

HP-85[®]

I/O PROGRAMMING GUIDE

$Z = \begin{bmatrix} 10 & 00 \\ 01 & 00 \\ 00 & V_{11}V_{12} \\ 00 & V_{21}V_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & Y \end{bmatrix}$

$X = (A^T A)^{-1} A^T Y$

$A =$	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0000	0.0000
$K =$	-0.0709	0.1861	-0.8383	-0.4806	0.1632
	0.0726	-0.4109	-0.5354	0.7253	-0.1144
	-0.2002	-0.6231	0.1027	-0.1411	0.7357
$R =$	1.4446	1.6867	1.2530		
	0.0	-1.3389	-0.1486		
	0.0	0.0	1.1831		



HP-85

I/O Programming Guide

August 1980

00085-90142

Contents

How To Use This Manual.....	vi
PART I: Beginner's Guide	1
Section 1: Who is I/O and What Is He Doing In My Computer?	1
Section Introduction.....	1
Installing the I/O ROM	2
Removing the I/O ROM.....	3
The Job of an Interface	3
Mechanical and Electrical Compatibility	4
Data Compatibility	4
Timing Compatibility	4
Choosing the Source or Destination.....	5
Using Interface Select Codes	5
Using a Primary Address.....	5
Printing to Peripheral Devices	5
Section 2: Simple I/O Operation	7
Section Introduction.....	7
Using Simple OUTPUT Statements.....	8
Using Simple ENTER Statements	9
Entering Numeric Data	9
Entering String Data	10
Section 3: Formatted I/O Operations	13
Section Introduction.....	13
Formatted OUTPUT	13
Numeric Images	14
String Images	17
Binary Images	18
End-of-Line Sequence Images.....	19
Formatted ENTER	20
Data Images	20
Skipping Unwanted Characters.....	23
Eliminating the Line-feed Requirement.....	24
Advanced Use of Terminator Images	25
Field and Statement Terminators	25
Terminator Images	26
A Word of Advice About Images	28
Converting I/O Data	28
Section 4: Error Handling	31
PART II: Bits and Bytes and Such	33
Section 5: Why Worry About Bits?	36
Section Introduction.....	36
Review of Base 2.....	36
Review of Alternate Representations	37
Review of Logical Operations.....	38
Section 6: Binary Functions	41
Section Introduction.....	41
The Binary AND Function	42
The Binary Inclusive OR Function	42
The Binary Exclusive OR Function	43
The Binary Complement Function	43
The Bit Test Function	44
Section 7: Base Conversion Functions	46
Section Introduction.....	46
Conversions From Base 10 to an Alternate Base	46
From Base 10 to Base 2	46
From Base 10 to Base 8	46
From Base 10 to Base 16	46
Conversions From an Alternate Base to Base 10	47
From Base 2 to Base 10	47
From Base 8 to Base 10	47
From Base 16 to Base 10	47
Converting From One Alternate Base to Another	48

PART III: Advanced I/O Operations	49
Introduction	49
Section 8: Specialized Transfers	51
Section Introduction.....	51
The Care and Feeding of Buffers	56
The Pointers.....	57
Buffer Activity	58
Buffer Status and Control.....	58
Data TRANSFERS	61
Output TRANSFER	61
Input TRANSFER.....	63
Section 9: End-of-Line Branching	65
Section Introduction.....	65
Some Background on Interrupts	65
End-of-Line Branch Programming	66
Introduction	66
Interface Interrupts	66
Timeouts	68
TRANSFER Terminations	70
Interactions and Permutations	71
Section 10: Keyboard Control	76
Section Introduction.....	76
Key Mask Programming	76
Section 11: Direct Interface Communications	79
Section Introduction.....	79
Checking the Status	79
Interface Control.....	81
PART IV: Interface Programming Techniques.....	83
Section 12: Using the HP-IB Interface	85
Section Introduction.....	85
A Basic Introduction to the HP-IB	85
First Things First	86
General Structure of the HP-IB	86
Data Transfers on the HP-IB	87
Controlling the HP-IB	89
Handling Requests for Service.....	90
HP-IB Operations	91
Introduction	91
Turn-on and Check-out	91
Controlling the Bus.....	92
HP-IB Data Transfers	94
Advanced I/O Operations.....	96
Handling Service Requests	102
Non-Controller Operations	104
Handling Interface Problems	106
HP-IB For the Specialist.....	108
HP-IB I/O Statements	108
Typical HP-IB Output Sequence	111
Typical HP-IB Enter Sequence	111
HP-IB Control Registers	112
User-Defined EOL Sequence Registers	115
HP-IB Status Registers	116
Mnemonic Conventions	120
Message Concepts	120
HP-IB Control Lines	122
HP-IB Control Responses	126
HP-IB Universal Commands	126
Available Bus Addresses and Codes	127
HP-IB Statement Summary	128
HP-IB Errors	130
Section 13: Using the Serial Interface	131
Section Introduction.....	131
Introduction to Serial Interfacing	132
Data Terminal Equipment	132
Data Communication Equipment	132
Asynchronous Data Transmission	133

Start Bit	134
Data Character	134
Parity	134
Stop Bits	134
Transfer Rates (or Baud)	135
Handshakes	135
Printer and Terminal Interfacing	136
Printer Interfacing	136
Terminal Interface	138
Teletype Interface	139
Advanced Serial Interfacing	140
Handshakes	140
Modems	141
Auto-originate and Auto-answers Routines	143
Registers	145
Register 0	145
Register 1	145
Register 2	148
Control Register 3	148
Status Register 3	149
Register 4	150
Register 5	152
Register 6	153
Register 7	153
Register 8	154
Register 9	154
Control Register 10	156
Status Register 10	156
Control Register 11	157
Status Register 11	158
Control Registers 12 through 15	158
Control Register 16	159
Control Registers 17 through 23	160
Troubleshooting Hints	161
Serial I/O Statements	162
Serial Interface Errors	162
Section 14: Using the BCD Interface	163
Section Introduction	163
Binary Coded Decimal	163
Data and Handshake Lines	164
Operating Modes	164
Default Formats	164
Data Rates	166
Program Statements	166
Using the Interface	166
Programming with Default Formats	166
Partial Fields	170
Port 10	171
Interrupts	173
Multiple Interrupts	174
Registers	176
Read Register 0	176
Read Register 1	176
Read Register 2	176
Write Register 0	176
Write Register 1	177
Write Register 2	177
Register 3	178
Register 4	179
Register 5	179
Register 6	180
Register 7	180
Register 8	181
Register 9	182
Register 10	183
Non-Standard Formatting	184
Single Channel Formatting	184
Dual Channel Formatting	185
Data Output	187
Transfers	188
BCD I/O Statements	188
BCD Interface Errors	189

Section 15: Using the GPIO Interface	191
Section Introduction	191
Essentials of a Parallel Interface	192
Direction of Data Flow	192
Number of Bits and Ports	193
Using Primary Addresses	194
Handshake Methods	195
Selecting the Handshake Method	202
Setting the Logic Polarity	205
Why Won't This Thing Output?	207
Choosing the Method of Transfer	208
Advanced Capabilities	209
FHS and INTR Transfers	209
EOL Sequence	211
Direct Use of Control Lines	212
Parity	214
Event Interrupts	216
The Trigger Function	219
GPIO I/O Statements	222
GPIO Interface Errors	223
PART V: Appendix	226
Syntax Reference	227
Interface Register Maps	283
Sales and Service Offices	296
Subject Index	299
Error Messages	305
ASCII Character Set	310

How To Use This Manual

Please take a minute to read this introduction so that you can better understand how this manual is organized and how to get the most utility from it. The HP-85 I/O ROM can open a whole new world of capabilities and applications for your desktop computer. This manual takes you from the simple, beginning concepts on through to many advanced and more complex operations.

Part I is the I/O beginner's guide. If I/O programming is a new experience to you, it is recommended that you start reading at the beginning of Part I and keep going until you have learned enough to perform simple operations with your external device. It is not necessary to understand everything in Part I before you can do successful I/O programming. However, if things don't work as you had expected, that is probably an indication that you need to read more. Even if you are an accomplished I/O programmer, it won't hurt to make a quick pass through Part I. The concepts presented are basic, but you still need to know how they are implemented on the HP-85.

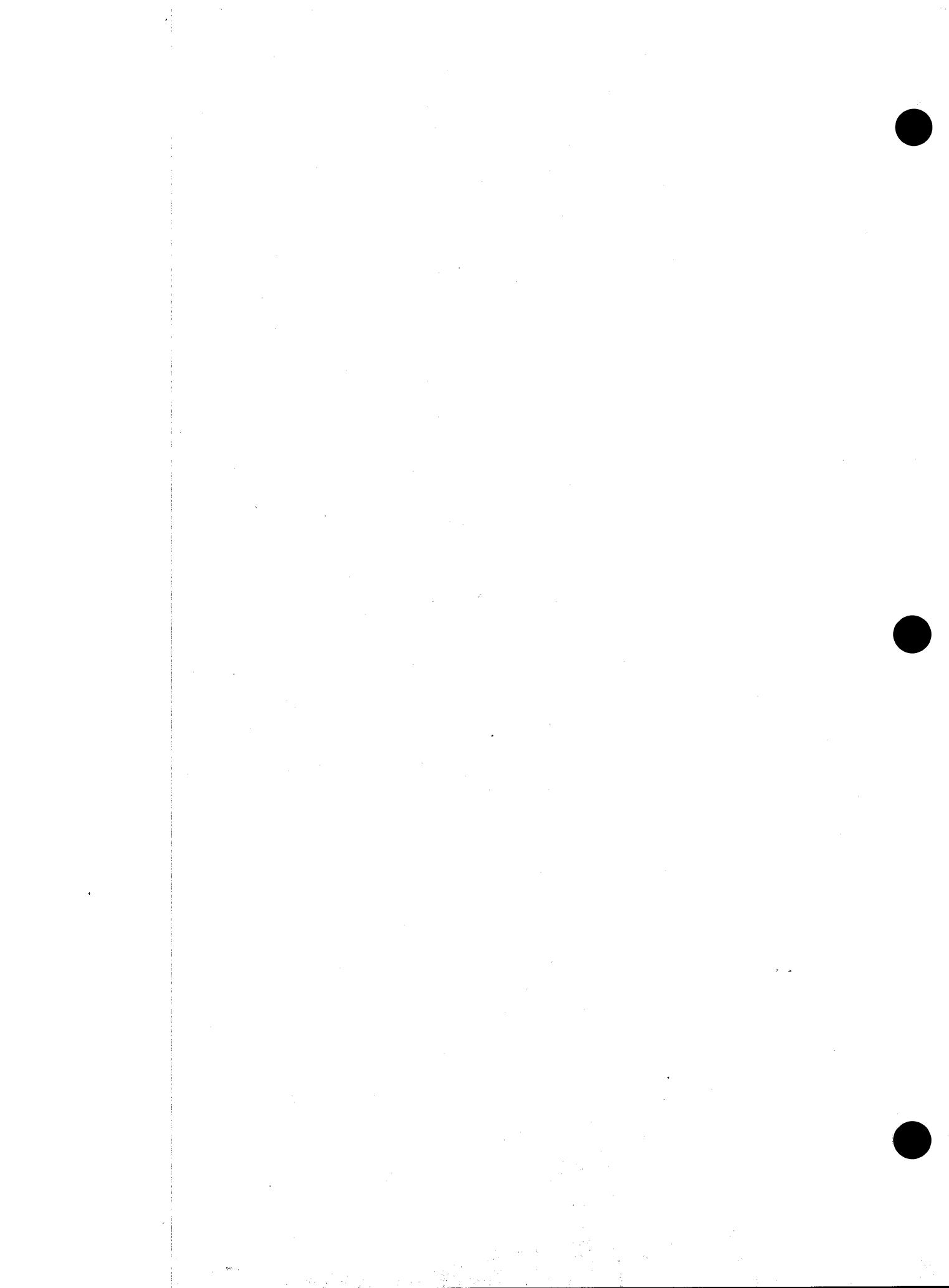
Part II describes the utility functions available for working in alternate number bases. The I/O ROM works just fine in base 10; it is not necessary to use any alternate number bases to do I/O on an HP-85. However, alternate bases can be very helpful, especially when there are so many useful utility functions. If you are already familiar with concepts like base 2, one reading of Part 2 should give you a working understanding of how alternate number bases are handled by the I/O ROM. If Hex and Octal are new to you, don't worry about Part 2 right now. You can always come back to it later.

Part III covers the advanced capabilities of this I/O ROM. If you are an accomplished I/O programmer, this is where you will be spending your time. The HP-85 I/O ROM has an impressive variety of features. It takes careful reading and some practical experience to get optimal benefit from them. If you are a beginner, try to get your program working with the tools presented in Part I. Come to Part III if you find that you need more capability.

Part IV focuses on the "business end" of I/O: the interfaces. A functional I/O system is a blending of software (the program) and hardware (the interface). Part IV talks about the interfaces from a programming point of view. Although an interface like HP-IB keeps you "insulated" from the hardware, HP-IB has a vocabulary of its own. You should at least read the first half of Section 12 to understand the programming statements used with HP-IB. With an interface like the 16-bit GPIO, you have a handful of unconnected wires, and you want all the information possible. Read the Section in Part IV that pertains to your interface. Also, be sure to read the Interface Installation and Theory of Operation manual.

Part V is the Quick Reference Appendix, primarily intended for experienced programmers. It is a reference section, designed to provide rapid access to information. All I/O ROM statements are summarized alphabetically. The syntax of each statement is stressed in this reference, making it handy for those times when you need to see a statement's proper form. In addition to syntax information, this appendix includes descriptions of all parameters, concise descriptions of all statement actions, maps of all interface registers, descriptions of error messages, and other useful tables.

Part I
Beginner's Guide



Who Is I/O and What Is He Doing In My Computer

Section Introduction

You have heard about "IRS", "UFO", and "IOU". Now you are faced with the challenge of learning about "I/O". This stands for "Input/Output", a powerful capability of your HP-85 computer. The Owner's Guide explains all of the many ways to perform computations using your computer. These computations place results into variables. Variables can be chosen to hold numbers or letters. You can see the current contents of any variable by printing it or displaying it. Being human, you can enter numbers or letters into the computer by using the keyboard. These familiar activities can be used to help define some I/O terminology. I/O operations are always referenced to the computer. **Input** means that the data comes from an external device, called the **source**, to the computer. **Output** means that data goes from the computer to an external device, called the **destination**. The external device which is communicating with the computer is often called a **peripheral device**.

When a human is acting as the peripheral device, the keyboard is the source and the CRT (or printer) is the destination. As long as a human is the only peripheral device, this system works well. Humans have little trouble reading data from a CRT. Humans have little trouble inputting data to a keyboard, although some of us have more trouble than others. But the difficulties experienced by a novice typist are very small compared to the difficulty of building a machine that can successfully enter data using a keyboard. To stretch your imagination even further, try to design a machine that can correctly read data from a CRT. Obviously, the keyboard and CRT are not effective vehicles for communicating with other machines. Something else is needed. That **something else** is an appropriate system of I/O interfaces, and the means of communicating with those interfaces. The following sections discuss these two elements of an I/O system in greater detail.

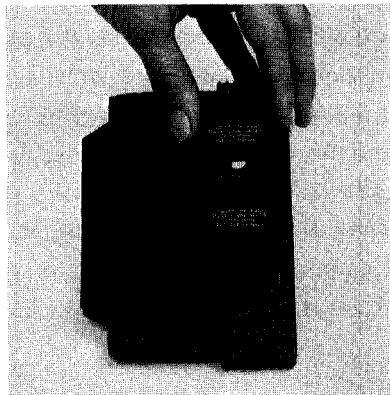
Installing the I/O ROM

CAUTION

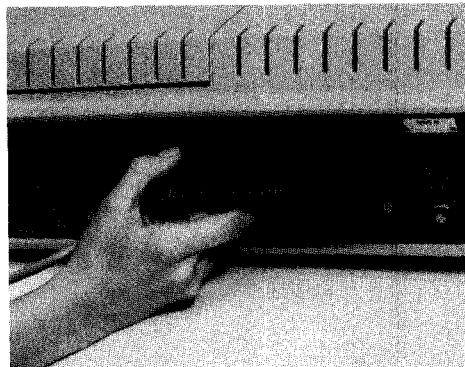
ALWAYS SWITCH OFF COMPUTER POWER BEFORE INSERTING OR REMOVING ROMS AND INTERFACES. FAILURE TO DO SO CAN DAMAGE THE COMPUTER.

The HP-85 has four slots located in the back of the computer. These slots are used to hold extra memory, option ROMs and interfaces. The I/O ROM should be installed in the ROM drawer, which is then installed in one of the slots. The installation procedure is as follows:

1. If the ROM drawer is already plugged into the computer, TURN OFF power to the computer and remove a ROM drawer.
2. Remove the plastic cover from an empty ROM socket.
3. The ROM and its socket are keyed so that the ROM can only be inserted one way. Align the ROM so that its chamfered end matches the chamfered end of the socket.
4. Press the ROM lightly into the socket until it is even with the top of the drawer.



5. With the ROM labels facing up, press the ROM drawer firmly into one of the slots. The drawer and slot are keyed so that the drawer cannot be installed upside down.



Removing the I/O ROM

The procedure to remove a ROM is as follows:

1. TURN OFF power to the computer.
2. Remove the proper ROM drawer.
3. Turn the drawer over and remove the ROM by using a pen, pencil, or small screwdriver to push gently through the hole underneath the ROM.

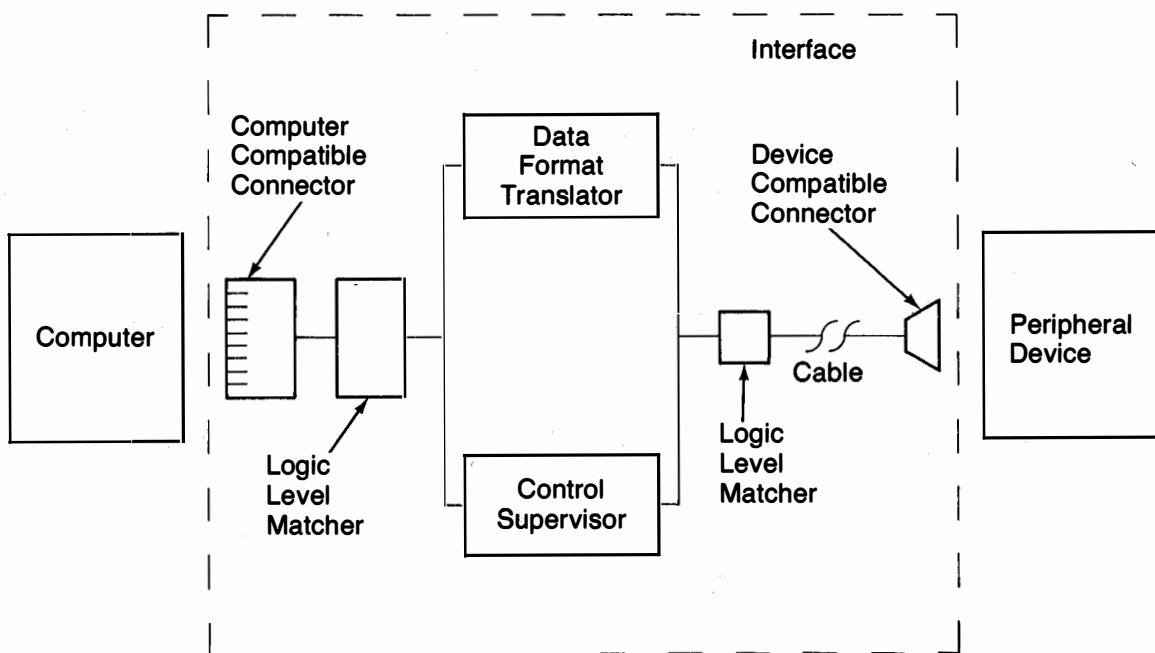
The Job of an Interface

An interface is the hardware link that is needed to allow efficient communication with peripheral devices. The job of an interface is to provide compatibility in four major areas. These are:

- Mechanical Compatibility
- Electrical Compatibility
- Data Compatibility
- Timing Compatibility

The following diagram shows these general roles of an interface in its position between the computer and the peripheral device.

