by interface configuration switches.

*STATUS 24* Protocol dependent (supported on BASIC/WS only). Returns value set by preceding CONTROL statement to Control Register 24.

*CONTROL 24* Protocol dependent (supported on BASIC/WS only). Functions as follows:

Data Link protocol: Set outbound block size limit.

Value	Block size	Value	Block size
0	512 bytes	4	8 bytes
1	2 bytes	-	-
2	4 bytes	-	-
3	6 bytes	255	510 bytes

Reset outbound block size limit=512 bytes

Async Protocol: Set mask for control characters included in receive data message queue.

Bit set: transfer character(s).

Bit cleared: delete character(s)

5

Bit set	Value	Character(s) passed to receive queue
0	1	Handshake characters (ENQ, ACK, DCI, DC3)
1	2	Inbound End-of-line character(s)
2	4	Inbound Prompt character(s)
3	8	NUL (CHR\$(0))
4	16	DEL (CHR\$(127))
5	32	CHR\$(255)
6	64	Change parity/framing errors to underscores (.) if bit is set.
7	128	Not used

Reset value=127 (bits 0 through 6 set)

*STATUS 25*

Returns number of received errors since power up or reset (supported on BASIC/WS only).



---

**Note**

Control Registers 26 through 35, Status Registers 27 through 35, and Control and Status Registers 37 and 39 are used for ASYNC protocol *only*. They are not available during Data Link operation.

---

**STATUS 26**

Protocol dependent.

Data Link protocol: Returns number of transmit errors (NAKs received) since last interface reset (supported only on BASIC/WS).

Async protocol: Returns first protocol handshake character (ACK or DC1).

**CONTROL 26**

Sets first protocol handshake character as follows:

6=ACK, 17=DC1. (BASIC/UX supports only 17=DC1.) Other values used for special applications only. Reset value=17 (DC1). Use ACK when Control Register 22 is set to 1 or 2. Use DC1 when Control Register 22 is set to 3, 4, or 5.

**STATUS 27**

Returns second protocol handshake character.

**CONTROL 27**

Sets second protocol handshake character as follows:

5=ENQ, 19=DC2. (BASIC/UX only supports 19=DC2.) Other values used for special applications only. Reset value=19 (DC3). Use ENQ when Control Register 22 is set to 1 or 2. Use DC3 when Control Register 22 is set to 3, 4, or 5.

**STATUS 28**

Returns number of characters in inbound End-of-line delimiter sequence.

**CONTROL 28**

Sets number of characters in End-of-line delimiter sequence. Acceptable values are 0 (no EOL delimiter), 1, or 2. Reset value=2

**STATUS 29**

Returns first End-of-line character. (Async only)

- CONTROL 29** Sets first End-of-line character. Reset value=13 (carriage return) (Async only)
- STATUS 30** Returns second End-of-line character. (Async only)
- CONTROL 30** Sets second End-of-line character. Reset value=10 (line feed) (Async only)
- STATUS 31** Returns number of characters in Prompt sequence. (Async only)
- CONTROL 31** Sets number of characters in Prompt sequence. Acceptable values are 0 (Prompt disabled), 1, or 2. Reset value=1 (Async only)
- STATUS 32** Returns first character in Prompt sequence. (Async only)
- CONTROL 32** Sets first character in Prompt sequence. Reset value=17 (DC1) (Async only)
- STATUS 33** Returns second character in Prompt sequence. (Async only)
- CONTROL 33** Sets second character in Prompt sequence. Reset value=0 (null) (Async only)
- STATUS 34** Returns the number of bits per character. (Async only)
- CONTROL 34** Sets the number of bits per character as follows: (Async only)
- |                    |                    |
|--------------------|--------------------|
| 0=5 bits/character | 2=7 bits/character |
| 1=6 bits/character | 3=8 bits/character |
- Reset Value* is determined by the Datacomm Configuration for HP-UX.
- STATUS 35** Returns the number of stop bits per character. (Async only)

*CONTROL 35*

Sets the number of stop bits per character as follows:  
(Async only)

For BASIC/UX:

0=1 stop bit 2=2 stop bits Reset Value is determined by the Datacomm configuration for HP-UX.

For BASIC/WS:

0=1 stop bit 1=1.5 stop bits 2=2 stop bits Reset Value: 2 stop bits if 150 baud or less, otherwise 1 stop bit. Reset Value is determined by interface configuration switch settings.

*STATUS 36*

Returns current Parity setting.

*CONTROL 36*

For BASIC/UX:

Sets the parity for transmitting and receiving asynchronous protocol.

0=NONE; No parity bit is included with any characters.

1=ODD; Parity bit SET if there is an EVEN number of "1"s in the character body.

2=EVEN; Parity bit OFF if there is an ODD number of "1"s in the character body.

Reset Value is determined by the Datacomm configuration for HP-UX.

For BASIC/WS:

Sets Parity for transmitting and receiving as follows:

Data Link Protocol: 0=NO Parity; Network host is HP 1000 Computer.

1=ODD Parity; Network host is HP 3000 Computer.

Reset value=0

Async Protocol: 0=NONE; no parity bit is included with any characters.

1=ODD; Parity bit SET if there is an EVEN number of "1"s in the character body.

2=EVEN; Parity bit OFF if there is an ODD number of "1"s in the character body.

3="0"; Parity bit is always ZERO, but parity is not checked (supported only on BASIC/WS).

4="1"; Parity bit is always SET, but parity is not checked (supported only on BASIC/WS).

Default is determined by interface configuration switches. If 8 bits per character, parity must be NONE, ODD, or EVEN.

*STATUS 37* Returns inter-character time gap in character times.

*CONTROL 37* Sets inter-character time gap in character times.

Acceptable values: 1—255 character times.

0=No gap between characters. Reset value=0

*STATUS 38* Returns Transmit queue status.

If returned value=1, queue is empty, and there are no pending transmissions.

*STATUS 39*

Returns current Break time.

*CONTROL 39*

Sets Break time in character times. (Async only; supported on BASIC/WS only)

Acceptable values are: 2—255. Reset value=4



## The GPIO Interface

---

### Introduction

This chapter should be used in conjunction with the *HP 98622A GPIO Interface Installation* manual. *The best way to use these two documents is to read this chapter before attempting to configure and connect the interface* according to the directions given in the installation manual. The reason for this order of use is that knowing how the interface works and how it is driven by BASIC programs will help you to decide how to connect it to your peripheral device.

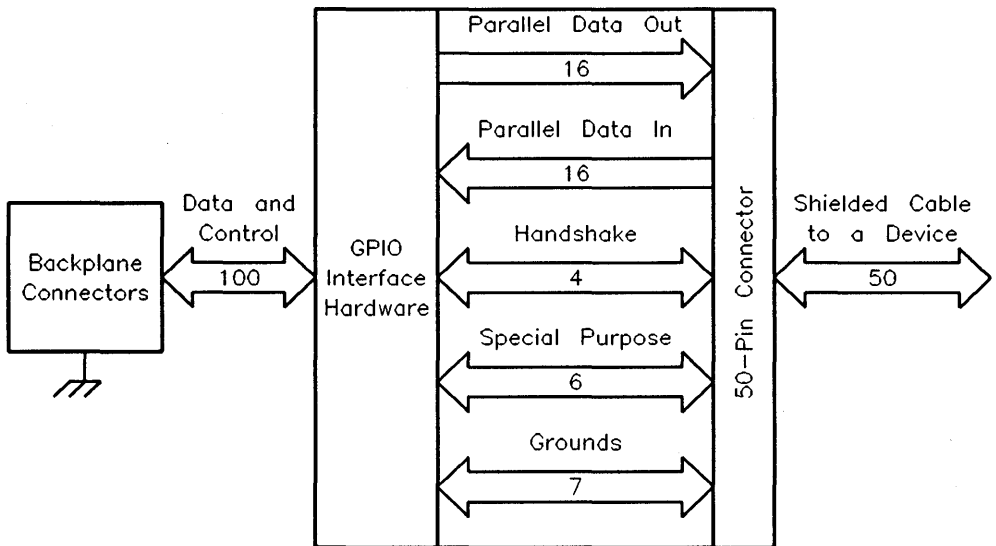
The HP 98622 Interface is a very flexible parallel interface that allows you to communicate with a variety of devices. The interface sends and receives up to 16 bits of data with a choice of several handshake methods. External interrupt and user-definable signal lines are provided for additional flexibility. The interface is known as the General-Purpose Input/Output (GPIO) Interface for these reasons. This chapter describes the use of the interface's features from BASIC programs.

Use of some statements or suggestions for interfacing requires that you load the TRANS BIN file.

## Interface Description

The main function of any interface obviously to transfer data between the computer and a peripheral device. This section briefly describes the interface lines and how they function. Using the lines from BASIC programs is more fully described in subsequent sections.

The GPIO Interface provides 32 lines for data input and output: 16 for input (DI0—DI15), and 16 for output (DO0—DO15).



**Block Diagram of the GPIO Interface**

Two lines are dedicated to *handshaking* the data from source to destination device. The Peripheral Control line (PCTL) is controlled by the interface and is used to initiate data transfers. The Peripheral Flag line (PFLG) is controlled by the peripheral device and is used to signal the peripheral's readiness to continue the transfer process.

One line is used to signal External Interrupt Requests to the computer (EIR). The interface must be enabled to initiate interrupt branches for the interface to detect this request. The state of the line can also be read by the program.



Four general-purpose lines are available for any purpose that you may desire; two are controlled by the computer and sensed by the peripheral (CTL0 and CTL1), and two are controlled by the peripheral device and sensed by the computer (STI0 and STI1).

Both Logic Ground and Safety Ground are provided by the interface. Logic Ground provides the reference point for signals, and Safety Ground provides earth ground for cable shields.

---

## Interface Configuration

This section presents a brief summary of selecting the interface's configuration-switch settings. It is intended to be used as a checklist and to begin to acquaint you with programming the interface. *Refer to the installation manual for the exact location and setting of each switch.*

The following sample program checks a few of these switch settings on a GPIO Interface already installed in the computer and displays the settings. However, many of the settings cannot be determined from BASIC programs. If any of the displayed settings are different than desired, or if any settings are not already known, refer to the installation manual for switch locations and settings.

```

100  PRINTER IS 1  ! Select printer device.
110  PRINT CHR$(12) ! Clear screen.
120  !
130  DISP "Enter GPIO Interface Select Code (CONT=12)"
140  OUTPUT 2 USING "#,DD";12
150  ENTER 2;Isc
160  DISP
170  !
180  ASSIGN @Gpio TO Isc ! FORMAT ON default.
190  !
200  ! Read STATUS registers 0 and 1.
210  STATUS Isc;Card_id,Intr_stat
220  !
230  ! Is this card a GPIO?
240  IF Card_id<>3 THEN
250      PRINT "The interface at select code";Isc
260      PRINT "is not a GPIO Interface."
270      PRINT "Program stopped."
280      STOP
290  ELSE
300      PRINT "The card ID of the GPIO at"
310      PRINT "interface select code";Isc
320      PRINT "is";Card_id
330  END IF
340  PRINT
350  !
360  ! Calculate hardware interrupt priority.
370  Bits_5_and_4=BINAND(Intr_stat,32+16)
380  Switches=Bits_5_and_4 DIV 16
390  Hd_prior=Switches+3
400  PRINT "Hardware Interrupt Priority is";Hd_prior
410  PRINT
420  !
430  END

```

6

## Interface Select Code

In BASIC, allowable interface select codes range from 8 through 31; codes 1 through 7 are already used for built-in interfaces. The GPIO interface has a factory default setting of 12, which can be changed by re-configuring the “SEL CODE” switches on the interface.

## Hardware Interrupt Priority

Two switches are provided on the interface to allow selection of hardware interrupt priority. The switches allow hardware priority levels 3 through 6 to be selected. **Hardware priority** determines the order in which simultaneously occurring interrupt events are *logged*, while **software priority** determines the order in which interrupt events are *serviced* by the BASIC program.

## Data Logic Sense

The data lines of the interface are *normally low-true*; in other words, when the voltage of a data line is low, the corresponding data bit is interpreted to be a 1. This logic sense may be changed to high-true with the Option Select Switch. Setting the switch labeled “DIN” to the “0” position selects high-true logic sense of Data In lines. Conversely, setting the switch labeled “DOUT” to the “1” position inverts the logic sense of the Data Out lines. The default setting is “1” for both.

## Data Handshake Methods

This section describes the data handshake methods available with the GPIO Interface. A general description of the handshake modes and clock sources is given first. A more detailed discussion of each handshake is then given to allow you to choose the handshake mode, clock source, and handshake-line logic sense that is compatible with your peripheral device.

As a brief review, a data handshake is a method of synchronizing the transfer of data from the sending to the receiving device. In order to use any handshake method, *the computer and peripheral device must be in agreement as to how and when several events will occur*. With the GPIO Interface, the following events must take place to synchronize data transfers; the first two are optional.

- The computer may optionally be directed to perform a one-time “OK check” of the peripheral before beginning to transfer any data.
- The computer may also optionally check the peripheral to determine whether or not the peripheral is “ready” to transfer data.
- The computer must indicate the direction of transfer and then initiate the transfer.
- During OUTPUT operations, the peripheral must read the data sent from the computer while valid; similarly, the computer must clock the peripheral’s data into the interface’s Data In registers while valid during ENTER operations.
- The peripheral must acknowledge that it has received the data.

### Handshake Lines

6

*The GPIO handshakes data with three signal lines.* The Input/Output line, I/O, is driven by the computer and is used to signal the direction of data transfer. The Peripheral Control line, PCTL, is also driven by the computer and is used to initiate all data transfers. The Peripheral Flag line, PFLG, is driven by the peripheral and is used to acknowledge the computer’s requests to transfer data.

### Handshake Logic Sense

Logic senses of the PCTL and PFLG lines are selected with switches of the same name. The logic sense of the I/O line is High for ENTER operations and Low for OUTPUT operations; this logic sense cannot be changed. The available choices of handshake logic sense and handshake modes allow nearly all types of peripheral handshakes to be accommodated by the GPIO Interface.

## Handshake Modes

There are two general handshake modes in which the PCTL and PFLG lines may be used to synchronize data transfers: Full-Mode and Pulse-Mode Handshakes. If the peripheral uses pulses to handshake data transfers *and* meets certain hardware timing requirements, the Pulse-Mode Handshake may be used. The Full-Mode Handshake should be used if the peripheral does not meet the Pulse-Mode timing requirements.

The handshake mode is selected by the position of the “HSHK” switch on the interface, as described in the installation manual. Both modes are more fully described in subsequent sections.

## Data-In Clock Source

Ensuring that the data are valid when read by the receiving device is slightly different for OUTPUT and ENTER operations. During OUTPUTs, the interface generally holds data valid while PCTL is in the Set state, so the peripheral must read the data during this period. During ENTERs, the data must be held valid by the peripheral until the peripheral signals that the data are valid (which clocks the data into interface Data In registers) or until the data is read by the computer. The point at which the data are valid is signalled by a transition of PFLG. The PFLG transition that is used to signal valid data is selected by the “CLK” switches on the interface. Subsequent diagrams and text further explain the choices.

## Optional Peripheral Status Check

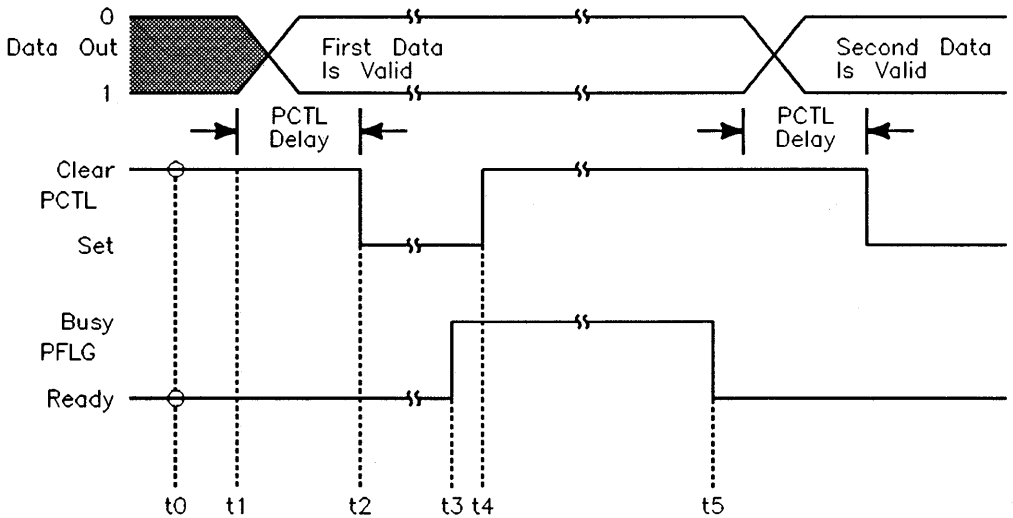
Many peripheral devices are equipped with a line which is used to indicate the device’s current “OK-or-Not-OK” status. If this line is connected to the Peripheral Status line (PSTS) of the GPIO Interface, and the computer may determine the status of the peripheral device by checking the state of PSTS. The logic sense of this line may be selected by setting the “PSTS” switch.

*If enabled*, the computer performs a *one-time check* of the Peripheral Status line (PSTS) *before initiating any transfers* as part of the data-transfer handshake. If PSTS indicates “Not OK,” error 172 is reported; otherwise, the transfer proceeds normally. If this feature is not enabled, this one-time check is never made. This feature is available with both Full-Mode and Pulse-Mode Handshakes. See “Using the PSTS Line” for further details.

## Full-Mode Handshakes

The Full-Mode Handshake mode is described first for two reasons. The first reason is that the PCTL and PFLG transitions must always occur in the order shown, so only one sequence of peripheral handshake responses needs to be shown. Secondly, this mode will generally work when the Pulse-Mode Handshake may not be compatible with the peripheral's handshake signals. The Pulse-Mode Handshake is described in the next section.

The following diagrams show the order of events of the Full-Mode OUTPUT and ENTER Handshakes. These drawings are not drawn to any time scale; only the order of events is important. The I/O line has been omitted to simplify the diagrams; in all cases, it is driven Low before any OUTPUT is initiated and High before any ENTER is initiated.



**Full-Mode OUTPUT Handshakes**

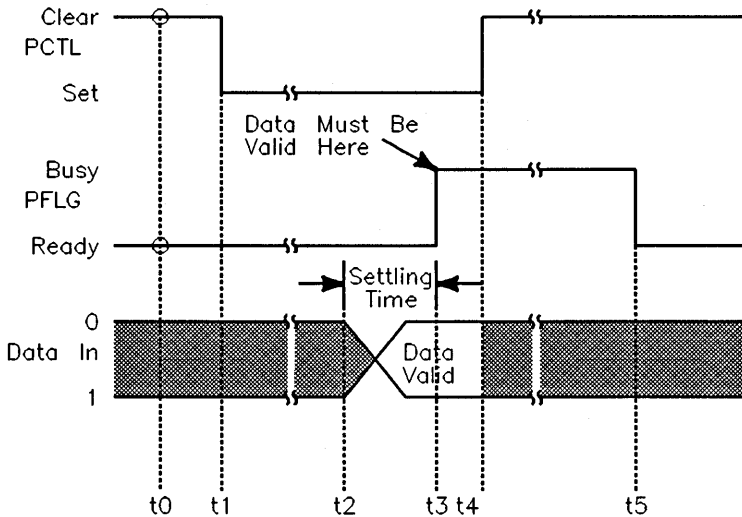
With Full-Mode Handshakes, the computer first checks to see that the peripheral device is Ready before initiating the transfer of each byte/word ( $t_0$ ); with this handshake mode, the peripheral indicates *Ready when both PCTL is Clear and PFLG is Ready*. If the peripheral does not indicate Ready, the computer waits until a Ready is indicated.

When a Ready is sensed, the computer places data on the Data Out lines ( $t_1$ ) and drives the I/O line Low (not shown). The interface then waits the PCTL Delay time before initiating the transfer by placing PCTL in the Set state ( $t_2$ ).

The peripheral acknowledges the computer's request by placing the PFLG line Busy ( $t_3$ ); this PFLG transition automatically Clears the PCTL line ( $t_4$ ). However, the computer cannot initiate further transfers until the peripheral is Ready with Full-Mode Handshake; the peripheral is not Ready until both PCTL is Clear and PFLG is Ready ( $t_5$ ).

The data on the Data Out lines is held valid from the time PCTL is Set until after the peripheral indicates Ready. The peripheral may read the data any time within this time period.

The PCTL and PFLG lines are used in the same manner in Full-Mode ENTER Handshakes as in Full-Mode OUTPUT Handshakes. However, there are three options available as to when the peripheral's data may be valid: at the Ready-to-Busy transition of PFLG (BSY clock source), at the Busy-to-Ready transition of PFLG (RDY clock source), and when the Data In lines are read with a STATUS statement (READ clock source). The first two of these options are shown in the following two diagrams; the READ clock source is discussed later in "Designing Your Own Transfers".



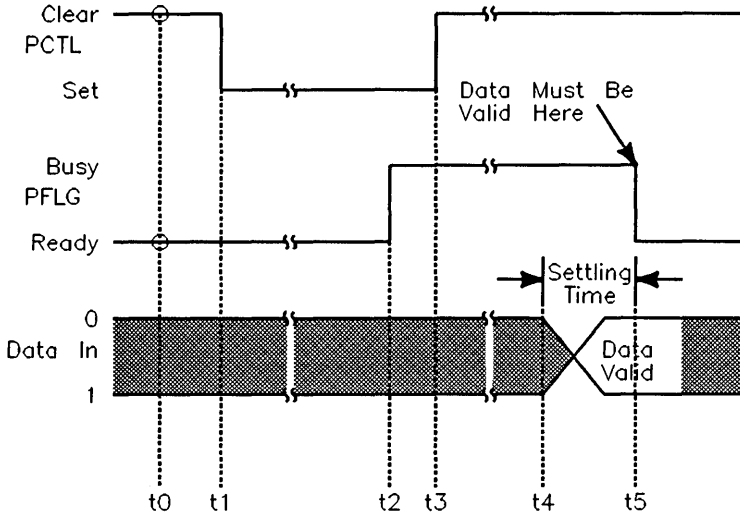
**Full-Mode ENTER Handshake with BSY Clock Source**

As with Full-Mode OUTPUT Handshakes, the computer first checks to see if the peripheral is Ready ( $t_0$ ); since PCTL is Clear and PFLG is Ready, the handshake may proceed. The computer places the I/O line in the High state (not shown) and then initiates the handshake by placing PCTL in the Set state ( $t_1$ ).

With the "BSY" clock source, the PFLG transition to the Busy state clocks the peripheral's data into the interface's Data-In registers; consequently, the peripheral must place data on the Data-In lines ( $t_2$ ), allowing enough time for the data to settle before placing PFLG in the Busy state ( $t_3$ ). This PFLG transition to the Busy state automatically Clears PCTL ( $t_4$ ). The next



handshake may be initiated when PFLG is placed in the Ready state by the peripheral (t5).



### Full-Mode ENTER Handshake with RDY Clock Source

As with other Full-Mode Handshakes, the computer first checks to see if the peripheral is ready (t0). Since PCTL is Clear and PFLG is Ready, the computer may drive the I/O line High (not shown) and initiate the handshake by placing PCTL in the Set state (t1).

The peripheral may acknowledge by placing PFLG Busy (t2), which automatically Clears PCTL (t3). Unlike the previous example, this transition does not clock data into the interface Data-In registers. With the "RDY" clock source, the peripheral must place the data on the Data-In lines (t4), allowing enough time for the data to settle before placing PFLG in the Ready state (t5). The computer may then initiate a subsequent transfer.

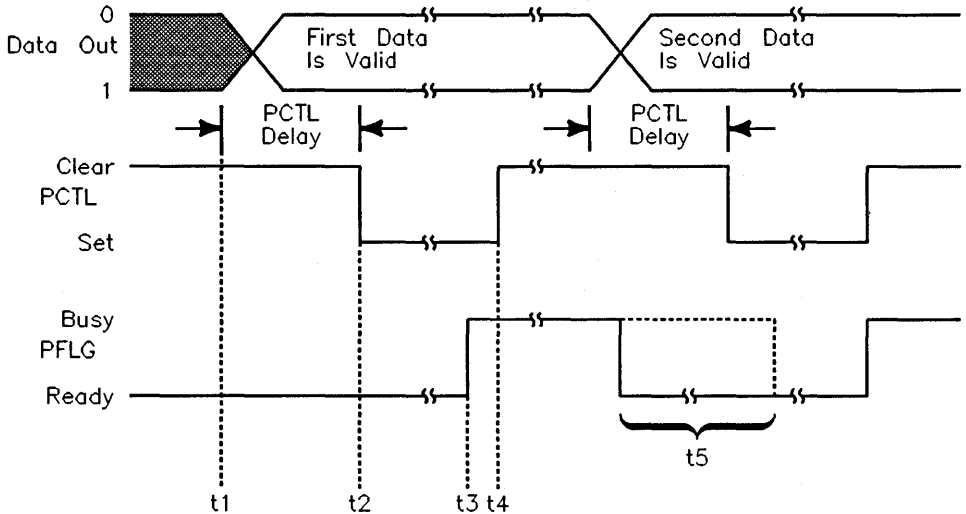
### Pulse-Mode Handshakes

The following drawings show the order of handshake-line events during Pulse-Mode Handshakes. Notice that the *main difference* between Full-Mode and Pulse-Mode Handshakes is that the *PFLG is not checked for Ready before the computer initiates Pulse-Mode Handshakes*; the computer may

initiate a subsequent data transfer as soon as the PCTL line is Cleared by the Ready-to-Busy transition of PFLG.

Two cycles of data transfers are shown in these diagrams to illustrate that the computer need not wait for the PFLG=Ready indication with the Pulse-Mode Handshake. The first cycle shown in each diagram is a typical example of the first transfer of an I/O statement. The dashed PFLG line at the beginning of the second cycle shows that computer disregards whether or not PFLG is in the Ready state before the next transfer is initiated.

This absence of the PFLG check allows a *potentially higher data-transfer rate* than possible with the Full-Mode Handshake; however, in some cases, it also places additional timing restrictions on the peripheral's response time, as described in the text.



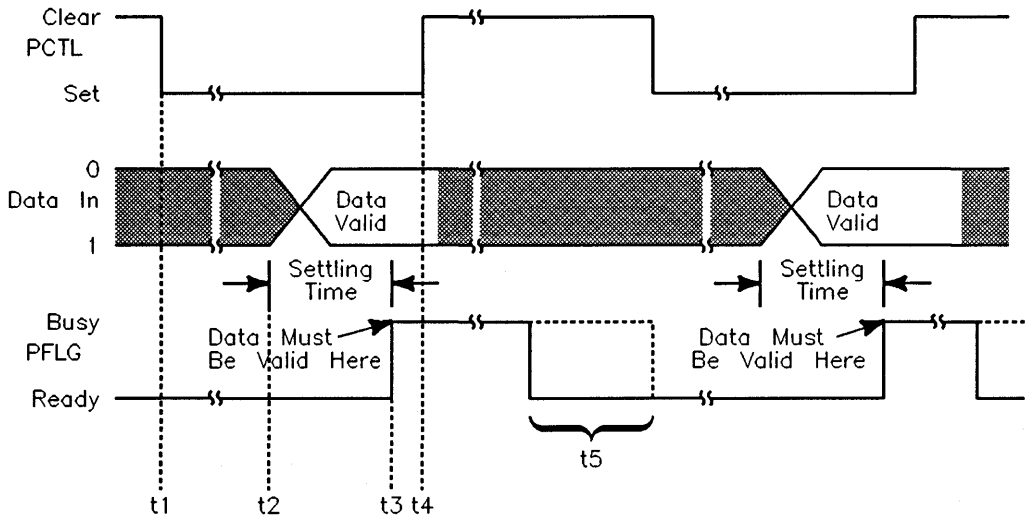
**Busy Pulses with Pulse-Mode OUTPUT Handshake**

The PFLG line is not checked for Ready before the computer drives the I/O line Low (not shown) and places data on the Data-Out lines ( $t_1$ ). A PCTL Delay time later, the interface initiates the transfer by placing PCTL in the Set state ( $t_2$ ).

The peripheral acknowledges by placing PFLG Busy ( $t_3$ ); this transition automatically Clears PCTL ( $t_4$ ). The dashed PFLG line shows that the

computer may initiate another transfer any time after PCTL is Clear, possibly before the peripheral places PFLG in the Ready state (t5).

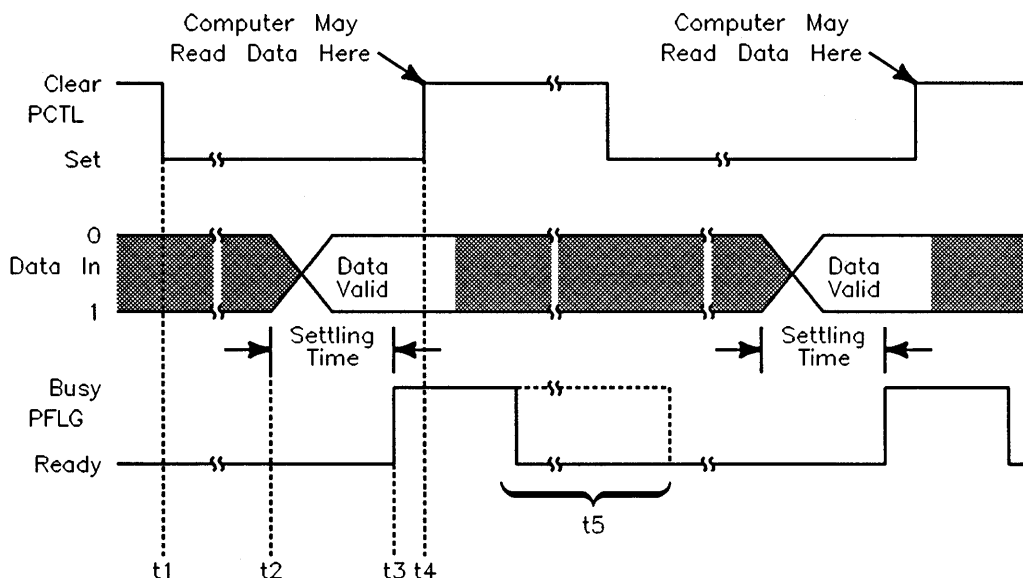
The Busy Pulse shown in the diagram is identical to the PFLG's response during the previous Full-Mode handshake; however, the Pulse-Mode Handshake works properly with this type of pulse *only* if the peripheral reads the data by the time PCTL is Clear (data should be read between t2 and t3). If the peripheral has not read the data by the time that PCTL is Clear, it might erroneously read the data for the second transfer, since the computer might have already changed the data and initiated the second transfer.



**Busy Pulses with Pulse-Mode ENTER Handshakes (BSY Clock Source)**

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state ( $t_1$ ).

The peripheral must place data on the Data In lines ( $t_2$ ), allowing enough time for the data to settle before placing PFLG in the Busy state ( $t_3$ ). This Ready-to-Busy transition of PFLG automatically Clears PCTL. The dashed PFLG signal shows that the next transfer may be initiated before PFLG indicates Ready.



### Busy Pulses with Pulse-Mode ENTER Handshakes (RDY Clock Source)

6

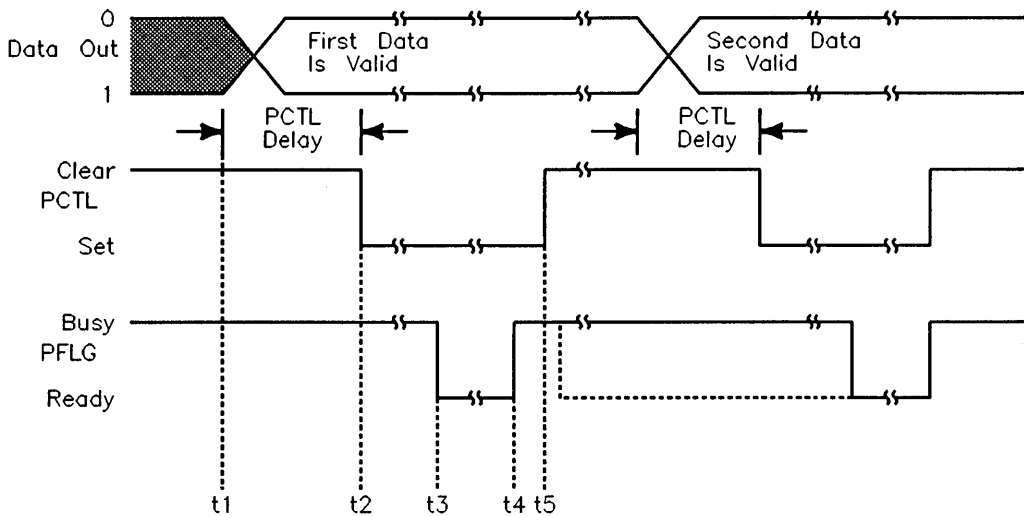
The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state ( $t_1$ ).

The peripheral must place data on the Data In lines ( $t_2$ ), allowing enough time for the data to settle before placing PFLG Busy ( $t_3$ ). This requirement *may seem contradictory*, since the clock source is the Busy-to-Ready transition of PFLG. However, with Pulse-Mode handshakes, the peripheral is assumed to be Ready whenever PCTL is Clear; consequently, the computer may read the data any time after PCTL is cleared by the Ready-to-Busy transition of PFLG. The PFLG transition to Busy Clears PCTL ( $t_4$ ), after which the peripheral may place PFLG Ready ( $t_5$ ).

#### Note



In order to use this type of pulse with the Pulse-Mode Handshake and RDY clock source, the peripheral must adhere to the stated timing restrictions.

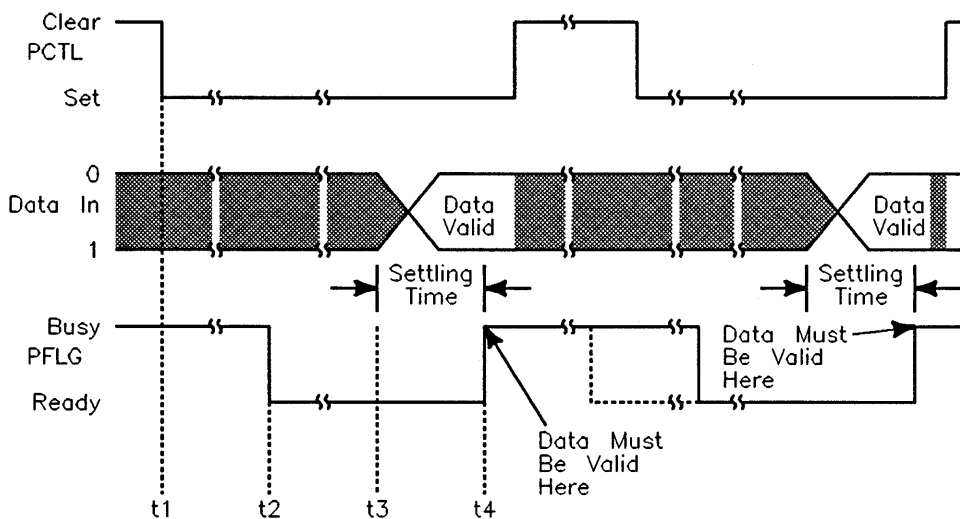


### Ready Pulses with Pulse-Mode OUTPUT Handshakes

The PFLG line is not checked for Ready before the computer drives the I/O line Low (not shown) and places data on the Data Out lines ( $t_1$ ). At a PCTL Delay time later, the interface initiates the transfer by placing PCTL in the Set state ( $t_2$ ).

The peripheral later acknowledges by placing PFLG in the Ready state ( $t_3$ ). The handshake is completed by the peripheral placing PFLG in the Busy state ( $t_4$ ), which automatically Clears PCTL ( $t_5$ ).

If the peripheral uses the type of Ready pulses shown, either the Pulse-Mode handshake with default PFLG logic sense or Full-Mode handshake with inverted PFLG logic sense may be used. With this type of pulse, the data being output may be read by the peripheral as long as PCTL is Set.

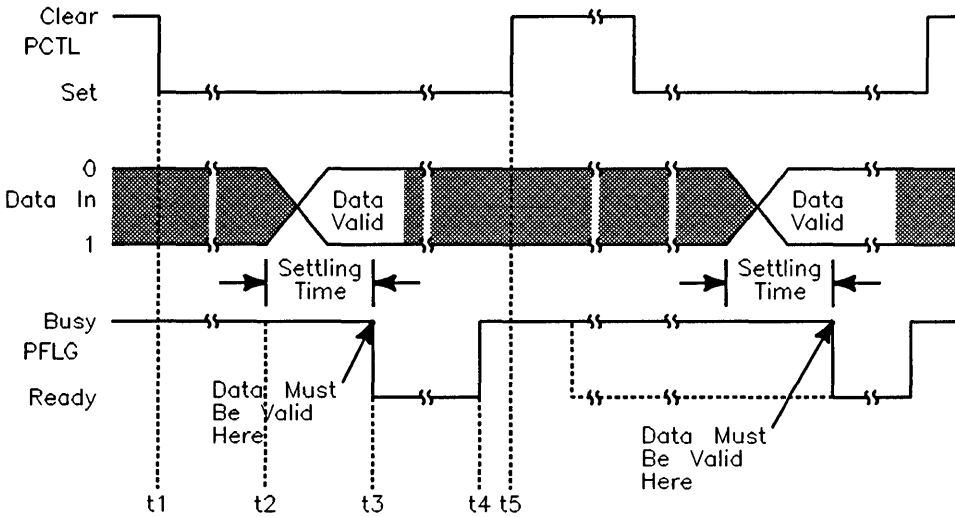


### Ready Pulses with Pulse-Mode ENTER Handshakes (BSY Clock Source)

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state ( $t_1$ ).

The peripheral acknowledges by placing PFLG in the Ready state ( $t_2$ ). The peripheral must place data on the Data In lines ( $t_3$ ), allowing enough time for the data to settle before placing PFLG in the Busy state ( $t_4$ ). With this type of pulse, events  $t_2$  and  $t_3$  may also occur in the reverse order.

The Ready-to-Busy transition of PFLG automatically Clears PCTL ( $t_4$ ). The dashed PFLG signal shows that the state of PFLG is not checked before the computer initiates a subsequent transfer.



### Ready Pulses w/ Pulse-Mode ENTER Handshakes (RDY Clock Source)

The computer does not have to check for PFLG to be Ready before placing I/O in the High state (not shown) and initiating the transfer by placing PCTL in the Set state (t1).

The peripheral must place data on the Data In lines (t2), allowing enough time for the data to settle before placing PFLG Ready (t3). The peripheral places PFLG in the Busy state (t4), which automatically Clears PCTL (t5).

---

## Interface Reset

The interface should always be reset before use to ensure that it is in a known state. All interfaces are automatically reset by the computer at certain times: when the computer is powered on, when **Reset** is pressed (**Shift-Reset** on an ITF keyboard), and at other times described in the Reset Table. (See “Useful Tables” in the *HP BASIC 6.2 Language Reference*.) The interface may be optionally reset at other times under control of BASIC programs. Two examples are as follows:

```
Gpio=12
CONTROL Gpio,0;1

Reset=1
CONTROL Gpio;Reset
```

The following action is invoked whenever the GPIO Interface is reset:

- The Peripheral Reset line (PRESET) is pulsed Low for at least 15 microseconds.
- The PCTL line is placed in the Clear state.
- If the DOUT CLEAR jumper is installed, the Data Out lines are all cleared (set to logic 0).
- The interrupt enable bit is cleared, disabling subsequent interrupts until re-enabled by the program.

The following lines are *unchanged* by a reset of the GPIO Interface:

- The CTL0 and CTL1 output lines.
- The I/O line.
- The Data Out lines, if the DOUT CLEAR jumper is not installed.



---

## Outputs and Enters through the GPIO

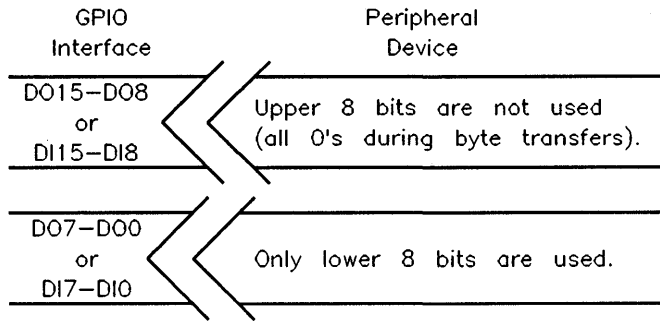
This section describes techniques for outputting and entering data through the GPIO Interface. The mechanism by which data are communicated are the electrical signals on the data lines. The actual signals that appear on the data lines depend on three things:

- the data currently being transferred,
- how this data is being represented,
- the logic sense of the data lines.