Interface Functional Diagram



Mechanical and Electrical Compatibility

Mechanical compatibility simply means that the plugs and connectors must fit together. The 829XX Series interfaces are designed to be mechanically compatible with your HP-85 computer. Certain interfaces, like HP-IB, are always mechanically compatible with their peripheral devices. Other interfaces, like the 16-bit parallel interface, are ordered without peripheral connectors. In these cases, it becomes your responsibility to install a mechanically compatible connector. If you need to do this, study the Installation and Theory of Operation manual supplied with your interface. Electrical compatibility means that the interface must change the voltage and current levels used by the computer to those used by the peripheral device. The situation here is similar to the mechanical compatibility case. That is, all HP-85 interfaces are electrically compatible with the computer, while only some are automatically compatible with the peripheral device. If you have questions about the electrical compatibility with your peripheral device, study the Installation and Theory of Operation manual.

Data Compatibility

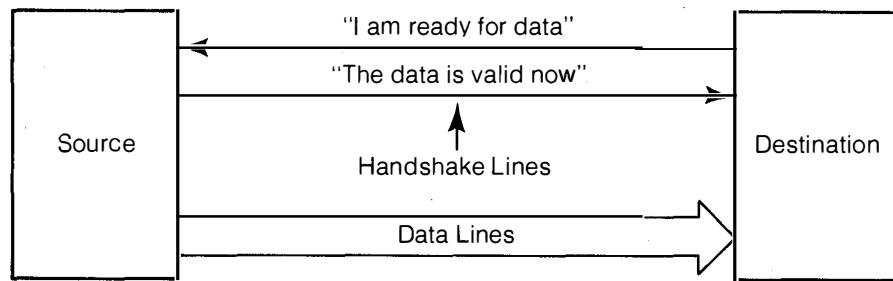
Mechanical and electrical compatibility alone do not guarantee that the computer and peripheral device will be able to communicate. Another requirement is that both devices must understand the data being sent by the other. Just as two humans who do not speak the same language need a translator, messages between the computer and the peripheral device may require some form of translation. The computer, with its versatile programming capability, usually performs this function. However, this job is sometimes given to the interface. The BCD interface is one example of giving the translation process to the interface. To handle those cases where the computer performs the data translation or conversion, the I/O ROM provides a wide variety of special functions and conversion capabilities. These capabilities are covered in subsequent sections of this manual.

Timing Compatibility

The speaking and listening rates of most humans are well matched. Computers and their peripheral devices, on the other hand, have such a wide range of operating speeds that an orderly mechanism is required for successful transfer of data. This timing mechanism is referred to as **handshake**. Although there are different varieties of handshake, the basic sequence can be summarized as follows:

1. The receiver signals that it is ready for an item of data, then waits for a signal from the sender that the data is available.
2. The sender outputs an item of data and signals the receiver when the data is available.
3. When this "data available" signal is recognized, the receiver inputs the data and signals that it is busy with this input operation.
4. The sender waits until the receiver is ready before it makes a new item of data available. When the receiver is ready, this process repeats.

The following simplified diagram further illustrates the general concept of handshaking.



To send a message to someone through the mail, you must specify their address before the post office will even attempt delivery. So it is when you want to communicate with peripheral devices. The device with which you want to communicate must be specified in your program. This selection process is called **addressing**. The HP-85 addresses its peripheral devices through the use of a **device selector** in the I/O statements. A device selector is a number that works alot like the address numbers you would use when mailing a letter. There are two basic kinds of device selectors used on the HP-85. Choosing the proper device selector depends upon the interface used and the way it is connected. The following two discussions detail the two types of device selectors.

Using Interface Select Codes

If you are not using an HP-IB interface, and you have only one device connected to the interface, the device selector can be simply the **select code** of the interface. An interface select code is analogous to a house number in a mailing address. It is a number between 3 and 10 (inclusive) that identifies the interface. Each type of interface is set to a different select code at the factory. The following table summarizes these factory settings.

Part Number	Name	Select Code Setting
82937A	HP-IB	7
82939A	Serial (RS-232)	10
82940A	GPIO (Parallel)	4
82941A	BCD	3

Note: For electrical reasons, you must **never** plug in two interfaces that are set to the same select code.

Serious electrical conflicts can result if the interfaces and select codes do not correspond uniquely. In other words, if there are two interfaces present with the same select code, neither one of them will work. As you can see from the preceding table, the factory settings prevent this from happening unless you are using two interfaces of the same type. If you need to use two interfaces that came with the same factory setting, you must change the select code on one of them. The procedure for changing a select code is covered in the Installation and Theory of Operation manual for your interface. Follow its instructions carefully.

Using a Primary Address

If you are using an HP-IB interface, or if you have more than one device connected to an interface, the device selector is a 3-digit or 4-digit number formed from the interface select code and a **primary address**. This method of addressing is like mailing a letter to someone in an apartment building. Giving the street address will get the letter to the right building, but you still need to specify an apartment number to get the letter to the final destination. When a primary address is used, it is analogous to the apartment number. It identifies a specific device to be selected from a group of devices serviced by one interface. Some examples:

- A device selector of **721** specifies device **21** on interface **7**.
- A device selector of **301** specifies device **1** on interface **3**.
- A device selector of **1002** specifies device **2** on interface **10**.

Printing to Peripheral Devices

One of the simplest ways to direct the computer's output to a peripheral device is the "PRINTER IS" statement. The Owner's Guide tells you about directing output to the CRT by specifying "PRINTER IS 1" or to the internal printer by specifying "PRINTER IS 2". The I/O ROM provides the capability of printing to external devices by using statements such as "PRINTER IS 4" or "PRINTER IS 720". Any valid device selector can be used with the "PRINTER IS" statement. The same holds true for the "CRT IS" statement. Your previous experience with "PRINTER IS" should help you understand why the interface select codes on the HP-85 start at 3, instead of 1. Device selectors 1 and 2 are assigned to the CRT and internal printer, respectively.

The "PRINTER IS" device is the destination for the output from all "PRINT" and "PLIST" statements. The "CRT IS" device is the destination for the output from all "DISP", "LIST", and "CAT" statements, as well as all "Error" and "Warning" messages. Graphics output always goes to the internal CRT, not to the "CRT IS" device. When programs are listed to a peripheral device, each program line is output as a single string. There is no indenting or 32-character wraparound as occurs when listings are done on the internal printer.

Note: If your only I/O requirement is to direct program listings or the output from "PRINT" statements to an external device, you need not read any further. Simply connect the desired interface and include the appropriate "PRINTER IS" statement in your program.

Section 2
Simple I/O Operations

Section Introduction

Section 1 talked about performing output operations with PRINTER IS and PRINT statements. Although this simple technique is very handy, it falls short of the mark in many circumstances. The most obvious shortcoming is that there is no corresponding “KEYBOARD IS” statement to allow input from external devices. Even when output is the only desired operation, it can be very inconvenient re-specifying the PRINTER IS device all the time when a program communicates with multiple peripheral devices.

The principal tools for using interfaces to move data in and out of the computer are the **OUTPUT** and **ENTER** statements. These statements are the “core” of I/O operations. They are usually the fastest and easiest ways of getting data from the source to the destination in its final form. Many applications require no more than the proper use of OUTPUT and ENTER.

Simple OUTPUT and ENTER statements (as described in this section) use ASCII representation for all data. **ASCII** stands for “American Standard Code for Information Interchange”. It is a commonly used code for representing letters, numerals, punctuation, and special characters. The ASCII code provides a standard correspondence between binary codes that are easily understood by the computer and alpha-numeric symbols that are easily understood by humans. A complete list of the characters in the ASCII set and their code values is included in the Appendix.

When special formatting is desired or when binary code is handled directly without using ASCII representation, the “OUTPUT USING” and “ENTER USING” forms are very convenient. These forms are discussed in Section 3.

Using Simple OUTPUT Statements

A simple OUTPUT statement can be used anywhere that a simple PRINT statement is proper. The OUTPUT statement contains the device selector(s) of the destination device(s) and a list of the items to be output. The primary difference between OUTPUT and PRINT is that PRINT statements do not contain a device selector. Here are some examples of properly syntaxed OUTPUT statements:

```
OUTPUT 1 ; "Hello"
OUTPUT 3 ; X
OUTPUT S1 ; A$,B$
OUTPUT 703,725 ; X;Y;Z
OUTPUT 1000 ; A(1);B(3),N$[2,7]
```

Notice that a semicolon is used to separate the device selector from the output list, and commas or semicolons can be used to separate items within the output list. Items in the output list may be numeric variables, numeric constants, string variables, or string constants. A Carriage Return/Line Feed (End of Line sequence) is output after the last item in the output list.

The difference between using a comma and a semicolon to separate items in the output list is the spacing, or **field** of the items. The simple OUTPUT statement uses the same field widths as the PRINT statement. The semicolon calls for a compact field, while the comma produces free field. These fields are summarized in the following table.

	Numeric Data	String Data
Compact Field	Digits of the number are output, preceded by a space (if plus) or a minus sign (if minus), and followed by one space.	Characters of the string are output with no leading or trailing spaces.
Free Field	Digits of the number (with leading space or minus sign) are output left-justified in a field of 11,21, or 32 characters. Trailing spaces are output as necessary to fill the unused portion of the field.	Characters of the string are output with no leading spaces and no more than 20 trailing spaces.

The actual field width in free field is determined by the same process used when items are output with the PRINT statement. Therefore, the computer "pretends" that it is displaying items on the CRT and sets a field width that would cause the items to start in column 1 or 22 of the 32-column display. Chances are that this type of free field will be of little use on most peripheral devices. For example, columns of numbers will not be lined up if they are sent to an 80-column wide printer. However, it is easy to circumvent this "problem" by separating items in the output list with semicolons, or by using formatted output as explained in Section 3.

Using Simple ENTER Statements

A simple ENTER statement can be used anywhere that an INPUT statement is proper. The ENTER statement contains the device selector of the source device and a list of items to be entered. Remember that INPUT statements always use the keyboard as the source and contain no device selector, while ENTER statements always use a peripheral device as the source and contain the device selector of that device. Here are some examples of properly syntaxed ENTER statements:

```
ENTER 3 ; X
ENTER S1 ; A$,B$,C$
ENTER 703 ; X,Y,Z
ENTER 1000 ; A(1),B(3),N$
```

Notice that a semicolon is used to separate the device selector from the enter list and commas are used to separate items within the enter list. Items in the enter list may be numeric variables or string variables.

To use the ENTER statement effectively, it is important to understand what constitutes the beginning and ending of an entry into a variable. The simple ENTER statements just shown use a “free field format” for processing incoming characters. This format operates differently with string and numeric data.

Entering Numeric Data

The computer enters numeric values by reading the ASCII representations of those values. For example, if the computer reads an ASCII “1”, then an ASCII “2”, and finally an ASCII “5”, it places the value one hundred twenty five into a numeric variable.

Understanding the process that the computer uses to read a free field number can help you remove much of the “mystery” from I/O. Suppose your program has the statement:

```
ENTER 3 ; X,Y
```

Now assume that when this statement is executed, the following character sequence is received through the interface at select code 3:

T	U	E	S	D	A	Y		D	E	C		1	1	,		1	9	7	9	cr	lf
---	---	---	---	---	---	---	--	---	---	---	--	---	---	---	--	---	---	---	---	----	----

The computer ignores all leading non-numeric characters, so the “TUESDAY DEC” characters do nothing. Then the “11” is read. Once the computer has started to read a number, a non-numeric character signals the end of that number. Therefore, the comma after the 11 causes the computer to place the value eleven into X and start looking for the next value. The space in front of “1979” is ignored and the computer reads the “1979”. The carriage-return character causes the computer to place the value nineteen hundred seventy nine into variable Y. Finally, the computer keeps reading until it finds a line-feed character. This terminates the ENTER statement, so the computer goes on to the next program line with X=11 and Y=1979.

The process just described can be easily summarized. When entering numeric data using free-field format, the computer:

1. Ignores leading non-numeric characters.
2. Ignores all spaces — leading, trailing, or imbedded.
3. Uses numeric characters to build a numeric value.
4. Terminates the building of a value when a trailing non-numeric character is encountered.
5. Inputs characters until a line-feed character is encountered

The discussion so far has referred to numeric and non-numeric characters without being specific. The digits "0" thru "9" are always numeric characters. Also, the decimal point, plus sign, minus sign, and the letter "E" can be numeric if they occur at a meaningful place in a number. For example, assume that the following character sequence is read by an ENTER statement:

-	-	T	E	S	T		1	2	.	5	E	-	3
---	---	---	---	---	---	--	---	---	---	---	---	---	---

If a numeric value is being entered, the leading minus signs and the "E" in "TEST" will be ignored. They have no meaningful numeric value when surrounded by non-numeric characters. However, the characters "12.5E-3" will be interpreted as 12.5×10^{-3} . In this case, the minus sign and the exponent indicator ("E") occur in a meaningful numeric order, so they are accepted as numeric characters.

Entering String Data

The computer enters string data by placing ASCII characters into a string variable. The process used for free-field entry is straightforward. All characters received are placed into the string until:

1. The string is full or,
2. A line-feed character is received or,
3. A carriage-return/line-feed sequence is received

Assume that the computer is executing the statement:

ENTER 4 ; A\$,B\$,C\$

The following character sequence is received:

H	E	L	L	O	If	cr	If	T	H	E	R	E	cr	If
---	---	---	---	---	----	----	----	---	---	---	---	---	----	----

The letters "HELLO" are placed into A\$ when the first line-feed is encountered. Note that the line-feed itself is not placed into A\$; it acts only as a terminator for the entry into A\$. The entry into B\$ begins. However, a carriage-return/line-feed sequence is read immediately. This terminates the entry into B\$. Since neither the carriage-return or the line-feed is placed into B\$, B\$ becomes the null string. Next, the entry into C\$ begins. The characters "THERE" are placed into C\$, terminated by the carriage-return/line-feed following those characters. With the enter list now satisfied and a line-feed detected at the end of the data, the computer will go on to the next program line.

Note that carriage-return characters are only ignored when they are immediately followed by a line-feed character. If a carriage-return is received and not followed by a line-feed, the carriage-return is placed into the string.

Another example can be used to show termination on a full string. This time, suppose the program contains the following statements:

```
DIM X$[3]
ENTER 4 ; X$
```

The following characters are sent to the computer:

B	O	Y	C	O	T	T	cr	lf
---	---	---	---	---	---	---	----	----

The computer places the characters "BOY" into X\$, which fills the dimentioned length of 3. Then the computer continues to read the incoming characters until a line-feed is encountered. At that time, the ENTER statement is completed, and the computer goes on to the next program step with X\$="BOY".



Section 3

Formatted I/O Operations

Section Introduction

Although free-field format works well for some I/O situations, there are times when more control over format is necessary. Perhaps the data is some binary pattern which has nothing to do with ASCII, or a line-feed terminator is not wanted or expected, or a column of numbers with the decimal points in line is desired, or numbers with only 2 exponent digits instead of 3 are required. There is a wide variety of reasons for desiring format control during I/O operations.

The format of information sent or received through interfaces is controlled by the use of **image specifiers**. These image specifiers can be placed in an image statement or can be included directly in an OUTPUT or ENTER statement. This section of the manual provides details on the meaning and use of image specifiers.

Formatted OUTPUT

An output image can control all major characteristics of output data, including spacing, appearance of the field, form of data representation, and use of end-of-line sequences. The HP-85 uses an output image when some form of the OUTPUT USING statement is encountered. There are two forms of this statement:

1. 10 IMAGE <output image>
20 OUTPUT ds USING 10 ; <output list>
2. OUTPUT ds USING <output image> ; <output list>

The examples above show the general forms of the OUTPUT USING statement. Here are some specific examples:

10 IMAGE "Total =",ZZ.D

20 IMAGE 5A,2X,17A

60 OUTPUT 4 USING 10 ; C1,C2,C3

70 OUTPUT 701 USING 20 ; A\$,B\$

80 OUTPUT 9 USING "#,B" ; X

90 OUTPUT S3 USING "MDDD.DD" ; T(1),T(2)

100 OUTPUT 710,711 USING I\$; N\$,A

In the general forms, the “ds” stands for “device selector”. Device selectors are explained in Section 1. The symbol “<output image>” represents a proper list of image specifiers. The image specifier list may be a literal enclosed in quotes or the name of a string variable which contains the specifier list. The specifiers within the list must be separated by commas. The list of items to be output is shown by “<output list>”. It does not matter whether you use commas or semicolons to separate items within the list. All spacing is controlled by the image specifiers, so a semicolon has the same effect as a comma.

Numeric Images

The image specifiers in this group are used to control the form of numbers which are output. Most of these image specifiers are the same as the PRINT image specifiers that may already be familiar to you. Since there are many numeric images, these specifiers are broken down into three categories in the following discussion. The categories are “digit characters”, “sign character”, and “punctuation characters”.

Digit Characters

These are the image specifiers which form the digits of the number. They allow you to determine the number of digits before and after the decimal point, display or suppress leading zeros, and control the inclusion of exponent information.

Image Specifier	Meaning
D	Causes one digit of a number to be output. If that digit is a leading zero, a space is output instead. If the number is negative and no sign image has been provided, the minus sign will occupy one digit place. If any sign is output, the sign will “float” to a position just left of the left-most digit.
Z	Same as “D”, except leading zeros are output.
*	Same as “D”, except leading zeros are replaced by asterisks.
E	Causes the number’s exponent information to be output. This is a 5-character sequence including the letter “E”, the exponent sign, and three exponent digits.
e	Same as “E”, except only two exponent digits are output.
K	Causes the number to be output in compact format. No leading or trailing spaces are output.

Sign Character

These are the image specifiers used to control the output of sign information. Note that if no sign specifier is included in the image, negative numbers will use a digit position to output the minus sign.

Image Specifier	Meaning
S	Causes the output of a leading plus or minus sign to indicate the sign of the number.
M	Causes the output of a leading space for a positive number or a minus sign for a negative number.

Punctuation Characters

These are the image specifiers used to control the output of punctuation within a number, such as the inclusion of a decimal point.

Image Specifier	Meaning
•	Causes an American radix point to be output (a decimal point).
R	Causes a European radix point to be output (a comma).
C	Usually placed between groups of three digits. Causes a comma to be output to separate the groups of digits (American convention).
P	Same as "C", except a period is used to separate the groups of digits (European convention).

It would be unrealistic to attempt examples of all possible combinations of these numeric image specifiers. The following examples show some of the many ways of combining these specifiers and the resulting output when numbers are sent to a typical printer. Additional examples for many of the specifiers can be found in the "Printer and Display Formatting" section of the HP-85 Owner's Manual.

Example Statement**Printed Output**

OUTPUT 701 USING "ZZZZ.DD" ; 30.336	0030.34
OUTPUT 701 USING "4Z.2D" ; 30.336	0030.34
OUTPUT 701 USING "4Z.2D" ; -30.336	-030.34
OUTPUT 701 USING "3DC3DC3D" ; 1E6	1,000,000
OUTPUT 701 USING "3DC3DC3D" ; 1.2345E4	12,345
OUTPUT 701 USING "3DC3DC3D" ; 1.2E9	(Overflow Error) , ,
OUTPUT 701 USING "S2.DDD" ; .5	+0.500
OUTPUT 701 USING "MZ.DDD" ; .5	0.500
OUTPUT 701 USING "MD.DDD" ; .5	.500
OUTPUT 701 USING "Z.DDE" ; .00456	4.56E-003
OUTPUT 701 USING "Z.DDe" ; .00456	4.56E-03
OUTPUT 701 USING "Z.DDe" ; -.00456	-.45E-02
OUTPUT 701 USING "M2.DDe" ; -.00456	-4.56E-03

Notice in these examples that the image "ZZZZ" and the image "4Z" mean the same thing. The same is true for the "D" and "*" specifiers. You can indicate the number of digits desired by simply placing that number in front of the specifier. The use of parentheses, as in "3(D)", means something different. The image "3D" means "output one numeric quantity in a three-digit field". The image "3(D)" means "output three numeric quantities, putting each one in a 1-digit field".

Be careful of overflow conditions when using these image specifiers. An overflow occurs when the number of digits required to accurately represent a number is greater than the number of digits allowed for in the image. If this happens, a warning is issued and something is output so that the program can continue. However, exactly **what** is output is difficult to predict and will probably bear little or no resemblance to the number that caused the overflow.

String Images

The image specifiers in this group deal with the output of string characters. They can also be used in combination with the numeric image specifiers for spacing and labeling purposes. All of these image specifiers are the same as PRINT image specifiers that may already be familiar to you.

Image Specifier	Meaning
A "literal"	<p>Causes the output of one string character. If all the characters in the current string have been used already, a trailing blank is output.</p>
	<p>A "literal" is a string constant formed by placing text in quotes, using the CHR\$ function, or a combination of the two. The character sequence specified is output when a literal image is encountered. When the literal is enclosed in quotes, the quote marks themselves are not output.</p> <p>Literal images are commonly used for labeling other output.</p> <p>Literal images cannot be placed directly into OUTPUT statements. An IMAGE statement must be used if literal images are desired.</p>
X	<p>Causes the output of one space.</p>
K	<p>Causes the string to be output in compact format. No leading or trailing spaces are output.</p>

The following examples show some of the many ways of using these specifiers and the resulting output when the characters are sent to a typical printer. Additional examples for these specifiers can be found in the "Printer and Display Formatting" section of the HP-85 Owner's Manual.

Example Statements

Printed Output

```
OUTPUT 701 USING "5A,A" ; "X", "Y"
OUTPUT 701 USING "K,3X,K" ; "UNCLE", "SAM"
OUTPUT 701 USING "K,3X,K" ; 98.6,99.9
```

X	Y
UNCLE	SAM
98.6	99.9

```
10 IMAGE "TOTAL = ",3D,X,K
20 T=125 @ A$="CARS"
30 OUTPUT 701 USING 10 ; T,A$
```

TOTAL = 125 CARS

Notice that the "X" and "A" image specifiers allow a number before them in the same fashion as the "D", "Z", and "*" specifiers. The "K" specifier works equally well with string data or numeric data. String and numeric image specifiers can be combined in the same image statement. If literal (string constant) images are desired, they must be placed in an IMAGE statement.

Binary Images

These image specifiers are not available without the I/O ROM, so they may not already be familiar to you. These images are used to cause information to be output as one or two binary bytes, rather than as a character representation. Part II of this manual explains the details of binary (base 2) representation. If you are unfamiliar with binary numbers, it is suggested that you read Section 5 before trying to use the binary image specifiers.

The items to be output using these images must be numbers in the proper range. If a value to be output is not an integer, it will be rounded to the nearest integer before being sent as a binary value.

Image Specifier	Meaning
B	Outputs a value as a single 8-bit byte. The value must be in the range of 0 thru 255. If the value to be output is out of range, the value MOD 256 is output.
W	Outputs a value as two 8-bit bytes comprising a 16-bit word. The most significant byte of the word is output first, followed by the least significant byte. The value to be output must be in the range of -32 768 to +32 767. Negative numbers are output in 16-bit 2's complement form. If the value to be output is out of range and positive, 32 767 is output. If the value is out of range and negative, -32 768 is output.

Example Statement

Example Statement	Bit Pattern Output
OUTPUT 3 USING "B" ; 127	01111111
OUTPUT 3 USING "B" ; 3	00000011
OUTPUT 3 USING "W" ; 3	00000000 00000011
OUTPUT 3 USING "W" ; -1	11111111 11111111

Note that specifying a binary image does not automatically suppress the end-of-line sequence after the last byte is output. Therefore, in the examples just given, the bit pattern shown is output followed by a carriage-return/line-feed.

End-of-Line Sequence Images

These image specifiers control the output of end-of-line sequences. An end-of-line sequence is one or more characters that is normally output after the last item in an output list, and/or a signal on an interface wire concurrent with the last byte output. Exactly which characters or signal is used depends upon the programming of the interface responsible for the output. Part IV of this manual covers this and all other interface-dependent details. If your program does not change the end-of-line sequence in the interface, the default is a 2-character sequence; a carriage-return followed by a line-feed. The following images do not alter the end-of-line sequence. They simply control whether or not it is output.

Image Specifier	Meaning
/	Causes the output of an end-of-line sequence. Often used for skipping lines in a printout.
#	Suppresses the output of the final end-of-line sequence. This specifier is frequently used with binary image specifiers to prevent the destination device from interpreting the end-of-line characters as binary data.

The “/” may be placed anywhere in the image list and may have a number before it to indicate how many EOL (end-of-line) sequences are desired. The “#” must be the first item in an image list and can only be specified once. Note also that the “#” only suppresses the EOL sequence that would ordinarily occur after the last item in the output list. It does not suppress any imbedded EOL sequences caused by the “/” specifier.

A typical use of the “#” image is to output one byte, and only one byte. The following statement does this:

OUTPUT 6 USING “#,B” ; X

This statement outputs the binary representation of X with no carriage-return, line-feed, or any other potentially unwanted bit patterns.

A typical use of the “/” image is shown by the statement:

OUTPUT 701 USING “K,4/,K” ; A\$,B\$

- If the destination is a printer, A\$ is printed, followed by four blank lines, then B\$ is printed. If A\$=“HI” and B\$=“JOE”, the character sequence output looks like this:

H	I	cr	lf	cr	lf	cr	lf	cr	lf	J	O	E	cr	lf
---	---	----	----	----	----	----	----	----	----	---	---	---	----	----

Formatted ENTER

Using ENTER statements with image specifiers gives you a high degree of control in two areas:

1. Accurately describing to the computer what the incoming data looks like and what should be done with it.
2. Precisely specifying what condition(s) constitutes the end point of an entry to a variable and the end point of the ENTER statement itself.

This discussion deals with data formatting images first, then presents the terminator images. The HP-85 uses an ENTER image when some form of the ENTER USING statement is encountered. There are two forms of this statement:

1. 10 IMAGE <enter image>
20 ENTER ds USING 10 ; <enter list>
2. ENTER ds USING <enter image> ; <enter list>

The examples above show the general forms of the ENTER USING statement. Here are some specific examples:

```
10 IMAGE 2(A),K
20 IMAGE 5D,2X,3De
.
.
.
60 ENTER 4 USING 10 ; A$,B$,X
70 ENTER 711 USING 20 ; I,J
80 ENTER 9 USING "#,B" ; A(1),A(2)
90 ENTER S2 USING "%,8A,/,K" ; Q$,R$
100 ENTER 712 USING I$ ; N$,A
```

The general forms use the same type symbols which were used to represent the OUTPUT statement. These are "ds" for "device selector", "<enter image>" for the list of image specifiers, and "<enter list>" for the list of variables to be entered. As with simple ENTER statements, the enter list must contain either string or numeric variables. You can't enter into a constant.

Data Images

The image specifiers in this group are used to tell the computer what to do with the incoming data stream. The basic choices are:

1. Use characters to build a numeric variable.
2. Place characters into a string variable.
3. Input bytes as binary values.
4. Skip over a number of characters.

Numeric Image Specifiers

These specifiers are used to control the input of numeric characters, including digits, sign, exponent, and punctuation.

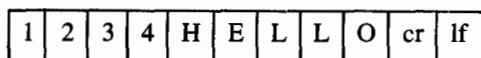
Image Specifier	Meaning
D } Z } * } • } S } M }	These specifiers all do the same thing. They tell the computer to accept one character to be used in building a numeric quantity. The incoming characters do not have to follow the specified format, there just has to be the right number of characters. The six different specifiers are provided so that your program can document the expected format of the characters, and so that ENTER and OUTPUT statements can share the same IMAGE statement, if desired.
E	Tells the computer to accept five characters to be used for building a number. The five characters do not have to be exponent information, but they can be.
e	Same as "E", except the computer accepts four characters to be used in building a number.
C	This specifier also tells the computer to accept one character to be used in building a numeric quantity. However, if a "C" is present anywhere in a number's image, all commas will be ignored while the number is being entered. Without this specifier, a comma would end the entry of a numeric quantity.
K	Tells the computer to enter a string or numeric variable using free-field format (explained in Section 2).
R } P }	These specifiers are used with the OUTPUT statement to provide a European radix point and digit separator. However, these images are NOT permitted for an ENTER statement. If you need to enter numbers in European format, you can use the CONVERT statement (covered later in this Section) to change the number into American format.

String Image Specifiers

These specifiers are used to enter characters into string variables.

Image Specifier	Meaning
A	Tells the computer to enter one string character.
K	Tells the computer to enter a string or numeric variable using free-field format (explained in Section 2).

Some examples are in order. Suppose the following character sequence is received by the computer:



Any of the following ENTER statements can be used to enter a numeric variable followed by a string variable:

```
ENTER 720 USING "4D,5A"; X,Y$  
ENTER 720 USING "Z.DD,5A"; X,Y$  
ENTER 720 USING "e,K"; X,Y$
```

Notice that any numeric image that accepts four characters will properly enter the "1234". String data can be entered with an "nA" image if n (number of characters) is known, or with a "K" if the number of characters is unknown.

Suppose instead that the incoming data was:



The ENTER image would now have to include a "C" for the entire "1234" to be entered. For example:

```
ENTER 720 USING "C4D,K"; X,Y$  
ENTER 720 USING "DDDC,5A"; X,Y$
```

Notice that the "C" does not have to appear at the same place in the image as the comma does in the incoming data. However, the comma is counted as a character.

Binary Image Specifiers

These specifiers are used to enter data that is received in binary format.

Image Specifier	Meaning
B	Tells the computer to enter one byte of binary data and enter its equivalent decimal value into a numeric variable.
W	Tells the computer to enter two bytes of binary data to be used in building a 16-bit, 2's complement binary word. The equivalent decimal value of the resulting word is entered into a numeric variable. The first byte entered is used as the most-significant byte of the word.

Skipping Unwanted Characters

These specifiers can be used with incoming numeric or string data to skip over any characters not wanted for the input.

Image Specifier	Meaning
X	Tells the computer to skip over one character.
/	Tells the computer to skip to a line-feed. Thus, after the variable has been satisfied, the computer "throws away" incoming characters until a line-feed is received.

The "X" specifier should only be used when you have a good understanding of the structure of the incoming data, but can very useful in formatting operations. For example, suppose that text is being entered from a remote computer that sends a line number at the beginning of every string. You know that the line number information always appears in the first 8 characters of each string, and you don't want these line numbers in your data. The following format could be used to strip off the line numbers:

ENTER 720 USING "8X,K" ; A\$

The "/" specifier is used to demand a line-feed field terminator before going on to the next variable. To see the effect of this specifier, assume that the incoming data is as follows:

1	2	3	H	I	lf	B	Y	E	cr	lf
---	---	---	---	---	----	---	---	---	----	----

Using the statement:

```
ENTER 720 USING "3D,K" ; X,Y$
```

causes X to get the value 123 and Y\$ becomes "HI". However, if the statement:

```
ENTER 720 USING "3D,/K" ; X,Y$
```

is used, then X gets the value 123 and Y\$ becomes "BYE". The "/" specifier caused the computer to skip all characters after X was entered until it saw the line-feed. Then the entry into Y\$ began with the first character after the line-feed. Without the "/" specifier, the entry into Y\$ began as soon as the "3D" field was exhausted.

Eliminating the Line-feed Requirement

The ENTER statement must "see" a line-feed character at the end of the incoming data before the program can go on to the next statement. If there is no line-feed character at the end of the data, the computer will be "hung up" waiting for one. If your incoming data does not have a line-feed at the end, you can get the ENTER statement working properly by using an image specifier.

Image Specifier	Meaning
#	Eliminates the requirement for a line-feed to terminate the ENTER statement. When this specifier is present, the ENTER statement terminates as soon as the last variable in the statement has been satisfied.

When the "#" specifier is used for this purpose, it must be listed as the first specifier in the image list. For example:

```
ENTER 3 USING "#,K" ; A$  
ENTER 720 USING "#,4D,6D" ; X,Y
```

The first example statement shows an entry into a string variable using free-field format with the line-feed requirement removed. This statement terminates when the string is full. The second example shows a formatted entry into numeric variables with the line-feed requirement removed. This statement terminates after inputting ten characters.

Advanced Use of Terminator Images

The ENTER image specifiers discussed in the preceding sections are sufficient to handle the great majority of requirements. However, there are some special situations that demand an even greater amount of flexibility. Most of these special cases involve the EOI line on the HP-IB. The following discussion is probably of no concern to most programmers. If you are one of those who must consider the EOI line, or if you have an unusual problem with line-feeds, then read carefully. This is the most complex part of the "Beginner's Guide".

Field and Statement Terminators

The purpose of an ENTER statement is to read a "record". To the programmer, a **record** is a logical grouping of data items. To the computer, a record is an incoming stream of data ended with a record terminator. Since the ENTER statement is ended when the record terminator is read, this manual refers to the record terminator as a "statement terminator". If there is a requirement for a statement terminator in effect, the ENTER statement does not end until that terminator is received. (The action is slightly different when using buffers. These are covered in Section 8.) If no terminator image is specified, the default statement terminator is a line-feed character. To allow a carriage-return/line-feed sequence as a statement terminator, the I/O ROM ignores a carriage-return if it is immediately followed by a line-feed.

An incoming record often contains multiple "fields". A **field** is the group of characters used to determine the input to a variable in the ENTER list. For example, an ENTER statement used to input a list of names and ages might look like this:

```
ENTER 720 ; N$,A
```

This statement reads a record containing a name and an age. This record has two fields. The first is a string field (the name), and the second is a numeric field (the age). A properly specified ENTER statement places the string field in N\$ and the numeric field in A.

It is not generally necessary to specify any terminator images to get the ENTER statement to perform properly. The system has built-in field terminators and a default statement terminator which are sufficient for most common applications. These normal terminators are:

- A string field ends when the string is full (check your DIM statements), the character count from an image field is exhausted, or a line-feed is received.
- A numeric field ends when any non-numeric character (except a space) is encountered or the character count from an image field is exhausted.
- The ENTER statement ends upon receipt of a line-feed character or a carriage-return/line-feed sequence. This can be the same line-feed that satisfied the last field in the ENTER list.

Given these normal terminating conditions, the ENTER statement mentioned previously properly separates the name field and the age field in two cases. One case is when there is a line-feed separating the string field from the numeric field. The other case is when the string field is always of fixed length and N\$ is dimensioned to that length.

Terminator Images

If the “normal” terminating conditions are not ideally matched to your application, the use of terminator images can help solve the problem. The following image specifiers apply to both field and statement terminating conditions. Field terminators are the conditions that end the entry of data into a variable. Statement terminators are the conditions that end the ENTER statement after the last variable is satisfied.

Image Specifier	As a Field Terminator	As a Statement Terminator
#	Eliminates line-feed as a terminating condition during free-field string entry. Line-feeds entered are placed into the string.	Suppresses the requirement for a line-feed terminator. Statement ends when last field is satisfied.
%	Allows EOI as an additional terminating condition.	Allows EOI or line-feed as terminating conditions.
#% (or %#)	Allows EOI as an additional terminating condition, and also eliminates line-feed as a terminator during free-field string entry.	Specifies that an EOI must be received to terminate the statement, and line-feed is not a terminator.

Whether an image specifier controls statement terminators or field terminators depends upon where it is placed in the image. Consider the following example statement:

```
ENTER 720 USING "%,%K" ; X
```

When the terminator image is specified by itself as the first item in the image list (like the first %), it specifies the statement terminator. When the terminator image is combined with another specifier (like the %K), it specifies a field terminator. The “#”, “%”, and “#%” images all follow this convention.

Because the built-in field terminators are always in effect, these special terminator images only alter the system’s action in a few cases. Let’s look at each of these meaningful field terminating combinations individually.

Entering Line-feeds Into a String

The image “#K” causes the computer to place all incoming characters (including line-feeds) into a string until it is full. If there is a line-feed forthcoming after the string is filled, this image is all that is necessary. If you wish the statement to end as soon as the string is filled (without waiting for a final line-feed), the image “#, #K” should be used.

Using EOI to Terminate a String Entry

The image “%K” allows the computer to terminate a free-field string entry with the EOI signal. However, a device which uses EOI as its end-of-line indicator may not output any other end-of-line characters, like line-feed. If this is the case, the proper image is “%,%K”. This allows the EOI signal to also terminate the ENTER statement. If you wish to enter line-feed characters into the string and also wish to terminate with EOI, the image “#%K” can be used. This may need to be expanded to “%,#%K” if no line-feed is expected to terminate the statement. The further expansion “#%,#%K” not only **allows** EOI to terminate the ENTER statement, but **requires** it as the only method of terminating the statement. Fixed-field entries can be checked for an expected EOI. For example, the image “%7A” inputs seven characters into a string and expects to have an EOI signal with the seventh character. Keep in mind that there are many valid combinations of these image specifiers. The combinations shown here are only some of the more common ones.