Brief explanations of ASCII and internal data representation are given in the “Interfacing Concepts” chapter. Complete details of the freefield convention and effects of IMAGE specifiers during OUTPUT and ENTER statements are described in the “Outputting Data” and “Entering Data” chapters, respectively. The section of the chapter “I/O Path Attributes” called “The FORMAT OFF Attribute” describes how internal-form data are represented during OUTPUT and ENTER. This section gives simple examples of how several representations are implemented during OUTPUTs and ENTERs through the GPIO Interface.

### ASCII and Internal Representations

When data are moved through the GPIO Interface, the *data are generally sent one byte at a time, with the most significant byte first*. This byte-mode transfer is independent of whether FORMAT ON or FORMAT OFF is the I/O path attribute. However, there are *two exceptions*; data are represented by words when the “W” image specifier is used and when numeric data are moved with reads of STATUS register 3 and writes to CONTROL register 3. The following diagrams illustrate which data lines are used during byte and word transfers.



### Byte Transfers

#### Example Statements that Output Data Bytes

The following diagrams show the actual logic signals that appear on the least significant data byte (DO7 thru DO0) as the result of the corresponding OUTPUT statement; the most significant byte is always zeros with byte transfers. The actual logic levels depend on how the data lines are configured (i.e., as Low-true or High-true).

```
ASSIGN @Gpio TO 12
OUTPUT @Gpio;"ASCII"
```

6

Signal Line		ASCII
DO7.....	DO0	Char.
0 1 0 0	0 0 0 1	A
0 1 0 1	0 0 1 1	S
0 1 0 0	0 0 1 1	C
0 1 0 0	1 0 0 1	I
0 1 0 0	1 0 0 1	I
0 0 0 0	1 1 0 1	C <sub>R</sub>
0 0 0 0	1 0 1 0	L <sub>F</sub>

```
Gpio=12
Number=-4
OUTPUT Gpio USING "MD.DD";Number
```

Signal Line		ASCII
DO7.....	DO0	Char.
0 0 1 0	1 1 0 1	-
0 1 1 0	0 1 0 0	4
0 0 1 0	1 1 1 0	.
0 0 1 1	0 0 0 0	0
0 0 1 1	0 0 0 0	0
0 0 0 0	1 1 0 1	C <sub>R</sub>
0 0 0 0	1 0 1 0	L <sub>F</sub>

```
ASSIGN @Gpio TO 12;FORMAT OFF
Integer_1=256*85+76
OUTPUT @Gpio;Integer_1
```

Signal Line		ASCII
DO7.....	DO0	Char.
0 1 0 1	0 1 0 1	U
0 1 0 0	1 1 0 0	L

```

ASSIGN @Gpio TO 12;FORMAT OFF
String$="1234"
OUTPUT @Gpio;String$

```

Signal Line		ASCII
DO7.....	DO0	Char.
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 1 0 0	E <sub>t</sub>
0 0 1 1	0 0 0 1	1
0 0 1 1	0 0 1 0	2
0 0 1 1	0 0 1 1	3
0 0 1 1	0 1 0 0	4

### Example Statements that Enter Data Bytes

6

The following diagrams show the variable values that result from the logic signals being present during the corresponding ENTER statement on the least significant data byte (DI7 thru DI0); the most significant byte is always ignored with byte transfers. The actual logic levels required depend on how the data lines are configured (i.e., as Low-true or High-true).

```

ENTER @Gpio USING "#,B";Byte
DISP "Value entered=";Byte

```

Value entered= 65

Signal Line		ASCII
DI7.....	DI0	Char.
0 1 0 0	0 0 0 1	A

```

ENTER 12;String$
DISP "String entered= ";String$

String entered= ruok?

```

Signal Line		ASCII
DI7.....	DI0	Char.
0 1 1 1	0 0 1 0	r
0 1 1 1	0 1 0 1	u
0 1 1 0	1 1 1 1	o
0 1 1 0	1 0 1 1	k
0 0 1 1	1 1 1 1	?
0 0 0 0	1 0 1 0	LF

```

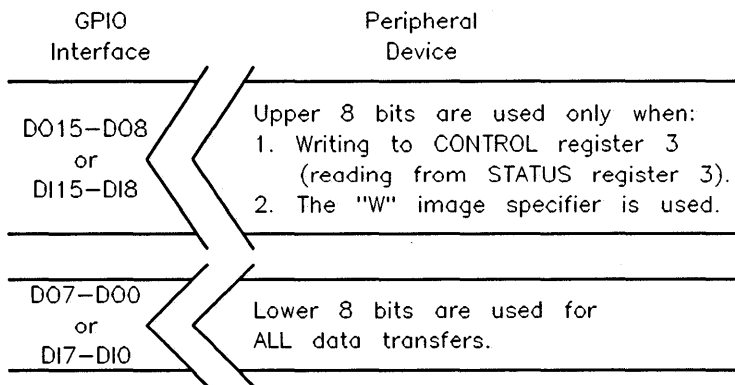
REAL Number
ASSIGN @Gpio TO 12
ENTER @Gpio;Number
DISP "Number=";Number

```

Number= 2

Signal Line		ASCII
DI7.....	DI0	Char.
0 1 0 0	0 0 0 0	@
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 0 0 0	N <sub>u</sub>
0 0 0 0	0 0 0 0	N <sub>u</sub>

6



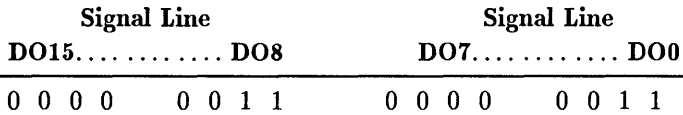
**Word Transfers**

### Example Statements that Output Data Words

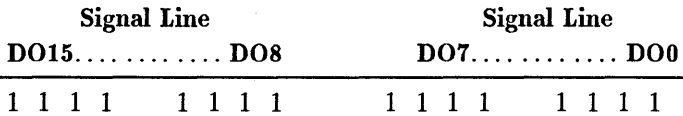
Data are automatically sent as words when using an I/O path with the WORD attribute. See the “I/O Path Attributes” chapter for further information.

The following diagrams show the logic signals that appear on the Data Out lines as a result of the corresponding BASIC statements and numeric values. All numeric values are first rounded to an INTEGER value before being placed on the Data Out lines. The actual logic level that appears on each line depends on how the lines have been configured (i.e., as High-true or Low-true).

```
Word=3*256+3
OUTPUT @Gpio USING "#,W";Output_word
```



```
Output_16_bits=-1
CONTROL Gp_isc,3;Output_16_bits
```



It is important to note that *no output handshake is executed when the CONTROL statement is executed*; only the states of the Data Out lines and the I/O line are affected. Handshake sequence, if desired, must be performed by BASIC statements in the program. See “Designing Your Own Transfers” for design suggestions.

### Example Statements that Enter Data Words

The following diagrams show the variable values that result from entering the logic signals on the Data In lines. Note that all sixteen-bit values entered are interpreted as INTEGER values.

Signal Line DI15..... DI8	Signal Line DI7..... DI0
0 0 0 0    0 0 0 1	1 1 1 1    1 1 1 1

```
ENTER 12 USING "#,W";Enter_16_bits
DISP "INTEGER entered=";Enter_16_bits
```

```
INTEGER entered= 511
```

Signal Line DI15..... DI8	Signal Line DI7..... DI0
1 1 1 1    1 1 1 0	0 0 0 0    0 0 0 0

```
STATUS Gp_isc,3;Enter_16_bits
DISP "INTEGER entered=";Enter_16_bits
```

```
INTEGER entered= -512
```

6

It is important to note that no enter handshake is performed when the STATUS statement is executed. The only actions taken are the I/O line being placed in the High state and the Data In registers being read. If an enter handshake is required, it must be performed by the BASIC program. See "Designing Your Own Transfers" for design suggestions.

Remember also that the Data In Clock source is solely determined by the switch setting on the interface card. Thus, when the STATUS statement is used to read the Data In lines, the data on the lines may or may not be clocked into the registers when the statement is executed. If the data are to be clocked in by the STATUS statement, the "READ" clock source must be selected. See the installation manual for further details.



## Using a GPIO Interface in the HP-UX Environment

This section explains the interface locking and burst I/O, which are useful when using an interface in the HP-UX environment.

### Locking an Interface to a Process

In a multi-user environment, interface cards are usually accessible to several users. BASIC/UX supports this sharing by making no attempt to guarantee exclusive access to an interface *unless it is directed to do so*. This allows you to access instruments, for instance, on an GPIO bus that is shared with other peripherals. Although this is not a recommended configuration, it is allowed.

BASIC/UX provides interface locking to support exclusive access to an interface. When an interface is locked to a process, all other processes are prevented from using that interface. For instance, this feature can prevent the loss of important data while a process is taking measurements from an instrument by keeping other users or processes from using the same interface.

Interface locking is enabled and disabled by using pseudo-register 255 and the interface's select code. For example:

`CONTROL 12,255;1` *Enables GPIO interface locking.*

`CONTROL 12,255;0` *Disables GPIO interface locking.*

In order to be a “good citizen” on a multi-user system, you should unlock an interface after you no longer need to have it locked.

Note that attempting to lock an GPIO connected to a system disc will result in an error.

In addition, attempting to lock an interface that is already locked to another process will cause a program to suspend execution until:

- The interface is unlocked (by the other process to which it is currently locked).
- A timeout occurs.
- You press **Reset** or **Clr I/O**.

## Using the Burst I/O Mode

The default mode of HP-UX I/O transactions requires many time consuming HP-UX system calls to send data to the destination.

Another method, “burst I/O”, maps the interface into your “user address space”, thereby bypassing the memory buffer. This direct-write method decreases the number of calls to HP-UX I/O system routines, which establishes a short, highly tuned path for performing I/O operations. The interface is also implicitly locked when burst mode is enabled (see above explanation of interface locking).

Burst I/O provides the fastest I/O performance available with BASIC/UX for the “smaller” I/O transactions that are typical of many instruments. For instance, an 8-byte ENTER operation is over an order of magnitude faster when burst mode is enabled. For larger I/O operations, of more than 4 000 bytes for example, burst mode becomes increasingly slower than the default (buffered or DMA) I/O modes.

Burst I/O is enabled and disabled by using register 255 and the interface’s select code. For example:

`CONTROL 12,255;3` *Enables GPIO interface burst I/O.*

`CONTROL 12,255;0` *Disables GPIO interface burst I/O.*

6

In order to be a “good citizen” on a multi-user system, you should unlock an interface after you no longer need to have it locked.

In addition, attempting to use burst mode with an interface that is already locked to another process will cause a program to suspend execution until:

- The interface is unlocked (by the other process to which it is currently locked).
- A timeout occurs.
- You press **Reset** or **Clr I/O**.

Note also that you cannot set up an ON TIMEOUT for an interface when using burst mode.

## GPIO Timeouts

Timeout events were generally discussed in the chapter “Interface Events”. However, specific details of the affects of the time parameter on the event’s occurrence were not described. This section explains how the time parameter is measured and describes typical service routines.

### Timeout Time Parameter

There are two general time intervals measured and compared to the specified TIMEOUT time. The first interval is measured between the computer initiating the first handshake (PCTL=Set) and the peripheral signalling Ready (with the PFLG line). If the peripheral does not indicate readiness by the specified TIMEOUT time parameter, a TIMEOUT event occurs. (The computer optionally reads the state of the PSTS line before initiating the transfer. See “Using the PSTS Line” for further details.)

The time elapsed during each handshake is also measured and compared to the TIMEOUT time. The timing begins when the transfer is initiated (PCTL Set by the computer) and, in general, ends when the peripheral responds on the PFLG line.

Keep in mind that the TIMEOUT time parameter specifies the *minimum* time that the computer will wait before initiating the ON TIMEOUT branch. However, the computer may occasionally wait an additional 25 percent of the specified time parameter before initiating the branch. For instance, if a time of 0.4 seconds is specified, the computer will wait at least 0.4 seconds for the handshake to be completed, but it may occasionally wait up to 0.5 seconds before initiating the ON TIMEOUT branch.

Note that timeouts do not occur when burst mode is enabled.

### Timeout Service Routines

The service routine usually responds by determining if the peripheral is functioning properly (“ok”) or is down (“not ok”). The simplest action that might be taken by the computer is to read the state of the PSTS signal line, as shown in the following service routine.

```

100 Gpio=12
110 ON TIMEOUT Gpio,.08 GOSUB Gpio_down
    .
    .
    .
200 OUTPUT Gpio;String$
210 ! Next line.
    .
    .
    .
300 Gpio_down: STATUS Gpio,5;Periph_status
310         Psts=BIT(Periph_status,3) ! Read PSTS.
320         IF NOT Psts THEN
330             PRINT "GPIO interface is "
340             PRINT "non-functional"
350             PRINT "Program paused."
360             PAUSE
370         ELSE
380             ! Take other action.
390             END IF
400     RETURN

```

6 A **TIMEOUT** has been set up to occur if the peripheral takes approximately more than .08 second to complete its response during a data transfer; how the peripheral completes its response depends on the handshake mode currently selected. With Pulse-Mode Handshakes, the peripheral completes its response by using PFLG to Clear PCTL; with Full-Mode Handshakes, the response is complete *only* after PCTL has been Cleared *and* PFLG is in the Ready state.

*When a TIMEOUT occurs, the computer automatically executes an Interface Reset; the PCTL line is Set and then Cleared, and the PRESET line is pulsed Low. See the section called "Interface Reset" for further effects. The Service routine checks the PSTS line to see if the peripheral is OK or not OK. If not OK, a message is displayed and the program is paused; if OK, program execution is returned to the line following that in which the TIMEOUT occurred. The service routine may be programmed to attempt the transfer again, if desired; however, the automatic Reset performed when the TIMEOUT occurred may make this type of response difficult to implement.*

---

## Using Alternate Data Representations

As with any other interface, representations other than the ASCII or internal representations may sometimes be more meaningful to the peripheral. This section briefly describes a few techniques for implementing alternate data representations.

### BCD Representation

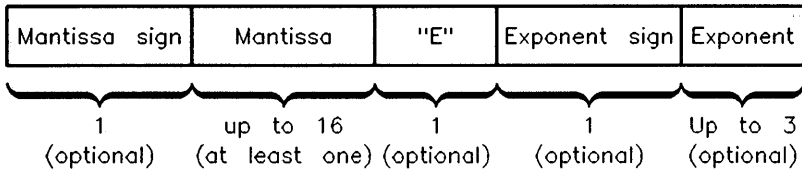
With OUTPUT and ENTER statements, numeric values are either represented by ASCII characters or by one of the internal representations (INTEGER or REAL). Another common method of representing numeric data is to use four-bit, binary-coded decimal (BCD) characters. Only ten of the available sixteen bit patterns need to be used to represent decimal digits “0” through “9”. The remaining six patterns can be used for sign, decimal point, exponent, and other special characters, as required by the application.

The following bit patterns have been chosen arbitrarily to correspond to numeric characters. This representation cannot be used if more than six other characters are to be represented. (Note that this is also the data representation used by the HP 98623 BCD Interface. See the “BCD Interface” chapter for further information.)

**Bit Patterns for Numeric Characters**

Decimal Digit	Bit Pattern			Other Character	Bit Pattern				
	MSB		LSB		MSB		LSB		
0	0	0	0	0	Line Feed	1	0	1	0
1	0	0	0	1	+	1	0	1	1
2	0	0	1	0	,	1	1	0	0
3	0	0	1	1	-	1	1	0	1
4	0	1	0	0	E	1	1	1	0
5	0	1	0	1	.	1	1	1	1
6	0	1	1	0					
7	0	1	1	1					
8	1	0	0	0					
9	1	0	0	1					

The following subprogram assumes that BCD numbers are to be entered through the GPIO Interface. Sixteen BCD characters are represented by four 16-bit words from the peripheral. The sixteen four-bit BCD characters have the following general format.



Each BCD character is represented by four bits of data. The first word entered contains the four most significant BCD characters, and the last word contains the least significant. The program changes the BCD characters to their ASCII representation and then uses the number builder to generate the corresponding numeric value.

```

100  ASSIGN @Gpio TO 12
110  !
120  ! Define conversion string.
130  Conv$="0123456789"&CHR$(10)&"+,-E."
140  !
150  CALL Enter_bcd(@Gpio,Conv$,Number)
160  OUTPUT 1;"The BCD number is ";Number
170  !
180  END
190  !
200  !
210  SUB Enter_bcd(@Device,Conv$,Number)
220  COM /Enter_bcd/ INTEGER Word(1:4)
230  !
240  ! Enter 4 words (=16 BCD digits).
250  ENTER @Device USING "#,W";Word(*)
260  !
270  FOR W=1 TO 4 ! Process four words.
280  !
290  ! Shift right multiples of four bits.
300  FOR Bits_rt=12 TO 0 STEP -4
310  Shifted_word=SHIFT(Word(W),Bits_rt)
320  Four_lsb=BINAND(Shifted_word,15) ! Mask MSB's.
330  Ascii_char$=Conv$[Four_lsb+1;1] ! LSB's = index.
340  Number$=Number$&Ascii_char$
350  NEXT Bits_rt
360  !
370  NEXT W
380  !
390  ENTER Number$;Number ! Use number builder.
400  SUBEND ! Returns BCD number as "Number".

```

## Character Conversions

One of the most common needs of a computer is to convert certain unused or disallowed bit patterns into meaningful or allowed bit patterns. A typical example is to change the radix character from a decimal point to a comma. For instance, the following ASCII characters represent the same number.

<b>U.S. Representation</b>	<b>European Representation</b>
1,234,567.89	1.234.567,89

A remedy is needed to allow these types of numbers to be entered through the number builder. To enter a number with the preceding European format, the commas must be changed to periods and the periods changed to spaces. The following routine changes the numeric radix from the European to the US convention when numeric data are entered through the GPIO.



```

100 ! Generate string with no conversions.
110 DIM Conv$[256]
120 FOR Code=0 TO 255
130     Conv$[Code+1]=CHR$(Code)
140 NEXT Code
150 !
160 ! Then define the conversions.
170 Conv$[NUM(".")+1;1]=" " ! Change "." to " "
180 Conv$[NUM(",")+1;1]="." ! Change "," to "."
190 !
200 !
210 Number$="123.456,789"
220 PRINT "Before conversion ";Number$
230 CALL Convert(Conv$,Number$)
240 PRINT "After conversion ";Number$
250 !
260 END
270 !
280 !
290 SUB Convert(Conv$,Data$)
300 !
310 FOR Char_pos=1 TO LEN(Data$)
320     Index=NUM(Data$[Char_pos])+1
330     Data$[Char_pos;1]=Conv$[Index;1]
340 NEXT Char_pos
350 !
360 ! Returns Data$ with converted characters.
370 SUBEND

```

6

If more characters are to be converted, simply change the default (standard ASCII) character in Conv\$ to the desired code. The speed of the conversion is not affected by the number of characters to be converted. This routine works for either input or output, but the characters to be converted must be in a string variable.

Conversions can also be made by using the CONVERT attribute. See the “I/O Path Attributes” chapter for further information.

---

## GPIO Interrupts

This section describes the types of and techniques for using the interrupts available on the GPIO Interface.

### Types of Interrupt Events

The GPIO Interface can sense *two interrupt events*: the first is the interface becoming “Ready” for subsequent handshakes, and the second is the External Interrupt Request line (EIR) being driven to logic low by the peripheral. As with all interfaces, both events initiate identical computer responses—the service routine must be able to determine which of these interrupts have occurred if both are enabled to initiate interrupts.

Both of these types of interrupt events are *level-sensitive*; in other words, the signal that caused the event should be maintained until the program has time to determine which event has caused the interrupt. Further explanation follows in this section.

### Setting Up and Enabling Events

When either event occurs, the interrupt is logged by the BASIC operating system. After logging the occurrence, any further interrupts from the GPIO Interface are automatically disabled until specifically enabled by a program. All further computer responses to either event depend entirely on the BASIC program currently in memory.

The following program segment shows the steps involved in setting up and enabling Ready interrupts.

```
100  Gpio=12
110  ON INTR Gpio GOSUB Gpio_serv
120  !
130  Mask=2
140  ENABLE INTR Gpio;Mask
```

The value of the interrupt mask determines which, if any, of the GPIO interrupt events are to be enabled to initiate the corresponding branch. Bits of the Interrupt Mask register have the following definitions.

*Interrupt Enable Register* (ENABLED INTR)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Enable interface ready interrupts	Enable EIR interrupts
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

- *Interface Ready*—Setting this bit (1) enables an interrupt to initiate the ON INTR branch when the interface detects that it is Ready to handshake data. If Full-Mode Handshake is selected (with the Option Select switch), the Ready event is PCTL=Clear and PFLG=Ready. With Pulse-Mode Handshake, the event is PCTL=Clear (independent of the state of PFLG).
- *External Interrupt Request*—Setting this bit (1) enables an interrupt to initiate the ON INTR branch when the interface senses an External Interrupt Request (EIR line=Low).

### Interrupt Service Routines

If both events are enabled, the service routine must be able to differentiate between the two. And, if both have occurred, the service routine must be able to service both causes. The following registers contain the current state of the Interface Ready flag and EIR signal lines, from which the interrupt cause(s) may be determined.

*STATUS Register 4* Interface is ready for a subsequent data transfer;  
1=Ready, 0=Busy.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR line low	STI1 line low	STI0 line low
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

As mentioned in preceding paragraphs, these two interrupt causes are both *level-sensitive* events, not edge-triggered events. This fact has *two important implications*. The *first* is that, for an event to be recognized, the corresponding signal line must be held in the interrupting state until the computer can interrogate the line's logic state. If the signal line's state is changed before the service routine checks the line, the interrupt may be "missed". This will happen only if both events are enabled; if only one event is enabled, determining the cause may not be necessary.

The *second implication* is that the service routine must be able to acknowledge the request in order for the peripheral device to remove the request. If the request is not removed after service, the same request may be serviced more than once.

The following program shows a simple example of servicing an External Interrupt Request. Note that only EIR-type interrupts have been enabled and that the peripheral device provides its own interrupt cause with signals on the STI0 and STI1 lines.

```

100  PRINTER IS 1
110  Gpio=12
120  CONTROL Gpio;1 ! Reset Interface.
130  !
140  ON INTR Gpio GOSUB Gpio_serv
150  ENABLE INTR Gpio;1 ! Enable EIR-type only.
160  !
170  ! Show concurrent processing.
180  Loop: Counter=Counter+1
190      DISP Counter
200      GOTO Loop
210  !
220  STOP

```

```

230  !
240 Gpio_serv: !
250 STATUS Gpio,5;Periph_status ! Check EIR line.
260 IF BIT(Periph_status,2) THEN ! EIR interrupt.
270  !
280   IF BIT(Periph_status,0) THEN ! STIO=True.
290     BEEP
300     PRINT "Improper value; type in correct"
310     PRINT "value, and press ENTER."
320     PRINT
330     ENTER 2;Value
340     OUTPUT Gpio;Value
350   END IF
360   !
370   IF BIT(Periph_status,1) THEN ! STI1=True.
380     BEEP
390     PRINT "Reading of: ";Reading;" out of range"
400     PRINT "No other action will be taken."
410     PRINT
420     WAIT 2
430     BEEP
440   END IF
450   !
460 END IF
470  !
480  ! Put Ready service routine here.
490  !
500  !
510  ENABLE INTR Gpio      ! Use same mask.
520  RETURN
530  !
540  END

```

A slightly different method that peripherals use to communicate the cause of their interrupt request is to place the interrupt cause on the data lines concurrent with the interrupt request. The service routine can determine the cause by reading STATUS register 3 and take the appropriate action.

Notice that the service routine indicates a likely place for a Ready-interrupt service routine. The Service routine must check for the Ready condition, acknowledge the interrupt, and then take the desired action. In this case, no service action has been defined because Ready interrupts have not been enabled. The next section provides an example of a Ready interrupt service routine.

---

## Designing Your Own Transfers

Other specialized methods of handshaking data can be designed according to your specific needs. The methods of synchronizing data transfers are as flexible as the GPIO Interface hardware. However, the general techniques will probably still require the fundamental handshake features: initiation by the sending device, acknowledgement from the receiving device, and agreement as to when the data are valid. The TRANSFER statement can be used to transfer data. See the chapter “Advanced Transfer Techniques” for further information.

A wide choice of initiating events is available; obvious possibilities include use of the PCTL, EIR, or CTL0 (or CTL1) lines to signal the start of the transfer. Data can be placed on the Data Out lines by writing to CONTROL register 3, or data can be clocked into the Data In registers by reading STATUS register 3. Sensing acknowledgement from the peripheral can be accomplished by reading the state of such lines as PFLG, PSTS, EIR, or STI0 (or STI1).

The feature common to all of these methods is that each byte (or word) of data must be transferred individually. If an entire block of data is to be entered or output, the BASIC program that implements the transfer must keep a “pointer” to which byte/word is to be transferred.

6

### Full Handshake Transfer

The following program implements a handshake similar to the Full OUTPUT Handshake by controlling the PCTL and sensing the PFLG and PCTL lines. The actual “Output” routine consists of lines 150 through 190. Timeout capability can easily be included in the routine, if so desired.

```

100 DATA 65,66,67,68,69
110 !
120 STATUS 12,5;Periph_status ! Check PSTS.
130 IF BIT(Periph_status,3) THEN ! PSTS True.
140 !
150 FOR Char=1 TO 5
160 READ Code
170 Wait: STATUS 12,4;Interface_ready
180 IF NOT Interface_ready THEN Wait
190 Output: CONTROL 12,3;Code ! Data onto lines.
200 CONTROL 12,1;1 ! Set PCTL.
210 NEXT Char
220 !
230 ELSE ! PSTS False.
240 PRINT "Peripheral error"
250 PAUSE
260 END IF
270 !
280 END

```

Notice that each byte of data must be output separately and that the program must keep track of which byte, of several, is to be sent. Keep in mind that the data written to CONTROL register 3 is *16-bit words*; in this case, the most significant eight bits (byte) is all zeros. Also, using FOR ... NEXT loops to index each byte/word to be sent may not be the most expedient way of sending data, so your particular application may use alternative methods for handling the data.

The following subprogram implements a handshake similar to the Full ENTER handshake.

```

170 SUB Enter_word(@Device,Data_word)
180 !
190 Wait1: STATUS 12,4;Interface_ready
200 IF NOT Interface_ready THEN Wait1
210 STATUS 12,3;Dummy_read ! I/O High.
220 CONTROL 12,1;1 ! Set PCTL.
230 Wait2: STATUS 12,4;Interface_ready
240 IF NOT Interface_ready THEN Wait2
250 STATUS 12,3;Data_word ! Enter word.
260 !
270 SUBEND

```

The appropriate Data-In Clock source should be selected to ensure the data are clocked into the registers when valid. Refer to the installation manual for further details.

## Interrupt Transfers

The interrupt capabilities of the GPIO Interface can be used to synchronize the transfer of data between the computer and peripheral. These examples describe simple methods of synchronizing the transfer of data by using both the EIR and the PFLG line. See the section of this chapter called “GPIO Interrupts” for further explanation of GPIO interrupts in general.

General interrupt transfers through the GPIO Interface involve the following elements:

- placing data on (or reading data from) the data lines
- signaling to the peripheral device to initiate the transfer
- continuing processing until an interrupt is received, at which time the handshake is finished and transfer of the next byte/word can be initiated.

Examples of using Ready interrupts to implement interrupt transfers are given in the remainder of this section.

6

### Ready Interrupt Transfers

The Ready interrupt event occurs when the GPIO Interface becomes “Ready”. Whether or not the GPIO Interface is Ready depends on the currently selected handshake mode. If Full-Mode Handshake is selected, the interface is Ready if *both* the PFLG line is Ready and the PCTL line is Clear; if Pulse-Mode is selected, the interface is Ready if PCTL is in the Clear state, regardless of the state of PFLG. The following program shows how to implement Ready interrupt transfers.

```
100  PRINTER IS 1
110  Gpio=12
120  CONTROL Gpio;1 ! Reset Interface.
130  ON INTR Gpio GOSUB Ready_xfer
140  !
150  DIM Data_out$(256)
160  Data_out$="123ABC"
170  Pointer=1
```



```

180 Size=LEN(Data_out$)
190 !
200 ! Initiate the transfer.
210 GOSUB Ready_xfer
220 !
230 ! Show concurrent processing.
240 Loop: Counter=Counter+1
250     DISP Counter
260     GOTO Loop
270 !
280 STOP
290 !
300 ! The branch to this subroutine is initiated
310 ! first by the program, but thereafter by
320 ! Ready Interrupt events.
330 !
340 Ready_xfer: !
350     !
360     IF Pointer<=Size THEN
370         Byte_out=NUM(Data_out$[Pointer;1])
380         PRINT Data_out$[Pointer;1];" sent"
390         CONTROL Gpio,3;Byte_out ! Place data on lines.
400         Pointer=Pointer+1
410         CONTROL Gpio,1;1         ! Set PCTL.
420         ENABLE INTR Gpio;2     ! Enable Ready INTR's.
430         RETURN
440     !
450     ELSE
460         DISABLE INTR Gpio     ! Disable after done.
470         RETURN
480     !
490 END IF
500 !
510 !
520 END

```

Interrupt transfers that use the EIR line are similar to Ready interrupt transfers. The main difference is that the interrupt-initiating event is the EIR line, rather than the PCTL line (and PFLG if in Full Handshake mode) indicating Interface Ready.

---

## Using the Special-Purpose Lines

Four special-purpose signal lines are available for a variety of uses. Two of these lines are available for output (CTL0 and CTL1), and the other two are used as inputs (STI0 and STI1).

### Driving the Control Output Lines

Setting bits 0 and 1 of GPIO CONTROL register 2 places a logic low on CTL0 and CTL1, respectively. The definition of this CONTROL register is shown in the following diagram.

*CONTROL Register 2*      Peripheral Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					PSTS error (1=report; 0=ignore)	Set CTL1 (1=low; 0=high)	Set CTL0 (1=low; 0=high)
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Ct10=0 ! Clear.  
Ct11=1 ! Set.  
CONTROL 12,2;Ct11\*2+Ct10

As indicated in the diagram, setting a bit in the register places the corresponding line Low, while clearing the bit places a logic High on the line. The logic polarity of these signals cannot be changed. The signal remains on these lines until another value is written into the CONTROL register, and Reset has no effect on the state of either line.

## Interrogating the Status Input Lines

The state of both status input lines STI0 and STI1 are determined by reading bits 0 and 1 of STATUS register 5, respectively. A logic “1” in a bit position indicates that the corresponding line is at logic Low, and a “0” indicates the opposite logic state. This logic polarity cannot be changed. The definition of GPIO STATUS register 5 follows.

*STATUS Register 5*                      Peripheral Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR line low	STI1 line low	STI0 line low
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

```
STATUS 12,5;P_status
Sti0=BIT(P_status,0)
Sti1=BIT(P_status,1)
```

Reading this register returns a numeric value that reflects the logic states of these lines *at the instant the computer reads the interface lines*; the state of these lines are not latched by any internal or external event.

6

## Using the PSTS Line

The Peripheral Status line (PSTS) is generally used to indicate whether or not the peripheral device is functional. The current state of PSTS may be checked by reading STATUS Register 5 (bit 3). It may also optionally be checked automatically at the beginning of an OUTPUT or ENTER statement; normally, it is not checked. This feature is only enabled by setting Bit 2 of CONTROL register 2.

*CONTROL Register 2*      Peripheral Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					PSTS error (1=report; 0=ignore)	Set CTL1 (1=low; 0=high)	Set CTL0 (1=low; 0=high)
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

When Bit 2 is set and PSTS is false at the beginning of either an OUTPUT or ENTER statement, Error 172 (Peripheral error) is reported. The error must be trapped with ON ERROR, since it generates no INTR or TIMEOUT branch.

---

## Summary of GPIO STATUS and CONTROL Registers

6

- STATUS Register 0*      Card Identification. Always 3.
- CONTROL Register 0*      Interface Reset. Any non-zero value causes a reset.
- STATUS Register 1*      Interrupt and DMA Status.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts are enabled	An interrupt is currently requested	Interrupt level switches (hard-ware priority)	Interrupt level switches (hard-ware priority)	Burst mode DMA	Word mode DMA	DMA channel 1 enabled	DMA channel 0 enabled
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*CONTROL Register 1*      Set PCTL Line. Any non-zero value sets the line.

*STATUS Register 2*

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake in process	Interrupts are enabled	Transfer in progress
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*CONTROL Register 2*      Peripheral Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					PSTS error (1=report; 0=ignore)	Set CTL1 (1=low; 0=high)	Set CTL0 (1=low; 0=high)
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*STATUS Register 3*      Data In (16 bits)

*CONTROL Register 3*      Data Out (16 bits)

*STATUS Register 4*      Interface Ready. Interface is Ready for a subsequent data transfer: 1=Ready, 0=Busy.

*STATUS Register 5*      Peripheral Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR line low	STI1 line low	STI0 line low
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Interrupt Enable Register*

(ENABLE INTR)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Enable interface ready interrupts	Enable EIR interrupts
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*STATUS Register 255*

- 0: GPIO interface unlocked and GPIO interface burst I/O disabled. (BASIC/WS and BASIC/DOS accept this command but always return the value "3".)
- 1: GPIO interface locked.
- 3: GPIO interface burst I/O enabled.

*CONTROL Register 255*

- 0: disables GPIO interface locking and GPIO interface burst I/O. (BASIC/WS and BASIC/DOS accept this command but always set the value "3".)
- 1: enables GPIO interface locking.
- 3: enables GPIO interface burst I/O.

6

---

## Summary of GPIO READIO and WRITEIO Registers

This section describes the GPIO Interface's READIO and WRITEIO registers. Keep in mind that these registers should be used *only* when you know the exact consequences of their use, as using some of the registers improperly may result in improper interface behavior. If the desired operation can be performed with STATUS or CONTROL, you should not use READIO or WRITEIO.

## GPIO READIO Registers

Register 0	Interface Ready
Register 1	Card Identification
Register 2	Undefined
Register 3	Interrupt Status
Register 4	MSB of Data In
Register 5	LSB of Data In
Register 6	Undefined
Register 7	Peripheral Status

*READIO Register 0*      Interface Ready. A 1 indicates that the interface is Ready for subsequent data transfers, and 0 indicates Not Ready.

*READIO Register 1*      Card Identification. This register always contains 3, the identification for GPIO interfaces.

*READIO Register 3*      Interrupt Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts are enabled	An interrupt is currently requested	Interrupt level switches (hardware priority)	Interrupt level switches (hardware priority)	Burst mode DMA	Word mode DMA	DMA channel 1 enabled	DMA channel 0 enabled
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*READIO Register 4*

MSB of Data In

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*READIO Register 5*

LSB of Data In

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*READIO Register 7*

Peripheral Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR line low	STI1 line low	STI0 line low
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

6

**GPIO WRITEIO Registers**

- WRITEIO Register 0      Set PCTL
- WRITEIO Register 1      Reset Interface
- WRITEIO Register 2      Interrupt Mask
- WRITEIO Register 3      Interrupt and DMA Enable
- WRITEIO Register 4      MSB of Data Out
- WRITEIO Register 5      LSB of Data Out



- WRITEIO Register 6**      Undefined
- WRITEIO Register 7**      Set Control Output Lines
- WRITEIO Register 0**      Set PCTL. Writing any non-zero numeric value to this register places PCTL in the Set state; writing zero causes no action.
- WRITEIO Register 1**      Reset Interface. Writing any non-zero numeric value to this register resets the interface.
- WRITEIO Register 2**      Interrupt Mask

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Enable interface ready interrupts	Enable EIR interrupts
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

**WRITEIO Register 3**      Interrupt and DMA Enable

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable interrupts	Not used			Enable burst mode DMA	Enable word mode DMA	Enable DMA channel 1	Enable DMA channel 0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*WRITEIO Register 4*

MSB of Data Out

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*WRITEIO Register 5*

LSB of Data Out

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*WRITEIO Register 7*

Set Control Output Lines

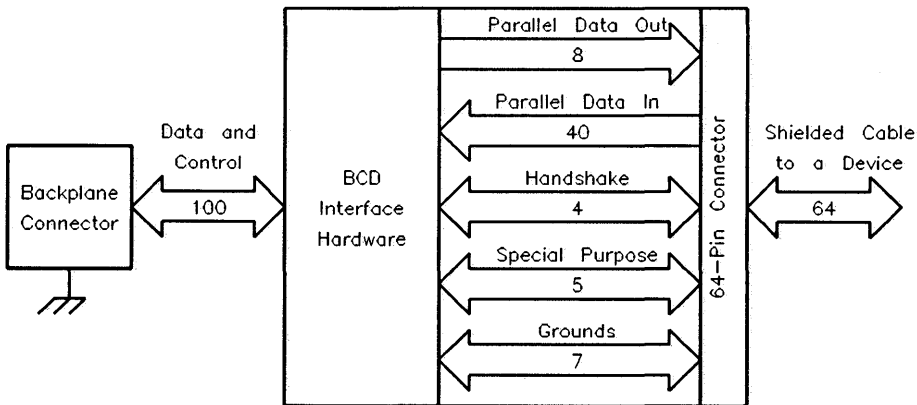
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Set CTL1 (1=low; 0=high)	Set CTL0 (1=low; 0=high)
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

6

## The BCD Interface for BASIC/WS

This chapter should be used in conjunction with the *HP 98623 BCD Interface Installation Note*. The best way to use these two documents is to first read the section of this chapter called “Brief Description of Operation” to see how the interface works with the BASIC language. Within this section is information about the interface’s modes of operation that will help you to understand how you might use the interface for your application. Second, read “Configuring the Interface” while referring to the Installation Note as necessary to configure and connect the interface according to your application’s needs. The reason for this order is that you will be able to configure and use the interface once you know a little about how it works.

The main section of the chapter presents several techniques for using the interface to move data between the computer and peripheral devices using BASIC programs.



**Block Diagram of the BCD Interface**

---

## Brief Description of Operation

The HP 98623 Interface consists of data registers and handshake circuitry required to transfer data to and from the computer using either BCD or binary data formats. The interface cable contains the following sixty-four conductors:

- forty data, two sign, and one overload signal lines used to enter data from the peripheral
- eight lines used to output data to the peripheral
- two sets of handshake lines (two wires per set)
- one reset line to the peripheral device
- one five-volt logic line
- five logic (signal) grounds and two safety (shield) grounds

## Data Representations and Formats

The BCD interface can be used to transfer data using one of two data representations: BCD (binary-coded decimal) and binary representations. BCD is the default data representation; the binary representation may be selected by software (as described in the configuration section).

### The BCD Data Representation

When the BCD representation is in use, data lines are handled in groups of four, with each group representing one BCD digit. The sixteen possible combinations of logic states and corresponding characters which each four-line group may represent are as follows:

### BCD Logic States

Data Line Logic Sense (MSB) (LSB)	Character Represented	Data Line Logic Sense (MSB) (LSB)	Character Represented
0 0 0 0	0	1 0 0 0	8
0 0 0 1	1	1 0 0 1	9
0 0 1 0	2	1 0 1 0	line-feed
0 0 1 1	3	1 0 1 1	+
0 1 0 0	4	1 1 0 0	,
0 1 0 1	5	1 1 0 1	-
0 1 1 0	6	1 1 1 0	E
0 1 1 1	7	1 1 1 1	.

When the BCD representation is in use, the data lines are read character read, a corresponding ASCII character (listed above) is generated. Operating system “drivers” control both the sequence of reading the BCD-character groups and the generation of the appropriate ASCII character which each group represents. The sequence used by the drivers and the resultant numeric value entered depends on which BCD format is currently in use: Standard or Optional format.

#### Standard Format

The Standard BCD format is used to connect one peripheral to the computer. The data lines are arranged as follows to form two numbers: one mantissa sign bit, eight BCD mantissa characters, one exponent sign bit, and one BCD exponent character form the first number; one overload-indicator bit and one BCD character are combined to form the second number.

The following diagram shows how the signal lines are organized in Standard format (i.e., the order in which the lines are read with ENTER statements). The notation used with these diagrams is as follows: SGN1, SGN2, and OVLD are individual signal lines, while DI1 through DI10 are groups of four lines each. The signal lines of group DIx (in which x denotes one of the BCD

characters 1 through 10) consist of DIx-8, DIx-4, DIx-2, and DIx-1; the 8, 4, 2, and 1 prefixes are used to denote the binary-weighted significance of each line.