Using EOI to Terminate a Numeric Entry

The image “%K” allows the computer to terminate a free-field numeric entry with the EOI signal. As mentioned in the preceding paragraph, the image “%,%K” may be necessary if the EOI signal is to terminate the ENTER statement also. Fixed-field entries can be checked for an expected EOI. For example, the image “%7D” inputs seven characters to build a number and expects to have an EOI signal with the seventh character. Binary fields work in a similar manner. The image “%W” inputs two bytes to make a 16-bit integer and expects an EOI signal with the second byte.

There's Always an Exception

Not all terminator problems are a proper job for terminator images. Consider again the example of a name field (string) followed by an age field (numeric). Suppose that the names are variable in length and separated from the age by a simple comma. If the ages came first, this would not be a problem since the comma would end the entry to the numeric variable. But since the string data is entered first in this example, the task is a bit trickier. You might be able to use a CONVERT statement (explained at the end of this section) to change the comma into a line-feed and terminate the string that way. If the application does not permit the blanket conversion of commas to line-feeds, then the entire record would have to be input into a temporary string variable. Once the record is entered, the POS function and string subscripts could be used to extract the name and age fields. This hypothetical situation emphasizes the importance of knowing the nature of the data you are trying to enter. Some problems are handled by terminator images, and some are solved by different means, but all require thought by the programmer.

A Word of Advice About Images

Choosing the proper image for your application can often mean the difference between success and failure for your program. However, considering the wide range of peripheral devices and the near-infinite variety of possible data formats, it is understandably difficult to pick just the right image. Even experienced programmers will go through a period of trial-and-error before finding the perfect combination of image specifiers.

There is an old, but true, saying in the world of computers: "You can't program a computer to do something that you don't know how to do yourself". This is an appropriate sentiment for formatted I/O. If you don't know exactly what character sequence needs to be output or what an incoming sequence contains, it is very unlikely that you will know exactly what image specifiers to use.

Deciding on an exact character sequence for an output is simply a matter of definition. You know what data is generated by your program, so all you need to do is pick a desirable form for its output. The primary caution here is to avoid image overflow conditions.

But how can a programmer determine the exact nature of incoming data when he or she can't get it into the computer to study it? If the only tools available were the string and numeric image specifiers, this might be a significant problem. Fortunately, there is a way to inspect a totally unknown character sequence. Any sequence of bytes, including potential terminators, can be entered with the "#,B" image. The values that are printed or displayed are the decimal equivalents of the binary value of for each byte. Admittedly, this is not the most convenient form of data to work with. However, you can use an ASCII table or the CHR\$ function to determine the exact character sequence which is being received. Then, knowing the exact nature of the incoming data, the job of choosing image specifiers will be much simpler. The following example program shows a typical use of this technique.

```

100 ! Program to inspect incoming data
110 S1=3 ! Interface select code
120 ! Establish a terminating condition
130 SET TIMEOUT S1,3000
140 ON TIMEOUT S1 GOTO 230
150 !
160 I=1 ! Initialize counter
170 ! Input 1 byte; display analysis
180 ENTER S1 USING "#,B"; X
190 DISP "BYTE";I;TAB(11); "VALUE =";X;TAB(24); "CHAR = ";CHR$(X)
200 I=I+1 ! Count bytes
210 GOTO 180
220 !
230 DISP "DATA INPUT HAS STOPPED"
240 RESET S1! Stop I/O operation
250 END

```

Converting I/O Data

The final type of "formatting" involves changing the data characters that are entered or output. An example cited earlier was incoming numbers in European format (with periods separating digit groups and a comma for the radix point). There is no image available to accept this type of data directly. The periods and commas need to be changed to other characters to give the computer what it wants. The tool for performing this kind of operation is the CONVERT statement. Its general form is:

CONVERT <direction> <select code> <access method> ; <string>

The parameters are defined as follows:

<direction> Indicates whether the conversion is to take place during an ENTER (choose "IN") or an OUTPUT (choose "OUT").

<select code> Indicates which interface will use the conversion. Note that the CONVERT operation applies to all devices on a particular interface. A device selector is not allowed. The parameter must be an interface select code, range 3 thru 10.

<access method> This specifies the method of accessing the conversion table. The conversion table is a string variable, and there are two access methods. If "PAIRS" is specified, the string is treated as a list of character pairs. The second character of a pair is substituted for the first character whenever the incoming or outgoing character matches a first character. This method is a good choice when only a few characters need to be converted.

If "INDEX" is specified, the string is treated as a sequential look-up table. The numeric value of each incoming or outgoing character is used as an index into that table. The first element in the string corresponds to the character with a value of 0. If the value of the character to be converted is too large for the number of characters in the string, no conversion is performed. This method is a good choice when a large number of characters need to be converted.

<string> This represents the actual conversion table. It must be a string variable. A literal (string constant) is not allowed.

The use of the CONVERT statement should become more clear with a few examples. First, the European number format problem. This is a conversion for incoming data. One effective conversion is to replace a comma with a decimal point and replace a period with a space. The statements for doing this with an interface at select code 7 are:

```
A$="., "
CONVERT IN 7 PAIRS ; A$
```

The conversion has this effect:

Data before conversion:	12.345,6
Data after conversion:	12 345.6

Since the free-field format ignores spaces within a number and recognizes a decimal point, you do not even need an ENTER image to recognize the converted data. It is important to note that this CONVERT statement changes all periods to spaces and all commas to periods, whether they are part of a European number or simply part of a block of text. Since this could have some undesired effects, it is necessary to be able to "turn off" the conversion when it is no longer desired. The statement which cancels the conversion in this example is:

```
CONVERT IN 7
```

Giving only the direction and interface select code, without specifying "PAIRS" or "INDEX" or any other parameters cancels a previously selected conversion.

Control characters, such as carriage-return or line-feed, can also be converted. The following example shows the statements used to convert a carriage-return to a line-feed. This conversion is needed when entering data from a device which gives only a carriage-return, without a line-feed, as a delimiter.

```
A$=CHR$(13)&CHR$(10)
CONVERT IN 7 PAIRS ; A$
```

Another conversion example is the output of EBCDIC code instead of ASCII code. EBCDIC is another form of character representation used on certain types of computers. Since all the ASCII symbols have corresponding EBCDIC symbols, it is reasonable to choose the INDEX conversion mode using a string with 128 characters. In the following example, it is more important to understand the general process being used than to understand what the actual EBCDIC values are. The decimal equivalents of 128 EBCDIC characters are read from data statements and converted to string characters by the CHR\$ function. The resulting look-up table is included in a CONVERT statement for interface select code 7. The INDEX specifier tells the computer to use the outgoing ASCII character as an index to find the equivalent EBCDIC character. For example, an ASCII right brace (decimal value 125) will convert to a CHR\$(155), which is an EBCDIC right brace.

```
10 DIM A$[128]
20 A$=""
30 DATA 0,1,2,3,55,45,46,47,22,
      5,37,11,12,13,14,15,16,17,18
      ,19,53,61,50,38,24,25,63,36
40 DATA 28,29,30,31,64,90,127,1
      23,91,108,80,125,77,93,92,78
      ,107,96,75,97,240,241,242
50 DATA 243,244,245,246,247,248
      ,249,122,94,76,126,110,111,1
      24,193,194,195,196,197,198
60 DATA 199,200,201,209,210,211
      ,212,213,214,215,216,217,226
      ,227,228,229,230,231,232,233
70 DATA 173,21,189,95,109,20,12
      9,130,131,132,133,134,135,13
      6,137,145,146,147,148,149
80 DATA 150,151,152,153,162,163
      ,164,165,166,167,168,169,139
      ,79,155,74,7
90 FOR I=1 TO 128
100 READ X
110 A$=A$&CHR$(X)
120 NEXT I
130 CONVERT OUT 7 INDEX ; A$
```

Section 4

Error Handling

Run-time errors on the HP-85 can be trapped by using the ON ERROR statement. You may already be familiar with the ERRN and ERRL functions which provide essential error information in a program. These functions can be used with the I/O ROM. However, all HP-85 option ROMs share the error numbers starting at 101. So some other tools are necessary to identify the source of an error when more than one ROM is installed.

The I/O ROM provides two error functions in addition to the standard diagnostic capabilities of the HP-85. These functions give the programmer the extra information necessary to isolate error conditions in a program.

- | | |
|----------------|--|
| ERROM – | Provides a number which identifies the option ROM which generated the most recent error. If the most recent error was caused by the I/O ROM, the ERROM function returns a value of 192. Note that this function is only updated by option ROM errors. Therefore, ERROM “remembers” the last option ROM error, even if the most recent error in the system was not caused by an option ROM. |
| ERRSC – | Provides the select code of the interface which generated the most recent interface-dependent error. Note that this function is only updated by interface-dependent errors. Therefore, ERRSC “remembers” the last interface error, even if the most recent error in the system was not caused by an interface. |

Because other option ROMs share similar error numbers to those of the I/O ROM, and because these functions are not updated by every system error, it is important to interrogate the various error functions in the proper order. If you are looking for I/O errors in an error recovery routine, check first for $\text{ERRN} > 100$. If there is a ROM error, check ERROM to find which ROM. Having determined that the I/O ROM generated the error, check ERRN for an interface error before looking at ERRSC. Error numbers 101 and 112 will not occur during a running program. All other errors below 123 are interface-dependent errors. Therefore, a simple test for $\text{ERRN} < 123$ will tell if there was an interface error.

The following simple program segment shows the recommended order of function checks used to isolate I/O errors. This segment only displays an error message. An actual error recovery routine would also include statements to take whatever corrective action is appropriate in your specific situation.

```
10 ON ERROR GOSUB 100
20 !
30 !
100 ! Test for non-I/O errors
110 IF ERRN<101 THEN GOTO 350
120 IF ERROM<>192 THEN GOTO 350
130 !
140 ! Test for interface errors
150 IF ERRN<123 THEN GOTO 260
160 !
170 !
180 ! Process ROM errors
190 DISP "I/O ERROR";ERRN;"at li
ne";ERRL
200 !
210 ! Recovery routine goes here
220 !
230 RETURN
240 !
250 !
260 ! Process interface errors
270 DISP "I/O ERROR";ERRN;"at li
ne";ERRL
280 DISP "Problem on select code
";ERRSC
290 !
300 ! Recovery routine goes here
310 !
320 RETURN
330 !
340 !
350 ! Process non-I/O error here
360 RETURN
```

There is a complete listing of all I/O errors, their meaning, and some debugging hints in the back of the manual.

Part II
Bits and Bytes and Such



Section 5
Why Worry About Bits?

Section Introduction

Numbers are concepts to humans. They are conceptual points on a number-line continuum that can be associated with visual images, sounds, and tactile sensations. Most humans are trained to think in base 10. In contrast to this, numbers are electronic patterns of ones and zeros to a computer. The computer performs many of its operations in base 2. Most significant to I/O applications is the fact that these electronic patterns of ones and zeros can be used directly as control signals in other electronic devices.

Some I/O operations treat groups of bits as characters or values, a concept which most programmers learn early. The ASCII character set is an example of this. On the other hand, I/O operations often involve handling bits on an individual basis. This delving into base 2 occurs often enough that a programmer entering the area of I/O and process control is well advised to become familiar with bit patterns and binary operations.

This section of the manual gives a review of the basic concepts of base 2, alternate base representation, and logical operators. The following two sections present the binary and alternate base functions available from the I/O ROM.

Review of Base 2

Before looking at base 2, it is helpful to take a careful look at the familiar base 10. The number one hundred twenty five is represented as follows:

125

The digits have a **place value** corresponding to powers of ten. The representation above really means:

$$1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

The concept of place value also exists in base 2. The difference being that powers of two are represented instead of powers of ten. The number one hundred twenty five is represented as:

1111101

Base 2 uses only the digits “1” and “0”; a 1 indicates that a place value is included, while a 0 indicates that a place value is not used in the value. Therefore, the binary representation shown above means:

$$2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0$$

This is the same as:

$$64 + 32 + 16 + 8 + 4 + 1$$

The term **bit** comes from the words “binary digit”. A bit is a single digit in base 2 that must be either a 1 or a 0. The grouping of 8 bits together is in such common usage for character representation, internal storage, and interfacing that it has been given a special name — a **byte**. The term byte refers to 8 bits processed as a unit.

Notice in both the previous examples that the right-most digit represents the “0th” power of the base. Because of this, bit patterns are usually numbered starting at Bit 0, instead of Bit 1. By doing this, the bit number and the power of two it represents are the same. The following table shows the bit positions in a byte and their corresponding values.

Bit Position:	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Meaning:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value:	128	64	32	16	8	4	2	1

- Examples: 130 in base 10 is 10000010 in base 2
 3 in base 10 is 00000011 in base 2
 25 in base 10 is 00011001 in base 2

The term **word** is also widely used in computer applications. A “word” is usually the number of bits that can be handled in one operation by the internal architecture of the computer. Although the HP-85 has an 8-bit internal architecture, it also has operations defined for 16-bit and floating point numbers. Therefore, 16-bit integers are often referred to as “words” in the HP-85 because the system can handle them as a basic data unit. Another characteristic of a word in the HP-85 is that 2’s complement representation is used. 2’s complement representation is a method of storing either positive or negative numbers in a word. It works like this:

Positive numbers: If bit 15 is 0, then the word is a positive number represented in normal binary form.

Negative numbers: If bit 15 is 1, then the word is a negative number represented in 2’s complement form.
 To find the absolute value of a negative number, invert all the bits and add 1.

Problem: What is the value of 11111111 11110000 ?

Solution: Bit 15 tells that this is a negative number.

Inverting all the bits gives: 00000000 00001111

Adding 1 results in: 00000000 00010000

So the value of the given bit pattern is -16.

Review of Alternate Representations

When text contains values represented in more than one base, it is extremely important to distinguish between the concepts of **value** and **representation**. Consider the number one hundred. The **value** is the number of beans in a jar of one hundred beans. The **representation** in base 10 is the digit “1” followed by two zeros. The value one hundred can also be represented as “64” in base 16, as “01100100” in base 2, and as “10” in base 100.

The representation of a number is merely the characters used to communicate the number's value. Numbers are often represented in bases other than 10 when the use of an alternate base more clearly communicates the number's value. For example, suppose that up to 16 small pumps and valves are controlled by a single 16-bit word from the HP-85. If the control pattern were represented in base 10, it could be very difficult to understand the number in terms of pumps and valves. However, suppose the number is represented in base 2, further defined so that the most-significant byte is pumps and the least-significant byte is valves. The base 2 representation “00100000 10000000” clearly shows one pump on and one valve open. That same value is “8320” in base 10. How quickly does “8320” communicate to you that one pump is on and one valve is open?

The problem with using base 10 to represent a binary number is that one base 10 digit does not represent an integral number of bits. A base 10 pattern does not readily reflect which bits are “1” and which are “0”. The problem with using base 2 is that there are simply too many characters to read and write. To circumvent these problems, persons who work at the bit and byte level in computers commonly use base 8 or base 16 to represent binary numbers. These bases have place values directly related to powers of two, making it easy to trace bits with a little practice. They also provide representations that are three and four times more compact than binary, reducing the number of characters needed to a more manageable small group. For example, an entire byte is never more than 2 characters in base 16.

Base 8, known as “octal”, uses one octal digit for three binary digits. Base 16, known as “hex” (short for hexadecimal), uses one hex digit for four binary digits. The following tables show the decimal (base 10), binary (base 2), octal (base 8), and hex (base 16) representations for the numbers 0 thru 16.

Decimal	Binary	Octal	Decimal	Binary	Hex
0	000	0	0	0000	0
1	001	1	1	0001	1
2	010	2	2	0010	2
3	011	3	3	0011	3
4	100	4	4	0100	4
5	101	5	5	0101	5
6	110	6	6	0110	6
7	111	7	7	0111	7
8	1 000	10	8	1000	8
9	1 001	11	9	1001	9
10	1 010	12	10	1010	A
11	1 011	13	11	1011	B
12	1 100	14	12	1100	C
13	1 101	15	13	1101	D
14	1 110	16	14	1110	E
15	1 111	17	15	1111	F
16	10 000	20	16	1 0000	10

Notice that the Arabic numeral system (designed for base 10) does not have any single-character symbols to represent quantities above nine. Single-character representation of all quantities less than the base value is essential to the concept of place value. Therefore, base 16 representation “borrows” the characters A thru F to represent values from 10 thru 16. The following examples help to illustrate the way that octal and hex numbers use bit groupings to represent binary values.

Octal:	1 2 5	3 6 0	0 7 0	2 0 1	2 0 2
Binary:	0 1 0 1 0 1 0 1	1 1 1 1 0 0 0 0	0 0 1 1 1 0 0 0	1 0 0 0 0 0 0 1	1 0 0 0 0 0 1 0
Hex:	5 5	F 0	3 8	8 1	8 2

Review of Logical Operations

In this discussion, “logical operations” refers to operations from Boolean algebra, such as AND and OR. The outstanding I/O feature of these operations is that they can modify individual bits without affecting surrounding bits. In this respect, they can be contrasted to the arithmetic operations, such as addition and subtraction. Addition and subtraction generate carries and borrows that can propagate through an entire word, changing the state of numerous bits many places away from the bit where the arithmetic was performed. Although this is exactly what is desired for numerical quantities, many of the bytes and words used in I/O are not numerical quantities. Consider the previous example about 16 pumps and valves. The carry bits generated by an addition to such a word could literally destroy the hydraulic system being controlled. When bits are used as individual control elements, the programmer must have access to tools that allow individual control of bits. The specific tools available with the HP-85 are presented in Section 6. This section merely reviews the action of the common logical operators.

The operators AND, OR, NOT, and EXOR are available in the standard language of the HP-85. These operators treat an entire variable as one entity. A value of “0” is considered “false”, while any other value is considered “true”. Although these operators perform the same Boolean function as the binary logical operators, they do not operate on individual bits. The binary logical operators available with the I/O ROM work on a bit-by-bit basis across an entire word. Without these binary tools, isolating an individual bit requires an involved combination of tests, branches, and arithmetic operators.

The simplest logical operation is the **complement** operation. When binary data is complemented, all the 1's are changed to 0's, and all the 0's are changed to 1's. This operation is also known as “1's complement”, or “inversion”. The Boolean notation for this operation is a horizontal bar drawn over the variable. The truth table is as follows:

A	\overline{A}
0	1
1	0

When used on an entire byte, the complement operator inverts each bit individually.

Binary value of A: 10011101

Binary complement of A: 01100010

The other logical operators combine two inputs to create a result. Let's look at the AND operator first. A binary AND produces a "1" in the result only if both inputs are "1". The Boolean notation for this operation is \wedge , although you may also see the symbol \bullet used. The truth table for AND is as follows:

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

The important thing to notice is that the result is 0 when A is 0, while the result is equal to B when A is 1. Because of this, a binary AND is a convenient method for clearing selected bits. For example, assume that you wanted to clear the two lowest bits in a byte without disturbing the other bits. This can be done by ANDing the byte with an appropriate mask.

Original byte: 10011101

Byte ANDed: 11111100

Result: 10011100

This operation not only preserves the state of the top six bits, but also clears the bottom two bits no matter what their original state. That saves a lot of testing and branching.

The next operator is the binary OR, most correctly called the "inclusive OR". In English this means: you can have pie OR ice cream for dessert, and it is possible to have both at the same time. To the computer this means that the result bit is "1" when either input bit is "1". The Boolean notation for an inclusive OR is \vee , although you may also see the symbol $+$ used. The inclusive OR truth table is:

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

The important thing to notice is that the result is 1 when A is 1, while the result equals B when A is 0. Because of this, the inclusive OR is a convenient method for setting selected bits. For example, assume you wanted to set the two lowest bits in a byte without disturbing the other bits. This can be done by ORing the byte with an appropriate mask.

Original byte: 10011101

Byte ORed: 00000011

Result: 10011111

This operation sets the lower two bits no matter what their original state and also preserves the state of the top six bits.

The final operator is the binary EXOR, or "exclusive OR". In English this means: you can take the plane OR the train to Chicago, but you can't do both at the same time. To the computer this means that the result bit is "1" if a single input bit is "1", but the result bit is "0" if both input bits are the same. The Boolean notation for an exclusive OR is \neq , although you may see the symbol \oplus used. The exclusive OR truth table is:

A	B	$A \neq B$
0	0	0
0	1	1
1	0	1
1	1	0

The important thing to notice is that the result is equal to B when A is 0, while the result is the complement of B when A is 1. Because of this, the exclusive OR is a convenient method for inverting selected bits. For example, assume that you wanted to invert the lower two bits of a byte without disturbing the rest of the bits.

Original byte: 10011110

Byte EXORed: 00000011

Result: 10011101

This operation complements the lower two bits no matter what their original value and leaves the top six bits unchanged.

Section 6

Binary Functions

Section Introduction

As explained in the previous section, I/O programming often involves the use of binary functions and bit-level operations. The I/O ROM provides several useful tools to assist you with these tasks. These binary functions are often used when manipulating status and control registers in the interface cards. They may also be very handy for processing I/O data in device control applications. This section explains the binary functions available on the HP-85.

It is important to remember that all the binary functions provided by the I/O ROM operate on 16-bit words. For example, the binary complement of zero is 11111111 11111111 (base 2); the range of bits that can be tested is 0 thru 15; and the range of values for binary arguments is -32 768 thru 32 767. There is no problem using these functions to operate on binary values with less than 16 bits. The unused high-order bits are simply assumed to be zero.

In the following explanations, the term "integer" is used frequently to identify the arguments for many of the functions. To the I/O ROM, an integer is a 16-bit binary number with a range of -32 768 thru 32 767. This contrasts to the definition of an integer given in the Owner's Manual, where it states that an integer is a 5-digit number with a range of -99 999 thru 99 999. Please keep this distinction in mind to avoid confusion about the term "integer".

If you are not familiar with the binary operators, such as AND and EXOR, study the review in Section 5 before continuing with this section.

The Binary AND Function

The binary AND function performs a bit-by-bit AND using two integers as arguments and producing an integer result. Here are some examples of properly syntaxed binary AND functions:

```
X=BINAND(Y,15)
A=C+BINAND(D,E)
PRINT BINAND(255,Z-32)
```

Notice that the arguments must be enclosed in parentheses and separated by a comma. The arguments may be numeric constants, numeric variables, numeric expressions, or any combination. The arguments are assumed to be in base 10 representation. If you wish to express the arguments in an other base, refer to Section 7. Each bit of the result is computed according to the following truth table:

First Argument	Second Argument	Function Result
0	0	0
0	1	0
1	0	0
1	1	1

The Binary Inclusive OR Function

The binary inclusive OR function performs a bit-by-bit inclusive OR using two integers as arguments and producing an integer result. Here are some examples of properly syntaxed binary inclusive OR functions:

```
X=BINIOR(Y,15)
A=C+BINIOR(D,E)
PRINT BINIOR(255,Z-32)
```

Notice that the arguments must be enclosed in parentheses and separated by a comma. The arguments may be numeric constants, numeric variables, numeric expressions, or any combination. The arguments are assumed to be in base 10 representation. If you wish to express the arguments in an other base, refer to Section 7. Each bit of the result is computed according to the following truth table:

First Argument	Second Argument	Function Result
0	0	0
0	1	1
1	0	1
1	1	1

The Binary Exclusive OR Function

The binary exclusive OR function performs a bit-by-bit exclusive OR using two integers as arguments and producing an integer result. Here are some examples of properly syntaxed binary exclusive OR functions:

```
X=BINEOR(Y,15)
A=C+BINEOR(D,E)
PRINT BINEOR(255,Z-32)
```

Notice that the arguments must be enclosed in parentheses and separated by a comma. The arguments may be numeric constants, numeric variables, numeric expressions, or any combination. The arguments are assumed to be in base 10 representation. If you wish to express the arguments in an other base, refer to Section 7. Each bit of the result is computed according to the following truth table:

First Argument	Second Argument	Function Result
0	0	0
0	1	1
1	0	1
1	1	0

The Binary Complement Function

The binary complement function performs a bit-by-bit complement of an integer argument, producing an integer result. Here are some examples of properly syntaxed binary complement functions:

```
A=BINCMP(B)
X=BINCMP(Y-Z)
PRINT BINCMP(N*8)
```

Notice that the argument is enclosed in parentheses. The argument may be a numeric constant, a numeric variable, a numeric expression, or any combination. The argument is assumed to be in base 10 representation. If you wish to express the argument in an other base, refer to Section 7. Each bit of the result is computed according to the following truth table:

Argument	Result
0	1
1	0

You should keep in mind that the binary complement function operates on a full 16-bit word. This may, in some cases, give an unexpected result if you are dealing exclusively with 8-bit bytes. The 16-bit complement of an 8-bit byte is always a negative number. You can generate 8-bit complements by using the binary exclusive OR function. An exclusive OR with 255 complements the lower 8 bits and leaves the upper 8 bits as zeros. This technique prevents the unintentional generation of negative values when dealing with single bytes.

The Bit Test Function

The bit test function is used to indicate whether a specific bit in an integer is set (1) or clear (0). The general form for the bit test instruction is:

BIT(integer,bit number)

The integer argument must be the first expression and the two expressions must be separated by a comma. The bit number must be in the range 0 thru 15, where 0 is the least-significant bit and 15 is the most-significant bit. Either argument may be a numeric constant, a numeric variable, a numeric expression, or any combination. Here are some examples of properly syntaxed bit test functions:

```
A=BIT(B,3)  
X=BIT(Y,Z-1)  
IF BIT(N,1) THEN GOSUB 220
```

The bit test function is very useful in decision making and branching. It is easily used with the IF statement to direct program flow based on the state of individual bits. The function returns a 0 (false) if the specified bit is 0 and returns a 1 (true) if the specified bit is 1.

Section 7

Base Conversion Functions

Section Introduction

A programmer who works at the bit and byte level soon develops a preference for the base in which bytes and words are represented. In some cases, base 2 offers the clearest display of a bit pattern. Base 8 built a large following in the days when computers could not easily handle alphabetic characters as numeric input. Base 16 has gained much popularity in recent years because most computers use a word length that is an integral multiple of 4, and modern systems have no trouble converting the symbols A thru F used in hex.

To accomodate these various preferences, your HP-85 provides conversion functions that allow the input and output of integers using any of the alternate representations mentioned above. The base conversion functions have certain characteristics in common:

- All conversions go from base 10 to an alternate base or from an alternate base to base 10. You can't convert directly from one alternate base to another without passing through base 10.
- The base 10 side of the conversion is always a numeric quantity, while the alternate base side is always a string.
- Because the alternate base representations are string data, they can be input, output, compared, stored, and manipulated to some degree. However, the string representations cannot be used in arithmetic operations.
- All arguments for the base conversion functions must be in the range of 16-bit integers. This includes the alternate representations as well as the base 10 values.

Conversions From Base 10 to an Alternate Base

These functions use a base 10 numeric quantity as an argument and produce a string as a result. The primary use of these functions is the printing, display, or output of data in an alternate base, although other applications are possible. The argument for the function may be a numeric constant, numeric variable, numeric expression, or any combination. The argument must be in the range of -32 768 thru 32 767. Functions are available to convert to base 2, base 8, or base 16.

From Base 10 to Base 2

This is the "Decimal to Binary String" function. It converts an integer argument to a string of 16 ones and zeros. The string is the base 2 representation of the integer argument. If the argument is out of range and positive, the function yields "0111111111111111". If the argument is out of range and negative, the function yields "1000000000000000". The following are examples of properly syntaxed expressions:

```
PRINT DTB$(X)
A$=DTB$(N*2)
OUTPUT 701;DTB$(32+Y)
DISP DTB$(255)
```

From Base 10 to Base 8

This is the "Decimal to Octal String" function. It converts an integer argument to a 6-character string. The string is the octal representation of the integer argument. If the argument is out of range and positive, the function yields "077777". If the argument is out of range and negative, the function yields "100000". The following are examples of properly syntaxed expressions:

```
PRINT DTO$(X)
A$=DTO$(N*2)
OUTPUT 701;DTO$(32-Y)
DISP DTO$(255)
```

From Base 10 to Base 16

This is the "Decimal to Hex String" function. It converts an integer argument to a 4-character string. The string is the hex (hexidecimal) representation of the integer argument. If the argument is out of range and positive, the function yields "7FFF". If the function is out of range and negative, the function yields "8000". The following are examples of properly syntaxed expressions:

```
PRINT DTH$(X)
A$=DTH$(N*2)
OUTPUT 701;DTH$(32-Y)
DISP DTH$(255)
```

Conversions From an Alternate Base to Base 10

These functions use a string as an argument and produce a numeric result. The primary use of these functions is the input of data in an alternate base, although other applications are possible. The argument for the function may be a string constant (literal), string variable, string expression, or any combination. The argument must represent a value in the range of 16-bit integers. Functions are available to convert from base 2, base 8, or base 16.

From Base 2 to Base 10

This is the "Binary to Decimal" function. The argument is a string which is the binary representation of an integer. The argument cannot have more than 16 characters, and only the numerals "1" and "0" are valid. The result of the function is the base 10 value of the number represented by the argument. Since the function result is numeric, it can be used in arithmetic operations or numeric functions. The following are examples of properly syntaxed expressions:

```
PRINT BTD("100101")
X=BTD(A$)
Y=255-BTD(N$)
```

From Base 8 to Base 10

This is the "Octal to Decimal" function. The argument is a string which is the octal representation of an integer. The argument cannot have more than 6 characters. Only the numerals "0" thru "7" are valid. If all six characters are used, the most-significant character can only be a "1" or a "0". The result of the function is the base 10 value of the number represented by the argument. Since the function result is numeric, it can be used in arithmetic operations or numeric functions. The following are examples of properly syntaxed expressions:

```
PRINT OTD("371")
X=OTD(A$)
Y=255-OTD(N$)
```

From Base 16 to Base 10

This is the "Hex to Decimal" function. The argument is a string which is the hex representation of an integer. The argument cannot have more than 4 characters. Only the numerals "0" thru "9" and the letters "A" thru "F" are valid. The result of the function is the base 10 value of the number represented by the argument. Since the function result is numeric, it can be used in arithmetic operations or numeric functions. The following are examples of properly syntaxed expressions:

```
PRINT HTD("1F4")
X=HTD(A$)
Y=255-HTD(N$)
```

Converting From One Alternate Base to Another

Conversions between alternate representations are easily done by nesting two conversion functions. The following short program is an example of this technique. It inputs a hex representation and displays the corresponding binary representation.

```
10 DISP "Hex Number";
20 INPUT A$;
30 DISP "Binary = "; DTB$(HTD(A$));
40 GOTO 10
```

Part III

Advanced I/O Operations

Introduction

This part of the manual deals with I/O operations that are typically used by experienced programmers for solving complex I/O problems. You may never use some of the operations discussed here at all. All of the operations give you capabilities that extend above and beyond those required to solve typical I/O problems.

The topics covered here include the "what" and "how" of: buffered I/O, end-of-line branching, keyboard masking, and interface control and status. If these terms are new to you, then take time to read the introduction to each section. If you are familiar with the terms just used, then skip the introduction and read the programming information of interest to you.

Specialized Transfers

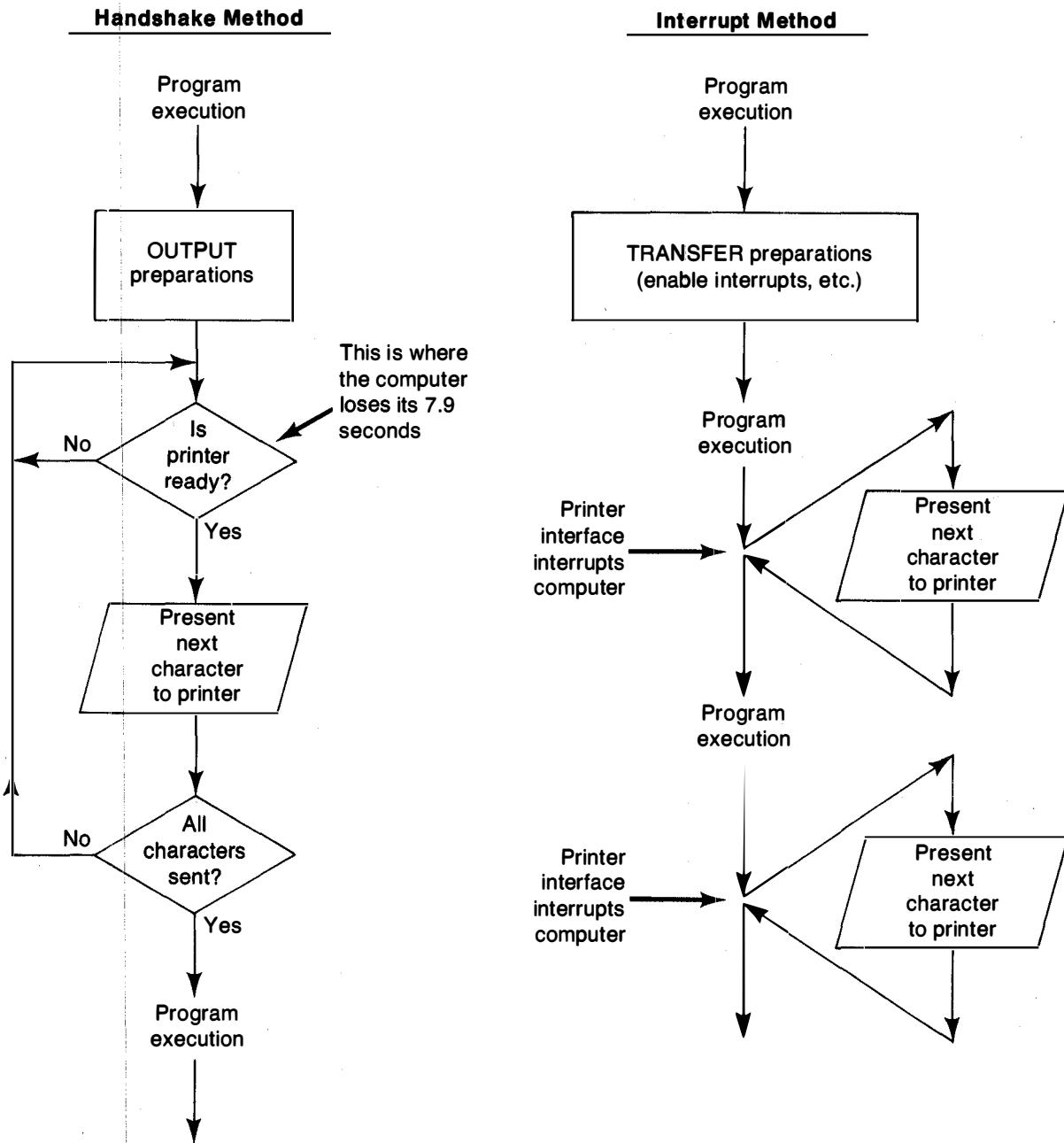
Section Introduction

This section deals with data transfers as they are implemented by the TRANSFER statement. The basic purpose of the TRANSFER statement is to provide a flexible tool for moving data into and out of the computer. The key word here is flexibility.

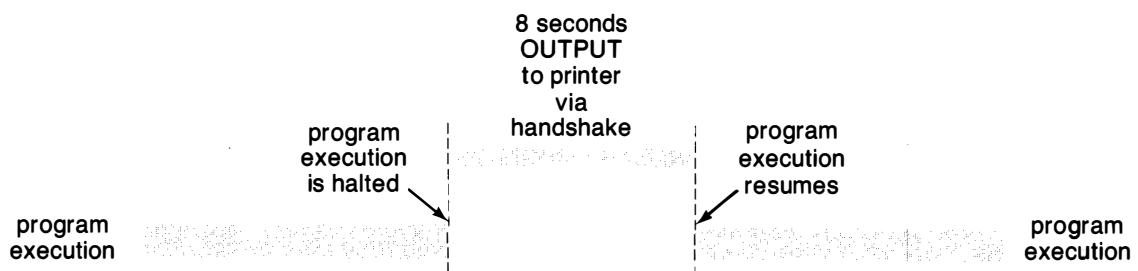
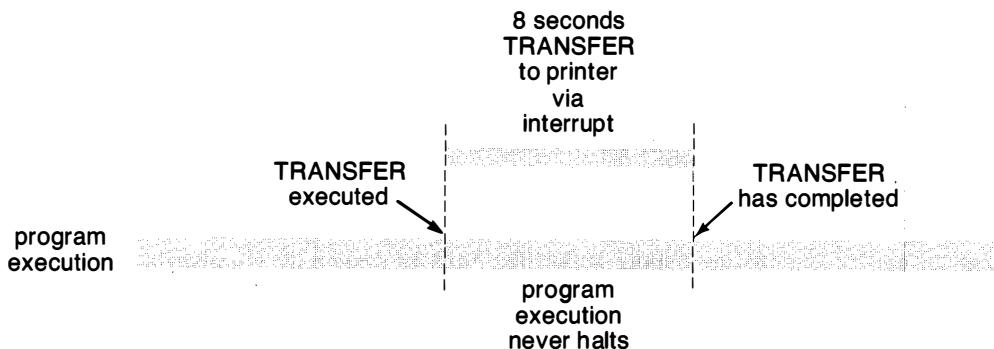
This flexibility allows you to better match the computer's speed to that of the peripheral it is communicating with. Take the case of a very slow device, such as a 10 character-per-second printer. It takes such a printer 8 seconds to print an 80 character line, but our computer could send those same 80 characters in less than .1 second. If the computer is forced to wait on the printer, then the computer is losing 7.9 seconds of computation time out of every 8 seconds! The computer's power can obviously be increased by gaining back that 7.9 seconds. Let's see how.

The following diagrams contrast the default "handshake" method used by OUTPUT and ENTER with the "interrupt" method of TRANSFER. When the computer executes the OUTPUT statement (for example), it is forced to handshake **each** character of the data list until all the data has been sent. Only then is the computer free to execute the next program statement, about 8 seconds later. On the other hand, the interrupt TRANSFER statement sets up some special pointers to the data and enables the printer interface to interrupt the computer. Then the computer is free to execute the next program line, about 10 **milliseconds** later! (Enabling an interrupt is like hanging up a telephone receiver : the telephone is now able to "interrupt" you by ringing whenever someone calls.)