### Standard Format (Read One BCD Device)

Signal Name	Information	BCD Char. (Pos. True)	ASCII Char.
SGN1	Mant. Sign	+1011, -1101	+ -
DI1	MSD	0000 thru 1111	X
DI2	↑	↑	X
DI3			X
DI4			X
DI5			X
DI6			X
DI7	↓	↓	X
DI8	LSD	0000 thru 1111	X
	Exp. Char.	1110	E
SGN2	Exp. Sign	+1011, -1101	+ -
DI9	Exp. Digit	0000 thru 1111	X
	Comma	1100	,
OVL D	0= $\overline{\text{OVL D}}$ , 8=OVL D	0000, 1000	0 or 8
DI10	Fn. Digit	0000 thru 1111	X
	Line-Feed	1010	LF

Let's take a closer look at how data is entered into the computer with a BASIC-language ENTER statement while using the Standard format. (Standard format is selected when the Peripheral Status Switch marked "OF" is in the "ON" position; further details will be given in the subsequent configuration section.) Suppose the following logic signals are present on the lines from the peripheral device:

#### 7-4 The BCD Interface for BASIC/WS

### BCD-Mode Standard Format

Signal Name	Logic Level	BCD Character
SGN1	1	—
DI1—8	0	1
DI1—4	0	
DI1—2	0	
DI1—1	1	
DI2—8	0	2
DI2—4	0	
DI2—2	1	
DI2—1	0	
DI3—8	0	3
DI3—4	0	
DI3—2	1	
DI3—1	1	
DI4—8	0	4
DI4—4	1	
DI4—2	0	
DI4—1	0	
DI5—8	0	5
DI5—4	1	
DI5—2	0	
DI5—1	1	

### BCD-Mode Standard Format (continued)

Signal Name	Logic Level	BCD Character
DI6—8	0	6
DI6—4	1	
DI6—2	1	
DI6—1	0	
DI7—8	0	7
DI7—4	1	
DI7—2	1	
DI7—1	1	
DI8—8	1	8
DI8—4	0	
DI8—2	0	
DI8—1	0	
SGN2	0	+
DI9-8	1	9
DI9-4	0	
DI9-2	0	
DI9-1	1	
OVL D	0	0
DI10-8	0	2
DI10-4	0	
DI10-2	1	
DI10-1	0	

7

Number = -1.2345678E+16 Function = 2

Let's further assume the following: the Peripheral Status Switch settings are DATA=ON, SGN1=ON, SGN2=ON, OVL D=ON; and the following ENTER statement has been executed (with the BCD Interface as the source):

ENTER Bcd;Number,Function

The ENTER statement is executed as follows. The computer first initiates a handshake with the CTLA signal (handshake operation is also described in the configuration section). The peripheral responds to the request by placing data



on the lines and then completing the handshake. The states of all data lines are now stored in registers on the interface (i.e, the data signals are “latched”).

The Standard-format driver reads the state of the SGN1 line and generates an ASCII “+” character. The “number builder” routine of the free-field ENTER statement (described in Chapter 5) is used to construct the number as characters are entered for the variable **Number**.

The BCD digits DI1 through DI8 are then read and used to form the mantissa. The “E” character is generated automatically by the driver, after which it reads the SGN2 line and generates a “-” character. BCD digit DI9 is read; the driver generates a “3” for the exponent character. A comma is automatically generated by the driver, terminating entry into **Number**. The number builder then constructs the internal representation of -0.4205, which is placed in **Number**.

Since one additional numeric variable has been specified in the ENTER statement, the computer continues to enter characters from the interface. The OVLD signal line is read, and a “0” is generated and entered. BCD digit DI10 is read, and the resultant ASCII “2” is entered by the number builder. The driver automatically generates the line-feed character, which terminates both entry of characters into the **Function** variable and the ENTER statement. The variable **Function** is assigned a value of 2, and the ENTER has finished execution. Further examples of sending and receiving data through the BCD Interface are given in the main section of this chapter.

### **Optional Format**

With the Optional format, two peripherals may be connected to the interface. One four-digit and one five-digit mantissa are generated with this format. The signal lines are organized as follows with Optional format:

### Optional Format (Read Two BCD Devices)

Signal Name	Information	BCD Char. (Pos. True)	ASCII Char.
SGN1 <sup>1</sup>	Mant. Sign	+1011, -1101	+ -
DI4 <sup>1</sup>	MSD	0000 thru 1111	X
DI2 <sup>1</sup>	↑	↑	X
DI6 <sup>1</sup>	↓	↓	X
DI8 <sup>1</sup>	LSD	0000 thru 1111	X
	Comma	1100	,
SGN2 <sup>2</sup>	Mant. Sign	+1011, -1101	+ -
DI10 <sup>2</sup>	MSD	0000 thru 1111	X
DI1 <sup>2</sup>	↑	↑	X
DI5 <sup>2</sup>			X
DI3 <sup>2</sup>	↓	↓	X
DI7 <sup>2</sup>	LSD	0000 thru 1111	X
	Exp. Char	1110	E
OVLD	FD	0000 thru 1000	0 or 8
DI9	SD	0000 thru 1000	0 or 8
	Line-Feed	1010	LF

7

<sup>1</sup> First Device (FD) <sup>2</sup> Second Device (SD)

Let's take a closer look at how data is entered into the computer by a BASIC-language ENTER statement while using the Optional format ("OF"=OFF). Suppose the following logic signals are present on the lines from the peripheral device:

### BCD-Mode Optional Format

Signal Name	Logic Level	BCD Character
SGN1	1	—
DI4—8	0	4
DI4—4	1	
DI4—2	0	
DI4—1	0	
DI2—8	0	2
DI2—4	0	
DI2—2	1	
DI2—1	0	
DI6—8	0	6
DI6—4	1	
DI6—2	1	
DI6—1	0	
DI8—8	1	8
DI8—4	0	
DI8—2	0	
DI8—1	0	

### BCD-Mode Optional Format (continued)

Signal Name	Logic Level	BCD Character
SGN2	0	+
DI10—8	0	0
DI10—4	0	
DI10—2	0	
DI10—1	0	
DI1—8	0	1
DI1—4	0	
DI1—2	0	
DI1—1	1	
DI5—8	0	5
DI5—4	1	
DI5—2	0	
DI5—1	1	
DI3—8	0	3
DI3—4	0	
DI3—2	1	
DI3—1	1	
DI7-8	0	7
DI7-4	1	
DI7-2	1	
DI7-1	1	
OVL D	1	8
DI9-8	0	1
DI9-4	0	
DI9-2	0	
DI9-1	1	

7

Number\_1 = -4268 Number\_2 = 1.537E+84

Let's further assume that the Peripheral Status Switches are set as follows: DATA=ON, SGN1=ON, SGN2=ON, OVL D=ON; and that the following ENTER statement has been executed (with the BCD Interface as the source):

```
ENTER Bcd;Number_1,Number_2,
```

The computer initiates a handshake with the first peripheral (or device A) by using the  $\overline{\text{CTLA}}$  and  $\overline{\text{CTLB}}$  signals (handshake operation is described in the configuration section). The first peripheral responds to the request by placing data on the lines and then completing the handshake. The states of all data lines from the first device are now stored in registers on the interface (i.e, the data signals are “latched”).

As with Standard format, the Optional-format driver reads the states of the signal lines from the peripheral and generates the appropriate ASCII characters. The computer uses the “number builder” routine of the free-field ENTER statement (described in the chapter “Entering Data”) to enter the ASCII characters from the interface and to generate the internal representation of the number(s) represented by the BCD characters.

In this example, the logic state of SGN1 (1, or True) is read by the driver, which generates a “-” character (see table). The BCD digits DI4, DI2, DI6, and DI8 are read, and corresponding characters are generated. The comma (generated by the driver) terminates entry into the first numeric variable, called `Number_1`. In this case, the value assigned to `Number_1` is -4268.

Since another number has been specified in the ENTER statement, the computer continues to enter characters through the interface until the line-feed is entered. A value of 1.537E+84 is assigned to the variable `Number_2`. The line-feed character (also generated by the driver) terminates both entry of characters into `Number_2` and the ENTER statement. Further examples of entering data through this interface are given in in the main section of this chapter.

## The Binary Data Representation

A binary data representation is available on the HP 98623 BCD Interface. With this representation, the forty data lines (groups DI1 through DI10) are treated as five individual data bytes which can be entered using ENTER or STATUS statement(s).

## The Binary Mode

Unlike the BCD representation, the Binary Mode has no Standard and Optional format; thus, the setting of the Option Format switch has no effect while in the Binary Mode.

To select the Binary Mode, write a non-zero numeric value into BCD Control register 3; the following statement shows a typical method.

**CONTROL 11,3;1**

To see how the ENTER statement enters data through the BCD Interface while in Binary Mode, let's suppose the logic signals on the data lines are as follows.

**Binary Mode ENTER**

Signal Name	Logic Level	Decimal Value	ASCII Character
DI1—8	0	49	1
DI1—4	0		
DI1—2	1		
DI1—1	1		
DI2—8	0		
DI2—4	0		
DI2—2	0		
DI2—1	1		
DI3—8	0	50	2
DI3—4	0		
DI3—2	1		
DI3—1	1		
DI4—8	0		
DI4—4	0		
DI4—2	1		
DI4—1	0		
DI5—8	0	51	3
DI5—4	0		
DI5—2	1		
DI5—1	1		
DI6—8	0		
DI6—4	0		
DI6—2	1		
DI6—1	1		

### Binary Mode ENTER (continued)

Signal Name	Logic Level	Decimal Value	ASCII Character
DI7—8	0	69	E
DI7—4	1		
DI7—2	0		
DI7—1	0		
DI8—8	0		
DI8—4	1		
DI8—2	0		
DI8—1	1		
DI9-8	0	53	5
DI9-4	0		
DI9-2	1		
DI9-1	1		
DI10-8	0		
DI10-4	1		
DI10-2	0		
DI10-1	1		

Let's make the same assumptions that have been made in the previous examples: the logic sense of the data lines is positive-true (the "DATA" switch is set to "ON"). Assume that the following ENTER statement has been executed.

```
ENTER Bcd USING "B";Byte1,Byte2,Byte3,Byte4,Byte5
```

The Control signal line ( $\overline{CTLA}$ ) is placed in the Set state by the computer to signal to the peripheral that a data transfer is to take place. The peripheral responds on the Data Flag line ( $\overline{DFLGA}$ ), completing the handshake and clocking ("latching") the data on the lines into interface registers.

The Binary-Mode driver begins reading the line states as bytes in the order DI1 through DI10; the first byte contains DI1 as the most significant bits and DI2 as the least significant bits. The second byte contains DI3 and DI4, and so forth. In this case, the values 49, 50, 51, 69, and 53 are given to the variables Byte1 through Byte5, respectively.

In this example, the “B” image is used to direct the computer to enter the data on the input signal lines as bytes. A line-feed character is generated by the driver to terminate the ENTER statement.

As another example, suppose that the data on the input lines and the switch settings are as in the preceding example. Let’s look at how the computer would enter the data with the following statement.

```
ENTER Bcd;Number
```

As in the preceding example, the ENTER statement latches the data into the interface registers with the same handshake. The Binary-Mode driver begins reading the line states as bytes in the order DI1 through DI10; the first byte contains DI1 as the most significant bits and DI2 as the least significant bits. The second byte contains DI3 and DI4, and so forth. In this case, the characters “123E5” are entered, followed by a line-feed generated by the driver. In this case, the variable Number receives a value of 1.23E+7.

### Alternate Methods of Entering Data

As with other interfaces, the data signal lines’ logic states can be read with STATUS statements. However, no handshake is performed with this method of entering data.

With the BCD Interface, STATUS registers 5 through 9 contain digits DI1 through DI10, and STATUS register 4 contains SGN1, SGN2, and OVLD. Examples are given in the main section of this chapter.

### Outputting Data

Data may be output through the BCD Interface by using the OUTPUT statement. Data are sent through the eight output lines in byte-serial fashion. The eight lines are called DO-7 through DO-0, in which DO-7 is the most significant bit. Numeric data are sent with the most significant digits first; string data are sent with the lowest-subscripted string characters sent first. Representation depends on whether FORMAT ON or FORMAT OFF is in effect.

Let’s look at how data are output through the BCD Interface with the following OUTPUT statement.

```
OUTPUT 11;"A2C"
```



With OUTPUT, each byte is transferred under control of a handshake which is identical to a corresponding ENTER handshake. The Binary-Mode driver does not send four-bit BCD digits, it sends entire bytes of data; so the driver does not perform any ASCII-to-BCD translation. The items specified in the OUTPUT list are evaluated and sent to the BCD Interface byte-serially. The following diagram shows the logic signals that appear on the Data Output signal lines:

**Data Output**

ASCII Char.	Decimal	DO-7	DO-6	DO-5	DO-4	DO-3	DO-2	DO-1	DO-0
	Value								
A	65	0	1	0	0	0	0	0	1
2	50	0	0	1	1	0	0	1	0
C	67	0	1	0	0	0	0	1	1
<sup>C</sup> R	13	0	0	0	0	1	1	0	1
<sup>L</sup> F	10	0	0	0	0	1	0	1	0

Notice that the free-field convention is used, since the free-field form of the OUTPUT statement was used. The CR-LF (default) EOL sequence is sent after all items have been output. The same data may be sent with the following statement.

```
OUTPUT 11 USING "#,B";65,50,67,13,10
```

Other examples are given in the main section of the chapter.

---

## Configuring the Interface

This section describes the range of or recommended interface's switch settings for use with BASIC language. The switch locations are described in the *HP 98623 BCD Interface Installation Note*.

## Determining Interface Configuration

If the interface is already installed in a computer which currently has the BASIC-language system resident, you can determine the configuration by running the following program. If the interface is not yet installed, you may want to check the switch settings as you read this section to see that they are set for use with your particular application.

```
100  PRINTER IS 1
110  PRINT CHR$(12)    ! Clear screen.
120  !
130  DISP "Enter select code of BCD Interface."
140  ENTER 2;Isc
150  DISP
160  !
170  ON ERROR GOTO Skip_status ! Skip if bad isc.
180  STATUS Isc;Id
190  Skip_status:  OFF ERROR
200  !
210  PRINT "The Interface at select code ";Isc;
220  IF Id=4 THEN
230      PRINT "is a BCD Interface."
240  ELSE
250      PRINT "is NOT a BCD Interface."
260      PRINT "Program terminated."
270      STOP
280  END IF
290  PRINT
300  !
310  CONTROL Isc;1      ! Reset interface.
320  !
330  STATUS Isc,1;Intr_status
340  Mask=2^5+2^4      ! Mask out all but bits 5 and 4.
350  Bits_set=BINAND(Intr_status,Mask)
360  Hd_prior=(Bits_set MOD 16)+3 ! Shift Rt. and add 3.
370  PRINT "Hardware priority (Interrupt Level) is ";Hd_prior;". "
380  PRINT
390  !
400  STATUS Isc,3;Binary_mode
410  IF Binary_mode THEN
420      PRINT "Binary mode selected."
430  ELSE
440      STATUS Isc,4;Switches
450      IF BIT(Switches,7)=1 THEN
```

```

460         PRINT "BCD mode, Optional format selected (2 devices).\"
470     ELSE
480         PRINT "BCD mode, Standard format selected (1 device).\"
490     END IF
500 END IF
510 PRINT
520 !
530 PRINT "Logic sense of signals:\"
540 IF BIT(Switches,6)=1 THEN
550     PRINT "    Input data: Low=1, High=0.\"
560 ELSE
570     PRINT "    Input data: Low=0, High=1.\"
580 END IF
590 !
600 IF BIT(Switches,5)=1 THEN
610     PRINT "    SGN1: High=""+""", Low=""-""\"
620 ELSE
630     PRINT "    SGN1: High=""-""", Low=""+""\"
640 END IF
650 !
660 IF BIT(Switches,4)=1 THEN
670     PRINT "    SGN2: High=""+""", Low=""-""\"
680 ELSE
690     PRINT "    SGN2: High=""-""", Low=""+""\"
700 END IF
710 !
720 IF BIT(Switches,3)=1 THEN
730     PRINT "    OVLD: High=0, Low=8.\"
740 ELSE
750     PRINT "    OVLD: High=8, Low=0.\"
760 END IF
770 PRINT
780 !
790 END

```

## Setting the Interface Select Code

The interface's select code setting determines the value of the interface select code parameter in which is used in ENTER and OUTPUT statements to specify the interface through which data is to be sent. The allowable range is 8 through 31, since internal interfaces already use select codes 1 through 7. Keep in mind that *no two interfaces should be set to the same select code*.

The default select code is 11. If a different select code is desired, set the switches as described in the installation note.

## Setting the Hardware Priority (Interrupt Level)

The hardware priority assigned to an interface determines the order in which the interrupts from the interface are logged by the system. The software priority of interrupts determines the order of interrupt service, which is independent of this hardware priority.

A default setting of 3 is generally used. See the installation note for switch location and settings.

## Setting the Peripheral Status Switches

The peripheral status switches are used to select the format of BCD data and the logic sense of data input lines. The  $\overline{OF}$  switch selects between the Optional BCD format and the Standard BCD format. Set the switch to ON (default) if Standard is desired, or to OFF if Optional format is desired. The setting of this switch is irrelevant if the interface is only to be used in the Binary mode.

The **DATA** switch determines the logic sense of all 40 data input lines. If set to ON, positive-true logic is used (logic high is a 1). If set to OFF, negative-true logic is used; (logic low is a 1).

The **SGN1** and **SGN2** switches determine the logic sense of the respective sign-bit signal lines. If set to ON, a logic high signifies a “-” and logic low signifies a “+”. If set to OFF, a logic high signifies a “+” and logic low signifies a “-”.

The **OVLD** switch determines the logic sense of the OVLD signal line. If set to ON, a logic high is entered as an “8” and a low is entered as a “0”. If set to OFF, a logic high is entered as a “0” and low is an “8”.

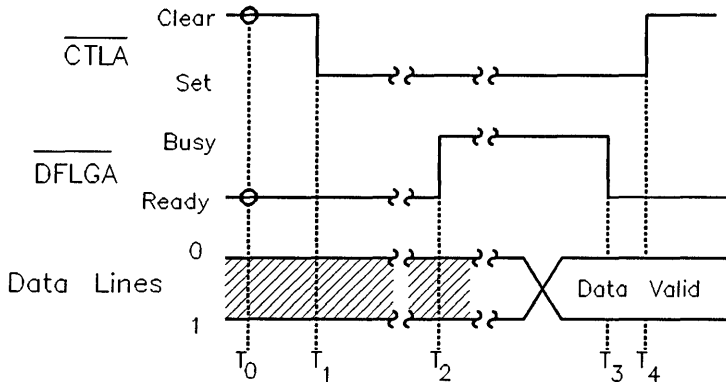
## Setting the Handshake Configuration

The handshake used by the BCD Interface is a two-wire handshake that synchronizes the exchange of data in one of two general manners: Type 1 timing or Type 2 timing. Type 1 and Type 2 timing differ in when the peripheral's data are clocked (latched) into the interface's data registers.

The logic sense of both the Control lines from the computer ( $\overline{CTLA}$  and  $\overline{CTLB}$ ) and Data Flag lines from the peripheral ( $\overline{DFLGA}$  and  $\overline{DFLGB}$ ) are switch-selectable.

### Type 1 Timing

With Type 1 handshake timing, the Busy-to-Ready transition of the peripheral's data flag line ( $\overline{DFLGA}$  or  $\overline{DFLGB}$ ) Clears the Control line ( $\overline{CTLA}$  or  $\overline{CTLB}$ ) from the computer and clocks the data into the interface's Data In registers. The following timing diagram shows an example of how this sequence of events takes place. Note that the  $\overline{CTLA}$  and  $\overline{DFLGA}$  switches are set to OFF (Low-true).



**Type 1 Handshake Timing Diagram**

At time  $t_0$ ,  $\overline{CTLA}$  is Clear and  $\overline{DFLGA}$  is Ready, indicating that a transfer may be initiated. At time  $t_1$ , the computer initiates the handshake. At  $t_2$ , the peripheral responds by placing  $\overline{DFLGA}$  Busy. The peripheral then places the data on the data lines. When data have settled, the peripheral completes the handshake by placing  $\overline{DFLGA}$  Ready, which also Clears  $\overline{CTLA}$  and clocks the data into the interface registers (at time  $t_4$ ). Another handshake cycle may then be initiated by the computer.

---

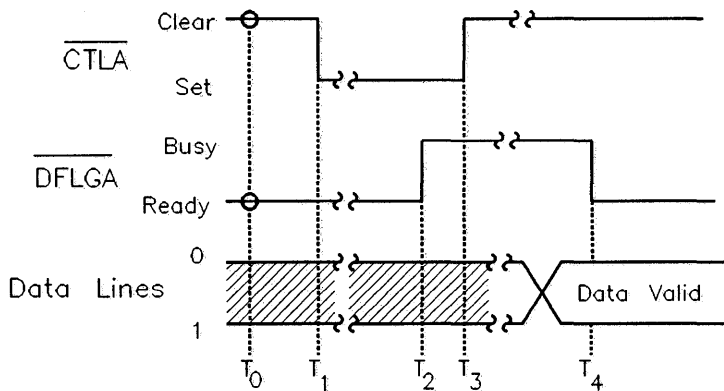
**Note**

If only one peripheral is connected to the interface, connect the  $\overline{CTLB}$  line to the  $\overline{DFLGB}$  line and set both CTLB and DFLGB switches to the OFF positions. If this is not done, the handshake cannot be completed.

---

**Type 2 Timing**

With Type 2 handshake timing, the Ready-to-Busy transition of the peripheral's data flag line ( $\overline{DFLGA}$  or  $\overline{DFLGB}$ ) Clears the Control line from the computer; however, the Busy-to-Ready transition still clocks the data into the interface's Data In registers. The following timing diagram shows an example of how this sequence of events takes place. Note that the CTLA and DFLGA switches are set to OFF (Low-true).



**Type 2 Handshake Timing Diagram**

At time t0,  $\overline{CTLA}$  is Clear and  $\overline{DFLGA}$  is Ready, indicating that a transfer may be initiated. At time t1, the computer initiates the handshake. At t2, the peripheral responds by placing  $\overline{DFLGA}$  Busy, which also Clears  $\overline{CTLA}$ . When ready, the peripheral places  $\overline{DFLGA}$  Ready (at time t4), which also clocks the data into the interface registers. Another handshake cycle may then be initiated by the computer.

---

**Note**

If only one peripheral is connected to the interface, connect the CTLB line to the DFLGB line and set both CTLB and DFLGB switches to the OFF positions. If this is not done, the handshake cannot be completed.

---

## Configuring the Cable

The installation note describes how to connect the cable wires. Any unused lines should be connected as follows: connect the line to the “+5 Ref” signal line if the line is to be read as high, or to logic ground if the line is to be read as low. With lines such as SGN1, SGN2, and OVLD the line may be tied either to ground or to +5V, because the logic-sense switch allows either sense to be selected independent of other signals.

---

**Note**

Be sure to follow the recommendations in the installation note **exactly** to ensure signal integrity and operator safety.

---

---

## Interface Reset

The interface should always be reset to ensure that it will be in a known state before use. All interfaces are automatically reset by the computer at certain times: when the computer is powered on, when the **RESET** (**Shift-Reset** on an ITF keyboard) key is pressed, and at other times described in the Reset Table (in the Useful Tables). The interface may also be reset by BASIC programs, as in the following examples.

```
Bcd=11
CONTROL Bcd;1
```

```
Reset_value=1
CONTROL Bcd,0;Reset_value
```

```
RESET Bcd
```

The following action is take when the BCD Interface is reset:

- The peripheral reset signal line ( $\overline{\text{PRESET}}$ ) is pulsed low for at least 15 microseconds.
- The  $\overline{\text{CTLA}}$  and  $\overline{\text{CTLB}}$  handshake lines are Cleared.
- The Data Out register is cleared (set to all 0's).
- The Interrupt Enable bit is cleared, disabling subsequent interrupts until re-enabled by the program, and the Interrupt Request bit is set.

The state of the BCD/Binary Mode register (STATUS and CONTROL Register 3) is unchanged by the Interface Reset.

---

## Entering Data Through the BCD Interface

This section describes BASIC programming techniques useful for entering data through the BCD Interface. Several examples of entering data were given in the first section to show how the interface works in BCD Mode with Standard and Optional formats and Binary Mode. This section gives additional general techniques for entering data from peripheral devices. If you need further explanation of how the ENTER statement works, refer to the chapter "Entering Data"; the chapter entitled "Registers" discusses the STATUS statement.

7

The diagrams and corresponding BASIC-language statements in this section show how data on the interface signal lines get read by the ENTER statement and corresponding values assigned to BASIC-language variable(s). The notation used in the examples is that the name of the interface signal line (or group of lines) is shown above the decimal value and ASCII character that the driver produced by reading the line(s). The logic sense of the lines is not shown here; see the preceding configuration section for a description of selecting the logic sense of the interface signals.

As an example, the following drawing shows that an ASCII "+" was generated by the driver when it read the SGN1 signal line; similarly, the four signals of the group DI5 produced a period character. The driver produces the "E" (exponent), comma, and line-feed characters automatically.



SGN1	DI1	DI2	DI3	DI4	DI5	DI6	DI7	DI8	SGN2	DI9	OVLD	DI10			
+	1	2	3	4	.	6	7	8	E	-	6	,	0	4	LF

The following statements show how the preceding data might be entered and the resultant values assigned to program variables.

```
Bcd=11
ENTER Bcd;Number,Function
PRINT "Number= ";Number
PRINT "Function= ";Function
```

The following display is the result of executing the preceding statements.

```
Number= 1.234678E-3
Function= 4
```

## Entering Data from One Peripheral

There are several methods of entering data through the BCD Interface when only one peripheral device is connected. The Standard BCD format can be used with many devices; the Binary mode must be used with others, and some require that you write your own “drivers.”

### Entering with BCD-Mode Standard Format

Using the Standard format of BCD mode usually provides the most convenient means of entering data from one device. This format allows up to 8 BCD digits for mantissa and one BCD digit for exponent. The state of an overload-indicator signal and one optional BCD digit can also be entered, if desired.

7

SGN1	DI1	DI2	DI3	DI4	DI5	DI6	DI7	DI8	SGN2	DI9	OVLD	DI10			
+	0	.	3	4	5	6	0	0	E	+	4	,	8	4	LF

```
100 ENTER 11;Number,Function
110 PRINT "Number= ";Number
120 IF Function>=80 THEN
```

```
130 PRINT "Overload of function ";Function-80
140 ELSE
150 PRINT "Function= ";Function
160 END IF
```

The following results would be printed by the preceding program segment:

```
Number= 3456
Overload of function 4
```

The ENTER statement calls the Standard-format driver, which reads the BCD characters on the interface lines in the order shown and generates the appropriate ASCII characters. Characters are entered until the “,” is read, which terminates entry into the variable `Number`. The characters after the comma are used to build the value of `Function`. The ENTER statement is properly terminated when the line-feed (an ENTER-statement terminator) is encountered.

Notice that an “8” is generated by the driver when the OVLD line is true. The BASIC program must “separate” this from the “function” digit (DI10). The method shown in the example is only one of many methods available.

If a second variable would not have been included in the preceding ENTER statement, ENTER would have continued asking the driver for characters until it encountered the line-feed, which terminates the statement.

To contrast the preceding example, suppose that the following statement has been executed:

```
ENTER 11 USING "#,K";Number
```

7

In this case, the # specifier directs the ENTER statement to suppress its default requirement of looking for a line-feed character (or other statement-termination condition) to terminate the ENTER. Thus, the comma terminates both entry of data into `Number` and the ENTER statement. Consequently, a subsequent ENTER statement would begin entering characters beginning with the “8” character (OVLD), which may not be the desired action.

In such a case, several remedies are possible. The simplest is probably to go ahead and include a second variable so that the driver is left pointing to the first character after the ENTER is completed. The second variable is thus used for a “dummy” read operation. Another remedy is to write a non-zero value to BCD CONTROL register 1, which “resets” the driver pointer to the first

character of the format (SGN1). Executing the following statement performs the driver reset.

```
CONTROL 11,1;1
```

This type of “problem” may also occur when the BCD device sends a line-feed as one of the BCD characters.

SGN1	DI1	DI2	DI3	DI4	DI5	DI6	DI7	DI8	SGN2	DI9	OVLD	DI10
-	1	2	LF	4	5	6	7	LF	E	+	0	, 0 0 LF

In such case, two numbers are sent separated by line-feeds. The following statements would read these two numeric values and then reset the driver pointer to the first character (the SGN1 character).

```
ENTER 11;Number_1,Number_2
CONTROL 11,1;1
```

If the CONTROL statement had not been executed, the driver would have been left pointing to the “E” character.

As another example, suppose the exponent is to be ignored but the overload and function digits are to be read. The following statement would be appropriate in such a situation.

```
ENTER 11;Number_1,Number_2,Dummy,Function
```

The variable Dummy is so named to show that it is included in the ENTER statement only to ensure that the overload and function digits are read and assigned to a variable (i.e., it is not used for any other purposes). Of course, the value could be used if desired.

If your application requires reading only certain characters or groups of characters, you may want to read the chapter “Entering Data” to see more examples of using images with ENTER statements.

### Entering with the Binary Mode

If your application represents data with eight-bit ASCII characters or has a data format that is not compatible with the Standard BCD format, the Binary Mode can be used. With the Binary Mode, data are entered in groups of eight

bits each, rather than in groups of four-bit BCD digits. Five bytes are latched with each handshake; the driver reads the bytes sequentially until the fifth byte is read, after which it sends a line-feed character to terminate the ENTER. Another handshake operation is required if more data are to be entered.

As an example, let's assume that the following logic signals are present on the interface lines. Only 16 signals are shown here because that is all that we will be using for this example.

### Sixteen Signals

Signal Name	Logic Level	Decimal Value	ASCII Character
DI1—8	0	65	A
DI1—4	1		
DI1—2	0		
DI1—1	0		
DI2—8	0		
DI2—4	0		
DI2—2	0		
DI2—1	1		
DI3—8	0	49	1
DI3—4	0		
DI3—2	1		
DI3—1	1		
DI4—8	0		
DI4—4	0		
DI4—2	0		
DI4—1	1		

Assume also that the I/O path name "@Bcd" is assigned to the select code of a BCD Interface. The following ENTER statement enters these two bytes of data as numbers in the range 0 through 255.

```
ENTER @Bcd USING "B";Di1_di2,Di3_di4
```

The "B" specifier directs the computer to enter one byte of data from the interface and place it into the corresponding numeric variable, which happens two times in this case. The variables Di1\_di2 and Di3\_di4 receive values of 65 and 49, respectively. The ENTER statement continues to request characters

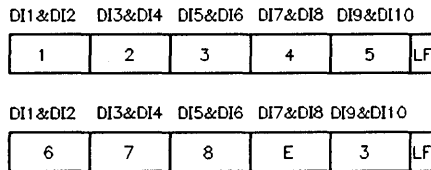
from the Binary-Mode driver until a line-feed (generated by the driver) is returned, which terminates the ENTER statement. Even though only two bytes were used to fill variables in this example, all five bytes of data were read from the interface.

The following statement could be used to enter the two bytes as one 16-bit word.

```
ENTER @Bcd USING "W";Word1
```

The variable "Word1" receives a value of 16 689 (256\*65 + 49).

As another example, suppose that the data on the lines are to be interpreted as ASCII characters. The following diagrams show the ASCII representations of data read from the interface; entering ten bytes of data in this mode requires two handshake cycles.



The following ENTER statement enters characters until an item terminator is found and then calls the "number builder" routine to construct the number; this sequence is performed for each numeric variable in the statement.

```
ENTER Bcd;Number_1,Number_2
```

In this case, Number\_1 is assigned a value of 12345, and Number\_2 is assigned 6.78E+5. With a Binary-Mode ENTER, the driver does not read SGN1, SGN2, or OVLD, and does not insert any E's, or commas; only a line-feed is generated as a sixth character to terminate the ENTER statement.

If your application has a data format that is not compatible with the Standard BCD format, the Binary Mode can be used in conjunction with a routine of your own design that is tailored for your application's requirements. Let's look at an example of how this might be accomplished.

Suppose that your peripheral requires five digits of mantissa but must have three exponent digits and two function digits. A program will be used to read the data using the desired format. If the peripheral's handshake method is compatible with one of the handshake types available on the BCD Interface, the Binary mode may be used to enter the data; if not, see the example of implementing a handshake in the next section.

For this example, suppose the following conditions exist: the mantissa is entered from DI1 through DI5, the exponent is entered from DI6 through DI8, and function is entered from DI9 and DI10. The mantissa and exponent signs and the overload indicator are still available as individual signal lines, but they must be read with the STATUS statement.

The subroutine shown in the following program reads the data on the lines with ENTER and STATUS statements and then formats the data as required for the application. The formatted information is then entered from a string variable into the desired numeric variables.

```

100 ! This program executes a subroutine which enters data from the
110 ! BCD Interface using Binary mode and formats it as follows:
120 !
130 ! SGN1 DI1 DI2 DI3 DI4 DI5 E SGN2 DI6 DI7 DI8 ,
140 ! OVLD DI9 DI10 LF
150 !
160 ! Define ordering of BCD characters.
170 Bcd_chars$="0123456789+,-E"
180 !
190 Bcd=11
200 CONTROL Bcd,3;1 ! Set Binary mode.
210 !
220 GOSUB New_format ! Execute subroutine.
230 ENTER Format$;Number,Function ! Use results for ENTER.
240 PRINT "Number=";Number
250 PRINT "Function=";Function
260 !
270 STOP
280 !
290 New_format: ! ***** Beginning of Subroutine. *****
300 !
310 ! Perform a Binary-mode ENTER.
320 ENTER Bcd USING "5A";Bytes$ ! 5 bytes read.
330 !
340 ! Use STATUS to read SGN1, SGN2, OVLD.
350 STATUS Bcd,4;Sgns_and_ovld

```

```

360  !
370  ! Generate two ASCII characters from each byte.
380  FOR Byte=1 TO 5
390    !
400    ! Get numeric value of single byte from Bytes$.
410    Char=NUM(Bytes$[Byte])
420    !
430    ! Upper 4 bits form first ASCII char.
440    Up_4_bits=Char DIV 16 ! Shift right 4 places.
450    ! Use numeric value as index into Bcd_chars$.
460    First_char$=Bcd_chars$[Up_4_bits+1;1]
470    !
480    ! Lower 4 bits form 2nd ASCII char.
490    Lo_4_bits=Char MOD 16 ! Mask upper 4 bits.
500    ! Use numeric value as index into Bcd_chars$.
510    Second_char$=Bcd_chars$[Lo_4_bits+1;1]
520    !
530    ! Now append characters onto format string.
540    Digits$[2*Byte-1]=First_char$&Second_char$
550    !
560  NEXT Byte
570  !
580  !
590  ! Calc. SGN1's and SGN2's ASCII representations.
600  Sgn1=BIT(Sgns_and_ovld,2)
610  Sgn1$=CHR$(43+2*Sgn1) ! "+" if Lo; "-" if Hi.
620  Sgn2=BIT(Sgns_and_ovld,1)
630  Sgn2$=CHR$(43+2*Sgn2) ! "+" if Lo; "-" if Hi.
640  !
650  ! Calc. Overload's ASCII representation.
660  Ovld=BIT(Sgns_and_ovld,0)
670  Ovld$=CHR$(48+8*Ovld) ! "0" if Lo; "8" if Hi.
680  !
690  Number$=Sgn1$&Digits$[1,5]&"E"&Sgn2$&Digits$[6,8]
700  Function$=Ovld$&Digits$[9,10]
710  Format$=Number$&"", "&Function$&""
720  !
730  RETURN ! ***** End of Subroutine. *****
740  !
750  END

```

## Entering with STATUS Statements

The preceding examples assumed that the handshake options available with the BCD Interface are compatible with your peripheral. This section shows

examples of designing enter operations using STATUS and CONTROL statements to implement your own handshakes.

```
100 Bcd=11
110 CALL Enter_bytes(Bcd,Bytes$)
120 PRINT Bytes$
130 STOP
140 !
150 END
160 !
170 SUB Enter_bytes(Isc,Return_string$)
180 !
190 CONTROL Isc,2;1 ! Initiate handshake.
200 !
210 Check: STATUS Isc,1;Intr_stat
220 Irq=BIT(Intr_stat,6)
230 IF NOT Irq THEN Check ! Wait for response.
240 !
250 ! Now read bytes in registers 5 -> 9.
260 STATUS 11,5;R5,R6,R7,R8,R9
270 !
280 ! Return bytes as a string.
290 Return_string$=CHR$(R5)&CHR$(R6)&CHR$(R7)
300 Return_string$=Return_string$&CHR$(R8)&CHR$(R9)
310 !
320 SUBEND
```

7

Note that the program uses the Interrupt Request bit (bit 6 of register 1) to determine when the handshake is completed by the peripheral. This bit is cleared (0) when a Request is performed (i.e., when the handshake is initiated by writing a non-zero value to CONTROL register 2). The bit is set when the peripheral acknowledges the Control ( $\overline{\text{CTLA/B}}$ ) signal by responding with Data Flag ( $\overline{\text{DFLGA/B}}$ ). The acknowledgement occurs when the Control line is Cleared by the leading edge of Data Flag (Type 2 timing) or by its trailing edge (Type 1 timing).

The transfer of data can also be implemented with interrupts, which is described in the BCD Interrupts section.



## Entering Data from Two Peripherals

Data can be entered from two devices by using either BCD-mode Optional format or by using STATUS statements. Optional format allows up to 4 BCD digits from the first peripheral and up to 5 BCD digits from the second peripheral; overload from either device may also be detected. Data from each device is handshaked independently.

### Optional Format

This section describes how to use the Optional format with BASIC programs. In order to use this format, the peripheral's handshake convention must be compatible with one of the handshake options available on the BCD Interface. Since the preceding section described how to implement handshake routines with STATUS and CONTROL statements, you should refer to that discussion if your application requires that type of solution.

With the BCD-Mode Optional format, the data, sign, and overload signals are read and formatted into ASCII characters in the following sequence:

SGN1	DI4	DI2	DI6	DI8		SGN2	DI10	DI1	DI5	DI3	DI7		OVLD	DI9	
+	4	2	6	8	,	-	0	1	5	3	7	E	0	0	LF

The following program segment shows an example of how these characters might be entered and stored in variables.

```
100 ENTER 11;Number_1,Number_2
110 PRINT "Number 1= ";Number_1
120 PRINT "Number 2= ";Number_2
```

The following results would be printed by the preceding program segment:

```
Number 1= 4268
Number 2= -1537
```

The ENTER statement calls the Optional-format driver, which reads the signals on the interface lines in the order shown and generates the appropriate ASCII characters. Characters are entered and sent to the "number builder" until the "," is read, which terminates entry into the variable Number\_1; the internal representation of the numeric value is then generated. The characters

after the comma are used to build the value of `Number_2`. The `ENTER` statement is properly terminated when the line-feed (an `ENTER`-statement terminator) is encountered.

If a second variable would not have been included in the preceding `ENTER` statement, `ENTER` would have continued asking the driver for characters until it encountered the line-feed, which terminates the statement.

It is important to note that an "8" is generated by the driver when the `OVL`D line is true or when *any* of the bits of `DI9` are true, making the possibilities of exponent values 0, 8, 80, or 88; consequently, the `BASIC` program must "separate" these overload indicators.

Separating overload information from the mantissa may be a problem when one number can be represented in two ways; for instance, ".0001E8" and "10000" both represent the value "1.0E+4", but the two representations have entirely different meanings. The first representation indicates an overload on the second device, while the second value does not.

To solve this potential problem, the second number can be entered into a string variable, as shown in the following segment.

```
100  ENTER 11;Number_1,Number_2$
110  !
120  ! Separate 2nd mantissa and exponent.
130  Exponent$=Number_2$[8,9]
140  Number_2$=Number_2$[1,6]
150  !
160  ! Place 2nd mantissa in numeric variable.
170  ENTER Number_2$;Number_2
180  !
190  PRINT "Number 1=";Number_1
200  ! Check overload information.
210  IF Exponent$[1;1]="8" THEN
220      PRINT "Overload on device 1."
230      PRINT
240  END IF
250  !
260  PRINT "Number 2=";Number_2
270  ! Check overload information.
280  IF Exponent$[2]="8" THEN
290      PRINT "Overload on device 2."
300      PRINT
310  END IF
```

```

320 PRINT
330 !
340 END

```

The program checks the exponent digits separately and indicates an overflow on either device.

To contrast the preceding examples, suppose that the following statement has been executed:

```
ENTER 11 USING "#,K";Number
```

In this case, the # specifier directs the ENTER statement to suppress its default requirement of looking for a line-feed character (or other statement-termination condition) to terminate the ENTER. Thus, the comma terminates both entry of data into Number and the ENTER statement. Consequently, a subsequent ENTER statement would begin entering characters beginning with the character following the comma (i.e., the first character of the second number), which may not be the desired action.

In such a case, several remedies are possible. The simplest is probably to go ahead and include a second variable so that the driver is left pointing to the first character after the ENTER is completed. The second variable is thus used for a “dummy” read operation. Another remedy is to write a non-zero value to BCD CONTROL register 1, which “resets” the driver pointer to the first character of the format (SGN1). Executing the following statement performs the driver reset.

```
CONTROL 11,1;1
```

This type of situation may also occur when the BCD device sends a line-feed as one of the BCD characters.