The computer continues program execution until the printer is ready for another character. The printer interface interrupts the computer from whatever it was doing, and the computer then fetches the next character, updates its pointers, checks to see if all data has been sent, then continues on with what it was doing. If all the data has now been sent, the computer disables anymore interrupts from the interface (like taking the telephone receiver off the hook — no more rings) before continuing on with the program.



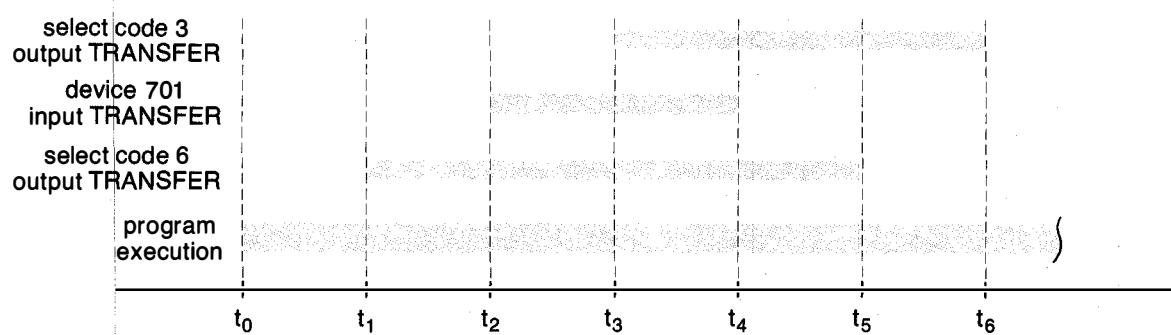
If the two methods are looked at in a broader sense, it is possible to see the real difference : the handshake method is a linear, or **sequential** operation, while the interrupt method is a parallel, or **overlap** operation. Consider the following diagram:

Sequential**Overlap**

The sequential method effectively PAUSES the program for the duration of the OUTPUT operation, while the overlap method continues both with program execution and TRANSFER operation.

An interesting possibility brought about by this overlap is that of multiple, simultaneous I/O operations. Suppose that the next program statement after the TRANSFER statement is another TRANSFER to a different device (a large-screen CRT monitor, for instance)? Then three things are happening at once : the program is being executed, the printer is printing, and the external CRT monitor is displaying new characters.

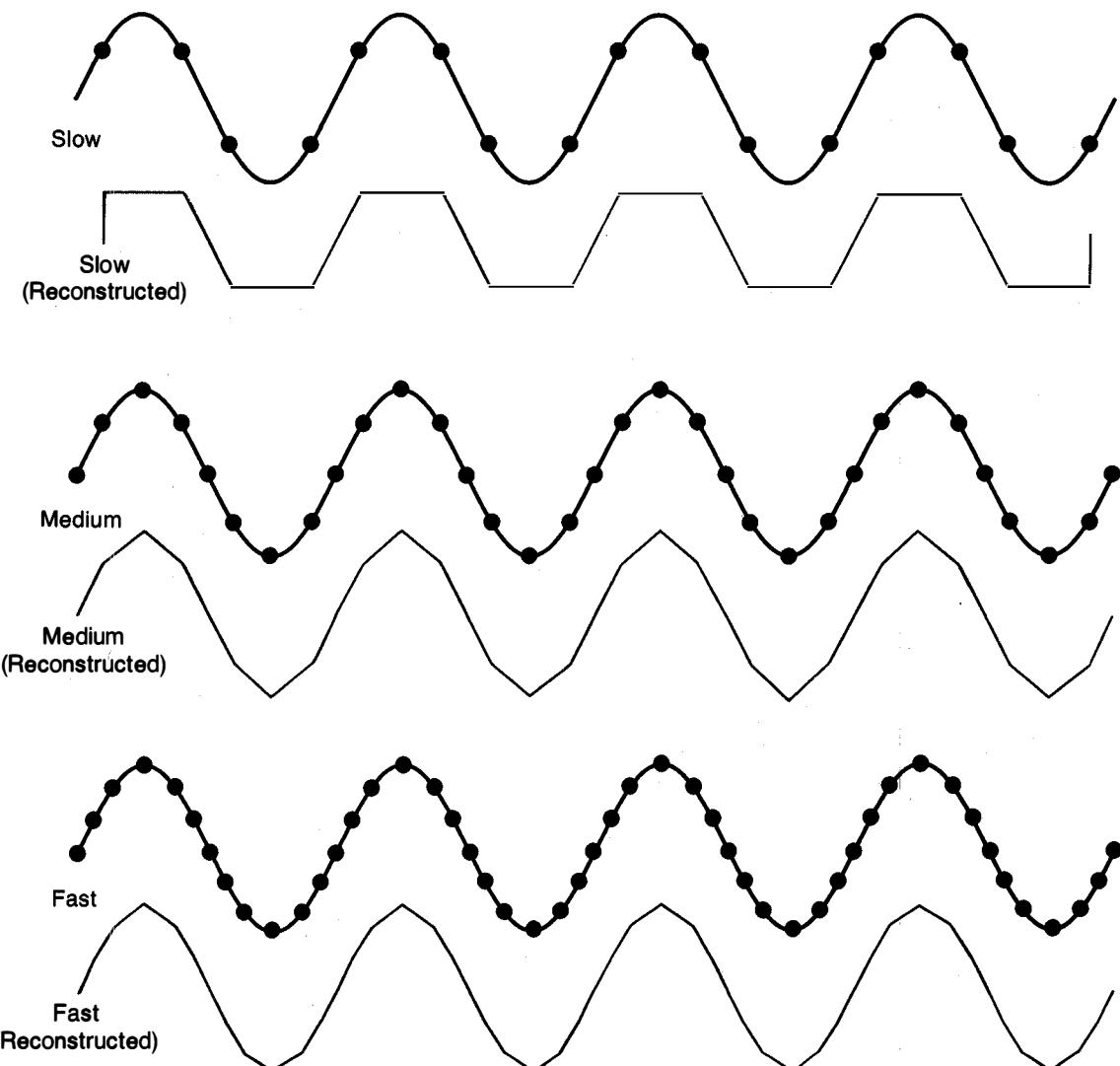
The following diagram illustrates some of the power of overlapped I/O operations made possible with TRANSFER. (Please note that this is for "interrupt" TRANSFER only, as explained later.)



Program execution begins at time t_0 , and some time later an output TRANSFER to select code 6 is initiated (time t_1). At time t_2 an input TRANSFER from HP-IB device 701 is started, and at time t_3 another output TRANSFER is started, this time to select code 3. By now, three TRANSFERS and program execution are all going on at once. At time t_4 , the input TRANSFER terminates, and the two output TRANSFERS finish at t_5 and t_6 . This overlap capability demonstrates some of the flexibility of system design made possible with the TRANSFER statement.

There is another side to the TRANSFER statements flexibility: speed. Certain operations are ineffective or impossible at slow or medium speeds, and require instead a high-speed transfer. Take the case where it is desired to analyze a signal's waveform by using an HP3437A voltmeter. This voltmeter is capable of producing a $3\frac{1}{2}$ digit voltage reading up to 3600 times per second (which is a transfer rate of about 25 000 characters per second! The HP-85 is not quite that fast however; its transfer rate is closer to 20 000 characters per second).

The following diagram illustrates the effect of sampling a signal at slow, medium, and high rates of speed (sample points are represented by dots):



The slow sample rate provides at best an inaccurate picture of the signal, while the high sample rate comes much closer to approximating the actual shape of the signal.

So where does TRANSFER fit into this picture? Consider the ENTER and OUTPUT statements with their extensive formatting and conversion capabilities as being like a Rolls-Royce automobile with electric windows, television, liquor cabinet, automatic transmission, and other accessories.

A fast-handshake TRANSFER is then comparable to a Formula I race car, with no windows, manual transmission, one seat, a spine-jolting ride, and that is capable of speeds over 300 kilometers per hour. Its main objective is performance.

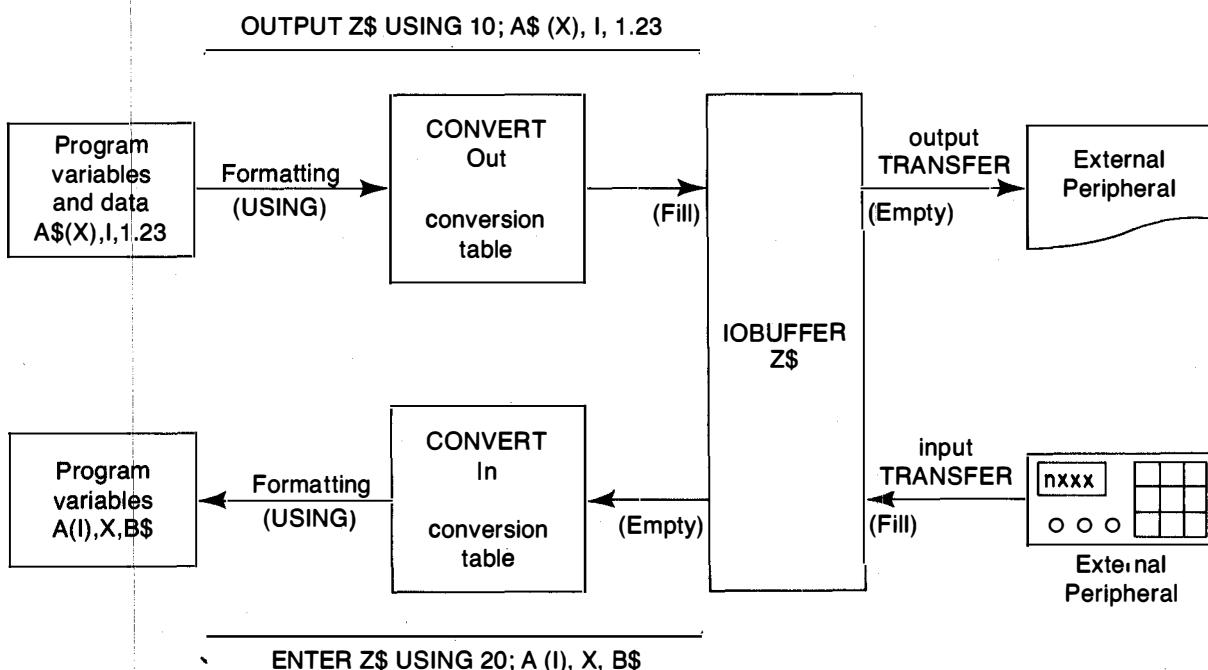
The fast-handshake TRANSFER delivers the highest data transfer rate possible with the HP-85A computer. When a fast-handshake TRANSFER operation is in progress, **ALL** other activities stop. Even the computer's RESET key is disabled and will have no effect until the TRANSFER completes.

As with the Formula I race car, you pay a price for performance.

The Care and Feeding of Buffers

A buffer is a section of read/write memory set aside for the purpose of temporary data storage. It is used to either input data, output data, or both by means of the TRANSFER statement. There is no formatting or data conversion done by TRANSFER, so what is in the buffer is what is sent to the peripheral device. The same holds true for data being input by TRANSFER; it is placed in the buffer exactly as it is received from the device.

To illustrate how OUTPUT, ENTER, conversion, formatting, IOBUFFER, and TRANSFER work together consider the following diagram.



The OUTPUT statement takes data from program variables and does any necessary formatting while placing that data into its ASCII representation. Then, if an output conversion is in effect, the ASCII characters are converted accordingly and placed into the IOBUFFER at the position specified by the **fill pointer**. The IOBUFFER is **full** when the fill pointer is at the end of the buffer (the string Z\$ in this case). You should be aware that a default OUTPUT operation to a buffer places a carriage-return and a line-feed at the end of the data in the buffer.

The output TRANSFER takes characters from the IOBUFFER at the position specified by the empty pointer. These characters are sent to the specified interface and its associated peripheral. This is done either by interrupt or fast-handshake as specified by the programmer. When the TRANSFER completes, the interface's end-of-line character sequence is sent.

The input TRANSFER accepts characters from the specified interface (and its associated peripheral) and places them into the IOBUFFER at the position specified by the fill pointer. Again, this is done either by interrupt or fast-handshake as specified by the programmer.

The ENTER statement takes characters from the IOBUFFER at the position specified by the empty pointer. If an input conversion is in effect, these characters are converted accordingly, formatted as necessary, and changed into the proper internal representation for the program variables. If you are entering data from an active buffer, errors can be avoided by using the form ENTER Z\$ USING “#, #K”;A\$.

This form of ENTER removes the requirement for statement and variable terminators, which may not be in the buffer yet.

The Pointers

When a string variable is first designated as an IOBUFFER (by executing an IOBUFFER statement), its dimensioned length is effectively reduced by 8 characters. This is to provide room for 4 "pointers." There is a fill pointer, an empty pointer, an active-out select code, and an active-in select code.

The **fill pointer** first equals zero (0). This pointer always contains the same value as that returned by the LEN function for the string variable. Placing a character into the buffer goes as follows: 1) increment the fill pointer, 2) store the character. This operation is **automatically** handled by the OUTPUT (fill pointer equals LEN function) statement and also by any string variable assignment operations such as Z\$=Z\$&A\$. You do not normally need to assign values to the fill pointer.

The **empty pointer** first equals one (1). Taking a character from the buffer goes as follows: 1) read the character, 2) increment the empty pointer. This operation is performed automatically by the ENTER statement.

A buffer is **full** when the fill pointer equals the string's dimensioned length minus 8. A string with a dimensioned length of 8 would not be a very useful buffer, as it would be full and empty at the same time, without any data being placed in it at all! Any OUTPUT operation to a full buffer will result in an error.

A buffer is **empty** when the empty pointer equals the fill pointer plus one. This has no relationship to the dimensioned length of the string. When a buffer is emptied, the fill pointer is reset back to 0 and the empty pointer is reset back to 1. This is exactly the same effect as executing the IOBUFFER statement, except that the IOBUFFER statement also initializes (destroys) conversion table pointers.

Buffer Activity

When a TRANSFER statement is executed, the specified buffer is then an **active** buffer. The buffer may be **active-out**, **active-in**, or both, depending upon the direction(s) of the transfer(s). The buffer is assigned an **active-in select code** when an input TRANSFER statement is executed. An **active-out select code** is assigned when an output TRANSFER statement is executed. For example when the following interrupt transfer to select code 6 is executed —

TRANSFER Z\$ TO 6 INTR

the active-out select code equals 6.

A buffer is made **inactive** when the TRANSFER completes. This is a direction-specific inactive state; that is, a buffer may be active-out but inactive-in, or vice-versa. When an input TRANSFER completes, the buffer's active-in select code is set to zero (0). Similarly, when an output TRANSFER completes, the buffer's active-out select code is set to zero (0).

Buffer Status and Control

The four buffer pointers can be checked by means of the STATUS statement. The four I/O buffer status registers are as follows:

Status Register	Default Value	Register Function	Statement used to read register value of buffer Z\$.
SR0	1	Buffer empty pointer	STATUS Z\$,0;T0
SR1	0	Buffer fill pointer	STATUS Z\$,1;T1
SR2	0	Active-in select code	STATUS Z\$,2;T2
SR3	0	Active-out select code	STATUS Z\$,3;T3

These registers may be read at any time on an active or inactive buffer, but attempting to read the status of a non-buffer string variable (that is, if no IOBUFFER statement has been executed for that string variable) results in an ERROR.

An example of using buffer status registers to control program flow follows. In the example, a string variable is dimensioned to 88 characters (to allow for 80 characters of data after becoming a buffer) and declared as an I/O buffer. Data is OUTPUT to the buffer, a TRANSFER out to an HP-IB printer is initiated, then the buffer's active-out select code is checked to determine when the TRANSFER has completed. The program ends when the transfer completes.

```

10 DIM Z$[88]
20 IOBUFFER Z$ ! Usable size of Z$ is 80
30 FOR I=0 TO 1 STEP .1
40 OUTPUT Z$ USING "#,0.000,X" ; SIN(I)
50 NEXT I
60 ! Show values of buffer registers
70 STATUS Z$,0 ; T0,T1,T2,T3
80 PRINT "Registers before TRANSFER",T0,T1,T2,T3
90 TRANSFER Z$ TO 701 INTR
100 STATUS Z$,0 ; T0,T1,T2,T3
110 DISP T0,T1,T2,T3
120 IF T3<>0 THEN GOTO 100
130 PRINT "Transfer completed"
140 END

```

The variable T0 shows how many characters have been taken from the buffer. T1 shows how many characters total are in the buffer, and T3 indicates select code activity.

For another example, assume device X is to send three numeric values followed by a carriage-return, line-feed. The following program displays the buffer registers until the TRANSFER completes. An ENTER is then executed to take the values out of the buffer, which are then printed.

```

10 ! Our device X is at HP-IB address 721
20 DIM Z$[88]
30 IOBUFFER Z$
40 TRANSFER 721 TO Z$ INTR ; DELIM 10
50 ! Note that 10 is the decimal value for line-feed
60 STATUS Z$,0 ; T0,T1,T2,T3
70 DISP T0,T1,T2,T3
80 IF T3<>0 THEN GOTO 60
90 ENTER Z$ ; X,Y,Z
100 DISP X,Y,Z
110 END

```

If device X is a very slow device, you could "watch" characters come into the buffer by means of variable T1. T0 is not altered until the ENTER is executed, and T2 reflects the TRANSFER statement activity. When the TRANSFER is initiated, T2 = 7; when the TRANSFER completes, T2 = 0.

The buffer empty pointer and the buffer fill pointer can be assigned new values by using the CONTROL statement. This gives you the capability of sending the same data over and over again without having to re-compute the data, for example. The following table shows these registers and how they are accessed:

Control Register	Default Value	Register Function	Statement used to write register value of buffer Z\$
CR0	1	Buffer empty pointer	CONTROL Z\$,0;V0
CR1	0	Buffer fill pointer	CONTROL Z\$,1;V1

These buffer registers may be written to at any time, but attempting to write to a control register of a non-buffer string variable (that is, if no IOBUFFER statement has been executed for the string variable) results in an ERROR.

In the following example, 360 values are computed for SIN(X) to represent one complete cycle of a sine wave. These values are OUTPUT to the buffer Z\$ and subsequently sent to device X, a digital-to-analog converter, by a fast-handshake TRANSFER.