SGN1	DI4	DI2	DI6	DI8	SGN2	DI10	DI1	DI5	DI3	DI7	DI9	OVLD			
+	1	2	3	LF	,	+	0	1	5	3	LF	E	0	0	LF

In such case, two numbers are sent separated by line-feeds. The following statements would read these two numeric values and then reset the driver pointer to the first character (the SGN1 character).

```
ENTER 11;Number_1,Number_2
```

```
CONTROL 11,1;1
```

If the CONTROL statement had not been executed, the driver would have been left pointing to the "E" character.

---

## Outputting Data Through the BCD Interface

All data outputs through the BCD Interface are made through the eight output lines. There are two general methods of sending data to devices through the BCD Interface—by using CONTROL statements and by using OUTPUT statements. With CONTROL statements, the data are latched on the output lines, but the handshake (if desired) must be performed with STATUS and CONTROL statements. With the OUTPUT statement, each data byte is sent individually under handshake control. With both methods, neither the setting of the Optional Format switch nor the current Mode (BCD or Binary) has any effect on how data are output through the interface.

### Output Routines Using CONTROL and STATUS

Many applications do not require that data be sent with a handshake operation. In such cases, the following example shows how one byte of data may be sent to the peripheral.

```
100 Byte=2^6+2^4      ! Set Bits 6 and 4.
110 CONTROL 12,4;Byte ! Send data w/o handshake.
```

- 7** If your application requires a handshake which is not compatible with the handshake options available on the BCD Interface, you can program your own. The following program shows an example handshake. The transition of the Data Flag signal that Clears the Control signals is still determined by the setting of the CTLA-2 and CTLB-2 switches. See the configuration section for further details.

```
100 Bcd=11
110 Chars$="1A2B"
120 Eol$=CHR$(10) ! LF is EOL sequence.
130 CALL Output_bcd(Bcd,Chars$,Eol$)
140 !
150 END
160 !
```

```

170 SUB Output_bcd(Isc,Characters$,Eol$)
180 !
190 Output_data$=Characters$&Eol$
200 FOR I=1 TO LEN(Output_data$)
210 CONTROL Isc,2;1 ! Initiate handshake.
220 !
230 ! Now output byte(s) to registers 4.
240 CONTROL Isc,4;NUM(Output_data$[I;1])
250 !
260 ! See if Ready for next byte.
270 Check: STATUS Isc,1;Intr_stat
280 Irq=BIT(Intr_stat,6)
290 IF NOT Irq THEN Check ! Wait for response.
300 !
310 NEXT I
320 !
330 SUBEND

```

The data are output on the Data Output lines in byte-serial fashion. The program uses the Interrupt Request indicator (Bit 6 of STATUS Register 1) to indicate the interface's and peripheral's joint readiness for a subsequent handshake operation. Interrupts can also be used; for more details, see the discussion of BCD Interrupts.

## Sending Data with OUTPUT

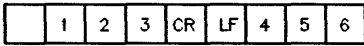
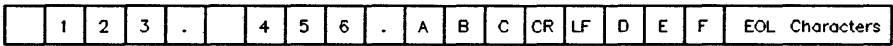
With the OUTPUT statement, data are output byte-serially, one byte per handshake cycle. The following program shows an example of outputting data through the BCD Interface.

```

100 Bcd=11
110 OUTPUT Bcd;123,456,"ABC","DEF"
120 OUTPUT Bcd;123,456;"ABC";"DEF"
130 OUTPUT Bcd;"123","456";
140 !
150 END

```

The following diagram shows the sequence of ASCII characters sent to the destination device with the preceding program. The notation indicates that each ASCII character is sent through the output lines DO-7 through DO-0.



Notice that when a comma follows an output item in a free-field OUTPUT statement, a numeric item in the output data is terminated by a comma and a string item is terminated by a CR/LF sequence (one carriage-return and one line-feed character). If an item is followed by a semicolon, no item terminator is sent. If an item is the last one in the output list, an end-of-line (EOL) sequence is sent instead of the item terminator; the default EOL sequence is a CR/LF with no time delay. Changing the EOL sequence is described in the chapter "Outputting Data".

In the preceding program, the FORMAT ON attribute was in effect so the ASCII representation of each data item was generated and sent to the peripheral device. It is also possible to OUTPUT with FORMAT OFF in effect by using I/O path names. See Chapter 10 for further details.

It is interesting to note that all handshake cycles latch *both* input and output data. In the following example, an OUTPUT statement is used to place one byte on the Data Out lines under handshake control. A STATUS statement is then used to read the Data In lines, since the handshake operation also latched the data on the input lines into STATUS Registers.

7

```

100  Byte=64+32                ! Set bits 6 and 5.
110  OUTPUT 11 USING "#,B";Byte ! Handshake byte 1 out.
120  ! Now read SGN1, SGN2, OVLD, and DI1 thru DI10.
130  STATUS 11,4;Reg4,Reg5,Reg6,Reg7,Reg8,Reg9
140  Sgn1=BIT(Reg4,2)
150  Sgn2=BIT(Reg4,1)
160  Ovld=BIT(Reg4,0)
170  Di1=Reg5 DIV 16
180  Di2=Reg5 MOD 16
190  Di3=Reg6 DIV 16
200  Di4=Reg6 MOD 16

```

.  
.  
.

The program determines the states of the sign, overload, and data lines. The data may then be formatted as desired.

---

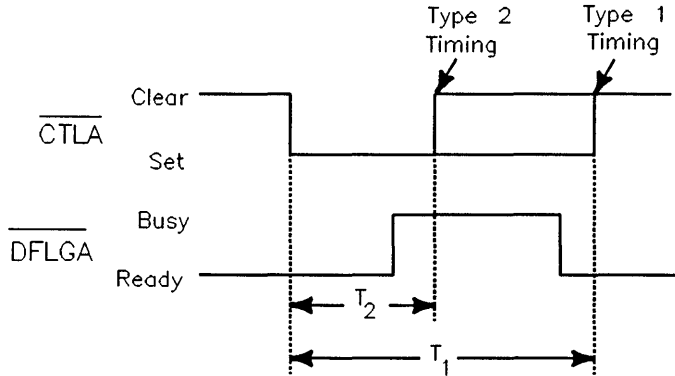
## BCD Interface Timeouts

When a peripheral device does not respond to a handshake request from the computer, it is convenient to be able to sense this condition and respond accordingly. Using the ON TIMEOUT statement sets up and enables a branch which will be initiated when the computer determines that the interface has taken too much time to respond.

Timeout events were generally discussed in the chapter "Interface Events". However, specific details such as the effects of the TIMEOUT event's occurrence on each interface and how the time parameter is measured were not described. This section describes such topics.

## Timeout Time Parameter

When an ON TIMEOUT is set up for an interface, the time required to complete each handshake is measured and compared to the time specified in the ON TIMEOUT statement. The interval measured is shown in the following diagram.



**Measuring the BCD Interface's TIMEOUT Parameter**

Timing begins when the CTLA (or CTLB) signal is placed in the Set state to initiate a handshake cycle. The computer continues to check the time elapsed against the specified time (TIMEOUT time parameter). Timing ends when the peripheral has completed its response; with both Type 1 and Type 2 timing, this occurs *only* when the Control line is cleared *and* the Data Flag line is placed in the Ready state by the peripheral.

7

## Timeout Service Routines

When a TIMEOUT occurs, the computer automatically executes an **Interface Reset**. The Peripheral Reset line to the peripheral (Preset) is pulsed low for at least 15 microseconds, and CTLA and CTLB are then Cleared. This action should “get the peripheral’s attention”, if it is functional. The service routine should then take the appropriate corrective action. See a previous section called “Interface Reset” for further effects of the reset.

Timeout service routines generally determine whether or not the peripheral is still functional. If so, the computer may take corrective action such as to



re-initiate the preceding transfer. If not, perhaps the program may inform the operator of the condition and then proceed.

The following program shows an example of setting up a branch to a service routine upon detecting a TIMEOUT on the BCD Interface. When a TIMEOUT occurs while trying to send the first message, the service routine attempts to send an escape character to the peripheral, which here is a request for status of our fictitious peripheral. If the peripheral does not respond, the destination of data is changed to the CRT.

```
100 Bcd=11 ! Interface select code of BCD.
110 Dest=Bcd ! Destination is device is BCD.
120 ON TIMEOUT Bcd,2 GOSUB Try_bcd_again
130 !
140 Message$="This sent to BCD."
150 OUTPUT Dest;Message$
160 ! If TIMEOUT, this line is executed upon RETURN.
170 !
180 ! All subsequent data sent to Dest=CRT (if TIMEOUT).
190 Message$="This sent to CRT."
200 OUTPUT Dest;Message$
210 !
220 STOP
230 !
240 Try_bcd_again: ON TIMEOUT Bcd,3 GOTO Forget_it
250 !
260 ! See if escape character is accepted.
270 OUTPUT Bcd USING "#,B";27
280 ! If accepted, then 2nd TIMEOUT didn't occur;
290 ! so this segment might contain a routine
300 ! that interrogates a peripheral.
310 !
320 ON TIMEOUT Bcd,3 GOSUB Try_bcd_again
330 GOTO Exit_point
340 !
350 Forget_it: PRINT "BCD Down; Data will be sent to CRT."
360 PRINT
370 Dest=1
380 BEEP
390 OFF TIMEOUT Bcd ! No longer need active TIMEOUT.
400 !
410 Exit_point: RETURN ! to line following TIMEOUT's occurrence.
420 !
430 END
```

The timeout service routine may be programmed to attempt to continue the transfer where it timed out; however, this action may be difficult to implement for two reasons: the computer may not be keeping track of where in the transfer the TIMEOUT occurred, and the automatic Interface Reset performed when the TIMEOUT occurred may have also reset the peripheral. How your program implements the transfer and how the peripheral responds to the reset will determine the feasibility of continuing the transfer.

---

## BCD Interface Interrupts

The BCD Interface can detect one type of interrupt condition: an interrupt can be generated when the interface is Ready for a subsequent data transfer, which generally occurs after the program initiates a handshake and the peripheral completes it.

### Setting Up and Enabling Interrupts

When an event occurs, the event is logged by the BASIC operating system. After the event is logged, any further interrupts from the interface are disabled until specifically re-enabled by a program. All further computer responses to the event depend entirely on the program.

The following segment shows a typical sequence of setting up and enabling a BCD interrupt to initiate its branch.

7

```
100 ON INTR 11 GOSUB Bcd_intr
110 Mask=1
120 ENABLE INTR 11;Mask
```

The value of the interrupt mask (**Mask** in the program) determines whether the interrupt is to be enabled or disabled. In this case, any non-zero value enables the Ready interrupt.

## Interrupt Service Routines

Since there is only one type of interrupt possible with the BCD Interface, the service routine does not need to determine the interrupt cause. In general, all the service routine needs to do is to determine whether another data item is to be transferred or the transfer is to be terminated. The following segment is a typical interrupt service routine.

```
100 Bcd=11
110 ON INTR Bcd GOSUB Get_bytes
120 !
130 CONTROL Bcd,2;1 ! Initiate 1st handshake.
140 ENABLE INTR Bcd;1 ! Enable Ready Interrupts.
150 !
160 ! Execute background routine.
170 WHILE Iteration<1.E+6
180     Iteration=Iteration+1
190     DISP Iteration
200 END WHILE
210 !
220 Get_bytes: !
230     STATUS Bcd,5;Reg5,Reg6,Reg7,Reg8,Reg9
240     PRINT Reg5,Reg6,Reg7,Reg8,Reg9
250     CONTROL Bcd,2;1 ! Initiate next handshake.
260     ENABLE INTR Bcd ! Re-enable (use same Mask).
270     RETURN
280     !
290 END
```

The main program sets up the branch location, initiates the first data-transfer handshake, and then enables the interface to interrupt when it is Ready; the peripheral is Ready when it has cleared the Control line(s) and placed the Data Flag line(s) in the Ready state.

The service routine reads the data on lines DI1 through DI10, initiates the subsequent handshake, and then re-enables another Ready interrupt. Since the mask parameter was not included, the last specified value (1) was used.

Obviously, this is a very simplistic example; however, the main topics of using interrupts have been shown. Your routine may need to format the data, keep track of how many bytes have been transferred, and check for terminator characters.

# Summary of BCD STATUS and CONTROL Registers

*STATUS Register 0*            Card Identification = 4.

*CONTROL Register 0*        Reset Interface (if non-zero value sent).

*STATUS Register 1*            Interrupt Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts are enabled	Interrupt Request	Hardware Interrupt Level Switches	Hardware Interrupt Level Switches	0	0	0	0
Value=128	Value=64	Value=32	Value=16	Value=0	Value=0	Value=0	Value=0

*CONTROL Register 1*        Reset driver pointer (if non-zero value sent).

*STATUS Register 2*            Busy Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake in progress	Interrupts Enabled	0
Value=0	Value=0	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Bit 0 is 1 when a handshake is currently in progress.

*CONTROL Register 2*        Request data by Setting CTLA and CTLB (if a non-zero value is sent); this operation also clears an Interrupt Request (clears bit 6 of Status Register 1).

*STATUS Register 3*            Binary Mode: 1 if the interface is currently operating in Binary mode, and 0 if in BCD mode.

*CONTROL Register 3*

Set Binary Mode: set Binary Mode if non-zero value sent, and BCD Mode if zero sent.

*STATUS Register 4*

Switch and Line States

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OF Switch Is ON	DATA Switch Is ON	SGN1 Switch Is ON	SGN2 Switch Is ON	OVL D Switch Is ON	SGN1 Input Is True	SGN2 Input Is True	OVL D Input Is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*CONTROL Register 4*

Data Out Lines

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Set DO-7 True	Set DO-6 True	Set DO-5 True	Set DO-4 True	Set DO-3 True	Set DO-2 True	Set DO-1 True	Set DO-0 True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*STATUS Register 5*

BCD Digits DI1 and DI2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI1-8 is True	DI1-4 is True	DI1-2 is True	DI1-1 is True	DI2-8 is True	DI2-4 is True	DI2-2 is True	DI2-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*STATUS Register 6*

## BCD Digits DI3 and DI4

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI3-8 is True	DI3-4 is True	DI3-2 is True	DI3-1 is True	DI4-8 is True	DI4-4 is True	DI4-2 is True	DI4-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*STATUS Register 7*

## BCD Digits DI5 and DI6

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI5-8 is True	DI5-4 is True	DI5-2 is True	DI5-1 is True	DI6-8 is True	DI6-4 is True	DI6-2 is True	DI6-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*STATUS Register 8*

## BCD Digits DI7 and DI8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7-8 is True	DI7-4 is True	DI7-2 is True	DI7-1 is True	DI8-8 is True	DI8-4 is True	DI8-2 is True	DI8-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI9-8 is True	DI9-4 is True	DI9-2 is True	DI9-1 is True	DI10-8 is True	DI10-4 is True	DI10-2 is True	DI10-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

## Summary of BCD READIO and WRITEIO Registers

This section describes the BCD Interface's READIO and WRITEIO registers. Keep in mind that these registers should be used *only* when an operation cannot be performed with a STATUS or CONTROL statement.

### BCD READIO Registers

- Register 1      Card Identification
- Register 3      Interface Status
- Register 17     DI1 and DI2
- Register 19     DI3 and DI4
- Register 21     DI5 and DI6
- Register 23     DI7 and DI8
- Register 25     DI9 and DI10
- Register 27     Peripheral Status

*READIO Register 1*      Card Identification (The contents of this register are always 4.)

*READIO Register 3*

Interrupt Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts are enabled	Interrupt Request	Hardware Priority (INT LVL Switches)	Hardware Priority (INT LVL Switches)	0	0	0	0
Value=128	Value=64	Value=32	Value=16	Value=0	Value=0	Value=0	Value=0

*READIO Register 17*

DI1 and DI2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI1-8 is True	DI1-4 is True	DI1-2 is True	DI1-1 is True	DI2-8 is True	DI2-4 is True	DI2-2 is True	DI2-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*READIO Register 19*

DI3 and DI4

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI3-8 is True	DI3-4 is True	DI3-2 is True	DI3-1 is True	DI4-8 is True	DI4-4 is True	DI4-2 is True	DI4-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1



*READIO Register 21*

DI5 and DI6

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI5-8 is True	DI5-4 is True	DI5-2 is True	DI5-1 is True	DI6-8 is True	DI6-4 is True	DI6-2 is True	DI6-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*READIO Register 23*

DI7 and DI8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7-8 is True	DI7-4 is True	DI7-2 is True	DI7-1 is True	DI8-8 is True	DI8-4 is True	DI8-2 is True	DI8-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*READIO Register 25*

DI9 and DI10

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI9-8 is True	DI9-4 is True	DI9-2 is True	DI9-1 is True	DI10-8 is True	DI10-4 is True	DI10-2 is True	DI10-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OF Switch Is ON	DATA Switch Is ON	SGN1 Switch Is ON	SGN2 Switch Is ON	OVLD Switch Is ON	SGN1 Input Is True	SGN2 Input Is True	OVLD Input Is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

**BCD WRITEIO Registers**

Register 1      Reset Interface

Register 3      Enable Interrupt

Register 5      Output Data

Register 7      Initiate Handshake

*WRITEIO Register 1*      Reset interface (any value causes reset).

*WRITEIO Register 3*      Enable interrupt if Bit 7 Set (1); disable if Bit 7 Clear (0).

*WRITEIO Register 5*      Set Data Output Lines

7

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Set DO-7 True	Set DO-6 True	Set DO-5 True	Set DO-4 True	Set DO-3 True	Set DO-2 True	Set DO-1 True	Set DO-0 True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*WRITEIO Register 7*      Initiate handshake: sending any value to this register initiates handshake cycle by setting CTLA and CTLB.

# EPROM Programming for BASIC/WS

---

## Introduction

With HP Series 200/300 BASIC, erasable programmable read-only memory (EPROM) devices are generally used like other mass storage devices. However, EPROM can also be accessed as individual bytes or words of data. This chapter describes both types of usage.

## Accessories Required

In order to program and read EPROM memory devices with HP Series 200/300 computers, you will need the following HP accessories:

- HP 98253 EPROM Programmer card
- HP 98255 EPROM Memory card(s) and compatible EPROM devices

## Hardware Installation

At this point, you should install the programmer and memory cards or verify that they have already been properly installed. The following manuals describe setting up your system to program EPROM devices.

- *HP 98253 EPROM Programmer Installation*—describes setting the select code switches on the programmer card and installing the card.
- *HP 98255 EPROM Memory Installation*—describes selecting compatible EPROM parts, loading the parts on the card, setting memory address switches, and installing EPROM memory cards.

The first example program in the chapter describes how to interrogate EPROM Programmer and Memory cards to determine their current configurations (and also to determine whether or not both are operational *before* installing EPROMs in the memory boards).

## Brief Overview of Using EPROM Memory

EPROM memory is organized and accessed like other mass storage devices from BASIC. Briefly, programs and data can be stored in EPROM memory with the following procedure:

1. Determine the EPROM Programmer card's select code. Determine the address of the EPROM Memory card to be programmed, relative to other cards' addresses, which determines its mass storage unit number.
2. INITIALIZE the memory unit, which writes directory and system information in the EPROM (see "EPROM Directories" for further information).

3. Store the information using whichever one of the following statements is appropriate:

CONTROL—store individual data words

COPY—store any type of file

SAVE—store the program as an ASCII file

STORE—store the program as a PROG file

STORE KEY—store typing-aid keys in a KEY file

4. Access the information with the corresponding one of the following statements:

CAT—get a catalog listing of the files in the EPROM unit

COPY—copy an EPROM file's contents into another file

ENTER—enter data from a file into a program variable

GET—load an ASCII program file into the computer

LOAD—load a BIN, KEY, or PROG file into the computer

LOADSUB—load SUB or FN subprograms from a PROG file

TRANSFER—transfer data from data file to a memory BUFFER

STATUS—read individual data words from EPROM memory

---

## Initializing EPROM Memory

Like other mass storage media, EPROM media must be initialized before being used for storage. Since EPROM Memory cards are organized as mass storage *units*, each card being *one unit*, EPROM memory must be initialized by units. The EPROM Programmer card is used to initialize and store other information in EPROM. This section describes how to specify EPROM units and programmer cards while initializing and accessing EPROM.

### EPROM Programmer Select Code

The EPROM Programmer card is accessed like other interface cards: you must specify its **select code** in BASIC statements. The factory default setting of the select code is 27, which is the select code assumed in the examples in this section. (Setting the select code is described in the *HP 98253 EPROM Programmer Installation* manual.) You don't usually need to specify the programmer card's select code when reading data from EPROM. This will be explained in a later section of this chapter.

### EPROM Addresses and Unit Numbers

With the BASIC system, EPROM Memory cards should be given memory addresses 20 000 through 3FF FFF (hexadecimal). Make sure that no other device is set to the same address range, or the EPROM memory (as well as the other device) will not work properly. Examples of conflicting devices are:

- BASIC 4.0 ROM Systems—which occupy addresses 80 000 through FF FFF (hexadecimal)
- BASIC 5.1 ROM Systems—which occupy addresses 100 000 through 200 000 (hexadecimal)
- HPL ROM Systems—which start at address 100 000 (hexadecimal)
- Series 300 Bit-mapped Display Frame Buffers—which start at address 200 000 and extend to 27F FFF (medium resolution), 2FF FFF (HP 98544 and HP 98545), or 3FF FFF (HP 98547, HP 98548, and HP 98550) (hexadecimal)
- The HP 98700 display controller frame Buffer—which may start at either 200 000 or 300 000 (hexadecimal)

The address switches on EPROM Memory cards can therefore be set in the range of 0 000 001 through 0 011 111, which result in hexadecimal base addresses of 20 000 through 3E 000, respectively, with intervals of 20 000 bytes (hex) between base addresses. When using addresses in this range, SW2 must be set to the "AD" position. Computers featuring HP-UX memory management capabilities, should have this switch set to the "GD" position. (Note that differences between base addresses of cards containing 27128 EPROMs must be at least 40 000 hexadecimal. See the *HP 98255 EPROM Memory Installation* manual for further explanation.)

At power-up, the system automatically gives unit numbers to all cards according to the initialized cards' *relative* memory addresses. The card with the lowest numbered address (which is initialized) is given unit number 0; the initialized card with the next higher address is given unit number 1, and so forth. (Note that un-initialized EPROM units are not given unit numbers by the system.)

As an example, suppose that two EPROM Memory cards are properly installed in the computer with hexadecimal base addresses of 100 000 and 180 000. Assume that they have already been initialized. At power-up, the former card will be given unit number 0 and the latter will be given unit number 1.

If an initialized card with base address 140 000 is then installed (with power off, of course), this card is given unit number 1 and the card at address 180 000 is given unit number 2 at power-up. (Note that, like disc media, the unit number is *not* written on the media. Unit numbers are a function of relative EPROM addresses only.)

It is a good idea to keep track of the addresses of all EPROM Memory cards in the system so that you will know the resultant unit number of each card.

## Verifying Hardware Operation

- 8 In order to INITIALIZE an EPROM unit, you will need to connect a programmer card to it. Connect the cable from the desired programmer card to the EPROM unit to be programmed; the power need not be turned off to make this connection. *All* EPROM devices on the unit to be initialized *must be completely erased*. Also, the address of the EPROM card to be initialized must be higher than all other initialized EPROM cards in the system, which results

in the card being given a unit number one greater than the largest unit number currently in the system.

If you have been keeping track of memory addresses, you should know the unit number of the EPROM Memory card to be programmed. If not, you can use the following program to determine the address of each EPROM Memory card in the computer by plugging the connector into each memory card in succession.

```
100 ! This program interrogates interfaces at select codes
110 ! 8 thru 31 to find an EPROM Programmer card. If one IS found,
120 ! the program reads and displays its STATUS registers; if one
130 ! is NOT found, the program reports this negative result.
140 !
150 ! Clear screen.
160 PRINT CHR$(12)
170 !
180 Sel_code=8           ! Start with select code 8.
190 Found_card=0
200 ON ERROR GOTO Next_sel_code ! Goto next select code if
210                               ! no interface at this one.
220 REPEAT
230   STATUS Sel_code;Id
240   IF Id=27 THEN
250     Found_card=1
260     PRINT "EPROM Programmer card found at Select Code";Sel_code
270     PRINT
280   END IF
290 Next_sel_code: IF NOT Found_card THEN Sel_code=Sel_code+1
300 UNTIL Found_card=1 OR Sel_code>=31
310 OFF ERROR
320 !
330 IF Found_card=0 THEN
340   PRINT "EPROM Programmer card not found."
350   PRINT "Program stopped."
360   STOP
370 END IF
380 !
390 ! Check to see if connected to memory card.
400 STATUS Sel_code,4;Capacity
410 IF Capacity=0 THEN
420   PRINT "EPROM Programmer is NOT connected ";
430   PRINT "to an EPROM Memory card"
440   STOP
450 END IF
```

```

460  !
470  ! Read STATUS Registers 0 thru 6.
480  STATUS Sel_code;Reg0,Reg1,Reg2,Reg3,Reg4,Reg5,Reg6
490  !
500  ! Show register contents.
510  PRINT "STATUS Register 0:"
520  PRINT "  Card ID of EPROM Programmer card=";Id
530  !
540  PRINT "STATUS Register 6:"
550  PRINT USING "#,K,8D";" Connected to EPROM card at address ";Reg6
560  Msb_hex$=IVAL$(Reg6/65536,16)           ! Get MSB's in hex.
570  PRINT " (";Msb_hex$[3,4];"0 000 hexadecimal)" ! Trim leading 0's.
580  !
590  PRINT "STATUS Register 4:"
600  PRINT "  Memory card size=";Reg4;" bytes";
610  Msb_hex$=IVAL$(Reg4/65536,16)           ! Get MSB's in hex.
620  PRINT " (";Msb_hex$[3,4];"0 000 hex)"      ! Trim leading 0's.
630  !
640  PRINT "STATUS Register 5:"
650  PRINT "  Number of contiguous, erased bytes=";Reg5;
660  Msb_hex$=IVAL$(Reg5/65536,16)           ! Get MSB's in hex.
670  PRINT " (";Msb_hex$[3,4];"0 000 hex)"      ! Trim leading 0's.
680  !
690  PRINT "STATUS Register 2:"
700  PRINT "  Current target address=";Reg2
710  !
720  PRINT "STATUS Register 3:"
730  Word$=IVAL$(Reg3,16)
740  PRINT "  Word at current target address=";Reg3;" (";Word$;" hex)"
750  !
760  PRINT "STATUS Register 1:"
770  IF Reg1=0 THEN
780     PRINT "  Programming time = 52.5 ms"
790  ELSE
800     PRINT "  Programming time = 13.1 ms"
810  END IF
820  !
830  END

```

8

The following display is a typical result of running the program.



EPROM Programmer card found at Select Code 27

STATUS Register 0:

Card ID of EPROM Programmer card= 27

STATUS Register 6:

Connected to EPROM card at address 1048576 (100 000 hexadecimal)

STATUS Register 4:

Memory card size= 262144 bytes (040 000 hexadecimal)

STATUS Register 5:

Number of contiguous, erased bytes= 0

STATUS Register 2:

Current target address= 0

STATUS Register 3:

Word at current target address= -1 (FFFF hex)

STATUS Register 1:

Programming time = 52.5 ms

### Program Results

The program interrogates interfaces until it finds an EPROM Programmer card. It then prints the values of the Programmer card's STATUS registers. Register 6 shows the memory address of the EPROM Memory card to which the programmer card is connected. The program also shows that it can determine the type of EPROM devices being used on the card (2764's or 27128's).

The "target address" register points to the memory location (an offset address to the card's base address) at which the next word of data will be read (STATUS register 3) or written (CONTROL register 3). Target address 0 is the first word on the EPROM card. STATUS register 1 indicates which programming time will be used (for each word) during subsequent programming operations; 0 indicates a program time of 52.5 milliseconds, and 1 indicates 13.1 milliseconds.

## Initializing Units

To INITIALIZE an EPROM unit, you must specify the select code of the EPROM Programmer card and the unit number of the EPROM Memory card. For instance, the following statement initializes the memory with unit number 0 through the programmer card at select code 27.

```
INITIALIZE ":EPROM,27,0"
```

Because the unit number defaults to 0 if not specified, an equivalent statement would be:

```
INITIALIZE ":EPROM,27"
```

An error is reported if the specified programmer card is not connected to the specified EPROM unit. Furthermore, if the specified EPROM memory unit is not completely erased, error 72 (drive not found or bad address) is reported. Note that the entire card need not be filled with EPROMs for it to appear as entirely erased, since empty sockets and erased EPROM memory read as "FF" data bytes. The following simple program determines whether or not the EPROM unit contains all erased EPROMs (or erased EPROMs and empty sockets).

```
10 CONTROL 27,2;0 ! Set target address to first byte.
20 STATUS 27,4;Total_capacity,Erased_bytes
30 PRINT "EPROM card is ";
40 IF Total_capacity=Erased_bytes THEN
50 PRINT "completely erased (or empty)."
```

```
60 ELSE
```

```
70 PRINT "NOT completely erased."
```

```
80 END IF
```

```
90 END
```

## EPROM Directories

8

The INITIALIZE operation writes a directory and system information in the EPROM unit. This information occupies the first "sectors" of EPROM memory (since the unit is treated like mass storage, it is logically divided into 256-byte records known as sectors). The following table shows how the BASIC system allocates EPROM sectors.

### EPROM Sector Allocation

EPROM Type	Usable Sectors	Sectors for System Use	Sectors for Users	Maximum No. of Files
2764	511	0 - 6	7 - 510	40
27128	1023	0 - 11	12 - 1022	80

Note that the figures given for Total Sectors and Sectors for User are for *fully loaded* memory cards. Note also that the Total Sectors is one less than you may have expected, which shows that one sector is required by the system for overhead.

### EPROM Catalogs

Performing a CAT of the EPROM card reveals that it has been initialized. You can either specify the select code of the programmer card or use 0, since reading the EPROM card does not require the programmer card be connected to it. However, if you do specify a select code, then that programmer card must be connected to the specified EPROM unit, or error 72 will be reported. The following statements perform the same function (specifying select code 27 would change the first line of the catalog listing accordingly):

```
CAT ":EPROM,0"
```

or

```
CAT ":EPROM,27" ''
```

```
:EPROM,0
```

```
VOLUME LABEL: B9836
```

```
FILE NAME PRO TYPE REC/FILE BYTE/REC ADDRESS
```

This directory has the same general format as internal-disc directories, as described in the *HP BASIC 6.2 Programming Guide*. You can also perform all operations on EPROM directories that you can with other mass storage directories (such as SKIP and COUNT files and CAT individual PROG files).

---

## Programming EPROM

Once an EPROM unit is initialized, you can store data and programs in it. The following storage operations are supported for EPROM memory:

- CHECKREAD—direct the system whether to perform an additional verify operation after all operations that write to mass storage
- CONTROL—store individual data words in EPROM
- COPY—copy any type of file (that already exists on another mass storage device) into EPROM
- SAVE—store the current program in an ASCII file
- STORE—store the current program in a PROG file
- STORE KEY—store the current typing-aid keys in a KEY file

Using these statements is described in the following sections of this chapter. The topic of reading EPROM information is described in a subsequent section.

### Storing Data

As a simple example of storing a data file in EPROM, suppose that you want to store the date that the EPROM was initialized and the current number of EPROM chips on the card in EPROM memory. The following program shows a simple example of how you might perform this operation.

```
100 ! This program stores the Date that the
110 ! EPROM Memory unit was initialized.
120 ! (An EPROM file name shows the date.)
130 !
140 ! Select EPROM mass storage unit.
150 Progrm_sc=27
160 Unit_no=0
170 Eprom_msus$=":EPROM,"&VAL$(Progrm_sc)&","&VAL$(Unit_no)
180 !
190 ! Determine date to write in EPROM.
200 Correct_date=0
210 REPEAT
220 DISP "Enter date to be stored in EPROM ";
230 DISP "(Press ENTER for time shown)."
```

```
240 OUTPUT KBD;DATE$(TIMEDATE);
250 ENTER KBD;Date_$
```

```

260     SET TIMEDATE DATE(Date_!) ! Set date.
270     DISP "Is this correct? ";DATE$(TIMEDATE)
280     ENTER KBD;Ans$
290     IF UPC$(Ans$[1,1])="Y" THEN Correct_date=1
300 UNTIL Correct_date
310 DISP
320 !
330 ! Format Date_! from "DD MMM YYYY"
340 ! to "MMM_DD_YY".
350 Month$=Date_![4,6]
360 Day$=TRIM$(Date_![1,2]) ! Strip leading space (if one).
370 Year$=Date_![10,11] ! Remove "19" from year.
380 File_name$=Month$&"_"&Day$&"_"&Year$
390 !
400 ! Create a one-record ASCII file on the internal disc
410 ! (use an external disc with Model 16)
420 ! with the DATE as the file's name.
430 Disc_msus$=":INTERNAL"
440 CREATE ASCII File_name$&Disc_msus$,1 ! Error if file exists.
450 !
460 ! Write info into EPROM file.
470 COPY File_name$&Disc_msus$ TO File_name$&Eprom_msus$
480 PURGE File_name$&Disc_msus$ ! Remove disc file after use.
490 !
500 ! Now read date with catalog of file names.
510 CAT Eprom_msus$
520 !
530 END

```

The program first prompts for the EPROM unit number (the programmer card is assumed to be at select code 27). The program then prompts for the date by presenting the current clock date to the user on the keyboard input line. The user can either modify the date or accept it as it is shown.

Assuming that the program is run on a Model 226 or 236, the program stores this information in an ASCII file on a disc in the internal drive. (It would be much faster to store the file in a MEMORY volume or in Bubble memory.) This information is then stored in EPROM, and the internal ASCII file is purged.

8

### Data Storage Rates

The information is stored in EPROM at *approximately* the following rates (program time is set by writing to CONTROL register 1):

### Approximate Storage Rates

Program Time	Seconds per Sector	Bytes per Second
13.1 ms	2	150
52.5 ms	7	38

Note that these times are for COPY, SAVE, and STORE operations. The storage rate when using CONTROL is lower slightly than these figures.

### Determining Unused EPROM Memory

A potential problem with the example program in the preceding section is that there are times when you are not sure whether or not there is enough erased EPROM memory to store your information. Unfortunately, the system generally cannot determine beforehand whether there is sufficient room left in an EPROM unit to store the information it has been directed to store. This consequence is due to the fact that both blank sockets and erased EPROM memory read as all 1's (hexadecimal FF bytes). The system can, however, determine when there is not enough room left on a *fully loaded* card (Error 64 is reported).

Thus, when the system is directed to store data in EPROM, it begins programming the EPROMs one word at a time at the "next available" location. After each word is programmed, the system reads the word and compares it to what was to be written; this operation is known as "verifying" the word. An error will be reported when the word is not verified (such as when a blank socket, a previously programmed word, or other hardware failure has been reached). So before you attempt to store any information in EPROM memory, you should determine whether or not there is enough erased memory to hold the data.

8

The general method of determining whether or not an EPROM memory unit has enough erased space to store your information is as follows: Determine the total number of usable sectors on the EPROM card, and then subtract the number already used. The result is the number of sectors available for storing additional information. This procedure is broken down into steps as follows (an example follows the procedure):

1. Determine the number of usable sectors on the EPROM card.
  - a. Determine the number of usable sectors of EPROM by using the following formulas:
 
$$\text{Total Sectors} = (\text{Chips on card}) * (\text{Bytes/Chip}) * (1 \text{ Sector}/256 \text{ Bytes})$$

$$\text{Usable Sectors} = \text{Total Sectors} - 1 \text{ sector (used by the system)}$$
 in which:
 
$$\text{Bytes/Chip} = 16\,384 \text{ (for 27128's)}$$

$$\text{Bytes/Chip} = 8\,192 \text{ (for 2764's)}$$
  - b. Use the CAT statement to determine how many EPROM sectors are already being used by files (already programmed).
2. Determine the number of sectors required to store your information.
  - a. For ASCII data files, this number will simply be the number of records specified in the CREATE ASCII statement that created the file.
  - b. For BDAT data files, multiply the number of records in the file by the record size (default=256 bytes), divide this product by 256, and round any non-integer result to the next larger integer. Add one to this result to account for the sector used by the system (for EOF pointer and number of records).
  - c. For programs, place the information in a mass storage file using STORE or SAVE (MEMORY volumes and BUBBLE memory are best suited for this purpose). Use the CAT statement to determine how many 256-byte sectors were required to store the information.
  - d. For all other types of files, a quick look at the directory of the media on which the file is presently stored shows how many sectors are required to store the file.
3. Compare the amount of usable memory in EPROM to the amount of memory required for your information. If there is sufficient memory, perform the storage operation. Otherwise, use another EPROM unit or mass storage device.

As an example, suppose that you want to store a BDAT file that contains 20 records of 20 bytes each in EPROM. Since  $20 * 20 = 400$ , and  $400 / 256 = 1.5625$

(which rounds up to the next larger integer of 2), two sectors of EPROM are required for the data. Add one sector for system use. Therefore, three sectors are required to store the file.

To determine how much EPROM memory is available, first calculate the total number of sectors on the card. For this example, suppose that only two 27128 chips are on the board. The total number of sectors on the card is calculated by applying the preceding formula:

$$\text{Total sectors} = 2 \times 16\,384 / 256 = 128$$

$$\text{Usable sectors} = \text{Total sectors} - 1 = 127$$

To see how many sectors have already been used, perform a CAT of the EPROM card; assume EPROM unit 0.

```
CAT ":EPROM,0"
```

```
:EPROM,0  
VOLUME LABEL: B9836  
FILE NAME PRO TYPE REC/FILE BYTE/REC ADDRESS  
  
Mar_8_83 ASCII 1 256 12  
EPROM_BITS ASCII 17 256 13  
EPROM_INIT ASCII 11 256 30
```

The CAT reveals that the last file begins at sector 30 and is 11 sectors in length. Thus, the next unused sector begins at sector 41. Since sector addresses start at 0, 42 sectors have already been used. Assuming that you have not written any data in subsequent sectors (such as with the CONTROL statement), there are 85 (=127-42) sectors of unused EPROM remaining. The BDAT file can be stored in the unit.

## Storing Programs

8

Storing programs in EPROM is a simple operation. Like storing programs in other mass storage media, you can use either the STORE statement or the SAVE statement. The one you will use depends on whether you want the program to be stored in a PROG or an ASCII file; the STORE statement stores the program in a PROG file, while the SAVE statement stores it in an ASCII file. If the program is already stored on another device, use the COPY statement.



Compiled Pascal subprograms, or “CSUBs,” can also be stored in EPROM with COPY. For instructions on how to write these compiled subprograms, see the *HP BASIC 6.0 CSUB Utility* manual.

As with storing data files, you must ensure that there is enough memory on the card to hold the program. First execute a CAT operation on the EPROM unit to determine how many sectors are unused. Then determine how many sectors will be required to store the program by using STORE or SAVE to store the program on another mass storage device. If there is enough unused EPROM memory, execute STORE or SAVE with the destination as the desired EPROM unit. For instance, the following statements are typical ways to store programs in EPROM.

```
STORE "Prog_1:EPROM,27,0"  
SAVE "Prog_1:EPROM,27"
```

## Programming Individual Words and Bytes

You can also program individual words and bytes in EPROM with the BASIC system. However, you should *not* use these techniques to program EPROMs which are to be used as mass storage “units” in this manner. In other words, if you are going to access the EPROM with mass storage statements, use only operations such as SAVE, STORE, and COPY to program the EPROM unit. If the EPROM is to be for another purpose, such as to store machine-language code in another system, you can use these techniques to program the EPROMs.

To program individual words, use CONTROL to set the target address and then write the desired word at that address. Repeat this process for as many words as you need to write. Note that you need to set the target address before every write operation. If you don’t, an error will be reported.

The automatic verify operation is still performed for each word written into EPROM memory. The following example program shows how you might perform this type of operation; the first 16 384 bytes of the EPROMs in sockets marked “0U” and “0L” are programmed. (The program takes approximately eight minutes to run.)

```

100 ! Assume data source is a BDAT file that contains exactly
110 ! 8192 INTEGER elements (written with FORMAT OFF).
120 !
130 ASSIGN @Source TO "EPROMWORDS:INTERNAL"
140 INTEGER Int_array(0:8191)
150 ENTER @Source;Int_array(*)
160 !
170 ! Write 8K words (16K bytes).
180 FOR Addr=0 TO 16382 STEP 2
190     CONTROL 27,2;Addr,Int_array(Addr/2) ! Write to EVEN addresses.
200 NEXT Addr
210 !
220 END

```

Notice that the target address (CONTROL register 2) begins at an even address (0) and is incremented by two for each subsequent word. Attempting to program a word at an odd address will generate an error.

To program individual bytes, you will need to mask the byte that is not to be programmed. For instance, suppose that you want to insert one EPROM chip in the board and program it with data bytes. Inserting the chip in one of the sockets marked with an "L" gives the memory odd addresses. The program will need to be modified so that it writes only the least significant eight bits of each word (since you can only program words, which begin at even addresses).

To program only the least significant byte, you would make the most significant byte all 1's so that the program operation will verify (remember that empty sockets and erased bits read as all 1's). The following program shows an example of programming single bytes of data in the EPROM located in the socket marked "0L." Note that the only difference between this program and the previous one is the manipulation of the upper eight bits of each integer.

```

100 ! Assume data source is a BDAT file that contains exactly
110 ! 8192 INTEGER elements (written with FORMAT OFF).
120 !
130 ASSIGN @Source TO "EPROMBYTES:INTERNAL"
140 INTEGER Int_array(0:8191)
150 ENTER @Source;Int_array(*)
160 !
161 ! Define mask for upper 8 bits.
162 Ff00=IVAL("FF00",16)
163 !
170 ! Write 8K bytes.

```

```

180  FOR Addr=0 TO 16382 STEP 2 ! Must still write to EVEN addresses.
181    Low_byte=BINIOR(Ff00,Int_array(Addr/2)) ! MSB=FF (LSB=unchanged).
190    CONTROL 27,2;Addr,Low_byte
200  NEXT Addr
210  !
220  END

```

To program bytes of an EPROM located in a socket marked “U”, you would left-shift the 8-bit value by eight places (which shifts the least significant byte to the most significant byte). For instance, the following statement shifts the least significant byte to the most significant byte and then makes the least significant byte all 1’s:

```
High_byte=BINIOR(SHIFT(Low_byte,-8),255)
```

Programming EPROM with such eight-bit values writes the eight bits into EPROM devices at even addresses (i.e., in sockets marked with “U”).

## Operations Not Allowed

Once data is written in EPROM, it cannot be selectively erased (without erasing the entire EPROM device). Consequently, the following mass storage operations are not allowed for EPROM mass storage:

- CONTROL (cannot be used to write to registers 7 and 8 of an I/O path name assigned to a BDAT file)
- CREATE
- CREATE ASCII
- CREATE BDAT
- COPY (of an entire mass storage unit *into* EPROM)
- OUTPUT
- PROTECT
- PURGE
- RENAME
- RE-SAVE
- RE-STORE
- RE-STORE KEY
- TRANSFER (*to* an EPROM file)

Note that HFS cannot be used with EPROM.

---

## Reading EPROM Memory

After an EPROM unit has been programmed, you can perform the following operations to read the information:

- CAT—get a catalog listing of the files in an EPROM unit
- COPY—copy an EPROM file's (or unit's) contents into another file (or unit)
- ENTER—enter data from an ASCII or BDAT data file into program variables
- GET—load an ASCII program file into the computer
- LOAD—load a BIN, KEY, or PROG file into the computer
- LOADSUB—load SUB or FN subprogram(s) from a PROG file
- TRANSFER—transfer data from a BDAT data file into a memory BUFFER
- STATUS—read individual data words from EPROM memory

## Retrieving Data and Programs

Reading data files stored in EPROM is similar to reading data files stored in other mass storage devices; the only difference is the mass storage unit specifier (msus), which will be of the form `:EPROM,Select_code,Unit_number`. Remember that both `Select_code` and `Unit_number` parameters can be any type of numeric expression. Also keep in mind that when reading EPROM units you do not need to specify the select code of the EPROM programmer card; you can specify a select code of 0. However, if you do specify the programmer card's select code, it must be connected to the specified EPROM unit. If the unit number parameter is omitted, a default value of 0 is used.

The broad subject of using `ENTER` to read data files is discussed in the *HP BASIC 6.2 Programming Guide*. This manual discusses the `ENTER` statement in detail, and also describes using the `TRANSFER` statement to transfer the contents of data files into memory `BUFFERs`.

Like reading data files, retrieving programs from EPROM is identical to performing these operations from other mass storage devices. Refer to the *HP BASIC 6.2 Programming Guide*.

## Summary of EPROM Programmer STATUS and CONTROL Registers

*STATUS Register 0* ID Register. This register contains a value of 27 (decimal) which is the ID of an EPROM Programmer card.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	1	1	0	1	1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

*CONTROL Register 0* Interface Reset. Writing any non-zero value into this register resets the card; writing a value of zero causes no action.

*STATUS Register 1* Read Program Time. A value of 0 indicates that the program time is 52.5 milliseconds for each 16-bit word (default); a non-zero value indicates that the program time is 13.1 milliseconds.

*CONTROL Register 1* Set Program Time. Writing a value of 0 into this register sets the program time to 52.5 milliseconds for each 16-bit word; any non-zero value sets program time to 13.1 milliseconds.

*STATUS Register 2* Read Target Address. This register contains the offset address (relative to the card's base address) at which the next word of data will be read (via STATUS Register 3) or written (via CONTROL Register 3). The default address is 0, which is the address of the first byte on the card.

- CONTROL Register 2* Set Target Address. Writing to this register sets the offset address at which the next word of data will be read (via STATUS Register 3) or written (via CONTROL Register 3). The target address must always be an *even* number.
- STATUS Register 3* Read Word at Target Address. This register contains the 16-bit word at the current target address.
- CONTROL Register 3* Write Word at Target Address. Writing a data word to this register programs a 16-bit word at the current target address. The target address must be set (via CONTROL register 2) before every word is written. Automatic verification is also performed after the word is programmed.
- STATUS Register 4* Current Memory Card Capacity (in bytes). This register contains the current capacity of a *fully loaded* card in bytes; it also indirectly indicates which type of EPROM devices are being used on the card. If 262 144 is returned, then 27128 EPROMs are being used; if 131 072 is returned, then 2764 devices are being used. A 0 is returned if the programmer card is not currently connected to any EPROM memory card.
- CONTROL Register 4* Undefined.
- STATUS Register 5* Number of Contiguous, Erased Bytes. Reading this register causes the system to begin counting the number of subsequent bytes, beginning at the current target address, that are erased (or are empty sockets). The counting is stopped when a programmed byte (i.e., one containing at least one logical 0) is found or when the end of the card is reached. If the byte at the current target address is not FF, then a count of 0 is returned. Error 84 is reported if the programmer card is not currently connected to any EPROM card.

*CONTROL Register 5*

Undefined.

*STATUS Register 6*

Base Address of EPROM Memory Card. This register contains the (absolute) base address of the EPROM memory card to which the programmer card is currently connected; this base address is also the absolute address of the first word on the card. Error 84 is reported if the programmer card is not currently connected to any EPROM memory card.

*CONTROL Register 6*

Undefined.





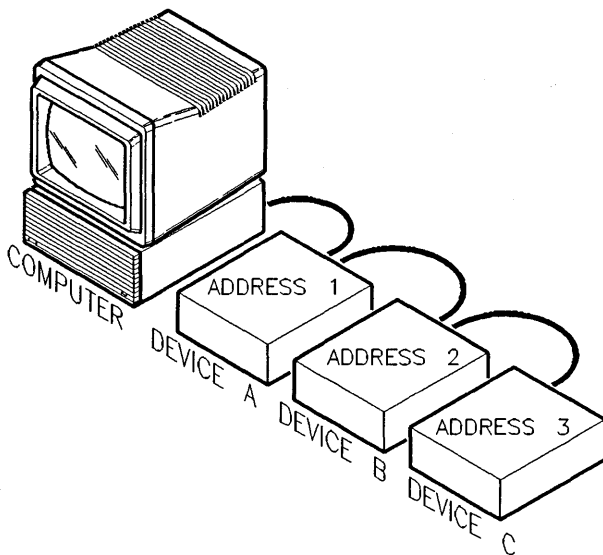
## HP-HIL Interface

---

### The Interface to HP-HIL Devices

This chapter describes communication with the HP-HIL interface. It is primarily a description of the use of HIL SEND and HIL EXT to handle the HP-HIL interface. This interface is capable of supporting up to seven devices, such as a Function Box, a system ID Module, and other peripherals generally related to human input.

Before launching into a discussion of the workings of the HP-HIL interface, a general overview needs to be presented. HP-HIL stands for “Hewlett-Packard Human Interface Link”. The following diagram illustrates the basic components.



Hewlett-Packard Human Interface Link

HP-HIL initialization takes place when BASIC is booted or when you execute SCRATCH A. BASIC logs HP-HIL devices which are present on the link. The link can deal with a maximum of seven devices at any one time (any devices present after the seventh one are not recognized). If you add an HP-HIL device to the HP-HIL link after the BASIC system has been booted, the device will not automatically be recognized by the system. If you replace a device on the link with a different device, the system may mis-interpret the data coming from the new device. Therefore, when adding, removing, or replacing a device on the link, either re-boot the system or execute a SCRATCH A.

The address of a particular device is merely its topological order of placement along the link. In the above diagram, Device A has address 1, B has address 2, and C has address 3. This is only a result of their physical order of connection. If Device C had been connected between Devices A and B, Device A would still have been address 1, but Device C would be address 2, and B would be address 3. The type of device is irrelevant to the address assigned to it.

After the link is operational, and during subsequent link operations, each device looks at the data being sent down the link. If a device notices that the destination address associated with the link data is the same as that device's address, that device receives and acts on the data. Otherwise, the data is merely shuttled along to the next device.

## **Preview of HP-HIL Devices**

HP-HIL devices can be divided into a number of categories. This section provides you with a table that includes these categories, as well as a list of high level and low level statements that apply to each category.

## HP-HIL Device categories

HP-HIL Device Categories	High Level BASIC Access	Low Level BASIC Access
HP-HIL Keyboards	BASIC Operating System normally handles keystrokes. Programs can enter text and numbers with the statements: INPUT, LINPUT, and ENTER.	ON/OFF KEY ON/OFF KBD KBD\$
Relative Positioner	BASIC Operating System handles as cursor-movement input. Can also be used with GRAPHICS INPUT IS.	ON/OFF KBD (traps movement as arrow keys and also traps mouse buttons) KBD\$ ON/OFF KNOB ON/OFF CDIAL CDIAL
Absolute Positioner	Can be used with GRAPHICS INPUT IS.	HIL SEND ON HIL EXT HILBUF\$
ID Module	One can be used with SYSTEM\$(“SERIAL NUMBER”)	HIL SEND ON HIL EXT HILBUF\$
Other Devices	None	HIL SEND ON HIL EXT HILBUF\$

## Communicating through the HP-HIL Interface

This section provides a brief description of the HP-HIL Interface Driver. The HP-HIL Interface Driver supports a set of statements which allow communications between the HP-HIL interface and HP-HIL devices. These statements and commands are as follows:

**HIL SEND *Address;HIL\_Command*** allows you to send HP-HIL commands to an HP-HIL device (e.g., HIL SEND

1;IDD). The BASIC HP-HIL commands can be found in the "HP-HIL Appendix" in this manual. The *HIL\_Address* is the location of the device in the HP-HIL link. Address 1 is assigned to the first device on the link that is addressable (i.e., any device except HP-HIL Extension Modules). Ascending address are assigned to subsequent devices on the link.

ON HIL EXT *Address\_mask Branch*

enables end-of-line interrupts from HP-HIL devices. This statement allows you receive interrupts from up to seven devices on the HP-HIL link. The *Address\_mask* is a bit-map of the locations of the device or devices in the HP-HIL link. The default *Address\_mask* is 254 which allows up to 7 devices to send interrupts. To select devices from which you want to receive interrupts, you need to raise 2 to the power of that device's address. For example, if the device is the second one on the HP-HIL link then you would raise 2 to the 2nd power which would result in an *Address\_mask* of 4. If you have two devices on the HP-HIL link, one at the second position and the other at the third position, then to enable interrupts from both of these devices you would add  $2^2$  and  $2^3$  together. The resulting *Address\_mask* would be 12. *Branch* refers to a branch to a program line number, label, subroutine or subprogram using the keywords GOTO, GOSUB, RECOVER, or CALL.

OFF HIL EXT

disables end-of-line interrupts. This statement *does not* require an address mask. It will disable all previously enabled end-of-line interrupts for HP-HIL devices.

HILBUF\$

is a function used to capture data returned from HP-HIL devices. This function provides a buffer for data to be stored in after execution of the first two statements listed above. Note that this buffer only holds up to 256 bytes of data. Once the buffer limit is reached it will not receive any new data until it has been emptied by reading the buffer. The first byte stored in the string buffer tells you if data has been lost. This byte is initially zero (the “null” character). It is only zero if no data has been lost; otherwise, it contains the number of packets lost. A packet is three or more bytes consisting of the packet length (first byte), the device address (second byte), and one or more bytes of data from the device (i.e., a poll record or a response to a command). A **poll record** is a set of data sent by an HP-HIL device to HILBUF\$ which accompanies an ON HIL EXT interrupt. This data first includes a poll record header byte which contains information about the bytes that follow it (covered in the “HP-HIL Appendix” under the section titled “Poll Record”).

The poll records (see HILBUF\$ above) for the devices listed below are not available through ON HIL EXT. An *Address\_mask* (ON HIL EXT) can include these devices, but no interrupts will be generated. The excluded devices are:

- any relative pointing device.
- the current GRAPHICS INPUT IS device.
- any system keyboard.

The HIL SEND statement operates under a different set of conditions than ON HIL EXT:

- HIL SEND *Address*;IDD is allowed with all devices.
- HIL SEND *Address*;HIL\_command is allowed if that device is not:
  - a relative pointing device.
  - currently a GRAPHICS INPUT IS device.

---

## Supported HP-HIL Devices

This section provides a brief description of those devices supported by the HIL Interface Driver, references to information for those devices not supported by the HIL Interface Driver, a program for identifying all devices on the HP-HIL link, and an explanation of that program. The topics are as follows:

- Selecting HP-HIL Devices (BASIC/UX only)
- Identifying All Devices on the HP-HIL Link
- Explanation of the HIL\_ID Program
- HP-HIL Devices

### Selecting HP-HIL Devices

When you enter BASIC/UX with the `rmb` command, the BASIC/UX system uses all available devices on the HP-HIL link. This prevents other HP-UX processes from accessing these HP-HIL devices. However, HP-HIL devices can be selected by using the `SET HIL MASK` command. This allows you to select only those HP-HIL devices required for your particular BASIC/UX task and relinquishes the remaining devices for use by other HP-UX processes. This section explains the use of this command.

## Enabling HP-HIL Devices

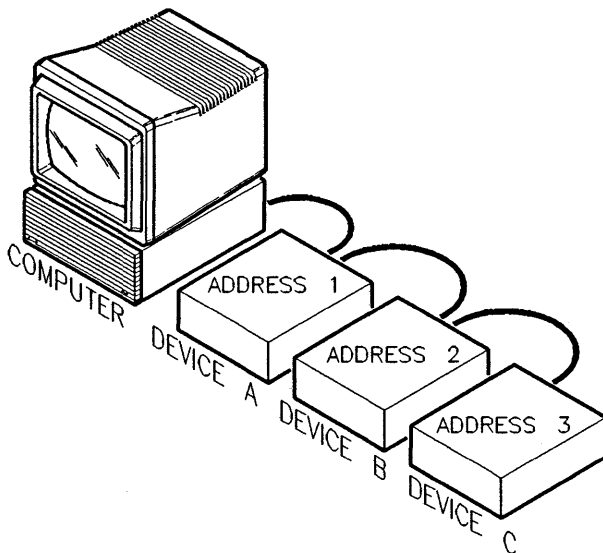
The SET HIL MASK statement enables the specified HP-HIL device for use by the BASIC/UX system. The syntax for this command is as follows:

```
SET HIL MASK address_mask
```

where the *address\_mask* is obtained by raising 2 to the power of each of the addresses of the desired devices and adding these values.

### Example

The following program uses the HP-HIL identify and describe command to determine the type of device that is located at address 3 in the link and gives the device's characteristics. The HP-HIL device shown below has address 3 assigned to it because it is the third device in the HP-HIL link.



### An HP-HIL Device with an Address of 3

The program then assigns the device to the current process. This prevents another process from using the device. Finally it prints the device's identification number and characteristics on the screen. You can use the "Device ID Byte Definitions" table in the "HP-HIL Appendix" in this manual and the identification number to determine the name of your device.

```

100 SET HIL MASK 2^3          ! Assigns the device at address 3 to the
110                          ! current BASIC/UX process (locks it).
120 HIL SEND 3;IDD ! Sends the IDD command to the device at address 3.
130 A$=HILBUF$           ! Assigns the IDD information found in the HP-HIL
140                          ! buffer string function to the string A$.
150 FOR I=1 TO LEN(A$)     ! This FOR loop prints the contents
160   B$=IVAL$(NUM(A$[I]),16) ! of A$ as hexadecimal values. The
170   PRINT B$[3];" ";     ! fourth element is the ID number
180 NEXT I                ! that identifies the device.
190 END

```

## Identifying All Devices on the HP-HIL Link

Each device in the HP-HIL link has a Device ID which identifies that device and a Describe Record which provides you with device characteristics. This information can be obtained by executing the HP-HIL IDD command and parsing the string value returned by the HILBUF\$ function. A program called HIL\_ID, located in the directory called /usr/lib/rmb/demo, makes use of the IDD command and HILBUF\$ function for:

- Determining if a device is recognized as being in the HP-HIL link,
- Identifying the device at a specific address, and
- Determining the device's characteristics (what it can do).

Assuming your HP-HIL link has a:

- Touchscreen located at address 1
- ITF Keyboard (HP 46020/21A) located at address 2
- Function Box located at address 3



Executing the HIL\_ID program would produce the following output:

HP 35723A (Touchscreen) located at address 1  
Describe Record Information  
  I/O Descriptor Information  
    Does not support Prompts/Acknowledges 1 thru 7  
    Supports Proximity Detection  
    Does not report buttons  
  X and Y axis information reported  
    Absolute positioning device  
    Returns 8 bits/axis

HP 46020/21A (ITF Keyboard) located at address 2  
Describe Record Information  
  No special features

HP 46086A (Function Box) located at address 3  
Describe Record Information  
  I/O Descriptor Information  
    Recognizes General Prompt and Acknowledge  
    Does not support Prompts/Acknowledges 1 thru 7  
    Does not report buttons  
  No axis information reported

NO MORE DEVICES.

Just what does the above information tell you? Let's look at the first device. It is a Touchscreen located at address 1 in the HP-HIL link. To determine what this device can do, you need to know its characteristics. The Describe Record provides you with this information. Describe Record information returned by this device is as follows:

- I/O Descriptor byte information is reported. The information supplied in this byte tells you that when you touch your finger on the screen it will be detected and when you remove it from the screen it will be detected. This is called proximity in/out detection.
- It is an absolute positioning device. This means that every coordinate position on the screen is referenced to the lower left-hand corner of the Touchscreen (X-coordinate = 0 and Y-coordinate = 0).
- X and Y axis information is reported. This tells you that Poll Records received when communicating with this device will contain X and Y coordinate information. These are absolute coordinate positions.
- Coordinate information is returned as 8 bits per axis. This means there will only be one byte of X coordinate information returned in the Poll Record and one byte of Y coordinate information returned in the Poll Record.

Putting the above information together, the Touchscreen makes a great device for option selection from screen menus. Other uses are left up to your creativity.

## **Explanation of the HIL\_ID Program**

The program called HIL\_ID is located on your *Manual Examples* disc for BASIC/WS, and in the directory `/usr/lib/rmb/demo` for BASIC/UX. This program has been divided into four segments. Each segment of the program will be given and explained in this section. It is not absolutely necessary for you to read this section to gain an understanding of how to communicate with HP-HIL devices. If you decide not to read this section, skip to the next one titled "HP-HIL Devices."

### **Segment 1 of HIL\_ID**

This segment of the program executes the HP-HIL command IDD for each device in the HP-HIL link. The system places the information returned from

executing this command in the string buffer used by the HILBUF\$ function. The buffer information has been stored in a string array to simplify future processing. The string buffer used by HILBUF\$ is cleared each time the function is executed.

```
1000 OPTION BASE 1
1010 DIM Idd$(7)[20]
1020 INTEGER Dev_id,Address_num,Des_header,Test,Count
1030 COM INTEGER Io_header
1040 !
1050 !*****
1060 ! This is segment 1 of the program. It stores Identify
1070 ! and Describe information in the array Idd$.
1080 !*****
1090 !
1100 ON ERROR GOTO Link_end
1110 FOR I=1 TO 7
1120   HIL SEND I;IDD
1130   Idd$(I)=HILBUF$
1140 NEXT I
1150 Link_end: !
1160 Count=I-1
1170 PRINT
1180 !
```

The following information is an explanation of program segment 1.

*Line 1000* sets a lowerbound of 1 for the string array Idd\$ in the program.

*Lines 1010 to 1030* declare the variables for the program.

*Line 1100* sets up a branch to the label Link\_end if an error occurs. Note that an error will occur in the FOR loop of lines 1110 to 1140 if there are less than 7 devices on the HP-HIL link. The branch to the label Link\_end is designed to prevent execution of the program from stopping when this condition occurs.

*Lines 1110 to 1140* are a FOR loop which executes the HP-HIL IDD command as many times as there are devices recognized plus one or until seven devices have been recognized on the HP-HIL link. Identify and Describe information from executing the IDD command is stored in the string array Idd\$ for future processing. If there are fewer than 7 HP-HIL devices in the link, then ON ERROR causes a branch to be taken outside of the FOR loop. If there are 7 devices in the loop, then ON ERROR branching *does not* take place.

*Line 1150* is the destination label `Link_end` for the `ON ERROR` branch from the `FOR` loop.

*Line 1160* sets the loop `Count` for the `FOR` loop in the second segment of the demonstration program.

## Segment 2 of HIL\_ID

This segment of the program is a large `FOR` loop with a `SELECT` statement in it for selecting and identifying the various devices in the HP-HIL link. Each `CASE` statement within the `SELECT` statement causes a message to be displayed for the device found on the link. This message contains the HP product number, device name and device address. After the `SELECT` statement a `CALL` is made to the subprogram `Describe_rec` which is used for determining the characteristics of the devices in the HP-HIL link. If the device is found to report I/O Descriptor information, the `Describe_rec` subprogram calls the subprogram `Io_descriptor`. This subprogram provides additional information about the device.

```
1190 !*****
1200 ! This is segment 2 of the program. It identifies all
1210 ! devices on the link and provides their address
1220 ! in the link. It also uses two subprograms called
1230 ! Describe_rec and Io_descriptor. These subprograms
1240 ! describe what each device can do.
1250 !*****
1260 !
1270 FOR I=1 TO Count
1280   Dev_id=NUM(Idd$(I)[4])
1290   PRINT
1300   !
1310   Address_num=NUM(Idd$(I)[3])
1320   Des_header=NUM(Idd$(I)[5])
1330   Io_header=NUM(Idd$(I)[LEN(Idd$(I))])
1340   !
1350   SELECT Dev_id
1352   CASE 0 TO 30 ! Vectra Keyboard
1354     PRINT "HP 46030A (Vectra Keyboard) located at address ";Address_num
1360   CASE 48 ! Function Box
1370     PRINT "HP 46086A (Function Box) located at address ";Address_num
1380   CASE 52 ! ID Module
1390     PRINT "HP 46084A (HP-HIL ID Module) located at address ";Address_num
1400   CASE 92 ! Bar Code Reader
```

```

1410     PRINT "HP 92916A (Bar Code Reader) located at address ";Address_num
1420 CASE 96 ! Rotary Control Knob
1430     PRINT "HP 46083A (Rotary Control Knob) located at address ";
        Address_num
1440 CASE 97 ! Control Dials and Quadrature Port
1450     ! This is a test to determine if the device is an HP 46085A
1460     ! or an HP 46094A. Note that both of these devices have the
1470     ! same ID number.
1480     IF (NOT BIT(Io_header,0)) THEN
1490         PRINT "One third of an HP 46085A (Control Dials) located at
            address ";Address_num
1500     ELSE
1510         PRINT "HP 46094A (HP-HIL Quadrature Port) located at address ";
            Address_num
1520     END IF
1530 CASE 104 ! HP Mouse
1540     PRINT "HP 46060A (HP Mouse) located at address ";Address_num
1550 CASE 140 ! Touchscreen
1560     PRINT "HP 35723A (Touchscreen) located at address ";Address_num
1570 CASE 147 ! Digitizer A
1580     PRINT "HP 46087A (Digitizer A) located at address ";Address_num
1590 CASE 148 ! Digitizer B
1600     PRINT "HP 46088A (Digitizer B) located at address ";Address_num
1610 CASE 149 ! Graphics Tablet
1620     PRINT "HP 45911A (Graphics Tablet) located at address ";Address_num
1630 CASE 160 TO 191 ! Integral Keyboard
1640     PRINT "Integral Keyboard located at address ";Address_num
1650 CASE 192 TO 223 ! HP 46020/21A Keyboard
1660     PRINT "HP 46020/21A (ITF Keyboard) located at address ";Address_num
1662 CASE 224 ! HP 98203C Keyboard
1664     PRINT "HP 98203C Keyboard located at address ";Address_num
1670 CASE ELSE
1680     PRINT "Unrecognized device located at address ";Address_num
1690     Unknown_dev$=IVAL$(Dev_id,16)
1700     PRINT "Device ID is ";Unknown_dev$[3]
1710 END SELECT
1720 CALL Describe_rec(Des_header,Address_num,Dev_id)
1730 !
1740 NEXT I
1750 PRINT
1760 PRINT "NO MORE DEVICES."
1770 !
1780 END
1790 !

```

The following information is an explanation of program segment 2.

*Lines 1270 to 1740* are a FOR loop which identifies all the devices specified by the Count variable. It also gives the address of these devices in the HP-HIL link.

*Lines 1350 to 1710* are a SELECT statement within the FOR loop. This statement contains CASE statements which cause a message to be printed for each type of device found on the HP-HIL link. An example of the type of message printed is as follows:

HP 46086A (Function Box) located at address 2

Assuming that there is a Function Box located at address 2 in your HP-HIL link.

Note that in *lines 1440 to 1520* a test is made for the type of device found with the ID number of 97 (decimal) because there are two devices which have that device ID number. They are the Control Dials box and the Quadrature Port. Also, in the case of a device not being recognized, *lines 1670 to 1700* will cause the following message to be displayed:

Unrecognized device located at address 2  
Device ID is 20

Assuming there is an un-recognized device located at address 2 and the ID number of that device is 20 (hexadecimal).

### Segment 3 of HIL\_ID

This segment of the program is a subprogram called Describe\_rec. The subprogram interprets each bit of the Describe Record byte to characterize the device at a specified address in the HP-HIL link. For a detail description of the Describe Record byte, read the section in the "HP-HIL Appendix" titled "Describe Record".

```
1800 !*****
1810 ! This is segment 3 of the program. It is a subprogram
1820 ! that provides information on each device. This
1830 ! information will help you determine what you can do
1840 ! with a particular device.
1850 !*****
1860 !
1870 SUB Describe_rec(INTEGER Des_header,Address_num,Dev_id)
```

```

1880   COM INTEGER Io_header
1890   !
1900   PRINT TAB(2),"Describe Record Information"
1910   IF Des_header=0 THEN
1920     PRINT TAB(5),"No special features"
1930     SUBEXIT
1940   END IF
1950   IF BIT(Des_header,2) THEN PRINT TAB(5),"Reports Security Code
information"
1960   IF BIT(Des_header,3) THEN PRINT TAB(5),"Supports the Extended
Describe command"
1970   !
1980   IF BIT(Des_header,4) THEN CALL Io_descriptor(Io_header)
1990   IF BIT(Des_header,7) THEN PRINT TAB(5),"Contains two independent
sets of coordinate axes"
2000   !
2010   SELECT Des_header MOD 4
2020   CASE 0
2030     PRINT TAB(5),"No axis information reported"
2040     SUBEXIT
2050   CASE 1
2060     PRINT TAB(5),"X axis information reported"
2070   CASE 2
2080     PRINT TAB(5),"X and Y axis information reported"
2090   CASE 3
2100     PRINT TAB(5),"X, Y and Z axis information reported"
2110   END SELECT
2120   !
2130   IF BIT(Des_header,6) THEN
2140     PRINT TAB(8),"Absolute positioning device"
2150   ELSE
2160     PRINT TAB(8),"Relative positioning device"
2170   END IF
2180   IF BIT(Des_header,5) THEN
2190     PRINT TAB(8),"Returns 16 bits/axis"
2200   ELSE
2210     PRINT TAB(8),"Returns 8 bits/axis"
2220   END IF
2230   SUBEND
2240   !

```

The following information is an explanation of program segment 3.

*Line 1900* prints the message:

Describe Record Information

This indicates that the information to follow this messages was taken from the Describe Record. Note that the Describe Record consist of up to 10 bytes of information. This information includes a Describe Record Header, Axes information, and an I/O Descriptor Byte.

*Lines 1910 through 1940* are an IF ... THEN statement which tests to see if there is information in the Describe Record Header byte . If the byte contains all zeros, then the following message is printed and the subprogram is exited:

**No special features**

If the Describe Record Header byte *is not* all zeros, then the subprogram continues to process the header information.

*Line 1950* tests bit 2 of the Describe Record Header byte to determine if the HP-HIL device reports security code information. If it *does not*, then the subprogram passes on the next test. However, if it does report security code information, then the following message is displayed:

**Reports Security Code information**

*Line 1960* tests bit 3 of the Describe Record Header byte to determine if the HP-HIL device supports the Extended Describe command. If it *does not*, then the subprogram passes on to the next test. However, if it does support the Extended Describe command, then the following message is displayed:

**Supports the Extended Describe command**

*Line 1980* tests bit 4 of the Describe Record Header byte to determine if the HP-HIL device reports I/O Descriptor information. If it *does not*, the subprogram passes on to the next test. However, if it does a CALL to the subprogram called `Io_descriptor` is made and I/O Descriptor byte information is displayed.

*Line 1990* tests bit 7 of the Describe Record Header byte to determine if the HP-HIL device contains two independent sets of coordinate axes. If it *does not*, the subprogram passes on to the next test. However, if it does the following information is displayed and the subprogram passes on to the next test:

**Contains two independent sets of coordinate axes**

*Lines 2010 through 2110* are a SELECT statement which test bits 0 and 1 for axis information. If no axis information is reported, then the following message is displayed and the subprogram is exited.



**No axis information reported**

However, if axis information is reported, you may receive one of the following messages:

**X axis information reported**

**X and Y axis information reported**

**X, Y, and Z axis information reported**

Once this test is completed program flow passes on to the next statement.

*Lines 2130 through 2170* test bit 6 of the Describe Record Header byte to determine if the device is a relative or absolute positioning device. If bit 6 is set, the following message is displayed:

**Absolute positioning device**

If bit 6 is clear (not set), the following message is displayed:

**Relative positioning device**

*Lines 2180 through 2220* complete the subprogram called Describe\_rec. They test bit 5 of the Describe Record Header byte to determine if the device returns 8 bits per axis or 16 bits per axis. If bit 5 is set, the following message is displayed:

**Returns 16 bits/axis**

If bit 5 is clear (not set), the following message is displayed:

**Returns 8 bits/axis**

#### **Segment 4 of HIL\_ID**

This segment of the program is a subprogram called Io\_descriptor. This subprogram provides information on the number of buttons the devices has and whether or not the device responds to general prompt and acknowledge commands. It also provides information on whether or not the device supports "proximity detection". Proximity detection checks for the in and out motion of a stylus or finger in relation to a digitizer or touchscreen. Io\_descriptor is called by the subprogram Describe\_rec.

```

2250 !*****
2260 ! This is segment 4 of the program. It is a
2270 ! subprogram that provides you with additional
2280 ! information on what a device can do.
2290 !*****
2300 !
2310 SUB Io_descriptor(INTEGER Io_header)
2320   PRINT TAB(5),"I/O Descriptor Information"
2330   IF Io_header=0 THEN
2340     PRINT TAB(8),"No features"
2350     SUBEXIT
2360   END IF
2370   IF BIT(Io_header,7) THEN PRINT TAB(8),"Recognizes General Prompt and
    Acknowledge"
2380   !
2390   Test_bits=(Io_header MOD 128) DIV 16
2400   SELECT Test_bits
2410   CASE 0
2420     PRINT TAB(8),"Does not support Prompts/Acknowledges 1 thru 7"
2430   CASE 1
2440     PRINT TAB(8),"Supports Prompt/Acknowledge 1"
2450   CASE 2
2460     PRINT TAB(8),"Supports Prompts/Acknowledges 1 and 2"
2470   CASE ELSE
2480     PRINT TAB(8),"Supports Prompts/Acknowledges 1 thru
    "&VAL$(Test_bits)
2490   END SELECT
2500   !
2510   IF BIT(Io_header,3) THEN PRINT TAB(8),"Supports Proximity Detection"
2520   !
2530   Test_bits=(Io_header MOD 8)
2540   SELECT Test_bits
2550   CASE 0
2560     PRINT TAB(8),"Does not report buttons"
2570   CASE 1
2580     PRINT TAB(8),"Reports 1 button"
2590   CASE 2
2600     PRINT TAB(8),"Reports buttons 1 and 2"
2610   CASE ELSE
2620     PRINT TAB(8),"Reports buttons 1 thru "&VAL$(Test_bits)
2630   END SELECT
2640 SUBEND

```

9

The following information is an explanation of program segment 4.

*Line 2320* displays the following message:

**I/O Descriptor Information**

*Lines 2330 through 2350* test the I/O Descriptor byte to determine if it contains any information. If this byte *is not* all zeros, then the subprogram continues on with the next test. However, if it *does* contain all zeros, then the message given below is displayed and program execution is passed to the subprogram (`Describe_rec`) which called it.

**No features**

*Line 2370* tests bit 7 of the I/O Descriptor byte (`Io_header`) to determine if the device being tested recognizes General Prompt and Acknowledge commands. A message is displayed if the test is true; otherwise, no message is displayed and the subprogram continues with the next test. The message displayed is:

**Recognizes General Prompt and Acknowledge**

*Line 2390* does a MOD 128 of the I/O Descriptor byte to mask off the bits above bit 6 of the byte and then does a DIV 16 to mask off the lower four bits of the byte and shifts the remaining bits to the right. The variable `Test_bits` is assigned the result of the above operation. The result is a value in the range of 0 through 7. Note that `Test_bits` is the decimal value of bits 4 through 6 of the IO Descriptor byte.

*Lines 2400 through 2490* are a SELECT statement which tests bits 4 through 6 of the I/O Descriptor byte to determine if Prompts/Acknowledges 1 through 7 are supported by the device being tested. If they *are not* supported, the following message is displayed:

**Does not support Prompts/Acknowledges 1 thru 7**

If they are supported, you will receive one of the following messages depending upon how many Prompts/Acknowledges your device supports:

Supports Prompt/Acknowledge 1

Supports Prompts/Acknowledges 1 and 2

Supports Prompts/Acknowledges 1 thru 3

Supports Prompts/Acknowledges 1 thru 4

Supports Prompts/Acknowledges 1 thru 5

Supports Prompts/Acknowledges 1 thru 6

Supports Prompts/Acknowledges 1 thru 7

*Line 2510* tests bit 3 of the I/O Descriptor byte to determine if the device being tested supports proximity detection. A message is displayed if the test is true; otherwise, no message is displayed and the subprogram continues with the next test. The message displayed is:

Supports Proximity Detection

*Line 2530* does a MOD 8 of the I/O Descriptor byte to mask off the bits above bit 2 the byte. The variable `Test_bits` is assigned the result of the above operation. The result is a value in the range of 0 through 7. Note that `Test_bits` is the decimal value of bits 0 through 2 of the IO Descriptor byte.

*Lines 2540 through 2630* are a SELECT statement which tests bits 0 through 2 of the I/O Descriptor byte to determine if Buttons 1 through 7 are reported by the device being tested. If they *are not* reported, the following message is displayed:

Does not report buttons

If they are reported, you will receive one of the following messages depending upon how many buttons your device reports:

Reports 1 button

Reports buttons 1 and 2

Reports buttons 1 thru 3

Reports buttons 1 thru 4

Reports buttons 1 thru 5

Reports buttons 1 thru 6

Reports buttons 1 thru 7

## **HP-HIL Devices**

A brief description will be provided for those devices supported by HIL SEND, ON HIL EXT, and HILBUF\$. For those devices not supported by these statements and function, there will be a reference given to help you locate further information on that device, as well as statements you may use to interact with it.

HP-HIL devices have been divided into the following categories:

- HP-HIL Keyboards
- Relative Positioners
- Absolute Positioners
- Security Device (the ID Module)
- Other Devices (i.e., Keyboards, Button Devices, Bar Code Reader)

## HP-HIL Keyboards

There are three HP-HIL keyboards supported (as Keyboards) on the HP-HIL link. They are the:

- HP 46020/21A (for information on this keyboard see the *Using BASIC* manual for your system.
- HP 98203C (this keyboard is not supported on BASIC/UX) (for information on this keyboard see the manual titled *Using HP BASIC/WS*).
- Integral Keyboard (for information on this keyboard see the *HP-UX Technical BASIC Getting Started Guide*).

These keyboards will not cause ON HIL EXT interrupts. Using HIL SEND to transmit a command (other than IDD) to one of these devices will cause an error only for the HP 98203C keyboard which is also a relative positioning device (see next section). To do interrupt branching with the keyboard keys, you need to use the following statements and function:

ON/OFF KEY	ON KEY defines and enables an event-initiated branch to be taken when a softkey is pressed. OFF KEY cancels event-initiated branches previously defined and enabled by an ON KEY statement. Without the KBD binary, subsequent softkey presses cause beeps. With the KBD binary, the action of subsequent softkey presses depends upon the typing-aid definitions.
ON/OFF KBD	ON KBD defines and enables an event-initiated branch to be taken when a key is pressed. OFF KBD cancels event-initiated branches previously defined and enabled by an ON KBD statement. Subsequent key presses are sent to the operating system in the normal manner.
KBD\$	This function returns the contents of the keyboard buffer when ON KBD is active.

For more information on keyboards, refer to the *Using BASIC* manual for your system.

## Relative Positioners

These devices will not cause ON HIL EXT interrupts. Using HIL SEND to transmit a command (other than IDD) to one of these devices will cause an error. Relative positioners can be categorized into two groups: those that are two axis devices and those that are three axis devices. A list of these devices and their statements and functions is given below.

Examples of two-axis relative positioning devices are:

- HP 46060A/B (HP-Mouse)
- HP 46083A (Rotary Control Knob)
- HP 46094A (HP-HIL/Quadrature Port)
- HP 98203C (this keyboard is not supported on BASIC/UX)

These devices support the following statements and functions. Note that in the case of the HP 46094A (HP-HIL/Quadrature Port) it supports statements and functions appropriate to the quadrature device connected to it (e.g., the HP 46095A 3-button Mouse).

**ON/OFF KBD**                      ON KBD defines and enables an event-initiated branch to be taken when a key is pressed. OFF KBD cancels event-initiated branches previously defined and enabled by an ON KBD statement. Subsequent key presses are sent to the operating system in the normal manner.

**KBD\$**                              This function returns the contents of the keyboard buffer.

**ON/OFF KNOB**                    ON KNOB defines and enables an event-initiated branch to be taken when the knob is turned. OFF KNOB cancels event-initiated branches previously defined and enabled by an ON KNOB statement. Subsequent use of the knob results in normal scrolling or cursor movement.

**KNOBX and KNOBY**              return the counts accumulated for X and Y motions of the relative positioning devices.

## DIGITIZE

This statement is used when:

```
GRAPHICS INPUT IS KBD,"KBD"
```

It inputs the X and Y coordinates of a digitized point from the locator specified by GRAPHICS INPUT IS.

## READ LOCATOR

This statement is used when:

```
GRAPHICS INPUT IS KBD,"KBD"
```

It samples the locator device, without waiting for a digitizing operation.

For more information on these statements, refer to the *HP BASIC 6.2 Language Reference*, and the *HP BASIC 6.2 Programming Guide*.

There are also three-axis devices, for example the HP 46085A (Control Dials) module contains three such devices. This device supports the following statements and function:

## ON/OFF CDIAL

ON CDIAL enables end-of-line interrupts in response to the rotation of one or more knobs on the HP-HIL Control Dials device. While such interrupts are enabled, pulses (rotation counts) are accumulated and returned via the CDIAL function (see below). OFF CDIAL cancels end-of-line interrupts previously enabled by an ON CDIAL statement. After an OFF CDIAL statement, left over counts may be read via CDIAL (but only once), and no further accumulation occurs.

## CDIAL (*Counter*)

This function is used to return counts from the Control Dials module or other 3-axis relative positioning devices. It is linked to a status word and 15 *counters*. Each of the 15 high order bits in the status word corresponds to one counter, the first counter being represented by bit 1 of the status word (bit 0 of the status word is unused). Normally the counters one through nine correspond to the nine knobs on the Control Dials module. Counter 1 is the knob in the lower left-hand corner of the module. The remaining counters are numbered from left to



right. The status word and counters are zeroed when an ON CDIAL statement is executed. Thereafter, whenever a count arrives from any of the knobs, the corresponding counter is incremented and its status bit is set. Reading a counter zeros both the counter and its bit in the status word. Reading the status word does not change its value. The status word is read as CDIAL(0).

For more information on these statements see the *HP BASIC 6.2 Programming Guide*.