When the TRANSFER completes ($T2=0$), the buffer empty pointer is automatically reset to 1, the fill pointer is set equal to 360, and the TRANSFER restarted. This continues indefinitely until the PAUSE key is pressed to stop the program. The effect of this program is to produce a near-continuous sine wave from device X. (Details of device X are purposefully left out here to avoid confusing the issue.)

```

10 ! Make room for 360 eight-bit values plus Pointers
20 DIM Z$(368)
30 IOBUFFER Z$
40 DEG
50 FOR I=0 TO 359
60 OUTPUT Z$ USING "#,B" ; SIN(I)*255
70 DISP I;0 NEXT I
80 TRANSFER Z$ TO 5 FHS
90 CONTROL Z$,1 : 360
100 GOTO 80
110 END

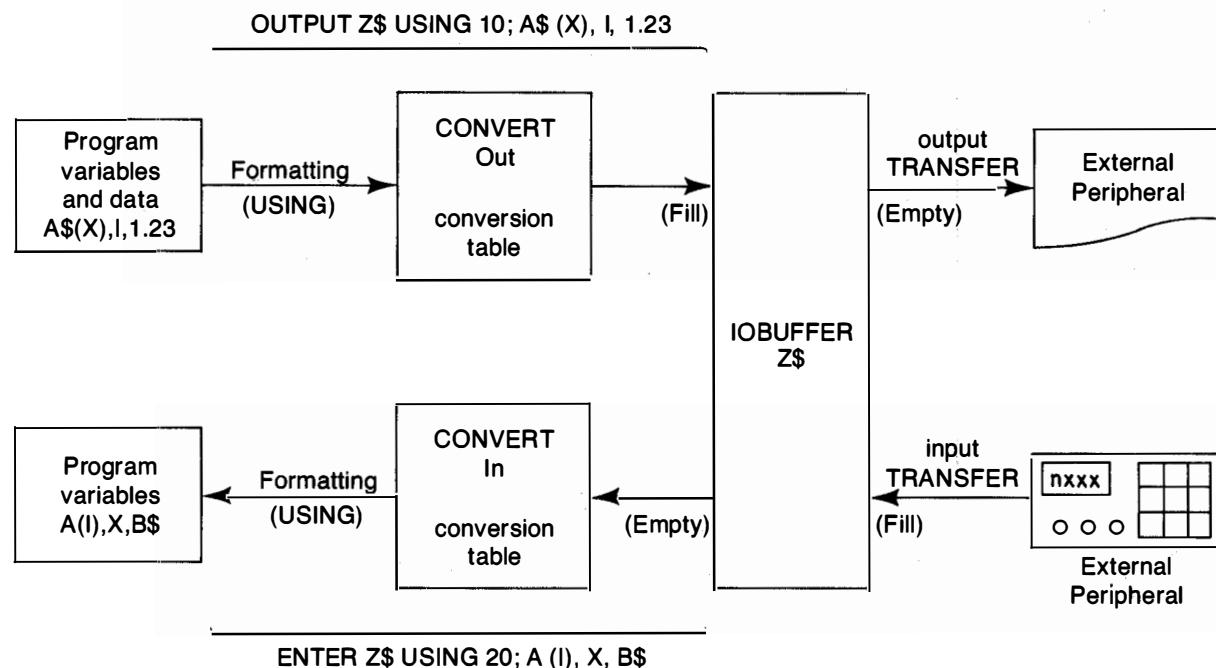
```

Lines 10-40 merely set up the buffer and place the computer in degrees mode. Lines 50-70 fill the buffer with the 8-bit value for each of 360 degrees (one complete sine wave), and line 80 starts the fast-handshake TRANSFER. Line 90 is executed when the TRANSFER is complete. The buffer fill pointer is set back to 360 (so it looks full), and the TRANSFER is restarted.

By exercising control over the buffer empty and fill pointer, it is possible to retransmit data, transmit any portion of the data in the buffer, to write data into any section of the buffer, read data out of any section of the buffer, etc. These operations may not be ones that you need to use in your application, but the flexibility they provide you could make feasible certain I/O operations not possible through any other means.

Data TRANSFERS

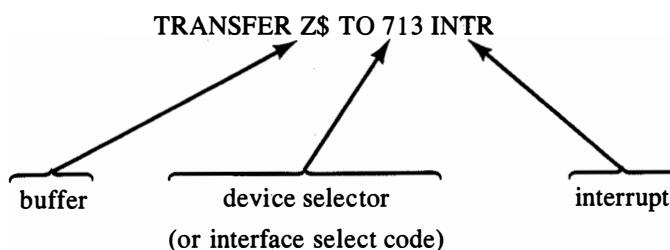
The TRANSFER statement has been mentioned several times up to this point. In combination with the IOBUFFER, it provides you with unmatched flexibility in tailoring and optimizing a program to exchange data with one or more peripheral devices. The diagram below shows the relationship of the TRANSFER with the IOBUFFER (or more simply, buffer), conversion tables, and program variables.



The TRANSFER itself is the easiest section of the entire picture to understand. Simply stated, an output TRANSFER takes characters, or bytes, out of the buffer from the position specified by the buffer empty pointer and sends them to the external peripheral. Conversely, the input TRANSFER takes characters from the external peripheral and places them in the buffer at the position specified by the buffer fill pointer.

Output TRANSFER

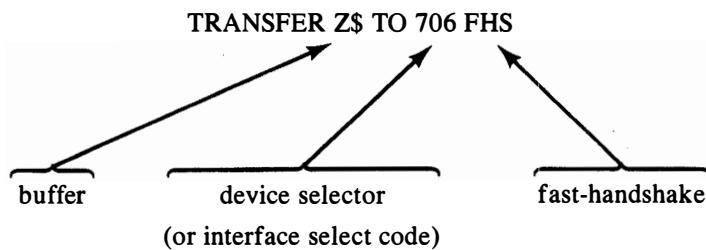
For an output TRANSFER, you merely specify whether an interrupt or a fast-handshake TRANSFER is to be performed. For example, to specify an interrupt output TRANSFER —



The operation of an interrupt output TRANSFER is as follows:

- When the statement is executed, the interface involved (in this case, an HP-IB Interface at select code 7) is automatically enabled to **interrupt¹** the computer when ready to accept a new character.
- Thereafter, each time the interface interrupts, the computer temporarily suspends program execution long enough to move the next character from the buffer to the interface. (It is then the interface's responsibility to see that the new character is properly sent to the peripheral.)
- When the buffer is finally emptied (described in "The Care and Feeding of Buffers"), the computer disables (or turns off) further interrupts from the interface. The TRANSFER is now essentially complete, although the interface may still be sending out the specified end-of-line character sequence (normally a carriage-return/line-feed: see the appropriate Interface Programming section).
- When the TRANSFER is terminated, a check is made for a user-defined end-of-transfer branch. If one is defined, then the branch is taken upon completion of the current program line. This is explained more thoroughly in the section "End-of-Line Branching."

A fast-handshake output TRANSFER is specified as —



The operation of a fast-handshake output TRANSFER is significantly different:

- The computer suspends program execution and dedicates itself to the task of moving all the characters in the buffer to the interface. The computer **totally** devotes itself to this task until the buffer is emptied. No other interrupts are allowed, not even a RESET! This means that once a fast-handshake TRANSFER begins the computer will see it through to completion, and **all** other operations are ignored until that time.
- When the buffer is empty, the TRANSFER is essentially finished. (The interface may still be in the process of sending the end-of-line character sequence.) If a user-defined end-of-transfer branch is specified, the branch is taken upon completion of the current program line.

¹ This is a hardware-level interrupt, and is distinct from the software-level end-of-line branch discussed in Section 9.

Input TRANSFER

The input TRANSFER is essentially the same as the output TRANSFER, except that data is being moved from the external peripheral to the computer. However, in addition to specifying whether an interrupt or fast-handshake TRANSFER is to be performed, you may also specify the terminating condition(s) for the TRANSFER. The terminating conditions for both interrupt and fast-handshake TRANSFER's are:

- COUNT — the number of characters or bytes of data to be input before the TRANSFER is considered complete. Use COUNT when the number of characters or bytes being sent from the peripheral device is known.
- EOI — an interface-specific terminating condition. In the case of the HP-IB Interface, some devices set the EOI control line when sending the last byte of data indicating end-of-data. This terminates the TRANSFER if EOI is specified.
- Default termination — a full buffer termination. An input TRANSFER terminates with the first occurrence of any specified terminating condition or a full buffer. (Obviously, if there is no room left in the IOBUFFER, the TRANSFER cannot continue.) If no other terminating condition has been specified, an input TRANSFER will terminate when the IOBUFFER becomes full.

The terminating conditions for interrupt TRANSFER's only are:

- DELIM — the numeric value of the character or byte of data that indicates all data has been sent. Use DELIM when the number of characters or bytes being sent from the peripheral device is either not known or varies for some reason. The peripheral device should not send the DELIM character as part of the data being sent, or else the TRANSFER will be terminated prematurely!
- Interface termination — an interface specific termination. Certain interfaces allow you to specify additional terminating conditions by writing to the interface control registers. These conditions may be either receipt of user-specified characters or line status, so consult the appropriate interface programming section to determine which if any, conditions your interface supports.

Programming with TRANSFERS

Every TRANSFER operation requires an IOBUFFER, which in turn requires a dimensioned string variable of adequate size. After executing the IOBUFFER statement, any CONVERT statement(s) for that buffer may be executed. This is because IOBUFFER initializes all the buffer pointers, including those to any conversion tables.

If there is any necessary interface initialization to do, it may be done anytime before executing the TRANSFER. If your program uses end-of-line branching (ON EOT : see the "End-of-Line Branching" section), the appropriate statement(s) should be executed before the TRANSFER statement.

An input TRANSFER may be executed at any point after the setup sequence described above. An output TRANSFER however, requires data in the buffer before anything can be sent. Therefore, an OUTPUT or a string assignment operation needs to be performed before attempting to execute the output TRANSFER.

It has been said that either an OUTPUT or a string assignment operation needs to be performed before executing an output TRANSFER so that there is data available in the buffer. This is not entirely true. Nor is it necessary to have all the data in the buffer before initiating an output interrupt TRANSFER. Also, it is not necessary to wait for an input interrupt TRANSFER to complete before reading data from the buffer. (Fast-handshake TRANSFERS are totally sequential operations and do not apply to the following discussion.)

If the external device is sufficiently slower than the computational speed of the HP-85, it is possible to initiate an output TRANSFER with the buffer only partially filled. Then, as more data becomes available to send to the device, the new data can be OUTPUT (for example) to the buffer while the TRANSFER is still in progress. This process can go on indefinitely until either: 1) all data has been sent or, 2) the buffer is filled (then the user's program must wait for more buffer space) or, 3) the buffer is emptied (then the TRANSFER terminates and it must be restarted).

Case 1 above is straightforward and needs no explanation. Case 2 is determined by the buffer status, and the program can wait for more buffer space by monitoring buffer status (with the STATUS statement). The occurrence of case 3 indicates that the peripheral device is actually faster than the HP-85 in this instance. (This may be due to excessive computation being performed or some other circumstance causing a delay in the program.) An end-of-line branch or a status check on the buffer indicates a TRANSFER completion, and the decision to continue the TRANSFER can be made at that time. The following example program illustrates how these conditions might be dealt with:

```

10 ! This Program shows how you might check for cases 1,2, and 3
20 ! The buffer B$ size is determined empirically to optimize
30 ! memory usage with transfer rate.
40 DEG @ DIM B$[160]
50 DEF FNA(X) = INT(SIN(X)*35+36.5)
51 CONTROL 7,16 : 0 ! This turns off CR/LF from the interface
52 IOBUFFER B$
53 !
54 FOR X=0 TO 359 STEP 10 ! The For-Next loop handles case 1
55 STATUS B$,1 : S1,S2,S3,S4
56 ! Test for buffer empty before putting in data (case 2)
57 Z=FNA(X)
58 IF S1+Z>145 THEN GOTO 90
59 I$=VAL$(Z)+"X,A"
60 OUTPUT B$ USING I$ : "*"
61 ! Test the active-out select code before starting TRANSFER
62 IF S4=0 THEN TRANSFER B$ TO 720 INTR ! (case3)
63 NEXT X
64 !
65 GOTO 54
66 END

```

Section 9

End-of-Line Branching

Section Introduction

The purpose of this introduction is to give you an understanding of end-of-line program branches. If you have a knowledge of this facility already, skip the background information and move on to "Using End-of-Line Branching." That section deals with the actual programming aspects of using end-of-line branches.

Some Background on Interrupts

If you've never heard the term "interrupt" with regards to a computer before, then a simple way to think of it is like a telephone ringing while you are working. It is a means of diverting your attention from whatever it is you are doing. "Servicing" an interrupt is similar to the act of going over to answer the ringing telephone. When the telephone business is completed, you typically resume the "interrupted" activity where you left off.

If you have a switch that can disconnect the telephone bell (so it can't ring), then you can "disable" or "enable" that interrupt by the setting of the switch. The computer has essentially the same facility.

In addition, the computer has two types of interrupts to deal with: low-level, or hardware interrupts; and high-level, or software interrupts. In general, the hardware interrupt is used by the computer for its own purposes, and is transparent to the user. Hardware interrupts make possible such things as the INTR type of TRANSFER, where program execution and I/O are concurrent operations. However, external (and internal) events can also trigger hardware interrupts that require specialized service routines to deal with them. Because there is such a wide variety of interrupt causes possible, it is necessary to make provisions for the programmer to custom tailor service routines for those interrupts specific to his system. This provision is the software interrupt, or end-of-line branch.

Imagine a "ghost" IF <event> THEN GOSUB <line#> statement appended to the end of each program line, like the ones below:

```
50 PRINT "Result ="; X@IF <event=true> THEN GOSUB <routine>
60 X=X+Y @IF <event=true> THEN GOSUB <routine>
```

In effect, at the end of each program line, a check is made for the "event," which may be an ERROR, a hardware interrupt, a select code timeout, a TRANSFER completion, a special function key-press, or a timer timeout. (End-of-line service for three of the above events is already provided for in the HP-85 mainframe: special function keys and timers.) When one of the events occurs, a special portion of the program can be executed to deal with that event.

The programmer also has the ability to disable or enable the end-of-line service facility for each event, as will be shown in the next section on programming for end-of-line branches.

End-of-Line Branch Programming

Introduction

This section shows you how to program service routines to deal with end-of-line branches for I/O related events. At the end of this section is a sub-section entitled "Interactions and Permutations." The discussion there deals with problems and questions you may encounter such as "GOSUB vs GOTO."

Interface Interrupts

Each interface (HP-IB, RS-232C, BCD, etc.) has a control register that allows you to specify an interrupting condition. This is control register CR1, the Interrupt Mask register. When you set a bit in that register, you enable the interface to interrupt the computer when the event corresponding to that bit occurs. The interrupt event is determined by referring to the programming section for the particular interface. For example, the HP-IB Interface provides for an interrupt on SRQ, or Service Request. The SRQ interrupt is enabled by setting bit 3 of control register CR1. To enable the SRQ interrupt for the HP-IB Interface at select code 7, execute

ENABLE INTR7;8

This is equivalent to

CONTROL 7,1;8

and both statements set bit 3 of control register CR1 on select code 7.

Suppose (for simplicity) that only one device is connected to the HP-IB Interface, and that device asserts SRQ only to indicate it is ready to send a numeric value to the computer. The statement we'll use to read this value is

ENTER 701;V1

This is the extent of our example service routine, which will take the form of a subroutine in the program.

Once the computer has read the new value from the device, it must again enable the SRQ interrupt so that another value can be read when the device is again ready. The service routine begins to take shape:

```

1000 ! SRQ Service Routine for select code 7
1010 STATUS 7,1 ; A ! Always read interrupt cause register
1020 ENTER 701 ; V1
1030 ! Re-enable interrupt and return on same line
1040 ENABLE INTR 7;8 @ RETURN

```

Note that the ENABLE and the RETURN are on the same program line. There are some occasions where this is a necessary practice, and this is discussed under "Interactions and Permutations."

It is necessary to specify **where** the service routine for an event is, and also whether the routine is a subroutine (GOSUB) or just a program segment (GOTO). This is accomplished by the ON INTR statement. Note that the computer must know where to go and what to do before an interrupt occurs, or else it will be forced to ignore it. Therefore, the ON INTR statement should be executed before the ENABLE INTR statement, as shown in the example below.

```

10 ON INTR 7 GOSUB 1000
20 OUTPUT 722 ;"F1R7T2T3D1"
30 V1=X=0
40 LOCAL 722
50 ENABLE INTR 7:8
60 ! The following lines are a dummy program
70 X=X+1 @ DISP X,V1
80 WAIT 100 @ GOTO 70
.
.
.
1000 ! SRQ Service Routine for select code 7
1010 STATUS 7,1 ; A ! Always read interrupt cause register
1020 ENTER 722 ; V1
1030 ENABLE INTR 7:8 @ RETURN
1040 END

```

Line 10 specifies the location (line 1000) and type (GOSUB, not GOTO) of the service routine for select code 7. Line 20 initializes the HP 3455A voltmeter to take readings whenever it is triggered. The LOCAL statement of line 40 allows the DVM to be triggered manually, and the ENABLE statement enables the interface for SRQ interrupt (bit 3 of the interrupt mask is set). Lines 70 and 80 merely display an incrementing counter and the last reading taken from the DVM.

Each time an SRQ is received, program execution of lines 70 and 80 is suspended, and the subroutine at line 1000 is executed. The ENTER is performed, which takes a new value for V1. The SRQ interrupt condition is again enabled in line 1030, and the RETURN allows the program display loop to resume. However, the value for V1 that is now displayed is the new value just read by the service routine.

What would happen if the ENABLE INTR on line 1030 were removed?

Simple. The first SRQ would be serviced as described above, and the program loop then resumed. However, no more SRQ interrupts would be detected, even if the HP-IB device were frantically sending the SRQ message to the interface! Without that bit set in CR1, the interface **ignores** the event — regardless of its importance.

To disable or cancel the end-of-line branch condition set up by ON INTR, simply execute an OFF INTR statement for the appropriate select code. No more branches will be taken until another ON INTR statement is executed.

Timeouts

It is possible for an external device to exhibit symptoms of "narcolepsy" (going to sleep unpredictably) from time to time. Of course, being an electronic device, this "falling asleep" is generally caused by an electronic malfunction or some other unusual condition, such as being switched off. It is not as important to focus here on the possible causes of device "failure" so much as the possible effects.

When a device fails, for whatever reason, the immediate effect on the computer is that it is no longer "handshaking" data to or from the computer. (Handshaking is a means of reliably transferring data between two devices. The sender makes data available, signals that data is ready, and the receiver accepts the data and signals that it has taken the data. This protocol is called "handshake.") If an ENTER, SEND or OUTPUT operation is in progress, a loss of handshake means no data can be transferred and the operation cannot complete. If the operation does not complete, the program "hangs" until the operator becomes aware that something has gone wrong. This is unacceptable for most I/O systems, and especially those where unattended operation is frequent.

The SET TIMEOUT statement gives you the capability of establishing a maximum time period for the computer to wait on an interface to handshake data. If an interface exceeds the SET TIMEOUT limit specified, two alternative courses of action may be selected. One method simply aborts the I/O operation in progress and continues program execution at the next line. The other method executes a timeout service routine after the I/O operation is aborted.

The following example shows the simpler form of SET TIMEOUT, where no timeout service routine is specified:

```

10 SET TIMEOUT 7:10000
20 OUTPUT 706 :"Simple test data"
30 DISP "Now at line 30"
40 END

```

If device 706 stops responding to handshake before the OUTPUT operation is complete, then after 10 seconds line 30 will be executed. (Line 30 would also be executed if the OUTPUT were successful, obviously.)