Note that when ON CDIAL interrupts are disabled, three-axis devices may be used with the two-axis statements and functions.

### **Absolute Positioners**

These devices can generate ON HIL EXT interrupts, but will not when:

```
GRAPHICS INPUT IS KBD,"TABLET"
```

is in effect. Moreover, due to the speed which data is returned from the digitizers, a BASIC program cannot keep up with them when using ON HIL EXT (HILBUF\$ overflows). Therefore, the only device in this group capable of using the ON HIL EXT statement is the Touchscreen. Using HIL SEND to transmit a command (other than IDD) to these devices while:

```
GRAPHICS INPUT IS KBD,"TABLET"
```

will result in an error.

The following statements should be used when:

```
GRAPHICS INPUT IS KBD,"TABLET"
```

is in effect.

```
DIGITIZE X_coord,Y_coord
```

inputs the X and Y coordinates of a digitized point.

```
READ LOCATOR X_coord,Y_coord
```

samples the locator device without waiting for a digitize operation.

For more information on these statements refer to the *HP BASIC 6.2 Programming Guide*. An explanation of each of these statements may also be found in the *HP BASIC 6.2 Language Reference*.

The following are absolute position devices:

- HP 35723A (HP-HIL/Touchscreen)—This module is a screen bezel which replaces the bezel of the HP 35731 (medium resolution black and white) and HP 35741 (medium resolution color) 12-inch video monitors. It can be programmed to select various functions by simply touching the screen. Note that this device is simply a lower resolution digitizer. The Touchscreen can be used as a GRAPHICS INPUT IS device or with the ON HIL EXT statement.
- HP 45911A (11 × 11 Graphics Tablet)—This device is best used as a GRAPHICS INPUT IS device.
- HP 46087A (A-size Digitizer)—This device is best used as a GRAPHICS INPUT IS device.
- HP 46088A (B-size Digitizer)—This device is best used as a GRAPHICS INPUT IS device.

If a three-axis absolute positioning device existed, it could always be used with HIL SEND and ON HIL EXT since it would not be recognized for use with:

```
GRAPHICS INPUT IS KBD,"TABLET"
```

## Security Device

The HP 46084A (HP-HIL ID Module) is an HP-HIL device that returns an identification number for identifying you as the computer user. The identification number is unique to your particular ID Module. This allows application programs to use the ID Module to control access to program functions, data bases, and networks. Note that the identification number is the product/exchange and serial numbers returned in a packed format as explained in the section "ID Module" found in this chapter.

This device can be used with SYSTEM\$("SERIAL NUMBER") or HIL SEND *device address*;RSC.

## Other Devices

These devices can generate ON HIL EXT interrupts and respond to various HIL SEND commands. They all have HP-HIL device IDs less than 96 (60 hexadecimal).

The HP 46086A (Function Box) provides 32 keys to select software-defined functions. It has an LED that acts as a visual prompt for any purpose you assign to it. This device uses a non-standard keycode set (Keycode Set 2) which is shown below.

	0/1	2/3	4/5	6/7		
8/9	10/11	12/13	14/15	16/17	18/19	
20/21	22/23	24/25	26/27	28/29	30/31	
32/33	34/35	36/37	38/39	40/41	42/43	
44/45	46/47	48/49	50/51	52/53	54/55	
	56/57	58/59	60/61	62/63		

**Keycode Set 2 for the Function Box  
(press value/release value)**

The HP 46086A (Function Box) responds to the following HP-HIL commands when sent by the HIL SEND statement:

- PRM
- ACK
- DKA
- EKA 1
- EKA 2

The HP 46030A (Vectra Keyboard) provides 103 keys to select software-defined functions. It has three LEDs which act as visual prompts for any purpose you assign to them. This device uses Keycode Set 3 which is described in the "HP-HIL Appendix." This keyboard is not supported on BASIC/UX.

The Vectra Keyboard responds to the following HP-HIL commands when sent by the HIL SEND statement:

- PRM 1 through 3

- ACK 1 through 3
- DKA
- EKA 1
- EKA 2

For BASIC/WS, to use an HP 46030A (Vectra Keyboard) as an auxiliary input device, you *must* have a computer capable of using the HP 98203C keyboard. It need not have an HP 98203C keyboard.

The HP 92916A (Bar-Code Reader) reads all standard bar-codes using a wand as the input device. It provides you with an effective and reliable alternative to the time consuming keyboard for data entry. Note that BASIC supports this device in both the ASCII transmit mode, where the input from the device is ASCII characters, and in the Keyboard mode, where it transmits the same keycodes as an HP 46020/21A Keyboard. (In Keyboard mode, this device also returns an HP-HIL ID in the same range as an HP 46020/21A Keyboard.) The codes that can be read by the Bar-Code Reader are: 3 of 9, Interleaved 2 out of 5, UPC/EAN, and Codabars USD-4 and ABC.

When the HP 92916A (Bar-Code Reader) is in the ASCII transmit mode use the following statement:

**ON HIL EXT**

When the HP 92916A (Bar-Code Reader) is in the Keyboard mode use the following statements:

**ON KBD**

**ENTER KBD**

**INPUT**

**LINPUT**

---

## Communicating with HP-HIL Devices

This section of the chapter covers the use HP-HIL devices which support the HIL SEND and ON HIL EXT statements. In the examples covered in this section, you will be looking at four HP-HIL devices and how to use them in the HP-HIL link.

- ID Module
- Function Box
- Touchscreen
- Bar Code Reader

### HP-HIL Device Characteristics

Once the HP-HIL device is in the link, you will need to verify its address and determine its characteristics. Accessing this information is the purpose of this section.

To verify a device's address and determine its characteristics, use the HIL SEND *address*;IDD statement and HILBUF\$ function. The HIL SEND *address*;IDD statement executes the HP-HIL Identify and Describe command. Data resulting from the execution of this command is placed in the buffer used by the HILBUF\$ function. Assuming that the address of your device is 1, entering this program and running it will give you the information you need. Note that the information returned is hexadecimal and will have to be interpreted using the information found in the "HP-HIL Appendix" of this manual.

```
100 HIL SEND 1;IDD
110 A$=HILBUF$
120 FOR I= 1 TO LEN(A$)
130   B$=IVAL$(NUM(A$[I]),16)
140   PRINT B$[3];" ";
150 NEXT I
160 END
```

Results from executing this program can be found under the topic heading "Device Characteristics" in each of these sections:

- ID Module
- Function Box and Vectra Keyboard
- Touchscreen
- Bar Code Reader

## ID Module

This module provides a means for securing your software. In this section, you will be:

- Determining ID Module characteristics,
- Verifying your ID Module's product/exchange and serial numbers,
- Learning how to install and remove the ID Module.

### Device Characteristics

This section provides and explains the results from executing the program found in the section titled "HP-HIL Device Characteristics". Remember that these results assume your ID Module is located at address 1. The program results are as follows:

```
00 04 01 34 04
```

where:

- 00 is a buffer overflow count. Zero means the buffer has not overflowed since last read. If the buffer of the HILBUF\$ function overflowed, this value would represent the number of packets of information lost.
- 04 is the number of bytes of data to follow this byte and including this byte. The number of bytes is 4.
- 01 is the address of the device in the loop. The address of the device in this case is 1 which means that it is the first device in the link with an address.
- 34 is the type of device located at the address given. The device in this case, as interpreted from the "Device ID Byte Definitions" table found in the "HP-HIL Appendix" in this manual, is the ID Module.
- 04 is the Describe Record for the device. This record helps you determine the device characteristics. To interpret this hexadecimal value, you need to turn to the "HP-HIL Appendix" found in this manual. Looking in the section titled "Describe Record", you will find that if bit 2 is set then the device reports security code information.

## Interpreting ID Module Data

In this section, you will learn how to verify the product/exchange and serial numbers for your ID Module.

To verify your product/exchange number, type in and execute the following program:

```
100 Sn$=SYSTEM$("SERIAL NUMBER")
110 OUTPUT Sn_disp$ USING "9D";256*(256*(256.*(NUM(Sn$[8]) MOD 64)
+NUM(Sn$[7]))+NUM(Sn$[6]))+NUM(Sn$[5])
120 PRINT VAL$(256*(256.*BIT(NUM(Sn$[4]),7)+NUM(Sn$[3]))+NUM(Sn$[2]))
&CHR$(NUM(Sn$[4]) MOD 128),Sn_disp$[1,4]&CHR$(NUM(Sn$[9])
MOD 128)&Sn_disp$[5]
130 END
```

The results from executing the above program look similar to this:

```
46084A    2529A10988
```

The same results can be obtained using the HP-HIL Report Security Code command (RSC) in the above program. This requires replacing program line 100 with three additional program lines as shown below. Note that you may need to replace line 120 with additional statements if your program is also using HILBUF\$ to return other data.

```
100 HIL SEND 3;RSC
110 Temp_sn$=HILBUF$
120 Sn$=Temp_sn$[4,12]
130 OUTPUT Sn_disp$ USING "9D";256*(256*(256.*(NUM(Sn$[8]) MOD 64)
+NUM(Sn$[7]))+NUM(Sn$[6]))+NUM(Sn$[5])
140 PRINT VAL$(256*(256.*BIT(NUM(Sn$[4]),7)+NUM(Sn$[3]))+NUM(Sn$[2]))
&CHR$(NUM(Sn$[4]) MOD 128),Sn_disp$[1,4]&CHR$(NUM(Sn$[9])
MOD 128)&Sn_disp$[5]
150 END
```

## Note about Installing and Removing ID Modules

The HP 46084 (ID Module) is an HP-HIL device which connects to the computer through the HP-HIL (HP Human-interface Link) interface. Normally you will be connecting this module to the computer before booting the system. When the KBD binary is loaded, the system recognizes that the module is installed. The SYSTEM\$ function reads the module's contents each time the function is accessed, rather than keeping the contents in memory.

The ID Module can also be installed while the computer is running. However, in order for BASIC to recognize that it has been connected, you must execute this statement:

**SCRATCH A**

Executing this statement performs a "re-configuration" of the link, after which the BASIC system recognizes and can properly talk to any additional HP-HIL device.

If your machine has both an ID PROM and an ID Module, the ID Module has precedence. In other words, if both are installed (and recognized at boot or SCRATCH A), then the ID Module's contents are read and returned by the SYSTEM\$ function.

If you remove the ID Module and do not re-boot or execute SCRATCH A, then the SYSTEM\$ function will return a null string (even if an ID PROM is present). This behavior is due to the fact that the system still expects the ID Module to be installed, and thus reads nothing when you attempt to read it with SYSTEM\$.

Conversely, if you install an ID Module in a machine with an ID PROM after booting BASIC and *without* performing a SCRATCH A, then SYSTEM\$("SERIAL NUMBER") will return the ID PROM's contents (because it does not recognize that the ID Module is present).



## Function Box and Vectra Keyboard

This section deals mainly with the Function Box and not the Vectra Keyboard. However, to use the Vectra Keyboard with BASIC/WS you would use the same techniques as used for the Function Box. The main difference between the two devices are the number of keys and the keycode sets used. The Vectra Keyboard has 103 keys and uses Keycode Set 3 found in the “HP-HIL Appendix.” The Function Box has 32 keys and uses Keycode Set 2 which is found in the section entitled “Other Keyboards and Button Devices.” Note that in order to use a Vectra Keyboard as an auxiliary input device, you must have a computer capable of using the HP 98203C Keyboard (or any keyboard for that matter). This section covers the following topics:

- Determining Function Box characteristics.
- Activating the Function Box.
- Trapping key presses.
- Assigning functions to keys.

### Determining Function Box Characteristics

This section provides and explains the results from executing the program found in the section titled “HP-HIL Device Characteristics”. These results assume your ID Module is located at address 1. The program results are as follows:

```
00 05 01 30 10 80
```

where:

- 00 is an overflow indicator. If the buffer to the HILBUF\$ function overflowed, this value would represent the number of packets of information lost.
- 05 is the number of bytes contained in the packet of information sent to the buffer used by the HILBUF\$ function including that byte.
- 01 is the address of the device within the HP-HIL link. The address of the device is 1 in this example.
- 30 is the ID number of the device. This number helps to determine what devices are connected in the HP-HIL link. Hexadecimal 30 is the ID number for the Button Box as found in the table titled “Device ID Byte Definitions” in the “HP-HIL Appendix.”

- 10 is the Describe Record Header. It returns information, such as what type of HP-HIL commands are supported by this device, proximity in/out information, and coordinate information. By use of the Describe Record Header information provided in the "HP-HIL Appendix" you will be able to interpret the information contained in this byte. In this case, the 4th bit of the Describe Record byte is set which indicates that the last byte in the packet of information is the I/O Descriptor Byte.
- 80 is the I/O Descriptor Byte. This byte contains information as found in the "I/O Descriptor Byte" table in the "HP-HIL Appendix." You will find that bit 7 of this byte has been set. This indicates that the HP-HIL General Prompt and Acknowledge are supported by this device.

### **Activating the Function Box**

A status light is located in the upper-right corner of your Function Box. You could use this light to indicate whether the buttons on the Function Box are active or non-active. The following program which is titled "Activate" can be found in the directory called `/usr/lib/rmb/demo`. Note that the program assumes your Function Box's address is 2. You may have to change this address if your Function Box's address is different from that found in the program.

```

100 CLEAR SCREEN
110 DISP "Do you want to activate the Function Box?";
120 DISP " Enter Yes or No.";
130 INPUT "",Response$
140 IF LWC$(Response$[1,1])="y" THEN
150     HIL SEND 3;PRM
160     ON HIL EXT 8 CALL Key_service
170     PRINT TABXY(15,10),"The status light is on and keys are active."
180 ELSE
190     HIL SEND 3;ACK
200     OFF HIL EXT
210     PRINT TABXY(15,10),"The status light is off and keys are not active."
220 END IF
230 Loop: GOTO Loop
240 END
250 !
260 SUB Key_service
270     PRINT "Key_service called."
280 SUBEND

```

This program executes the HP-HIL General Prompt and Acknowledge commands using the statements found on lines 150 and 190 of the above program. When the program is run you are prompt by the following message:

Do you want to activate the Function Box? Enter Yes or No.

You need to type in either Yes or No. If your answer is Yes, the status light on the Function Box lights up and this message is displayed:

The status light is on and keys are active.

Each time a button on the Function Box is pressed or released, this message is displayed:

Key\_service called.

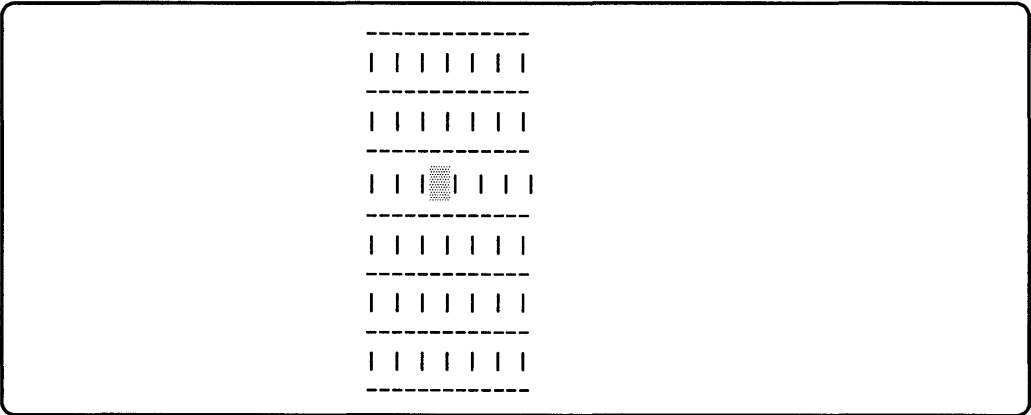
Note that the subprogram called Key\_service just prints a message that it has been called. It is left up to you to write your own subprograms to assign processes to the keys. If you answered No to the prompt, the status light either remains off if it was already off or is turned off if it was on and the following message is displayed:

The status light is off and keys are not active.

Press the **PAUSE** key which pauses the program and allows you to re-run it or to go on to the next example.

## Trapping Key Presses

Key presses are recognized as interrupts and cause end-of-line branching when the `ON HIL EXT` statement is executed in your program. Recognition of a key press and then branching to a subprogram is called “trapping” a key press. The program given in this section provides a good example of trapping key presses. This program is called “Mul\_press” and can be found in the directory called `/usr/lib/rmb/demo`. When you enter and run this program a Function Box key matrix is displayed on the screen. Each time you press a key that key’s location in the key matrix is displayed on the screen. Note that you should only press one key at a time because the Function Box *does not* provide for multi-key presses. For example, pressing key number 12 on the Function Box results in the following being displayed:



Releasing the same key you pressed causes key 12’s matrix location to go blank. The following program titled “Mul\_press” produced the above results. Note that this program assumes a Function Box address of 3. If your Function Box is not located at address 3, then you need to change the addresses on both lines 120 and 290 of the program to the proper address for your Function Box.

```
100 CLEAR SCREEN  
110 ! Assumes button box at location 3.  
120 HIL SEND 3;DKA ! Disable Key Auto-repeat.  
130 !  
140 COM INTEGER Array(31,1:2)
```

```

150  INTEGER Key,Packet_length,Packet_start,Index,Packet_end
160  INTEGER Keycode
170  DIM A$(256)
180  !
190  DATA 2,1,3,1,4,1,5,1
200  DATA 1,2,2,2,3,2,4,2,5,2,6,2
210  DATA 1,3,2,3,3,3,4,3,5,3,6,3
220  DATA 1,4,2,4,3,4,4,4,5,4,6,4
230  DATA 1,5,2,5,3,5,4,5,5,5,6,5
240  DATA 2,6,3,6,4,6,5,6
250  !
260  READ Array(*)
270  Framework
280  !           enable device at location 3 (button box)
290  ON HIL EXT 2^3 GOSUB Service_req
300  !
310  Loop:GOTO Loop
320  !
330  Service_req:!
340  A$=HILBUF$
350  IF LEN(A$)=1 THEN RETURN ! no data in buffer
360  Packet_start=2
370  REPEAT
380    Packet_length=NUM(A$(Packet_start))
390    Packet_end=Packet_start+Packet_length-1
400    FOR Index=Packet_start+3 TO Packet_end
410      Keycode=NUM(A$(Index))
420      Key=(Keycode DIV 2)
430      Disp_key(Key,NOT BIT(Keycode,0))
440    NEXT Index
450    Packet_start=Packet_end+1
460  UNTIL Packet_start>LEN(A$)
470  RETURN
480  !
490  END
500  SUB Disp_key(INTEGER N,On)
510    COM INTEGER Array(*)
520    IF On THEN           ! Even='downstroke'.
530      PRINT TABXY(2*Array(N,1)+17,2*Array(N,2)+4),"";
540    ELSE           ! Odd=>'upstroke'.
550      PRINT TABXY(2*Array(N,1)+17,2*Array(N,2)+4)," ";
560    END IF
570  SUBEND
580  SUB Framework
590    FOR I=0 TO 10 STEP 2

```

```

600     PRINT TABXY(18,I+5);"-----"
610     PRINT TABXY(18,I+6);"| | | | | |"
620     NEXT I
630     PRINT TABXY(18,17);"-----"
640     SUBEND

```

Here is an explanation of the above program.

*Line 100* clears the display.

*Line 120* disables the auto-keyswitch repeat mode by executing the HP-HIL command DKA.

*Lines 140 through 170* declare the variables for the program.

*Lines 190 through 240* provide values for the element locations in the two dimensional array called `Array`. Note that `OPTION BASE 0` is used for this program.

*Line 260* is a `READ` statement which assigns all of the values in the `DATA` statements of lines 190 to 240 to the elements in the array called `Array`.

*Line 270* calls the subprogram `Framework` which causes a 6 by 6 Function Box key matrix to be displayed on the screen. The subprogram `Framework` consist of lines 580 to 640.

*Line 290* enables end-of-line branching when a key on the Function Box is pressed.

*Line 310* is a continuous loop which allows the program to wait for key presses.

*Line 330* is the label for the beginning of the service routine called `Service_req`.

*Line 340* assigns the value of the buffer used by the function `HILBUF$` to the string array called `A$`.

*Line 350* tests the string length. If the string length is 1, then the data that generated this interrupt has already been read and a return from the subroutine is made.

*Line 360* assigns the value of 2 to the integer variable `Packet_start`. Note that `Packet_start` initially is the subscript for the second element in `A$`. This element tells how many elements there are in the first packet of information

including that element. This variable will also be the counter used to determine the starting position of each packet of information in the string (A\$).

*Lines 370 through 460* are a REPEAT loop used to search the data in the string (A\$) for each packet of key press information. *Lines 400 through 440* scan the packet for the up or down key presses. *Line 430* detects the up or down key press and passes this parameter to the subprogram called `Disp_key`. Note that the integer variable `Key` is the key which was either pressed or released.

*Line 470* is the return back from the subroutine.

*Lines 500 through 570* are the subprogram called `Disp_key`. This subprogram has a test in it for an up or down press of a key on the Function Box. Each time you press a key that key's location in the key matrix is displayed on the screen as an inverse video character. When you release that key a blank appears in the key matrix.

*Lines 580 through 640* are the subprogram called `Framework` which draws the key matrix on the display.

## Assigning Functions to Keys

It was previously mentioned that processes or functions can be assigned to each key on the Function Box. These functions are not assigned in the same manner as those assigned to typing-aids keys nor do they have softkey labels which appear at the bottom of the display.

A function is assigned by pressing a key which causes an interrupt. This interrupt is trapped and causes a branch to a subprogram which sets a process in motion. Once the process is completed the subprogram returns execution back to the main program and waits for another key press. An example of this can be seen by `LOADing` and executing the following program called "Button\_box" can be found in the directory called `/usr/lib/rmb/demo`. Keep in mind that only two keys are being used in this program. These keys are located in the top row starting from the left. Pressing the first key stops the program, pressing the second key draws a series of circles. Any other key press causes the following message to appear on the display:

This key is not implemented.

Note that you must have the graphics binary (GRAPH) loaded in order for this program to work. Also, lines 120, 130, and 400 may have to be changed if your Function Box is not located at address 3 in order for the program to work.

```
100  INTEGER Packet_length,Packet_start,Packet_end
110  DIM A$[256]
120  ON HIL EXT 2^3 GOSUB Service_req
130  HIL SEND 3;PRM
140  CLEAR SCREEN
150  GINIT
160  PEN 0
170  GRAPHICS ON
180  Loop:GOTO Loop
190  !
200  Service_req: !
210  A$=HILBUF$
220  IF LEN(A$)=1 THEN RETURN
230  Packet_start=2
240  REPEAT
250    Packet_length=NUM(A$[Packet_start])
260    Packet_end=Packet_start+Packet_length-1
270    FOR Index=Packet_start+3 TO Packet_end
280      Key_check(NUM(A$[Index]))
290    NEXT Index
300    Packet_start=Packet_end+1
310  UNTIL Packet_start>LEN(A$)
320  RETURN
330  !
340  Prog_done:END
350  !
360  SUB Key_check(INTEGER Key_num)
370    SELECT Key_num
380    CASE 0,1
390      DISP "The program has STOPPED!"
400      HIL SEND 3;ACK
410      STOP
420    CASE 2
430      MOVE 50,50
440      FOR I=1 TO 20
450        POLYGON I,20,20
460      NEXT I
470      FOR I=20 TO 1 STEP -1
480        POLYGON I,20,20
490      NEXT I
500      GCLEAR
```



```

510     CASE ELSE
520         IF (Key_num MOD 2)=0 THEN
530             PRINT TABXY(20,10),"This key is not implemented."
540             WAIT 1
550             CLEAR SCREEN
560         END IF
570     END SELECT
580 SUBEND

```

The following is an explanation of the above program. This program assumes your Function Box is located at address 3 in the HP-HIL link.

*Lines 100 and 110* declare the integer and string variables.

*Line 120* executes the statement `ON HIL EXT 8` which sets up a branch to be made to the subprogram `Service_req`. At the same time this branch is initiated Poll Record data is sent to the buffer used by the function `HILBUF$`. This data contains information on which key was pressed. You can trap these key presses and use them to activate various process.

*Line 130* turns on the status light of the Function Box.

*Line 140* clears the alpha display.

*Line 150* set the graphics parameters to their default values.

*Line 160* sets the graphics pen value to zero (0).

*Line 170* turns the graphics display on.

*Line 180* causes the program to loop until a key is pressed.

*Line 200* is the label for the beginning of the service routine called `Service_req`.

*Line 210* assigns the value of the buffer used by the function `HILBUF$` to the string called `A$`.

*Line 220* tests the string length. If the string length is 1, then the data associated with this interrupt has already been processed, and a return from the subroutine is made.

*Line 230* assigns the value of 2 to the integer variable `Packet_start`. Note that `Packet_start` initially is the subscript for the second element in `A$`. This element tells how many elements there are in the first packet of information

including that element. This variable will also be the counter used to determine the starting position of each packet of information in the string (A\$).

*Lines 240 through 310* are a REPEAT loop used to search the data in the string (A\$) for each packet of key press information. Lines 270 through 290 search the packet for up or down key presses. Line 280 calls the subprogram `Key_check` and passes it the value of the key you have pressed.

*Line 320* is the return back from the subroutine.

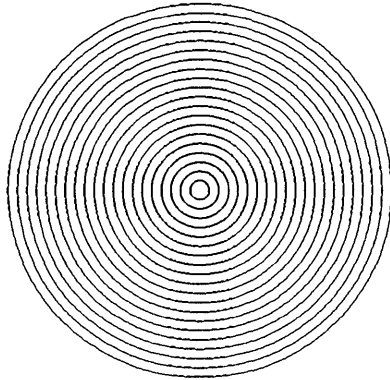
*Lines 360 through 580* are the subprogram called `Key_check`. This subprogram is a large SELECT structure starting at line 370 and going to line 570. This structure selects a particular process to be performed depending on which key has been pressed. One process can be found in each of the three different CASE segments.

*Lines 380 through 410* are the first CASE segment. This segment when executed causes the following message to be displayed:

**The program has STOPPED!**

It also executes the HP-HIL command `ACK` (General Acknowledge) which turns the status light on the Function Box off and terminates the program.

*Lines 420 through 500* are the second CASE segment. This segment causes circles to be displayed one inside the other starting with a small circle and going to a large one. It then erases these circles in the reverse order.



*Lines 510 to 560* are the third CASE segment. All key releases come through this CASE segment and are ignored due to line 520. This includes the release of key 2. Any key press coming here causes the following message to be displayed:

**This button is not implemented.**

Remember there are only two keys whose interrupts were recognized as a result of running this program. When you press one of the keys which does not cause a process to become activated the above message is displayed.

## Using a Touchscreen

As its name indicates, the Touchscreen responds to a touch of the screen. A touch of the screen will report you are in proximity and a release of this touch will report you are out of proximity. At the same time this device is reporting in and out proximity information it is also returning X and Y axis coordinate information for the screen touch. Combining both of these characteristics, the user is able to do location selection using the Touchscreen. Below is a list of the topics covered in this section:

- Determining Touchscreen characteristics,
- Plotting selected locations.

### Device Characteristics

Assuming the Touchscreen is located at address 2, it will return Identify and Describe information as follows:

```
00 0B 02 8C 52 0A 01 38 00 2A 00 08
```

where:

- 00 is the overflow counter.
- 0B indicates the number of bytes of data to follow including this byte (in this case there are 11).
- 02 is the address of the device.
- 8C is the device type. In this case, it is the Touchscreen as determined from the table titled, "Device ID Byte Definitions."
- 52 is the Describe Record. This gives information about the device. The bit pattern for a Describe Record of 52 is as follows:
  - Bit 0 is not set and bit 1 is. This says the device will return X and Y coordinates.
  - Bit 4 is set. This indicates that the last byte of the Describe Record is the I/O Descriptor byte.
  - Bit 5 is not set. This indicates that the X and Y coordinates returned will only be 8 bits each (one byte).

- Bit 6 is set. This indicates that Absolute Positional data will be returned by the device.

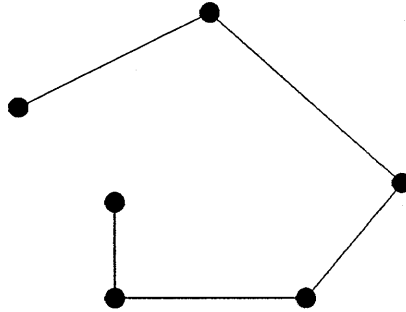
- 0A01 These two bytes are combined to give 010A (i.e., the 2nd byte is the more significant part of the number). Since bit 5 of the Describe Record is not set, this value is the number of counts per meter (in this case 266).
- 3800 These two bytes are combined to give 0038. (i.e., the 2nd byte is the more significant part of the number). This value represents the total number of absolute graphics units in the X-axis (in this case 56).
- 2A00 These two bytes are combined to give 002A. (i.e., the 2nd byte is the more significant part of the number). This value represents the total number of absolute graphics units in the Y-axis (in this case 42).
- 08 is the I/O Descriptor Byte. Bit 3 of this byte is set indicating that the device indicates changes in proximity in or out status in its Poll Record (only returned when the status changes).

### Plotting Selected Locations

This task requires the use of the statement **ON HIL EXT**. Information returned by the Touchscreen can be found in the buffer used by the **HILBUF\$** function.

The following program called "Touch\_plot", found in the directory called `/usr/lib/rmb/demo`, continuously displays the X and Y coordinates of your finger or stylus as you move it across the screen. The first release of your touch on the screen will cause a **MOVE** to that position. Any subsequent screen releases will cause a line to be draw from the last coordinate position to the present one. This particular program only allows you to plot and draw lines to 6 different locations on the screen.

Below are some sample results which you could receive from entering and running the program in this section.



Point 6 : X=40 Y=40

The following program called "Touch\_plot" returned the above results. Note that in order for this program to work on your BASIC system you need to change the address on line 210 to the address of your Touchscreen. The address currently assigned to this HP-HIL device in the program is 1.

```

100 CLEAR SCREEN ! Clear alpha.
110 GINIT          ! Initializes graphics.
120 GRAPHICS ON   ! Turn on graphics.
130 WINDOW 0,56,0,43 ! Scale to match Touchscreen resolution.
140 !

150 INTEGER Test,Point,Packet_start,Packet_length,Packet_end
160 INTEGER In_proximity,X_coord,Y_coord
170 DIM A$(256)
180 !
190 PRINT TABXY(16,12),"Touch the screen at 6 different locations."

```

```

200  !
210  ON HIL EXT 2 GOSUB Service_req ! Assumes the Touchscreen is the
220                                     ! first device on the link.
230  Point=1
240  !
250  Loop:GOTO Loop
260  !
270  Service_req:  !
280  IF Point=1 THEN CLEAR SCREEN
290  A$=HILBUF$
300  IF LEN(A$)=1 THEN RETURN
310  Packet_start=2
320  REPEAT
330    Packet_length=NUM(A$[Packet_start])
340    Packet_end=Packet_start+Packet_length-1
350    IF BIT(NUM(A$[Packet_start+2]),1)=1 THEN
360      X_coord=NUM(A$[Packet_start+3])
370      Y_coord=NUM(A$[Packet_start+4])
380      DISP "Point ";Point;" : X = ";X_coord;" Y = ";Y_coord
390    END IF
400    IF BIT(NUM(A$[Packet_start+2]),6)=1 THEN
410      In_proximity=NUM(A$[Packet_end])
420      IF In_proximity=142 THEN
430        IF Point=1 THEN
440          MOVE X_coord,Y_coord
450        ELSE      ! Point=2 thru 6.
460          DRAW X_coord,Y_coord
470        END IF
480      END IF
490    END IF
500    Packet_start=Packet_end+1
510  UNTIL Packet_start>LEN(A$)
520  IF In_proximity=143 THEN
530    Point=Point+1
540  END IF
550  IF Point<7 THEN RETURN
560  DISP "You're Done"
570  !
580  END

```

The following is an explanation of the above program.

*Line 100* clears the alpha screen.

*Line 110* initializes graphics to its default values.

*Line 120* turns graphics on.

*Line 130* sets the graphics scale to match the Touchscreen resolution.

*Lines 150 through 170* declare the program variables.

*Line 190* prompts the user to make 6 screen touches. This means moving your finger or stylus in and out of proximity 6 times.

*Line 210* enables end-of-line interrupts from the Touchscreen located at address 1.

*Line 230* initializes the counter variable to 1. The counter variable is called `Point` and it keeps track of the number of times you have released your finger or stylus from the screen.

*Line 250* is a continuous loop which allows the system to wait for either a screen touch or release.

*Lines 270 through 560* are a subroutine called `Service_req`. Whenever a touch or release of the screen is made, this service routine is called.

*Line 290* assigns the value of the buffer used by the function `HILBUF$` to the string called `A$`.

*Line 300* tests the string length. If the string length is 1, then the data associated with this interrupt has already been processed, and a return from the subroutine is made.

*Line 310* assigns the value of 2 to the integer variable `Packet_start`. Note that `Packet_start` initially is the subscript for the second element in `A$`. This element tells how many elements there are in the first packet of information including that element. This variable will also be the counter used to determine the starting position of each packet of information in the string (`A$`).

*Lines 320 through 510* are a REPEAT loop used to search the data in the string (`A$`) for each packet of screen touch and release information. Lines 350 through 490 check the packet for coordinate and proximity information. The REPEAT loop continues until the last element in the string `A$` is reached.

*Lines 400 through 480* test for proximity in and out. As long as proximity in is detected the coordinates of your present finger or stylus position are printed.

*Lines 430 through 470* determine whether to draw a line on the display or to move the graphics pen to the initial position before plotting.



*Line 560* is reached when the sixth point is plotted on the screen. This line will cause the following to be displayed:

You're Done

## Using a Bar Code Reader

A Bar Code Reader may either act as a keyboard or a transmitter of ASCII characters. In this section, you will assume it is a transmitter of ASCII characters. When your Bar Code Reader is acting as a keyboard it is returning keyboard presses. When it is acting as an ASCII transmitter it is sending ASCII characters.

To use this HP-HIL device as a reader of ASCII characters you need to program the switches on its underside for the proper settings. The settings for these keys are explained in the installation manual for this device. Below is a list of settings you need to verify on the Bar Code Reader before booting the system.

- The four switches used to define the Transmission Type (i.e., switches 5 through 8 on the right-hand set of switches) should be set to all zeros. This puts you in the non-keyboard mode.
- The Appended Key switch setting (i.e., switches 2 and 3 on the right-hand set of switches) should be set for none. This assures that no key operation will be appended to the end of your bar-code reading.
- The Bar Code Reader should have its Auto Recognition switch set (i.e., switch 1 on the right-hand set of switches) and the bar code you are going to be reading selected (use the eight left-hand set of switches). The following are possible bar code selections:
  - Interleaved 2/5
  - Code 3/9
  - Extended Code 3/9
  - CODABAR USD-4 and ABC
  - UPC/EAN/JAN
  - UPC E (8 digits)

Note that the Automatic code recognition does not mean that the bar-code reader will automatically know the codes you intend to read, you have to select them first. It does mean that it will automatically recognize the codes you have selected. For example, if you wanted to read bar codes that may be either the Interleaved 2 of 5 bar code or the 3 of 9 bar code, you would set the right-hand set of switches to the following:

- switch 8 to 1
- switch 7 to 1
- switches 6 through 1 to 0

Topics covered in this section are:

- Determining device characteristics, and
- Transmitting ASCII Characters.

### **Determining Bar Code Reader Characteristics**

The following results assume the Bar Code Reader is at address 4. The Bar Code Reader returns Identify and Describe information as follows:

```
00 04 04 5C 00
```

If this is not the case, you need to check the switch settings on the underside of your Bar Code reader again.

After the system has been re-booted you should do another Identify and Describe of the device to see that it is recognized as a Bar Code Reader. The program used to obtain this information is found in the section titled "HP-HIL Device Characteristics." Your results after entering and running this program should be as follows:

```
00 04 04 5C 00
```

where:

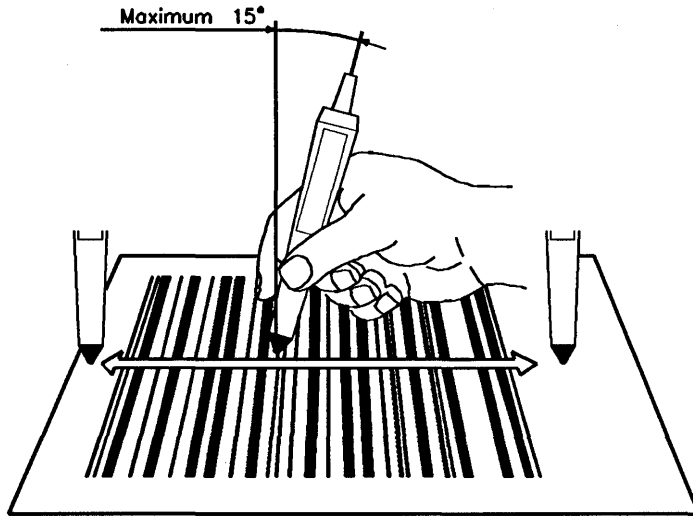
- 00 is the null character (packet overflow count).
- 04 is the number of bytes to follow including this byte.
- 04 is the address of the device.
- 5C is the type of device which in this case is the Bar Code Reader.
- 00 is the Describe Record Header with no special features.

## Reading a Selected Bar Code

Once you have selected the type of bar code you wish to read, you are ready to use your bar code reader. To do this, enter the program called "Bar\_code" found on your *Manual Examples* disc for BASIC/WS or in the directory called /usr/lib/rmb/demo for BASIC/UX and run it. Note that if your Bar Code Reader is not located at address 4, then you need to change line 100 of the program to match your devices address.

```
100  ON HIL EXT 2^4 CALL Disp_buf
110  Loop:GOTO Loop
120  END
130  !
140  SUB Disp_buf
150    DIM A$[256]
160    A$=HILBUF$
170    IF LEN(A$)=1 THEN RETURN
180    Packet_start=2
190    REPEAT
200      Packet_length=NUM(A$[Packet_start])
210      Packet_end=Packet_start+Packet_length-1
220      PRINT A$[Packet_start+3,Packet_end];
230      Packet_start=Packet_end+1
240    UNTIL Packet_start>LEN(A$)
250    PRINT
260  SUBEND
```

To have data displayed on the screen, you need to move the Bar Code Reader's wand rapidly and at a constant speed across the bar code. The wand should also be held as shown:



**The Correct Method for Holding the Bar Code Reader**

The following is an explanation of the program provided in this section.

*Line 100* enables end-of-line branching on movement of the wand across the bar code.

*Line 110* is a continuous loop which allows the program to idle while waiting for a bar-code reading.

*Line 140* is the beginning of the subprogram `Disp_buf`. This subprogram is used to display the data read by the Bar Code Reader.

*Line 160* dimensions the string `A$`.

*Line 170* tests for an empty buffer. If the buffer is empty the data that caused the interrupt has already been processed by a previous invocation of the service routine, so it simply returns to the idle loop on line 110.

**9** If the buffer is not empty, a REPEAT loop in lines 190 through 240 causes ASCII bar code information to be displayed.

*Line 180* assigns the value of 2 to the integer variable `Packet_start`. Note that `Packet_start` initially is the subscript for the second element in `A$`. This element tells how many elements there are in the first packet of information including that element. This variable will also be the counter used to determine the starting position of each packet of information in the string (`A$`).

*Lines 190 through 240* are a REPEAT loop used to search the data in the string (`A$`) for each packet of information in the string `A$`.

*Line 220* prints out the ASCII characters found in each packet.

To end the program press the `Stop` key.

## Interaction Among Multiple HP-HIL Devices

End-of-line interrupts can be handled when they come from more than one HP-HIL device during program execution. To demonstrate this a program called `Multi_dev`, found on your *Manual Examples* disc for BASIC/WS or in the directory `/usr/lib/rmb/demo` for BASIC/UX, has been provided for you to load and run. The program and its explanation are included in this section. Note that line 1030 of this program assumes that you have a Touchscreen located at address 1 and a Function Box located at address 3. If these are not the correct addresses for your devices, you will have to change the address mask on line 1030 of this program. For information on how to change the address mask, read the section in this chapter titled "Communicating through the HP-HIL Interface."

The interaction covered in this section is between a Touchscreen and a Function Box. Both of these devices could easily be replaced by two other HP-HIL devices which are supported by the `ON HIL EXT` and/or `HIL SEND` statements. To do this, you would have to make a few variables changes to suit the new program and write new subprograms which would be appropriate for the HP-HIL devices you are using.

```
1000 DIM Buf$[256],Packet$[15]
1010 INTEGER Index,Hil_addr
1020 !
1030 ON HIL EXT 10 GOSUB Disp_buf ! Set up interrupts for
1040 !                               addresses 1 and 3.
1050 GINIT ! Initialize
1060 GRAPHICS ON ! Turn on graphics.
1070 PRINT TABXY(16,4)," This program allows you to touch a point on"
```

```

1080 PRINT TAB(16),"the screen and draw a figure at that location"
1090 PRINT TAB(16),"by pressing a key on the Function Box. The"
1100 PRINT TAB(16),"keys are numbered from left to right starting"
1110 PRINT TAB(16),"with the top row of Function Box keys."
1120 PRINT
1130 PRINT TAB(21),"Key 1 draws a TRIANGLE."
1140 PRINT TAB(21),"Key 2 draws a SQUARE."
1150 PRINT TAB(21),"Key 3 draws a PENTAGON."
1160 PRINT TAB(21),"Key 4 draws a CIRCLE."
1170 PRINT
1180 PRINT TAB(16),"To continue with the program:"
1190 PRINT
1200 PRINT TAB(21),"Press 'Continue', or"
1210 PRINT TAB(21),"Type 'CONT' and press 'Return'."
1220 PAUSE
1230 CLEAR SCREEN ! Clear the alpha display.
1240 WINDOW 0,56,0,43 ! Scale to match Touchscreen resolution.
1250 !
1260 Loop:GOTO Loop
1270 !
1280 Disp_buf:!
1290 Buf$=HILBUF$
1300 IF LEN(Buf$)=1 THEN RETURN ! Data already processed.
1310 Packet_start=2 ! Skip "overflow" indicator.
1320 REPEAT
1330   Packet_length=NUM(Buf$[Packet_start]) ! Determine packet length.
1340   Packet_end=Packet_start+Packet_length-1 ! Find end of packet.
1350   Packet$=Buf$[Packet_start,Packet_end]
1360   !
1370   Hil_addr=NUM(Packet$[2])
1380   SELECT Hil_addr
1390   CASE 1 ! Touchscreen
1400     CALL Touchscreen(Packet$)
1410   CASE 3 ! Function box
1420     CALL Function_box(Packet$[4])
1430   END SELECT
1440   !
1450   Packet_start=Packet_end+1 ! Prepare for next packet.
1460   !
1470 UNTIL Packet_start>LEN(Buf$)
1480 !
1490 RETURN
1500 END
1510 !
1520 SUB Touchscreen(Coordinate$)

```

```

1530      !
1540      IF BIT(NUM(Coordinate$[3]),1)=1 THEN
1550          X_coord=NUM(Coordinate$[4])
1560          Y_coord=NUM(Coordinate$[5])
1570          MOVE X_coord,Y_coord
1580      END IF
1590  SUBEND
1600      !
1610  SUB Function_box(Key_press$)
1620      INTEGER Key_num
1630      WHILE LEN(Key_press$)
1640          Key_num=NUM(Key_press$)
1650          SELECT Key_num
1660          CASE 0,1
1670              POLYGON 5,3,3
1680          CASE 2,3
1690              POLYGON 5,4,4
1700          CASE 4,5
1710              POLYGON 5,5,5
1720          CASE 6,7
1730              POLYGON 5,50,50
1740          CASE ELSE
1750              BEEP
1760          END SELECT
1770          Key_press$=Key_press$[2]
1780      END WHILE
1790  SUBEND

```

The following is an explanation of the above program. This program, as *lines 1070 to 1110* state, allows you to touch a point on the screen and draw a figure at that location by pressing a key on the Function Box. The keys are numbered from left to right starting with the top row of Function Box keys.

*Line 1030* initiates the end-of-line interrupts for the HP-HIL devices located at addresses 1 and 3. To do this you need to know how to set up the mask which will cause end-of-line interrupts to be recognized by both devices. The mask value is obtained by raising 2 by the power of each of the addresses and adding these values. For example, 2 raised to the first power added to 2 raised to the third power results in the value 10 for your mask. When an interrupt is received from either of the HP-HIL devices, program execution branches to the subroutine called *Disp\_buf*.

The subroutine *Disp\_buf* includes *lines 1280 through 1490*. This subroutine separates the packets of data sent by each HP-HIL device to the string buffer

of the function `HILBUF$` and sends those packets to appropriate subprograms which process this data. In other words, packets containing the address 1 are sent to the subprogram called `Touchscreen` and those packets with address 3 are sent to the subprogram called `Function_box`. Once the string buffer has been completely processed the subroutine is exited.

`Touchscreen` is the subprogram located in *lines 1520 through 1590* which searches the string `Coordinate$` to determine if it has X and Y axis coordinate information. If coordinate information is available, it is assigned to the variables `X_coord` and `Y_coord`. The graphics pen is next moved to the location of these coordinates. Plotting of the figures will start at these locations. Once the pen move is made program execution is returned to the main program.

Now that a screen location has been selected pressing a key on the Function Box will cause a triangle, square, pentagon or circle to be drawn at that location. The subprogram called `Function_box` located at *lines 1610 to 1790* receives the string called `Key_press$` and looks at it for the number of the key which was pressed and assigns that value to the variable called `Key_num`. The `SELECT` structure uses the variable `Key_num` to choose which figure should be drawn at the last selected screen location. Note that the `CASE` segment will responded to both the press and release of a Function Box key. The `WHILE` loop on *lines 1630 through 1780* handles multiple keys in the packet, since the program only expects keycodes from the Function Box.

### **Modifying the Interactive Program**

The program explained in the previous section made the assumption that there was a `Touchscreen` located at address 1 and a `Function Box` located at address 3. If you didn't have those HP-HIL devices or they weren't located at the addresses given above, you would need a way of determining what devices were on your HP-HIL link and their address. This section provides a method for doing this.



Determining which HP-HIL devices are on the HP-HIL link and their address, can be accomplished by adding the following FOR loop to the your program:

```
1030   ON ERROR GOTO Link_end
1040   FOR I=1 TO 7
1050     HIL SEND I;IDD
1060     Buf$=HILBUF$
1070     Idd(I)=NUM(Buf$[4])
1080   NEXT I
1090 Link_end: ! OFF ERROR
```

These program lines can be inserted in the previous program just after line 1020. The FOR loop consisting of *lines 1040 through 1080* is designed to loop 7 times because that is the maximum number of addressable devices you may have on the HP-HIL link at any time. If there are less than 7 devices on the link an error occurs and the FOR loop exits to line 1090 labeled `Link_end`. This branch to the label `Link_end` is a result of the `ON ERROR` statement on line 1030.

*Line 1050* uses the `HIL SEND` statement along with the HP-HIL `IDD` command to determine the device's location in the HP-HIL link, as well as its Device ID. Using the Device ID number returned upon executing the `HIL SEND address;IDD` statement, you can determine what your device is by looking the number up in the "Device ID Byte Definition" table found in the "HP-HIL Appendix" in the back of this manual.

Information returned after executing the `HIL SENDaddress;IDD` statement is placed in the string buffer of the `HILBUF$` function. *Line 1060* takes the information found in this string buffer and assigns it to the string variable `Buf$`.

*Line 1070* assigns the integer value of the fourth element of `Buf$` to the integer array variable `Idd(I)`. The fourth element in `Buf$` is the Device ID number. "I" (device address) in the subscript portion of the array is incremented as many times as there are devices in the link.

A **SELECT** structure, *lines 1500 through 1610*, can be added to the program to access various subprograms which perform a process for a specified HP-HIL device on the link. A device address called `Hil_addr` is used as an index to the array `Idd` to obtain the device ID number associated with the index value. For example, if 1 is assign to the variable `Hil_addr` and `Hil_addr` is used as the index to the array `Idd` and the device ID number found at that index is 48 (decimal), the subprogram `Function_box` is called and its process is executed. Note that the additional **SELECT** structure should follow line 1370 of the program. The **SELECT** structure contains the following program lines:

```
1500 SELECT Idd(Hil_addr)
1510 CASE 0 to 31
1520   Vectra(Packet$)
1530 CASE 48
1540   Function_box(Packet$[4])
1550 CASE 92
1560   Bar_code(Packet$)
1570 CASE 140
1580   Touchscreen(Packet$)
1590 CASE ELSE
1600   ! Ignore
1610 END SELECT
```

## The Parallel Interface

---

### Introduction

This chapter describes the HP Parallel interface. The HP Parallel interface provides compatibility for Centronics compatible printers.

---

#### Note



HP BASIC/UX does not directly support the HP Parallel interface. However, the user can use parallel peripherals on HP-UX via unnamed pipes. For example, instead of typing `PRINTER is 23`, type `PRINTER IS "| lp"` where `lp` spools to the parallel printer. For more information about unnamed pipes, refer to *HP BASIC 6.2 Advanced Programming Techniques*.

---

The HP Parallel interface supports bidirectional data flow between the interface and peripherals with parallel interfaces. The HP Parallel interface *does not* support input from devices using IBM's parallel input protocol.

---

#### Note



For most printing applications you need not address the interface at a register level. Refer to the "Using a Printer" chapter in the *HP BASIC 6.2 Programming Guide* for more information.

---

### Required Software and Hardware

In order to drive the HP Parallel interface, you must first LOAD BIN the PLEEL binary.

An HP parallel port is provided with the HP 9000 Model 345, 375, 380, 362, and 382 computers.

## Bus Description

HP computers that provide support for the HP Parallel interface provide a 25 pin connection. Peripherals generally provide a 36 pin connection. There are 17 lines used for communicating data between the host and the peripheral, consisting of:

- Eight data lines
- Four handshake lines
- Two error lines
- Two device status lines
- One reset line

Some lines are used only by the peripheral or host while other lines are used by the active sender or receiver. The following table shows the HP Parallel interface pin outs. Note that the *n* preceding the line labels indicates this line is asserted *low* (e.g., *nStrobe*).

When discussing the setting or resetting of signals on the bus, this chapter uses the term **assert** to indicate the signal has been set, and **release** to indicate the signal has been reset. When a signal is asserted, it is driven to its active state. For example, when the Busy signal is asserted, it is driven *high*, and when the *nStrobe* signal is asserted, it is driven *low*. Alternatively, when a signal is released, it is driven to its inactive state. For example, when the Busy signal is released, it is driven *low*, and when the *nStrobe* signal is released, it is driven *high*.

### HP Parallel Interface Pin Outs

Host (25 pins) Pin No.	Line Label	Peripheral (36 pins) Pin No.
1	<i>nStrobe</i>	1
2	Data 1	2
3	Data 2	3
4	Data 3	4
5	Data 4	5
6	Data 5	6
7	Data 6	7
8	Data 7	8
9	Data 8	9
10	<i>nAck</i>	10
11	Busy	11
12	PErrror	12
13	Select	13
14	Wr/ <i>nRd</i> (sometimes <i>nAutoFd</i> )	14
15	<i>nError</i> (sometimes <i>nFault</i> )	32
16	<i>nInit</i> (sometimes <i>nReset</i> )	31
17	<i>nSelectIn</i>	36

#### The Data Lines

These lines carry the binary signals that make up the byte being transmitted. Because there is only one set of data lines, communication is half duplex (input and output cannot happen simultaneously).

## The Handshake Lines

Line Label	Description
nStrobe	This signal is used by the sender to qualify the data currently being asserted on the data lines.
nAck	This signal is a pulse used by the peripheral to inform the host that it is ready to receive data. Not all peripherals use this line, however all HP bidirectional devices must use it.
Busy	This signal is used by the receiver to indicate it is not ready to receive another byte of data.
Wr/nRd	This signal is used by the host to set the direction of data flow over the interface. Wr/nRd asserted ( <i>high</i> ) indicates an output data direction (out from the host to the peripheral).

## The Error Lines

Line Label	Description
PError	This signal is used by the peripheral to indicate to the host that there is currently a paper error of some sort. Generally this signal is expanded upon to indicate that an error has occurred that requires operator intervention. This signal is not released until the paper error has been cleared up. (The HP ScanJet optical scanner uses this line for all error conditions.)
nError	This signal is used by the peripheral to indicate to the host that an error other than a paper error has occurred. This signal is not released until the error condition has been cleared up. (The HP ScanJet optical scanner does not use this signal.)

## The Status Lines

Line Label	Description
Select	This line is used by the peripheral to indicate to the host that it is online. During error conditions this line is usually released.
nSelectIn	This line is used by the host to indicate to the peripheral that it is online.

## The Reset Line

Line Label	Description
nInit	This line is used by the host to cause the peripheral to clear its buffers and do a soft reset (restoring the peripheral to power on conditions). Not all peripherals use this line; however, all HP bidirectional devices must use it.

---

## Summary of Parallel Interface STATUS and CONTROL Registers

<i>STATUS Register 0</i>	Card Identification. 6 is always returned.
<i>CONTROL Register 0</i>	Interface Reset. Any non-zero value causes a reset.
<i>STATUS Register 1</i>	Interrupt and DMA Status.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupt enabled	Interrupt requested	Interrupt level	Interrupt level	0	0	DMA1	DMA0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* is set (1) if interrupts are currently enabled.

*Bit 6* is set (1) when the card is currently requesting service. (This bit is independent of Interrupt Enabled, bit 7).

*Bits 5 and 4* constitute the card's hardware interrupt level:

Hardware Interrupt		
Bit 5	Bit 4	Level
0	0	3
0	1	4
1	0	5
1	1	6

*Bits 3 and 2* are not used (always 0).

*Bit 1* is set (1) if DMA channel one is currently enabled.

*Bit 0* is set (1) if DMA channel is currently enabled.

On POR (Power on Reset), interrupts are disabled (Bit 7=0) and both DMA channels are disabled. The interrupt level reflects the hardware state and is always the same.

*STATUS Register 10*                      Peripheral Status.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	nError	Select	PError
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bits 7-3                                      Not used (always 0).

Bit 2 (nError)                              If this bit is set (1), nError is asserted low.

Bit 1 (Select)                                If this bit is set (1), Select is asserted high.

Bit 0 (PError)                                If this bit is set (1), PError is asserted high.

## 10-6 The Parallel Interface



These bus lines are controlled by the peripheral. This register merely reflects the state of these bus lines, and therefore does not have a default POR setting.

*STATUS Register 11*                      Communication Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	FIFO Full	FIFO Empty	nStrobe	Busy	nAck
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bits 7-5                                      Not used (always 0).

Bit 4 (FIFO Full)                      If this bit is set (1), the hardware FIFO is full.

Bit 3 (FIFO Empty)                      If this bit is set (1), the hardware FIFO is empty.

Bit 2 (nStrobe)                              If this bit is set (1), nAck is asserted low.

Bit 1 (Busy)                                      If this bit is set (1), Busy is asserted high.

Bit 0 (nAck)                                      If this bit is set (1), nAck is asserted low.

On POR the hardware FIFO (first in/first out register) is empty, the nStrobe line should not be asserted, and the remaining lines are controlled by the peripheral. This register reflects the state of the peripheral owned lines, and therefore these register bits do not have a default POR setting.

*STATUS Register 12*                      Host Line Control

*CONTROL Register 12*                      Host Line Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	nInit	nSelectIn	Wr/nRd
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bits 7-3                                      Not used (always 0).

Bit 2 (nInit)                                      If this bit is set (1), nInit is asserted low.

Bit 1 (nSelectIn) If this bit is set (1), nSelectIn is asserted low.

Bit 0 (Wr/nRd) If this bit is set (1), Wr/nRd is asserted high.

On POR, nInit is asserted low, nSelectIn is asserted high, and Wr/nRd is released high.

*STATUS Register 13* I/O Control.

*CONTROL Register 13* I/O Control.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	I/O Modifier	Input/ nOutput
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bits 7-2 Not used (always 0)

Bit 1 (I/O Modifier) If cleared, outbound transfers handshake with both BUSY and nAck and inbound transfers will use the FIFO. If set, outbound transfers will handshake with BUSY only and inbound transfers will only use one location in the FIFO (FIFO disabled).

Bit 0 (Input/nOutput) If this bit is set to 1, Input is selected. If this bit is reset (0), output is selected.


On POR bits 1 and 0 are reset to 0.

*STATUS Register 14* FIFO

*CONTROL Register 14* FIFO

In order to get valid information when reading the hardware FIFO, the I/O direction must be "input" and the FIFO must not be empty (see the Hardware I/O Status and Control register and the Communication Status register). If either of these conditions are not true, reading this register will not cause an error, but unpredictable results may occur.

For writing, the same rules apply. The I/O direction must be “output” and the FIFO must not be full. If either of these conditions are not true, writing this register will not cause an error, but the data written will not be entered into the hardware FIFO.

**Note**  This register should not be used unless the program has full control of this select code. For example, if this register is being used while the driver is attempting a transfer, it is very likely the transfer will fail.

*STATUS Register 20*      Peripheral Type

Decimal value	Peripheral type
0	No device attached.
1	Output-only device is currently attached.
2	An HP bidirectional device is attached.
10	User-specified no device.
11	User-specified output only device.
12	User-specified HP bidirectional device.

*CONTROL Register 20*      Peripheral Type

Decimal value	Peripheral type
0	No device attached.
10	User-specified no device.
11	User-specified output only device.
12	User-specified HP bidirectional device.

**CONTROL Register 22**      Peripheral Reset

Writing any non-zero value to this register causes the driver to attempt a hardware soft reset on the attached peripheral. The driver will assert the nInit line, wait, release the nInit line, and wait for Busy to be released.

**STATUS Register 23**      Interrupt State

Interrupt Enable Register (ENABLE INTR)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FIFO Full	FIFO Empty	0	Busy	nAck	nError	Select	PError
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

This register returns the interrupt requests that are currently being made by the driver.

- Bit 7 (FIFO Full)      If this bit is set (1), an interrupt will be requested when the hardware FIFO transitions to full.
- Bit 6 (FIFO Empty)      If this bit is set (1), an interrupt will be requested when the hardware FIFO transitions to empty.
- Bit 5      Not used (always 0).
- Bit 4 (Busy)      If this bit is set (1), an interrupt will be requested when the Busy signal is low.
- Bit 3 (nAck)      If this bit is set (1), an interrupt will be requested when the nAck signal transitions low.
- Bit 2 (nError)      If this bit is set (1), an interrupt will be requested when the nError signal transitions.
- Bit 1 (Select)      If this bit is set (1), an interrupt will be requested when the Select signal transitions.
- Bit 0 (PError)      If this bit is set (1), an interrupt will be requested when the PError signal transitions.

On POR the driver disables all interrupt conditions, thus this register will return a 0 on POR.

*STATUS Register 24*      Driver Options

*CONTROL Register 24*      Driver Options

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Ignore PError	Write Verify	Wr/nRd low	Use nAck
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bits 7-4      Not used (always 0).

Bit 3 (Ignore PError)      If this bit is set to 1, the interface will communicate with the device despite PError assertion.

If this bit is set to 0 (the default), an error occurs on a communication attempt with PError asserted.

Bit 2 (Write Verify)      If this bit is set to 1, the interface verifies that the peripheral receives data on each byte sent.

If this bit is set to 0 (the default), verification does not occur.

Bit 1 (Wr/nRd low)      If this bit is set to 1, Wr/nRd is always LOW.

If this bit is set to 0 (the default), Wr/nRd HIGH on output, LOW on input.

Bit 0 (Use nAck)      If this bit is set to 1, the interface uses nAck to complete the output handshake.

If this bit is set to 0 (the default), the interface uses Busy to complete the output handshake.

*STATUS Register 26*

## Driver State

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Disable by user	Inactive ERROR	Write	Read	0	0	0	Active Xfer
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

The driver states are:

DISABLED\_BY\_USER      =80h (hexidecimal)

INACTIVE\_ERROR        =40h

INACTIVE\_WRITE        =20h

ACTIVE\_WRITE          =21h

INACTIVE\_READ         =10h

ACTIVE\_READ            =11h

If the POR state of the peripheral type is not “user specified no device” (see register 20) then the POR state for this register is INACTIVE\_ERROR. Otherwise, the POR state is DISABLED\_BY\_USER.

*STATUS Register 27*

## Driver Information

nAck set = 1

When reading data from the HP Parallel interface one byte at a time, it may be necessary to determine if the peripheral has indicated end of transmission by pulsing the nAck line. Bit 0 of Status Register 27 is provided for this purpose.

## Summary of Parallel Interface READIO and WRITEIO Registers

This section describes the HP Parallel interface's READIO and WRITEIO registers. Keep in mind that these registers should be used *only* when you know the exact consequences of their use, as using some of the registers improperly may result in improper interface behavior. If the desired operation can be performed with STATUS or CONTROL, you should not use READIO or WRITEIO.

### Parallel READIO Registers

Register 1	ID Reset
Register 3	Parallel Interface Status
Register 5	Parallel Device Status
Register 7	Parallel Device Control
Register 9	Parallel Interrupt Status
Register 11	Parallel FIFO Data Register

*READIO Register 1*            Interface ID. Returns the interface ID byte for the HP Parallel interface (always 6).

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	1	1	0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

## READIO Register 3

## Parallel Interface Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts enabled	Interrupt requested	Interrupt level (IL1)	Interrupt level (IL0)	IO modifier	I/O direction	DMA1	DMA0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

*Bit 7* is set (1) if interrupts are currently enabled.

*Bit 6* is set (1) when the card is currently requesting service.

*Bits 5 and 4* constitute the card's hardware interrupt level:

Hardware Interrupt		
Bit 5	Bit 4	Level
0	0	3
0	1	4
1	0	5
1	1	6

*Bit 3*: if cleared, outbound transfers handshake with Busy and nAck. Inbound transfers use the FIFO. If set, outbound transfers handshake with Busy only and inbound transfers only use one location in the FIFO (FIFO disable).

*Bit 2*: 0 = output. 1 = input.

*Bit 1* is set (1) if DMA channel one is currently enabled.

*Bit 0* is set (1) if DMA channel zero is currently enabled.



*READIO Register 5*

## Parallel Device Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FIFO full	FIFO empty	STROBE	BUSY	nACK	nERROR	SELECT	PError
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

- Bit 7 (FIFO full)                    If this bit is 1, the FIFO is full.
- Bit 6 (FIFO empty)                If this bit is 1, the FIFO is empty.
- Bit 5 (STROBE)                    If this bit is 1, the nSTROBE line is asserted (data may be read from the parallel bus by the CPU).
- Bit 4 (BUSY)                        If this bit is 1, the BUSY line is asserted.
- Bit 3 (nACK)                        If this bit is 1, the nACK line is asserted.
- Bit 2 (nERROR)                    If this bit is 1, a hardware error on the peripheral has occurred.
- Bit 1 (SELECT)                     If this bit is 1, SELECT has been asserted.
- Bit 0 (PError)                     If this bit is 1, the PError (paper error) line is asserted.

*READIO Register 7*

## Parallel Device Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	nINIT	nSelectIn	WRnRD
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

- Bits 7-3                              Not used (always 0).
- Bit 2 (nINIT)                        If this bit is 1, the nINIT line is asserted (peripheral soft reset).

Bit 1 (nSelectIn)                      If this bit is 1, the nSelectIn line is asserted (CPU on line).

Bit 0 (WRnRD)                        If this bit is 1, the I/O direction is “output” (write).

*READIO Register 9*                      Parallel Interrupt Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FIFO full interrupt request	FIFO empty interrupt request	0	BUSY low interrupt request	nACK transition low interrupt request	nERROR transition (high/low) interrupt request	SELECT transition (high/low) interrupt request	PError transition (high/low) interrupt request
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bit 7                      If this bit is 1, a “FIFO full” interrupt is requested.

Bit 6                      If this bit is 1, a “FIFO empty” interrupt is requested.

Bit 5                      Not used (always 0).

Bit 4                      If this bit is 1, a “BUSYlow” interrupt is requested.

Bit 3                      If this bit is 1, an interrupt has been requested on an nACK transition low.

Bit 2                      If this bit is 1, an interrupt has been requested on an nError transition (high or low).

Bit 1                      If this bit is 1, an interrupt has been requested on a SELECT transtion (high or low).

Bit 0                      If this bit is 1, an interrupt has been requested on a PError transition (high or low).

**READIO Register 11**      Parallel FIFO Data Register

If register 3, bit 2, is set (input) I/O is set, a read from Register 11 will return the next byte from the FIFO.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
d7	d6	d5	d4	d3	d2	d1	d0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

**Parallel WRITEIO Registers**

- Register 1      ID Reset
- Register 3      Parallel Interface Control
- Register 5      Parallel Device Status
- Register 7      Parallel Device Control
- Register 9      Parallel Interrupt Control
- Register 11     Parallel FIFO Data Register

**WRITEIO Register 1**      Reset. Write any value to this register to reset the card.

**WRITEIO Register 3**      Parallel Interface Control.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupt enable	Not used	Not used	Not used	IO modifier	I/O direction	DMA1	DMA0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bit 7      Set this bit to 1 to enable interrupts.

Bit 6-4     Not used (indeterminant).

- Bit 3            If cleared, outbound transfers handshake with BUSY and NACK. Inbound transfers use the FIFO. If set, outbound transfers handshake with BUSY only and inbound transfers only use 1 location in the FIFO (FIFO disable).
- Bit 2            0 = output. 1 = input.
- Bit 1            Set this bit to 1 to enable DMA channel 1.
- Bit 0            Set this bit to 1 to enable DMA channel 0.

*WRITEIO Register 7*            Parallel Device Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					nINIT	nSelectIn	WRnRD
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

- Bits 7-3            Not used (indeterminant).
- Bit 2 (nINIT)            If this bit is 1, the nINIT line is asserted (peripheral soft reset).
- Bit 1 (nSelectIn)            If this bit is 1, the nSelectIn line is asserted (CPU on line).
- Bit 0 (WRnRD)            If this bit is 1, the I/O direction is "output" (write).

*WRITEIO Register 9*            Parallel Interrupt Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable FIFO full interrupt	Enable FIFO empty interrupt	0	Enable BUSY low interrupt	Enable nACK transition low interrupt	Enable nERROR transition (high/low) interrupt	Enable SELECT transition (high/low) interrupt	Enable PError transition (high/low) interrupt
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1

Bit 7            If this bit is 1, a “FIFO full” interrupt is enabled.

Bit 6            If this bit is 1, a “FIFO empty” interrupt is enabled.

Bit 5            Not used (always 0).

Bit 4            If this bit is 1, a “BUSYlow” interrupt is enabled.

Bit 3            If this bit is 1, an interrupt is enabled for an nACK transition low.

Bit 2            If this bit is 1, an interrupt is enabled for an nError transition (high or low).

Bit 1            If this bit is 1, an interrupt is enabled for a SELECT transition (high or low).

Bit 0            If this bit is 1, an interrupt is enabled for a PError transition (high or low).

#### *WRITEIO Register 11*      Parallel FIFO Data Register

If register 3, bit 2, is cleared (output), writing data to Register 11 writes data to the FIFO.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
d7	d6	d5	d4	d3	d2	d1	d0
value=128	value=64	value=32	value=16	value=8	value=4	value=2	value=1



## HP-HIL Appendix

---

This appendix contains information necessary for the development of drivers for HP-HIL devices. The contents of this appendix should be used in conjunction with the chapter in this manual entitled “HP-HIL Devices”. This appendix has been divided into the following section:

- HP-HIL Command Reference
- Device ID Byte
- Describe Record
- Extended Describe Record
- Poll Record
- Report Security Code Record
- Accessible Keycode Definitions

For more information on how HP-HIL devices work, order the *HP-HIL Technical Reference Manual* (HP product number 45918A).

---

### HP-HIL Command Reference

This section provides information for each of the existing HP-HIL commands which are supported by BASIC. The usage of the command is given first, followed by a brief listing of the characteristics of the command.

## A

The characteristics of the commands include:

- how or what the command is used for
  - device identification
  - data input
  - data output
- what commands are not supported by most devices
- a verbal description of the operation of the command

If a device does not support a particular command, it will ignore the command when sent to it.

### **Identify and Describe (IDD)**

**Usage:** The IDD command is used to determine the type of the attached devices, as well as some general characteristics of the device required to understand the data it reports.

**Characteristics:** Used for device identification.

**Description:** A device responds to the IDD command by first transmitting the device ID byte. The Device ID Byte is used to identify the general class of device and the nationality (in the case of a keyboard other than the HP 98203C). After the ID byte, a series of data bytes, referred to as the Describe Record, is transmitted. This record varies in length and is terminated by a null byte. This means that all bytes of the Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored. See the "Device ID Byte" and "Describe Record" sections of the "HP-HIL Appendix".

### **Read Register (RRG)**

**Usage:** Read Register provides the System with an alternate method of collecting data from a device supporting RRG. Note that RRG is not supported by most HP-HIL devices.

**Characteristics:** Used for data input. Not supported by most devices.



**Description:** A device indicates support of the Read Register command in the Extended Describe Record, also indicating the specific read registers contained in the device. To perform a register read, the System transmits the address of the register to be read, with the Read Register command. The device, upon receiving the command, transmits the contents of the register.

HP-HIL register errors are not supported. If an HP-HIL register error does occur, the command that caused the error is ignored.

Devices which do not support the Read Register command will ignore it (no Register I/O Error is sent).

## **Write Register (WRG)**

**Usage:** Write Register provides a means of setting the contents of individual registers in devices supporting this advanced feature.

**Characteristics:** Used for data output. Not supported by most devices.

**Description:** There are two forms of the Write Register command. Devices indicate support of either of these two forms (or both) in the Extended Describe Record. Both Write Register forms are supported to accommodate devices which support only one or the other form, but they are equivalent capabilities as supported by BASIC.

HP-HIL register errors are not supported. If an HP-HIL register error does occur, the command that caused the error is ignored.

Devices which do not support the Write Register command will ignore it (no Register I/O error is sent).

## **A Report Name (RNM)**

- Usage:** Report Name is used to request a string of up to 15 characters (8-bit ASCII) which would aid in describing the device to the user.
- Characteristics:** Used for device identification. Not supported by most devices.
- Description:** Characters returned are US ASCII. Devices indicate support of the Report Name command in the Extended Describe Record.
- This record varies in length and is terminated by a null byte. This means that all bytes of the Extended Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored.

## **Report Status (RST)**

- Usage:** Report Status is used to extract device-specific status information from devices configured on the Link.
- Characteristics:** Used for data input. Not supported by most devices.
- Description:** Devices indicate support of the Report Status command in the Extended Describe Record. This record varies in length and is terminated by a null byte. This means that all bytes of the Extended Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored. Interpretation of the status bytes will necessarily depend upon the device in question.

## **Extended Describe (EXD)**

- Usage:** Extended Describe provides additional information concerning more advanced device features which may not be required for basic operation.
- Characteristics:** Used for device identification. Not supported by most devices.

**Description:** Support of the Extended Describe command is indicated in the Describe Record. Devices supporting the EXD command respond with a series of data bytes referred to as the Extended Describe Record. This record varies in length and is terminated by a null byte. This means that all bytes of the Extended Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored. Detailed information on the Extended Describe Record can be found in the section of this appendix entitled “Extended Describe Record”.

### **Report Security Code (RSC)**

**Usage:** The Report Security Code command is used to extract a unique identifier from a device.

**Characteristics:** Used for data input. Not supported by most devices.

**Description:** Support of the command is indicated in the Describe Record. The Security Code Record consists of a Header and a variable number of bytes of data terminated by a null byte. This means that all bytes of the Describe Record are transmitted with the exception of trailing null bytes and these bytes are ignored. See the “Report Security Code Record” section of this appendix for further information.

### **Disable Keyswitch Autorepeat (DKA)**

**Usage:** This command is used to disable the “repeating keys” feature in the addressed device, reducing returned data to one report per keyswitch transition.

**Characteristics:** Not supported by most devices.

**Description:** The default condition of devices supporting DKA and EKA AutoRepeat Commands is Keyswitch AutoRepeat Disabled. More advanced key repeat features may be implemented using device specific commands.

Note that this AutoRepeat is independent of the normal Keyboard AutoRepeat implemented by Series 200 and 300 computers.

## Enable Keyswitch Autorepeat (EKA 1,EKA 2)

**Usage:** These two commands are used to enable the “repeating keys” feature in the addressed device (if the feature is supported).

**Characteristics:** Not supported by most devices.

**Description:** When Keyswitch AutoRepeat is enabled, most keys will repeat at the rate of one report every 40 milliseconds. Following a keyswitch down transition, a delay of 200 ms will occur and the key begins to repeat. Modifier keys (Shift, CTRL, Extend char, etc.) will not repeat, while based on the argument of the Enable Keyswitch AutoRepeat command the Cursor Keys (cursor left, right, up, and down) will repeat at either 20 millisecond or 40 millisecond intervals. Most keys repeat by generating repeated down transitions corresponding to the key position being repeated, although repeating cursor keys on an ITF Keyboard will report a keycode of 02(hexadecimal). Since the BASIC system does not recognize 02 as a valid Keycode, the effect is no cursor key autorepeat for either argument with the ITF Keyboard.

Note that this autorepeat is independent of the normal Keyboard AutoRepeat implemented by Series 200 and 300 computers.

## Prompt 1 thru Prompt 7 (PRM 1 .. PRM 7)

**Usage:** These commands are used to provide an audible or visual stimulus to the user, perhaps indicating that the System is ready for a particular type of input. Although intended to be directly associated with Acknowledge 1 thru Acknowledge 7 and Button 1 thru Button 7, this association is not a requirement.

Characteristics: Used for data output. Not supported by most devices.  
Description: The Prompts and Acknowledges supported are indicated in the Describe Record. All unsupported Prompts will be treated the same as other unsupported commands.

**Prompt (PRM)**

Usage: Intended as a general-purpose stimulus to the user. Prompt is not intended to be associated with a particular Button as are Prompt 1 thru Prompt 7.

Characteristics: Used for data output. Not supported by most devices.

Description: A device indicates support of Prompt in the Describe Record.

**Acknowledge 1 thru Acknowledge 7 (ACK 1 .. ACK 7)**

Usage: These commands, similar to the Prompt 1 thru Prompt 7 commands, are intended to provide an audible or visual response to the user, and are generally directly associated with the corresponding Prompt and Button of the same number, although this is not a requirement.

Characteristics: Used for data output. Not supported by most devices.

Description: Since there is no explicit “Prompt Off” function provided, this functionality may be part of the Acknowledge definition for a particular device.

The Prompts and Acknowledges supported by the devices are indicated in the Describe Record, and all unsupported Prompts will be treated the same as other unsupported commands.

## A Acknowledge (ACK)

- Usage: Similar to Prompt, Acknowledge is not associated with any particular Button, but is intended merely as a general purpose audio or visual response to the user.
- Characteristics: Used for data output. Not supported by most devices.
- Description: Since there is no explicit "Prompt Off" function provided, this functionality may be part of the Acknowledge definition for a particular device. Support of Prompt and Acknowledge is indicated in the Describe Record.

## Device-Dependent Commands (DDC 128 .. 239)

- Usage: A range of 112 commands has been reserved for use as "device-dependent" commands.
- Characteristics: Not supported by most devices.
- Description: These commands are intended for use by devices with special requirements which the other HP-HIL commands do not really support. Devices should use Read and Write Registers and the Prompts and Acknowledges for special functionality where possible.

---

## Device ID Byte

This section defines the device ID bytes for all types of devices currently defined or anticipated and lists the ID numbers which have currently been allocated. Nationalization for Keyboards is given in the second table.

The Device ID Byte is used to identify the general class of device and the nationality (language) in the case of a Keyboard. Since it is not possible to designate the characteristics of all future devices, the ID Byte should be used to identify only the basic type of device and the nationality (for a Keyboard).

The following table gives device ID Byte definitions for general classes of devices (keyboards, absolute positioners, etc.). For keyboard type devices other

than the HP 98203C, note that the ID has a range of 00 to 1F. This allows for the nationalization to be embedded in the ID Byte. The table of nationalized ID definitions gives the lower five bits of the ID Byte. Thus a French ITF keyboard (with an ID range of C0 to DF), would report its ID Byte as DB (C0 + 1B).

### Device ID Byte Definitions

Device Type	ID Range (hexadecimal)	Assigned Device IDs	HP-HIL Device Name	HP Product Number
Keyboard Group 1	A0 .. FF	C0 .. DF A0 .. BF	ITF Keyboard Integral Keyboard	46020/21A —
Absolute Positioners	80 .. 9F	95 94 93	11×11 Graphics Tablet Size-B Digitizer Size-A Digitizer	45911A 46088A 46087A
Relative Positioners	60 .. 7F	66 68 62 61 60	2-Button Mouse 3-Button Mouse Quadrature Port Control Dials Rotary Control Knob	46060A 46060B 46094A 46085A 46083A
Character Entry	40 .. 5F	5C	Barcode Reader	92916A
Other Devices	20 .. 3F	34 30	ID Module Function Box	46084A 46086A
Keyboard Group 2	00 .. 1F	00 .. 1F	Vectra Keyboard (BASIC/WS only)	46030A

### Keyboard Nationalized ID Definition

Lower 5 Bits of Device ID Byte (hexadecimal)	Nationality of Keyboard/Keypad
00	Other <sup>1</sup> <sup>2</sup>
01	reserved
02	Kanji
03	Swiss/French
04	Portuguese <sup>2</sup>
05	Arabic <sup>2</sup>
06	Hebrew <sup>2</sup>
07	Canadian/English
08	Turkish <sup>2</sup>
09	Greek <sup>2</sup>
0A	Thai (Thailand) <sup>2</sup>
0B	Italian
0C	Hangul (Korea) <sup>2</sup>
0D	Dutch
0E	Swedish
0F	German
10	Chinese-PRC (China) <sup>2</sup>
11	Chinese-ROC (Taiwan) <sup>2</sup>
12	Swiss/French II
13	Spanish
14	Swiss/German II
15	Belgian (Flemish)
16	Finnish
17	United Kingdom
18	French/Canadian
19	Swiss/German

<sup>1</sup> See the section "Extended Describe Record" for usage.

<sup>2</sup> Not supported by BASIC, treated as US ASCII.



### Keyboard Nationalized ID Definition (continued)

Lower 5 Bits of Device ID Byte (hexadecimal)	Nationality of Keyboard/Keypad
1A	Norwegian
1B	French
1C	Danish
1D	Katakana
1E	Latin American/Spanish
1F	United States

A

---

## Describe Record

The Identify and Describe command is used to determine the type of device(s) attached to the Link and also what their characteristics are.

When a device receives the IDD command, the device will respond by returning a device ID byte followed by the Describe Record. The Record consists of 1 to 10 bytes of information. The first byte of the Describe Record is the Describe Record Header. If the device reports positional information, then 2 bytes will follow containing the resolution of the device. If the device is an absolute positioner, then the maximum count per axis is then reported (for each axis), 2 bytes per axis. The last byte of the Describe Record is the I/O Descriptor Byte.

**A**

The Describe Record is shown graphically below:

Device ID
Describe Record Header
Number of counts / cm (m) Low Byte
Number of counts / cm (m) High Byte
Maximum Count X-axis Low Byte
Maximum Count X-axis High Byte
Maximum Count Y-axis Low Byte
Maximum Count Y-axis High Byte
Maximum Count Z-axis Low Byte
Maximum Count Z-axis High Byte
I/O Descriptor Byte

Every device will respond to the IDD command with at least 2 bytes of data, the Device ID Byte, and the Describe Record (1 to 10 bytes). Cursor positioning devices and devices containing buttons, proximity detection, and/or prompt/acknowledge functions will need to report additional information. The Describe Record Header contains some information about the device and provides an indicator of how much additional information is to follow the Header. The description of the Describe Record Header follows:

**Bit 7** Set if the device contains two independent sets of coordinate axes. Consider, for example, a device which interfaces two joysticks to HP-HIL, each with its own independent set of X, Y axes. It is assumed, however, that both sets of coordinate axes share common characteristics as identified in the remainder of the record. Default (clear) indicates a maximum of one set of axes.

- Bit 6 Set if the device is to return absolute positional data (unsigned integers). Default (clear) indicates relative data (2's complement).
- Bit 5 Set if the device returns all positional information at 16-bits/axis. Default (clear) is 8-bits/axis.
- Bit 4 Set if the I/O Descriptor Byte is to follow later in the Describe Record. Default (clear) indicates that the device has no buttons, no proximity detection, and no prompt/acknowledge functionality, with no I/O Descriptor Byte to follow.
- Bit 3 Set if the device supports the Extended Describe command. Default (clear) indicates Extended Describe command is not supported.
- Bit 2 Set if the device supports the Report Security Code command. Default (clear) indicates Report Security Code is not supported.
- Bit 1,0 Bit 1 and bit 0 indicate the coordinate axes the device will report. If non-zero, then following the header will be 16 bits describing the resolution of the device, and in the case of an absolute positioner, 16 bits/axis detailing the extent of each coordinate axis.

Bit 1	Bit 0	Axes Reported
0	0	none
0	1	X
1	0	X and Y
1	1	X, Y, and Z

If the Describe Record Header indicates a non-zero number of axes for which the device will report positional information, then following the Header will be 16 bits describing the resolution of the device in counts per centimeter if the device reports data in a 16-bit format, or in counts per meter if 8-bit format. In the case of an absolute positioner, following the Number of Counts/cm (m) will be 16 bits per axis indicating the maximum extent of each axis for which the device reports data, assuming an origin at the lower left. This is the maximum count per axis the device is capable of reporting, based on a

**A**

minimum value of 0. Note that these values are reported as 16 bits regardless of whether the device indicates 8-bit or 16-bit data reporting format.

The I/O Descriptor Byte indicates the buttons the device will report keycodes for, whether the device has proximity detection, and what Prompt/Acknowledge functions, if any, are implemented in the device. Note that Prompt and Acknowledge are treated as a set, and no device may indicate support of any particular Prompt or Acknowledge without also supporting its counterpart. If none of the above features are implemented, the I/O Descriptor byte may not be transmitted. The following is the definition of the I/O Descriptor byte:

- Bit 7** Set if the device implements the general purpose Prompt and Acknowledge functions. Default (clear) implies these functions are not implemented.
- Bits 6,5,4** Bits 6, 5, and 4 indicate specific Prompt/Acknowledges (Prompt 1 thru 7 and Acknowledge 1 thru 7) implemented by the device. Default (clear) indicates none.

<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Prompt/Acks. Implemented</b>
0	0	0	none
0	0	1	1
0	1	0	1 and 2
0	1	1	1, 2, and 3
1	0	0	1 thru 4
1	0	1	1 thru 5
1	1	0	1 thru 6
1	1	1	1 thru 7

- Bit 3** Set if the device will report the Proximity In/Out keycodes. Default (clear) indicates no proximity detection.

Bits 2,1,0

Bits 2, 1, and 0 indicate the buttons for which the device will report keycodes.

A

Bit 2	Bit 1	Bit 0	Buttons Reported
0	0	0	none
0	0	1	1
0	1	0	1 and 2
0	1	1	1, 2, and 3
1	0	0	1 thru 4
1	0	1	1 thru 5
1	1	0	1 thru 6
1	1	1	1 thru 7

---

## Extended Describe Record

Support of the Extended Describe command is indicated in the Describe Record Header. The Extended Describe Record provides additional information concerning more advanced features which may not be required for basic operation.

**A**

Devices supporting the Extended Describe command respond with a series of data bytes referred to as the Extended Describe Record. The record length may vary from 1 to 15 bytes (although only 6 bytes are currently defined). The Extended Describe Record has the following format:

Extended Describe Record Header
Maximum Read Register Support
Maximum Write Register Support
Maximum Write Buffer Length Low Byte
Maximum Write Buffer Length High Byte
Localization Code

Devices responding to the Extended Describe command return at least 1 byte of data, the Extended Describe Record Header. Devices supporting Read Register or Write Register or those returning a Localization Code will need to report additional information so that their capabilities may be more fully defined. The Extended Describe Record Header both supplies some of the parameters of the device and provides an indication of how much additional information is to follow. The meanings of the individual bits in the Header are as follows:

- Bit 7      Reserved for future use. Default will be clear.
- Bit 6      Set if the Localization Code is supported. If set, then following the Maximum Write Buffer Length High Byte will be one byte indicating the nationality of the device (keyboard). See the table in the previous section for a listing of the Localization Codes. Default (clear) indicates that the Localization Code is not supported.
- Bit 5      Set if the Report Status command is supported. Default (clear) indicates Report Status not supported.
- Bit 4      Set if the Report Name command is supported. Default (clear) indicates Report Name not supported.

- Bit 3 Reserved for future use. Default will be clear.
- Bit 2 Set if Read Register supported. If set, immediately following the Header is a byte indicating the registers supported for reading in the device. Default will be clear, indicating Read Register not supported.
- Bit 1,0 Bit 1 and bit 0 indicate support of the Write Register command. If bit 1 is set, Write Register Type 2 is supported by the device. If bit 0 is set, Write Register Type 1 is supported. If both bits are set, then the device supports both Type 1 and Type 2. If either bit 1 or bit 0 is set, then following in the Record will be information indicating the registers supported for writing in the device. If bit 1 is set, then an additional 16 bits will be returned indicating the maximum number of data bytes which may be written to the device at a time using Write Register Type 2 without data loss.

If the device indicated support for the Read Register command in the Header, then following the Header is a byte indicating the read registers supported by the device. This byte, the Maximum Read Register Supported byte, indicates the largest read register address supported. Note that it is assumed that all addresses less than this maximum are also supported. Thus a byte of 0Fh indicates that the device contains 16 read registers, addressed as read registers 0 thru 15. HP-HIL protocol allows for devices containing up to 128 read registers, addressed as 0 thru 127.

If Write Register (Type 1 or Type 2) support is indicated, then next is a byte indicating the write registers supported. The Maximum Write Register Supported byte indicates the largest write register address supported in the device. It is assumed that all addresses less than the maximum are also supported. Up to 128 write registers, addressed as 0 thru 127, are supported in the HP-HIL protocol.

If Write Register Type 2 is supported, as indicated by bit 1 of the Extended Describe Record Header being set, then following the Maximum Write Register Supported byte is 16 bits of data indicating the maximum number of bytes which may be transmitted to the device in a Type 2 transfer without overflowing the device's internal buffer. This number, transmitted first low byte, then high byte, represents the buffer length of the device minus 1. Thus a device capable of buffering 1024 bytes of data would transmit a Maximum

**A**

Buffer Length Low Byte of FFh and a Maximum Buffer Length High Byte of 03h.

If the Localization code is supported, then the Localization Code byte will be included in the Extended Describe Record. The Localization Code is an 8 bit number which corresponds to a nationality (language) of a keyboard. The table in the previous section, lists currently assigned Localization Codes and languages (values from 20 through FF are reserved).



---

## Poll Record

A

The Poll command is the fundamental means for extracting data from the input devices attached to the Link. Data is sent back to the host in the form of a record, which may contain character data, position data, or some status information.

Data returned from HP-HIL devices is in record form, similar to the response to the Describe command. Each device transmits its individual Poll Record. Note that it may not be required for the device to report all available information in response to a single Poll request; data may be split between Polls provided correct formatting is observed for each record reported. The Poll Record is structured as follows:

Poll Record Header
X-axis Data Low Byte
X-axis Data High Byte
Y-axis Data Low Byte
Y-axis Data High Byte
Z-axis Data Low Byte
Z-axis Data High Byte
Character Data
⋮
Character Data

**A**

The function of the Poll Record Header is to indicate to the System the type and quantity of information to follow, as well as to report simple status information. The bits of the Header are assigned as follows:

Bit 7            Set if the device is reporting data from the second set of coordinate axes. Default (clear) indicates data from set 1.

Bit 6,5,4        Based on the value of these 3 bits, following all position information will be character data (up to 8 bytes):

Bit 6	Bit 5	Bit 4	Character Data Description
0	0	0	No character data to follow
0	0	1	Reserved Character Set 1
0	1	0	US ASCII Characters
0	1	1	Binary Data
1	0	0	Keycode Set 1
1	0	1	Reserved Character Set 2
1	1	0	Keycode Set 2 *
1	1	1	Keycode Set 3

\* These keycodes are device dependent. They use the LSB to indicate the key transition (0 = Down, 1 = Up); 126 keys maximum.

Bit 3            Set indicates request for status check. Clear (default) indicates status unchanged.

Bit 2            Set indicates device ready for data. Default (clear) indicates not ready for data transfer at this time.

Bit 1,0

Bit 1 and bit 0 indicate the coordinate axes the device is reporting:

A

Bit 1	Bit 0	Axes Reported
0	0	none
0	1	X
1	0	X and Y
1	1	X, Y, and Z

Following the Header is the device data. If the device indicated that it would report coordinate information 16-bits/axis in the Describe Record, then for each axis reported will be first the low, then high byte coordinate data. Otherwise, the high byte will not be transmitted. In general, the Poll Record format indicates the maximum data which can be reported; most devices will transmit only a subset each time. Following the positional information will be up to 8 bytes of character data, as specified in the Poll Record Header. The different types of character data may not be mixed. Note that more than one device may respond to the Poll command; each will respond with an individual Poll Record, distinguishable from the previous by the address field of the data.

The BASIC system automatically sends poll commands at approximately 20 millisecond intervals.

---

## Report Security Code Record

The Report Security Code command is used to extract a unique identifier from the device. Support of the command is indicated in the Describe Record Header. The Report Security Code Record consists of a header (1 byte) which defines the format of the data following the header (the remaining 1 to 14 bytes of data). Bits 7 through 4 of the header byte describe the data format type. Currently, only one data format type is defined, Type 1. Bits 3 through 0 are reserved, and should be set to 0. Thus the only currently valid header is for a Type 1 format (hexadecimal 10). The Report Security Code Record is similar

**A**

in purpose to a serial number, it may also contain information related to user identity, network address, or other information which is unique to a particular user or environment.

The only data transmitted by the ID Module is in response to the Report Security Command. However, the following information applies to any device that supports the Report Security command.

The data format consists of a one byte header and eight bytes of binary data. The eight data bytes are the packed product and serial numbers of the HP-HIL device. In the case where an ID Module is an exchange module signified by a ten digit part number, the five digit prefix number remains the same and the product number letter is replaced by the least significant digit of the part number.

The product, exchange and serial number formats are:

Header : H (1 byte header)  
Product number is : DDDDDA (5 digits and 1 ASCII character)  
Exchange part number : DDDDDd (5 digits and 1 ASCII character)  
Serial number is : YYWW@NNNNN (9 digits and 1 ASCII character)

where:

H is the data header.  
DDDDD is the product number (e.g., 46084).  
A is the product number alpha character.  
d is the least significant numeric character of the exchange number.  
YY is the year code (year less 60).  
WW is the week code (0 to 51).  
@ is the serial country of manufacturing code.  
NNNNN is the serial suffix (0 to 99 999)

The header byte is transmitted before the eight data bytes. The header's purpose is to allow for other data formats, however none are currently implemented.

The five digits of the product or exchange part prefix number are converted to a two byte binary number and the high order bit of a third byte. The remaining lower seven bits of the third byte contain the ASCII character. In products where two alpha characters are used in the product number, only the first character is used in the data format. The order of the bytes have been arranged to transmit the least significant byte of the number first.

In a similar manner, the nine digits of the serial number are converted to a four byte binary number. The country code of manufacturing is in the last byte to be transmitted and is an ASCII character.

The Report Security data bytes are transmitted in the following order, starting with byte 1 and going through byte 9. Bits are numbered starting with bit 0 at the right most position of the byte (least significant bit) and going through bit 7 (most significant bit), left most position.

Header (10 hexadecimal)	
Product Number Bits 7 .. 0	
Product Number Bits 15 .. 8	
Product Number Bit 16	Product Letter Suffix ASCII (7 Bits)
Serial Number Bits 7 .. 0	
Serial Number Bits 15 .. 8	
Serial Number Bits 23 .. 16	
0 0	Serial Number Bits 29 .. 24
0	Country of Manufacture USASCII (7 Bits)

**A**

The report security data format is:

Byte	Bit(s)	Description
1	7 - 0	The first byte is the header containing the number 10 hexadecimal for the following format. The general scheme for the header is:  Bits 7 - 4 are assigned as format variations, where format 1 is the only assignment.  Bits 3 - 0 are undefined, but set to zero.
2 3 4	7 - 0 7 - 0 7	The second and third bytes and the 7th bit of the fourth byte represent the 5 digits of the product or exchange part number DDDDD in binary form. The least significant bit is bit 0 of byte two.
4	6 - 0	The least significant seven bits of byte four represent the product letter or the least significant digit of the exchange number numeric character. The character is the US ASCII 7 bit representation of the character.
5 6 7 8	7 - 0 7 - 0 7 - 0 5 - 0	The fifth, sixth, seventh bytes and the six least significant bits of byte eight represent the 9 digits of the serial number YYWWNNNNN in binary form, without the alpha character. The least significant bit is bit 0 of byte 5.
8	7 - 6	The two most significant bits of byte eight are reserved for future use and are set to zero.
9	6 - 0	The least significant seven bits of byte 9 represent the serial number letter. The character is the US ASCII 7 bit representation of the character.
9	7	The most significant bit of byte nine is reserved for future use and is set to zero.

## Sample of Report Security Format for a Product Module

The following information is returned upon receiving a Report Security command for a Product Module. The data is based on the data format described in the last section. Byte 1 is the first byte sent from the module to the host.

The sample results given are based on the product number 46084A and serial number 2519A00001. The serial number corresponds with the year of 1985, week 19, and serial number suffix 00001. Note that by adding 60 to the above serial numbers first two digits you get the year 85.

Byte No.	Data (hex)	Description
1	10	Header
2	04	Part of product number HP 46084
3	B4	Part of product number HP 46084
4	41	Product letter "A" and part of product number HP 46084
5	61	Part of serial number
6	B0	Part of serial number
7	03	Part of serial number
8	0F	Part of serial number
9	41	Country of manufacturing code

## Sample of Report Security Format for An Exchange Module

The following information is returned upon receiving a Report Security command for an Exchange Module. The data is based on the data format described in the section, "Report Security Code (RSC)". Byte 1 is the first byte sent from the module to the host.

The sample results given are based on the exchange number 46084-69901 and serial number 2519A00001. The serial number corresponds with the year of

**A**

1985, week 19, and serial number suffix 00001. Note that by adding 60 to the above serial numbers first two digits you get the year 85.

Byte No.	Data (hex)	Description
1	10	Header
2	04	Part of product number HP 46084
3	B4	Part of product number HP 46084
4	31	US ASCII character "1" which is part of the product number HP 460841
5	61	Part of serial number
6	B0	Part of serial number
7	03	Part of serial number
8	0F	Part of serial number
9	41	Country of manufacturing code

Since the sample is an exchange module, the exchange part number transmitted is 460841. Byte 4 is the hexadecimal value of 31 which represent the US ASCII character "1". Note, the prefix number 46084 does not change from the sample of the product module and the character "1" is really an ASCII character.



## Accessible Keycode Definitions

A

This section covers a subset of Keycode Set 1. Keycode Set 1 provides the keycodes for the down and up keystrokes of ITF keyboards (HP 46020/21A). Note that both the ITF Keycode Set 1 and the version of Keycode Set 2 used by the HP 98203C keyboard are always processed by the system and thus are never available through the buffer used by the HILBUF\$ function. The subset of Keycode Set 1 which is used by HP-HIL Graphics Tablets is the small portion of Keycode Set 1 covered in this section. The HP 46066A Function Box uses Keycode Set 2. Its keys are numbered 0 through 31 starting with the upper-left key and going from left to right, then down. Key down generates Keycode  $2 \times n$  and key up generates Keycode  $2 \times n + 1$  where  $n$  is the key number. Keycode Set 3 provides the keycodes for the down and up keystrokes of the HP 46030A Vectra Keyboard for BASIC/WS.

### Keycode Set 1

Keycode for Down Transition (hex)	Keycode for Up Transition (hex)	United States Keycap Legend	Notes
80	81	<BUTTON 1>	1
82	83	<BUTTON 2>	1
84	85	<BUTTON 3>	1
86	87	<BUTTON 4>	1
88	89	<BUTTON 5>	1
8A	8B	<BUTTON 6>	1
8C	8D	<BUTTON 7>	1
8E	8F	<PROXIMITY IN/OUT>	1

<sup>1</sup> Typically used in positioning devices and not found on keyboards.

### Keycode Set 3: Vectra Keycodes (BASIC/WS)

Keycode for Down Transition (hex)	Keycode for Up Transition (hex)	United States Keycap Legend	Notes
00	80		Reserved
01	81	ESC	
02	82	1 / !	
03	83	2 / @	
04	84	3 / #	
05	85	4 / \$	
06	86	5 / %	
07	87	6	
08	88	7 / &	
09	89	8 / *	
0A	8A	9 / (	
0B	8B	0 / )	
0C	8C	- / _	
0D	8D	= / +	
0E	8E	Back space	
0F	8F	Tab	
10	90	Q	
11	91	W	
12	92	E	
13	93	R	
14	94	T	
15	95	Y	

### Keycode Set 3: Vectra Keycodes (continued)

A

Keycode for Down Transition (hex)	Keycode for Up Transition (hex)	United States Keycap Legend	Notes
16	96	U	
17	97	I	
18	98	O	
19	99	P	
1A	9A	I / F	
1B	9B	I / F	
1C	9C	Enter	
1D	9D	CTRL	
1E	9E	A	
1F	9F	S	
20	A0	D	
21	A1	F	
22	A2	G	
23	A3	H	
24	A4	J	
25	A5	K	
26	A6	L	
27	A7	: / ;	
28	A8	' / "	
29	A9	' / ~	
2A	AA	Shift	Left side
2B	AB	\ /	

A

### Keycode Set 3: Vectra Keycodes (continued)

Keycode for Down Transition (hex)	Keycode for Up Transition (hex)	United States Keycap Legend	Notes
2C	AC	Z	
2D	AD	X	
2E	AE	C	
2F	AF	V	
30	B0	B	
31	B1	N	
32	B2	M	
33	B3	. / <	
34	B4	0 / >	
35	B5	/ / ?	
36	B6	Shift	Right side
37	B7	* / Prt Sc	
38	B8	Alt	
39	B9	<space bar>	
3A	BA	Caps lock	
3B	BB	F1	
3C	BC	F2	
3D	BD	F3	
3E	BE	F4	
3F	BF	F5	
40	C0	F6	
41	C1	F7	

### Keycode Set 3: Vectra Keycodes (continued)

A

Keycode for Down Transition (hex)	Keycode for Up Transition (hex)	United States Keycap Legend	Notes
42	C2	F8	
43	C3	F9	
44	C4	F10	
45	C5	Num lock	
46	C6	<b>Break</b> / ScrLck	
47	C7	Home / <b>7</b>	
48	C8	<b>▲</b> / <b>8</b>	
49	C9	Pg Up / <b>9</b>	
4A	CA	<b>□</b>	
4B	CB	<b>◀</b> / <b>4</b>	
4C	CC	<b>5</b>	
4D	CD	<b>▶</b> / <b>6</b>	
4E	CE	<b>+</b>	
4F	CF	End / <b>1</b>	
50	D0	<b>▼</b> / <b>2</b>	
51	D1	Pg Dn / <b>3</b>	
52	D2	Ins / <b>0</b>	
53	D3	<b>DEL</b> / <b>.</b>	
54	D4	Sysreq	
55	D5		Reserved
56	D6		Reserved
57	D7		Reserved

A

### Keycode Set 3: Vectra Keycodes (continued)

Keycode for Down Transition (hex)	Keycode for Up Transition (hex)	United States Keycap Legend	Notes
58	D8		Reserved
59	D9		Reserved
5A	DA		Reserved
5B	DB		Reserved
5C	DC		Reserved
5D	DD		Reserved
5E	DE		Left side <sup>3</sup>
5F	DF		Right side <sup>3</sup>
60	E0	⬆	Cursor pad
61	E1	⬅	Cursor pad
62	E2	⬇	Cursor pad
63	E3	➡	Cursor pad
64	E4	Home	Cursor pad
65	E5	Pg Up	Cursor pad
66	E6	End	Cursor pad
67	E7	Pg Dn	Cursor pad
68	E8	Ins	Cursor pad
69	E9	DEL	Cursor pad
6A	EA	<unlabeled>	Cursor pad
6B	EB		Reserved
6C	EC		Reserved

<sup>3</sup> Key position is not loaded. Position is covered by a non-positional filler key.

**Keycode Set 3: Vectra Keycodes (continued)**

**A**

Keycode for Down Transition (hex)	Keycode for Up Transition (hex)	United States Keycap Legend	Notes
6D	ED		Reserved
6E	EE		Reserved
6F	EF		Reserved
70	F0	f1	
71	F1	f2	
72	F2	f3	
73	F3	f4	
74	F4	f5	
75	F5	f6	
76	F6	f7	
77	F7	f8	
78	F8		Reserved
79	F9		Reserved
7A	FA		Reserved
7B	FB		Reserved
7C	FC		Reserved
7D	FD		Reserved
7E	FE		Reserved
7F	FF		Reserved





# Index

---

## A

abort message, 3-23  
ABORT statement, 3-12, 3-16, 5-76  
above-screen lines, 1-27  
absolute positioners, 9-25  
active controller, 3-34  
addressed to listen, HP-IB, 3-9  
addressed to talk, HP-IB, 3-9  
addressing multiple listeners on the HP-IB bus, 3-11  
addressing, non-active HP-IB controller, 3-43  
addressing, secondary, 3-12  
ALPHA HEIGHT statement, 1-6  
alpha pen colors, 1-12  
ALPHA PEN statement, 1-10, 1-36  
ASCII and non-ASCII keys, 2-5  
ASCII data transfers, 5-47  
ASCII representations, 6-19  
async and data link operation, BOTH, 5-15  
asynchronous communication protocol, 5-3  
asynchronous data communication, 4-2  
async operation ONLY, 5-15  
attention line (ATN), HP-IB, 3-54  
automatic answering applications, data-comm, 5-71  
automatic dialing with the HP 13265A modem, 5-34  
auto-poll on the HP 1000, disabling, 5-67  
auto-repeat, keyboard, 2-11

## B

background datacomm program routines, 5-41  
bar code reader, using a, 9-49  
baud rate (RS-232C), 4-12  
baud rate, RS-232C handshake and, 4-6  
baud rate select switches, 4-8  
BCD binary data representation, 7-11  
BCD binary mode, 7-11  
BCD binary mode entry, 7-25  
BCD cable configuration, 7-21  
BCD data entry, 7-22  
BCD data output, 7-14, 7-34  
BCD data representation, 7-2  
BCD ENABLE INTR, 7-40  
BCD handshake configuration, 7-18  
BCD hardware priority, 7-18  
BCD interface, 7-1  
BCD interface configuration, 7-15  
BCD interface interrupts, 7-40  
BCD interface reset, 7-21  
BCD interface select code, 7-17  
BCD interface timeouts, 7-37  
BCD interrupt service routines, 7-41  
BCD interrupts, setting up and enabling, 7-40  
BCD-mode standard format, 7-23  
BCD operation, 7-2  
BCD optional format, 7-7, 7-31  
BCD output routines using CONTROL and STATUS, 7-34  
BCD peripheral status switches, 7-18

## Index

BCD representation, 6-31  
BCD standard format, 7-3  
BCD STATUS and CONTROL registers, 7-42  
BCD STATUS statement entry, 7-29  
BCD timeout service routines, 7-38  
BCD timeout time parameter, 7-38  
BCD type 1 timing, 7-19  
BCD type 2 timing, 7-20  
BREAK message, 4-24  
break received, 4-18  
break timing, datacomm, 5-28  
burst I/O mode, 3-6, 6-28

## C

cable options and functions, datacomm, 5-76  
cable options, RS-232C, 4-37  
caps lock mode, 2-9  
CDIAL function, 9-25  
character conversions, 6-34  
character format definition, datacomm, 5-27  
character format and parity, RS-232C, 4-13  
character format parameters, RS-232C, 4-6  
character length (RS-232C), 4-6  
circuit driver/receiver functions, optional, 4-39  
clearing the Screen, 1-6  
clear lockout/local message, 3-23  
clear message, 3-23  
CLEAR SCREEN statement, 1-6  
CLEAR statement, 3-12, 3-16  
Clear to Send (CTS), RS-232C, 4-6  
closure keys, 2-24  
CMD secondary keyword, 3-31  
color enhancements, 1-24  
communicating with HP-IB devices, 3-3

communication between desktop computers, datacomm, 5-74  
computer as a non-active controller on the HP-IB bus, 3-34  
configuration switches, 4-57  
configuring parallel poll responses, 3-19  
control block contents, datacomm, 5-21, 5-30  
control-character functions, 1-20  
control characters, 1-18  
control characters, generating, 2-6, 2-8  
controller address, HP-IB, 3-35  
controller's address, changing the HP-IB, 3-36  
controller status, HP-IB, 3-35  
control, passing, 3-36  
cooperating programs, 5-51  
CRT STATUS and CONTROL registers, 1-40

## D

Data Carrier Detect (DCD or CD), RS-232C, 4-6  
datacomm adapter options and functions, 5-76  
datacomm automatic answering applications, 5-71  
datacomm, break timing, 5-28  
datacomm character format definition, 5-27  
datacomm communication between desktop computers, 5-74  
datacomm configuration for BASIC/UX, 5-16  
datacomm connection, 5-15  
datacomm control block contents, 5-30  
datacomm data transfers between computer and interface, 5-6  
datacomm ENABLE INTR, 5-21  
datacomm error detection and program recovery, 5-76

- datacomm error recovery, 5-75
- datacomm exit conditions, 5-45
- datacomm handshake, 5-31
- datacomm interface protocol, 5-3
- datacomm interfaces, 5-1
- datacomm interrupts, 5-38, 5-40, 5-42
- datacomm interrupt service routines, 5-42
- datacomm interrupt system, setting up the, 5-38
- datacomm line connection, 5-32
- datacomm line timeouts, 5-22, 5-30
- datacomm options for async communications, 5-20
- datacomm options for data link communication, 5-29
- datacomm parity, 5-32
- datacomm parity option
  - EVEN, 5-4
  - NONE, 5-4
  - ODD, 5-4
- datacomm programming, 5-11
- datacomm programming helps, 5-65
- datacomm program operator inputs, setting up, 5-39
- datacomm prompt recognition, 5-27
- datacomm protocol and link operating parameters, 5-15
- datacomm protocol selection, 5-19
- datacomm service routines for ON KEY interrupts, 5-50
- datacomm start bits, 5-4
- datacomm stop bits, 5-4
- datacomm time gap, 5-4
- datacomm timeouts, 5-22
- datacomm transfers, data formats for, 5-47
- datacomm transmitted block size, 5-32
- Data Communication Equipment (DCE), RS-232C, 4-37
- data entry, RS-232C, 4-15
- data formats for datacomm transfers, 5-47
- data-link baud rates, 5-31
- data link communication protocol, 5-4
- data link operation ONLY, 5-15
- data loss prevention on the HP 1000, 5-65
- data message, 3-9, 3-22, 3-30
- data on the HP-IB bus, sending, 3-30
- data output, RS-232C, 4-14
- DATA secondary keyword, 3-31
- Data Set Ready (DSR), RS-232C, 4-6
- Data Terminal Equipment (DTE), RS-232C, 4-37
- data to the keyboard, sending, 2-15
- data transfers, RS-232C, 4-14
- data valid (DAV), HP-IB, 3-54
- DCE cable option, 4-37
- DCE cable options, 4-39, 5-77
- DCE cable, RS-232C, 4-39
- dialing procedure for switched (public) modem links, 5-33
- DIGITIZE statement, 9-24
- direct connection links, datacomm, 5-33
- disabling auto-poll on the HP 1000, 5-67
- disabling the cursor character, 1-37
- display-enhancement characters, 1-21
- display enhancement guidelines, 1-25
- display features, overview of, 1-4
- display functions mode, 1-25
- DISPLAY FUNCTIONS statement, 1-25
- display interfaces, 1-1
- display line, output area and the, 1-6
- display regions, 1-4, 1-10
- display regions affected by pen color statements, 1-11
- display types, 1-2
- DISP line, 1-35
- DRS and SRTS modem lines, programming the, 4-25
- DRS modem line, programming the, 4-25
- DTE cable options, 4-39, 5-77

DTE cable, RS-232C, 4-37

**E**

ENABLE INTR, BCD, 7-40

ENABLE INTR, datacomm, 5-21

ENABLE INTR, GPIO, 6-37

ENABLE INTR statement, 3-18, 5-38, 5-40

enabling and setting up GPIO events, 6-36

enabling local control, 3-15

enabling the insert mode, 1-37

end-of-line recognition, datacomm, 5-26

end or identify line (EOI), 3-55

enhanced keyboard control, 2-29

entering data from the keyboard, 2-13

entering from the CRT, 1-32

EOI signal, sending the, 2-15

EPROM addresses and unit numbers, 8-3

EPROM catalogs, 8-9

EPROM data storage rates, 8-11

EPROM directories, 8-8

EPROM hardware operation, 8-4

EPROM memory initialization, 8-3

EPROM memory overview, 8-2

EPROM memory, reading, 8-18

EPROM memory which is unused, 8-12

EPROM programmer select code, 8-3

EPROM programmer STATUS and CONTROL registers, 8-19

EPROM programming, 8-1, 8-10

EPROM, programming individual words and bytes in, 8-15

EPROM, reading data files stored in, 8-18

EPROM, storing data in, 8-10

EPROM to store programs, using the, 8-14

EPROM unit initialization, 8-8

ERRL function, 5-76

ERRN function, 5-76

error detection and program recovery, datacomm, 5-76

error detection, RS-232C, 4-4

error recovery, datacomm, 5-75

exit conditions, datacomm, 5-45

external interrupt request, 6-37

**F**

framing error (RS-232C), 4-4, 4-18

function box, 9-33

function box, activating the, 9-34

function box key presses, trapping, 9-36

function box keys, assigning functions to, 9-39

**G**

globalized BASIC

display-enhancement characters, 1-21

GPIO burst I/O mode, 6-28

GPIO control output lines, driving the, 6-44

GPIO data handshake methods, 6-5

GPIO data-in clock source, 6-7

GPIO data logic sence, 6-5

GPIO ENABLE INTR, 6-37

GPIO events, enabling and setting up, 6-36

GPIO full handshake transfer, 6-40

GPIO full-mode handshakes, 6-8

GPIO handshake lines, 6-6

GPIO handshake logic sence, 6-6

GPIO handshake modes, 6-7

GPIO hardware interrupt priority, 6-5

GPIO interface, 6-1

GPIO interface configuration, 6-3

GPIO interface reset, 6-18

GPIO interface select code, 6-4

GPIO interrupts, 6-36

GPIO interrupt transfers, 6-42

GPIO optional peripheral status check, 6-

- GPIO OUTPUT of data, 7-35
  - GPIO, outputs and enters through the, 6-19
  - GPIO pulse-mode handshakes, 6-11
  - GPIO ready interrupt transfers, 6-42
  - GPIO special-purpose lines, 6-44
  - GPIO statements that enter data bytes, 6-22
  - GPIO statements that enter data words, 6-25
  - GPIO statements that output data bytes, 6-20
  - GPIO statements that output data words, 6-25
  - GPIO STATUS and CONTROL registers, 6-46
  - GPIO status input lines, interrogating the, 6-45
  - GPIO timeouts, 6-29
  - GPIO transfer design, 6-40
  - GPIO, types of interrupt events, 6-36
- H**
- half-duplex telecommunications, 5-68
  - handshake, 3-9
  - handshake and baud rate, RS-232C, 4-6
  - handshake character assignment, data-comm protocol, 5-26
  - handshake, datacomm, 5-23, 5-31
  - handshake lines, HP-IB, 3-54
  - HILBUF\$ function, 9-5
  - HIL Devices, re-configuring, 2-4
  - HIL.ID program, 9-9
  - HIL.ID program explanation, 9-10
  - HIL SEND statement, 9-4
  - HP 1000, disabling auto-poll on the, 5-67
  - HP 13265A modem, automatic dialing with the, 5-34
  - HP 13265 modem, 5-3
  - HP 13266A current loop adapter, 5-3
  - HP 35723A (HP-HIL/Touchscreen), 9-26
  - HP 45911A (11 × 11 Graphics Tablet), 9-26
  - HP 46020/21A keyboard, 9-22
  - HP 46060A (HP-mouse), 9-23
  - HP 46083A (rotary control knob), 9-23
  - HP 46084A (HP-HIL ID module), 9-26
  - HP 46086A (function box), 9-27
  - HP 46087A (A-size digitizer), 9-26
  - HP 46088A (B-size digitizer), 9-26
  - HP 46094A (HP-HIL/quadrature port), 9-23
  - HP 92916A (bar-code reader), 9-28
  - HP 98203C keyboard, 9-22
  - HP 98622 interface, 6-1
  - HP 98626 and HP 98644 card ID register, 4-56
  - HP 98626 optional driver receiver circuits, 4-56
  - HP 98626 RS-232 serial interface, 4-55
  - HP 98628 data communications interface, 5-1
  - HP 98628 serial interface STATUS and CONTROL registers, 4-46
  - HP 98642 4-channel multiplexer, 5-83
  - HP 98642 data communications interface, 5-1
  - HP 98642 serial interface STATUS and CONTROL registers, 4-46
  - HP 98644 baud-rate and line-control registers, 4-60
  - HP 98644 card ID register, 4-59
  - HP 98644 coverplate connector, 4-57
  - HP 98644 optional driver/receiver registers, 4-59
  - HP 98644 RS-232 serial interface, 4-55
  - HP-HIL device characteristics, 9-29
  - HP-HIL device preview, 9-2
  - HP-HIL devices, 9-21
  - HP-HIL devices, communicating with, 9-29

- HP-HIL devices, interaction between multiple, 9-53
- HP-HIL devices supported by the HIL interface driver, 9-6
- HP-HIL ID module data, interpreting, 9-31
- HP-HIL ID modules, note about installing and removing, 9-31
- HP-HIL initialization, 9-2
- HP-HIL interface, 9-1
- HP-HIL interface, communicating through the, 9-3
- HP-HIL interface driver statements, 9-3
- HP-HIL keyboards, 9-22
- HP-HIL link, identifying all devices on the, 9-8
- HP-HIL, other devices, 9-26
- HP-HIL security device, 9-26
- HP-IB
  - abort message, 3-23
  - clear lockout/local message, 3-23
  - clear message, 3-23
  - data message, 3-22
  - local lockout message, 3-23
  - local message, 3-23
  - pass control message, 3-23
  - remote message, 3-23
  - service request message, 3-23
  - status bit message, 3-23
  - status byte message, 3-23
  - trigger message, 3-22
- HP-IB ABORT, 3-12
- HP-IB active controller, 3-9, 3-34
- HP-IB address commands and codes, 3-26
- HP-IB addressed to listen, 3-9
- HP-IB addressed to talk, 3-9
- HP-IB attention line (ATN), 3-9, 3-54
- HP-IB burst I/O mode, 3-6
- HP-IB bus activity, aborting, 3-16
- HP-IB bus, addressing multiple listeners on the, 3-11
- HP-IB bus commands and codes, 3-24
- HP-IB bus-line states, determining, 3-56
- HP-IB bus management, 3-12
- HP-IB bus management, advanced, 3-22
- HP-IB bus messages, explicit, 3-28
- HP-IB bus message types, 3-22
- HP-IB bus sequences, 3-10
- HP-IB CLEAR, 3-12
- HP-IB controller address, 3-35
- HP-IB controller status, 3-35
- HP-IB control lines, 3-53
- HP-IB data movement, 3-4
- HP-IB data valid (DAV), 3-54
- HP-IB devices, clearing, 3-16
- HP-IB devices, communicating with, 3-3
- HP-IB device selectors, 3-3
- HP-IB devices, polling, 3-19
- HP-IB devices, triggering, 3-15
- HP-IB ENABLE INTR, 3-18
- HP-IB end-or-identify line (EOI), 3-55
- HP-IB handshake lines, 3-54
- HP-IB installation and verification, 3-2
- HP-IB interface, 3-1
- HP-IB, interface clear line (IFC), 3-55
- HP-IB interface-state information, 3-48
- HP-IB interlocking handshake, 3-54
- HP-IB interrupts that require data transfers, servicing, 3-50
- HP-IB LOCAL, 3-12
- HP-IB LOCAL LOCKOUT, 3-13
- HP-IB message mnemonics, 3-31
- HP-IB messages, 3-22
- HP-IB NDAC holdoff, 3-61
- HP-IB not data accepted (NDAC), 3-54
- HP-IB not ready for data (NRFD), 3-54
- HP-IB ON INTR, 3-17
- HP-IB PPOLL, 3-13
- HP-IB PPOLL CONFIGURE, 3-13
- HP-IB PPOLL UNCONFIGURE, 3-13
- HP-IB REMOTE, 3-13
- HP-IB remote enable line (REN), 3-55

HP-IB secondary addressing, 3-12  
 HP-IB select code, 3-4  
 HP-IB SEND, 3-13  
 HP-IB service request line (SRQ), 3-56  
 HP-IB service requests, 3-17, 3-44  
 HP-IB SPOLL, 3-13  
 HP-IB SRQ interrupts, 3-17  
 HP-IB STATUS and CONTROL registers, 3-58  
 HP-IB structure, 3-7  
 HP-IB system controller, 3-34  
 HP-IB TRIGGER, 3-13

**I**  
 inbound control blocks, datacomm, 5-7  
 inbound datacomm data messages, 5-10  
 initiating the datacomm connection, 5-36  
 INPUT statement, 5-39  
 integral keyboard, 9-22  
 interactive keyboard, 2-33  
 interface clear line (IFC), HP-IB, 3-55  
 interface differences, 4-55  
 interface ready, 6-37  
 interface reset, RS-232C, 4-11  
 internal representations, 6-19  
 interrupt mask bits for async operation, 5-40  
 interrupt mask bits for data link operation, 5-40  
 interrupt service routine (ISR), 5-43, 6-37  
 interrupt service routines, datacomm, 5-42  
 interrupts, non-active HP-IB controller, 3-37

## K

KBD\$ function, 2-29-30, 9-22  
 KBD LINE PEN statement, 1-10  
 KBD status and control registers, 2-36  
 keyboard auto-repeat, 2-11  
 keyboard CAPS LOCK mode, 2-9

keyboard ENTER, 2-13  
 keyboard features, 2-4  
 keyboard, interactive, 2-33  
 keyboard interfaces, 2-1  
 keyboard interrupts, servicing datacomm, 5-48  
 keyboard, locking out the, 2-34  
 keyboard operating modes, 2-9  
 keyboard OUTPUT, 2-15  
 keyboards, description of, 2-1  
 keyboard types, 2-1  
 KEY LABELS ON/OFF statement, 1-39  
 KEY LABELS PEN statement, 1-10  
 keystrokes, trapping, 2-30  
 knob rotation, 2-27  
 KNOBX function, 2-28  
 KNOBY function, 2-28

## L

line connection, datacomm, 5-32  
 line-control switches, RS-232C, 4-8  
 line speed (baud rate), datacomm, 5-23, 5-30  
 LINPUT statement, 5-39  
 local control, enabling, 3-15  
 local lockout message, 3-23  
 LOCAL LOCKOUT statement, 3-13  
 local message, 3-23  
 LOCAL statement, 3-12  
 locking an interface to a process, 3-6, 6-27  
 locking out local control, 3-14  
 locking out the keyboard, 2-34

## M

Model 216 and 217 built-in interface differences, 4-55  
 modem control register, RS-232C, 4-24  
 modem handshake lines, RS-232C, 4-24  
 modem-initiated ON INTR branching conditions, datacomm, 5-21  
 modem-line disconnect switches, 4-7

modem line handshaking, RS-232C, 4-15  
monochrome enhancement characters, 1-22  
mouse keys, 2-32  
multiplexer, HP 98642 4-channel, 5-83

**N**

NDAC holdoff, HP-IB, 3-61  
non-active HP-IB controller addressing, 3-43  
non-active HP-IB controller interrupts, 3-37  
non-ASCII data transfers, 5-47  
non-ASCII keystrokes, 2-16  
non-data datacomm characters, handling of, 5-25  
not data accepted (NDAC), HP-IB, 3-54  
not ready for data (NRFD), HP-IB, 3-54  
numeric outputs, 1-17

**O**

OFF HIL EXT statement, 9-5  
OFF KBD statement, 2-30  
ON ERROR statement, 4-19, 5-76  
ON HIL EXT statement, 9-4  
ON INTR branching conditions, datacomm, 5-30  
ON INTR branching conditions, datacomm modem-initiated, 5-21  
ON INTR statement, 3-17, 5-38, 5-40  
ON KBD statement, 2-29  
ON KEY interrupts, datacomm service routines for, 5-50  
ON KNOB statement, 2-27, 2-33  
ON/OFF CDIAL statement, 9-24  
ON/OFF KBD statement, 9-22  
ON/OFF KEY statement, 9-22  
ON/OFF KNOB statement, 9-23  
ON TIMEOUT statement, 7-38  
operating parameters, RS-232C, 4-6  
outbound control blocks, datacomm, 5-6

outbound datacomm data messages, 5-9  
output Area and the display line, 1-6  
output-area memory, 1-27  
output to the CRT, 1-17  
overrun error (RS-232C), 4-4, 4-18

**P**

parallel interface, 10-1  
parallel interface STATUS and CONTROL registers, 10-5  
parallel poll, conducting a, 3-20  
parallel poll responses, configuring, 3-19  
parallel poll responses, disabling, 3-20  
parallel polls, responding to, 3-45  
parity bit, RS-232C, 4-3  
parity, datacomm, 5-32  
parity enable (RS-232C), 4-6  
parity error (RS-232C), 4-4, 4-18  
parity option  
    EVEN, 4-4  
    NONE, 4-4  
    ODD, 4-4  
parity options, datacomm, 5-4  
parity, RS-232C character format and, 4-13  
parity sense (RS-232C), 4-6  
pass control message, 3-23  
passing control, 3-36  
pen colors, 1-12  
    changing, 1-36  
pen colors in display regions, changing, 1-10  
peripheral status line (PSTS), 6-45  
plotting selected locations on a Touchscreen, 9-45  
PPOLL CONFIGURE statement, 3-13, 3-19  
PPOLL statement, 3-13, 3-20  
PPOLL UNCONFIGURE statement, 3-13, 3-20  
primary addresses, 3-4