A more sophisticated method for dealing with a timeout condition is to execute a timeout service routine when an I/O operation is aborted due to a timeout. A separate service routine may be programmed for each select code to deal with the specific device or conditions for that select code. The ON TIMEOUT statement provides this capability, as shown below:

```

10 SET TIMEOUT 7:2000
20 SET TIMEOUT 5:4000
30 ON TIMEOUT 7 GOSUB 1000
40 ON TIMEOUT 5 GOSUB 1100

```

```

50 ENTER 706 ; V1,X
60 OUTPUT 5 ;V1,X
70 GOTO 50
.
.
.
1000 ! Timeout service for select code 7
1010 PRINT "HP-IB Timeout"
1020 RESET 7 ! Attempt to recover
1030 RETURN
1100 ! Timeout service for select code 5
1110 PRINT "Select code 5 failure"
1120 RESET 5
1130 RETURN

```

Obviously, in a dedicated system, more significant action could be taken than simply printing a message, aborting the operation with a RESET, and returning to the program. For example, a flag could be set by the timeout routine which would be checked before attempting an I/O operation to that select code. This flag might indicate printer out of paper or tape reader at end of tape, which would make I/O to such a device a useless endeavor. It might also be the only means available of determining when an operation has completed. The following example shows how a flag might be used:

```

10 SET TIMEOUT 6:1500
20 ON TIMEOUT 6 GOSUB 1000
21 ! Clear timeout flag (F1)
30 F1=0 @ DIM V(50)
40 FOR I=1 TO 50
50 IF F1<>0 THEN GOTO 80 ! Test timeout flag before ENTER
60 ENTER 6 ; V(I)
70 NEXT I
80 PRINT I;" Values entered" @ STOP
.
.
.
1000 DISP "Timeout : end of data"
1010 F1=1 ! Set timeout flag
1020 RETURN
1030 END

```

When a timeout occurs, the flag F1 is set equal to 1, and the IF statement of line 50 causes the ENTER statement of line 60 to be skipped. The loop is exited and the program continues. (In the example, only a PRINT and STOP are shown for simplicity.) In this case, the cause for the timeout is not device failure, but rather the end of data to be sent.

As with ON INTR, there is a corresponding disable statement for timeout end-of-line branches. The OFF TIMEOUT statement can be used to cancel end-of-line service for a specified interface timeout. The SET TIMEOUT statement with a time limit of 0 can also be used, which sets an (almost) infinite timeout limit for the interface. For more information regarding syntax rules, refer to the alphabetical syntax listing section of this manual.

TRANSFER Terminations

There are two means of determining when a TRANSFER operation has completed. Both are straightforward, but using an end-of-line branch service routine allows the greatest flexibility of program design.

As is mentioned in "The Care and Feeding of Buffers," buffer status registers SR3 and SR4 indicate input and output activity, respectively. When the appropriate register becomes equal to 0, that TRANSFER has terminated. However, this requires re-checking the status register until the TRANSFER terminates, and so inhibits program execution. If there is other work to be done, a fairly convoluted program logic will be the result of trying to monitor TRANSFER status and getting the other work done. There is a simpler way.

When an ON EOT (On End Of Transfer) statement is executed, an end-of-line branch is enabled for TRANSFER termination for the specified interface. Then you can program the operations necessary to deal with the TRANSFER termination into the ON EOT service routine. These operations might include placing new values into the buffer and re-initiating the TRANSFER (for output) or taking values from the buffer and placing them into program variables (for input).

To illustrate how the ON EOT statement might appear in a program, consider the following example. A voltmeter (HP 3437A) is programmed to take a reading every .9 seconds. An interrupt TRANSFER is set up for 100 readings of seven characters each, plus a final carriage-return/line-feed and the 8 extra bytes needed for pointers.

When the TRANSFER completes, the service routine at line 1000 is executed which saves the readings and initiates a new TRANSFER.

```

10 ! Transfer termination service routine example
20 ON EOT 7 GOSUB 1000
30 OUTPUT 724 ;"0.9SN100SE00R3T1F1"
40 I=1 @ X=0 @ DIM D$[E710],D(100)
50 IOBUFFER D$
60 ! This Program assumes 10 data files are already
70 ! created, and that the file names are in the next
80 ! data statement.
90 DATA data1,data2,data3,data4,data5,data6,data7,data8,data9,
data10
100 TRANSFER 724 TO D$ INTR
110 X=X+1 ! Simple "work" loop
120 DISP X
130 GOTO 110
:
:
1000 ! EOT service routine
1010 ! Read next file name and open file on tape.
1020 READ F$@ ASSIGN# 1 TO F$
1030 ! Take values from buffer
1040 FOR N=1 TO 100
1050 ENTER D$ ; D(N)

```

```

1060 NEXT N
1070 ! Now initiate a new TRANSFER
1080 TRANSFER 724 TO D$ INTR
1090 ! Save data on tape. NOTE: this temporarily halts the transfer!
1100 PRINT# 1 , D$ @ ASSIGN# 1 TO *
1110 ! See if all 10 data files have been saved.
1120 I=I+1 @ IF I>10 THEN GOTO 1140
1130 RETURN
1140 ! If done, then halt I/O and end program.
1145 ! Note that ABORTIO causes an EOT branch, so disable it
1150 OFF EOT 7 @ ABORTIO 7
1160 PRINT "All data saved"
1170 END

```

The program logic associated with the TRANSFER and the EOT service routine is quite simple (shaded program lines). The extra programming shown is included as one example of how the EOT service might be used for data-save operations to tape. The EOT service routine might just as well have sent the collected data to a remote central computer (with another TRANSFER, of course!)

End-of-transfer service can be cancelled by execution of an OFF EOT statement for the desired select code. For further details about these statements, refer to the alphabetical syntax listing in this manual.

Interactions and Permutations

This section is written so that you may be able to better predict operation of those programs that utilize end-of-line branching.

End-of-line service occurs in a specific order. That is, if more than one end-of-line branch is pending at the end of a program line, one of the branches will be taken before the other. The following table lists the types of end-of-line branches and the select codes, and gives the precedence order for combinations of branch type and select code.

Branch Precedence Table

Branch Type	Select Code								
	3	4	5	6	7	8	9	10	
ON ERROR				1					
ON INTR	2	3	4	5	6	7	8	9	
ON TIMEOUT	10	11	12	13	14	15	16	17	
ON EOT	18	19	20	21	22	23	24	25	
ON KEY				26					
ON TIMER					27				

For example, a pending ON INTR branch for select code 5 would be taken before a pending ON INTR branch for select code 9. Any pending ON EOT branch would be taken after any pending ON TIMEOUT branch, regardless of select codes.

You should take note that the term **precedence** is used here, not **priority**. This means that when the computer is executing one service routine, other service routines are not implicitly locked out. (In a priority system, any service routine having a lower priority than the routine currently being executed will not receive control until the current routine completes.) If two end-of-line branches are pending on the HP-85, the one having precedence is executed first. However, after the first line of that service routine executes, the still-pending end-of-line branch to the second service routine is taken!

The only way the first routine can guarantee uninterrupted execution is to disable the other routine's end-of-line branch with its first line. An example will help show this:

```

10 ON INTR 7 GOSUB 100
20 ON INTR 9 GOSUB 200
30 ENABLE INTR 7;8
40 ENABLE INTR 9;64
.
.
.
70 ! Program body goes here
.
.
.
100 OFF INTR 9 ! Now select code 9 cannot set EOL service.
.
.
.
130 ! This routine would do necessary service functions for
140 ! select code 7.
.
.
.
180 ! Now re-enable EOL service for select code 9
190 ON INTR 9 GOSUB 200 @ ENABLE INTR 7;8 @ RETURN
200 OFF INTR 7 ! Now select code 7 cannot set EOL service.
.
.
.
240 ! This routine would do necessary service functions for
250 ! select code 9.
.
.
.
280 ! Now re-enable EOL service for select code 7
290 ON INTR 7 GOSUB 100 @ ENABLE INTR 9;64 @ RETURN
300 END

```

Assume that while line 50 is executing, both interrupts occur. Both end-of-line branches are pending when line 50 completes. Select code 7's service routine has precedence, and so line 100 is executed. Line 100 disables select code 9's service routine, so the end-of-line branch that would have occurred is not taken. Line 190, the last line of the select code 7 service routine, re-enables the select code 9 service routine and when the RETURN is executed, line 200 is then executed. Note that the OFF INTR does not "eliminate" the pending end-of-line branch — it just defers the branch until an ON INTR is executed later.

Events such as interrupts, timeouts, TRANSFER terminations and ERRORS are "remembered" indefinitely, or until a RESET or STOP occurs. Therefore, once an event has occurred and remains unserviced, execution of an ON INTR, ON TIMEOUT or ON EOT results in an immediate end-of-line branch.

The type of branch taken can affect program operation. The most predictable program execution occurs when GOSUB end-of-line branches are taken. Use GOTO when the program is simple and only one or two end-of-line branches are expected. One very simple example of this would be:

```
ON INTR 7 GOTO 9999
ENABLE INTR 7;128
```

These two statements direct program execution to line 9999 (presumably an END statement) when an Interface Clear on HP-IB occurs. This kind of "bail-out" provision can be useful when developing certain types of programs such as ones using keyboard masking (see Section 10). Then, when something goes wrong you can halt the program by asserting IFC on the interface bus with another controller or a bus analyzer.

The following example is presented to show you how some of the ON event branches interact. The listing and results are shown with a brief explanation, but you will learn more from it if you experiment on your own. Try setting timers, timeouts, and keys up and then watch it work (or not work). Press the special function key while the computer is waiting, then watch the displays.

```
10 ! This program serves to illustrate end-of-line branching.
20 ! Substitute YOUR select codes and device numbers
30 ! on lines 80 and 90 to make this run successfully.
40 ! Remember that an ENTER and an input TRANSFER cannot
50 ! occur simultaneously on the same select code.
60 !
70 !
80 S1=10 ! This is the TRANSFER select code assignment.
90 S2=705 ! This is the ENTER select code assignment.
100 I=0 @ DIM A$[80],B$[88]
110 IOBUFFER B$
120 !
130 !
140 ON KEY# 1 GOSUB 390 ! Key 1 clears the counter
150 ON KEY# 2 GOSUB 450 ! Key 2 terminates the TRANSFER
160 ON KEY# 3 GOSUB 510 ! Key 3 ends the program
170 ON KEY# 4 GOSUB 770 ! Key 4 starts another ENTER
180 !
190 !
200 ON TIMER# 1,1300 GOSUB 560
210 ON TIMER# 2,1400 GOSUB 610
220 !
230 !
240 SET TIMEOUT S2,1000 ! Dont forget to set the select code!
250 ON TIMEOUT S2 GOSUB 660
260 !
270 !
280 ON EOT S1 GOSUB 710 ! Dont forget to set the select codes!
290 TRANSFER S1 TO B$ INTR
```

```
300 ENTER S2 : A$  
310 !  
320 !  
330 BEEP 100,20 @ WAIT 500 @ BEEP 20,20  
340 DISP "Looping ";I  
350 I=I+1 @ GOTO 330  
360 !  
370 !  
380 !  
390 ! Key 1 service clears the counter.  
400 I=0  
410 DISP "Serviced Key 1"  
420 RETURN  
430 !  
440 !!  
450 ! Key 2 service terminates the active TRANSFER.  
460 RESET S1  
470 DISP "Terminated TRANSFER"  
480 RETURN  
490 !  
500 !  
510 ! Key 3 service ends the program.  
520 DISP "End of Program"  
530 END  
540 !  
550 !  
560 ! Timer 1 service.  
570 DISP "Timer 1 serviced."  
580 RETURN  
590 !  
600 !  
610 ! Timer 2 service.  
620 DISP "Timer 2 serviced."  
630 RETURN  
640 !  
650 !  
660 ! Select code S2 ENTER Timeout service  
670 DISP "ENTER operation timed out"  
680 RETURN  
690 !  
700 !  
710 ! Select code S1 TRANSFER termination service.  
720 DISP "End of transfer service"  
730 TRANSFER S1 TO B$ INTR  
740 RETURN  
750 !  
760 !  
770 ! Key 4 service executes another ENTER operation  
780 ENTER S2 : A$  
790 RETURN  
800 END
```

Section 10
Keyboard Control

Section Introduction

In certain applications, the operation of a system could be adversely affected by a curious passerby pressing keys which could halt or alter program execution. In other applications the system operator must have access to certain keys, but not others.

Traditionally, the dedicated computer's keyboard was covered by a "mask" of plastic, metal or wood which covered all keys unnecessary to the operation of the system. Cut-outs were provided to allow access to those keys the operator used to control the system. Obviously, the extra cost of designing and manufacturing such a mask added to the cost of a dedicated system.

The HP-85 I/O ROM provides the keyboard mask in software, which — short of turning the computer off — offers far less chance of user interference than does a mechanical mask. In addition, greater flexibility is possible with the software masks, as is discussed in the following section.

Key Mask Programming

There are three basic modes of operation for the HP-85: halted (or idle), program execution, and keyboard input.

In the idle mode, the computer accepts commands and is available for program modification/entry. This is the mode that the computer normally is in when not executing a program (when first turned on, or after executing a STOP, PAUSE, or END type statement). The idle mode is generally used for program development or debugging, so a keyboard mask for the idle mode is not very useful and is not provided.

In the program execution mode, pressing any key other than an ON KEY defined special function key will halt the program. A keyboard mask can be specified for four classes of keys while in the program execution mode. These classes are:

1. RESET
2. PAUSE
3. Special Function Keys and KEYLABEL key
4. Other keys (all remaining keys not in classes 1, 2, or 3)

Any or all four classes of keys can be masked out for the program execution mode.

In the keyboard input mode, the computer is temporarily halted awaiting operator response. This is the mode of operation for the INPUT statement. The four classes of key masks for the keyboard input mode are:

1. RESET
2. PAUSE
3. Special Function Keys and KEYLABEL key
4. Other keys (all remaining keys not in classes 1, 2, or 3)

The ENABLE KBD statement takes as a parameter the key mask desired for the appropriate operating mode. The upper four bits of the mask parameter specify the program execution keyboard mask. The lower four bits specify the keyboard input key mask. The bits of the mask parameter are shown below. Setting a bit **enables** the corresponding key(s), while clearing a bit **disables** the key(s). That is, only those keys having a bit set in the ENABLE KBD mask will respond.

Bit Number	Decimal Value	Operating Mode	Keys Masked
7	128	↑ Program Execution ↓	RESET PAUSE Special Function Keys and KEYLABEL Other keys
6	64		
5	32		
4	16		
3	8	↑ Keyboard Input ↓	RESET PAUSE Special Function Keys and KEYLABEL Other keys
2	4		
1	2		
0	1		

The following example illustrates the use of the keyboard mask to disable all keys except the Special Function Keys for program execution. The RESET, PAUSE, and Special Function Keys are masked out for the keyboard input mode. This still allows the operator to respond to an INPUT statement, but he cannot affect program execution in that mode.

```

10 ENABLE KBD 1+32 ! Select allowed keys
20 REMOTE 706,710 @ LOCAL LOCKOUT 7
30 ON KEY# 1 GOTO 1000
40 ON KEY# 2 GOTO 2000
50 ON KEY# 3 GOTO 3000
:
:
100 PRINT "Select operating mode"
:
1000 DISP "Enter time" @ INPUT M,D,Y
1010 ! and etc.

```

The operator can select the desired program section by pressing the appropriate special function key — but only the special function keys will respond to being pressed. When the operator presses special function key #1, the program branches to line 1000. When the INPUT statement of line 1000 is executed, the special function keys no longer respond but the numeric keys do, allowing him to set the time as requested. When the program continues on from the INPUT, once again all keys except special function keys are disabled.

The following example gives the operator a ten second time span in which he can PAUSE or RESET the computer if necessary:

```
10 ! Line 50 enables the operator to pause or reset the program
20 ! whenever an INPUT statement is executed.
30 ! Line 100 exists specifically to allow the operator to halt
40 ! the program if necessary.
50 ENABLE KBD 12
60 !
70 !
80 !
90 BEEP @ DISP "You have 10 sec to PAUSE or RESET the computer"
100 ON TIMER# 1,10000 GOTO 110 @ INPUT X9$
110 OFF TIMER# 1 @ DISP "Computer beginning automatic operations"
120 !
130 ! Now the operator cannot halt the program!
140 !
150 !
```



Section 11
Direct Interface Communication

Section Introduction

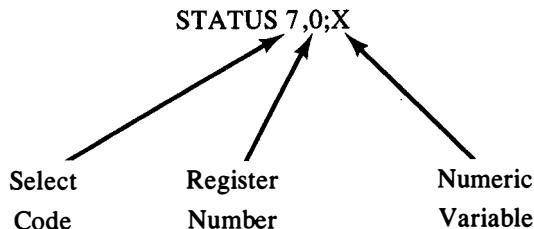
This section deals with the statements available to the programmer for tailoring the operations of an interface to the specific requirements of his system.

The status of interface operations can be monitored with the STATUS statement. This status may reflect the actual hi/low voltage level of external I/O lines or it may indicate the interface's internal state, depending upon which status register is being read.

The mode of operation of an interface can be directed by the CONTROL statement. The interface generally provides automatic control of external I/O lines according to the mode selected, and also may provide for manual override of certain of the I/O lines for custom sequences.

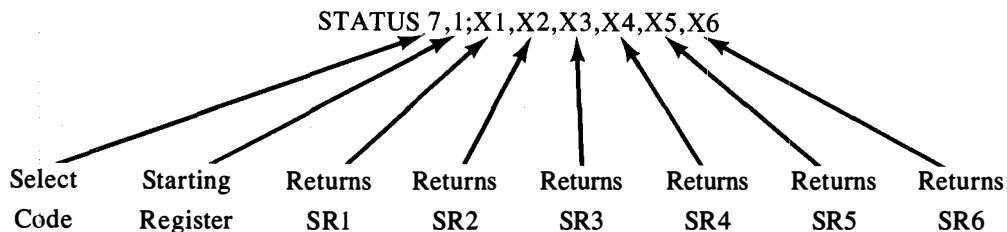
Checking the Status

Interface status (and also buffer status) is monitored by the STATUS statement. The STATUS statement is a very straightforward one to use, requiring only the interface's select code and desired register number to be specified. The following statement obtains the value of the HP-IB interface (select code 7) identification register, SR0:



In this case, the value returned is always 1, the identification code for an HP-IB interface. Each interface returns a different identification code, so consult the appropriate interface programming section for specific details.

The values of multiple status registers can be obtained with just one STATUS statement. Simply specify the first register number and give many variables as necessary to obtain the status needed. For example, to read status registers SR1 through SR6 of the HP-IB interface at select code 7:



What you do with the contents of these registers depends upon your application, needs, and type of interface. The HP-IB interface above, for example, returns

- Interrupt Cause - SR1
- HP-IB Control Lines - SR2
- HP-IB Data Lines - SR3
- HP-IB Address - SR4
- HP-IB State - SR5
- Secondary Command - SR6

for the six registers just read. Obviously the BCD Interface returns different information for those same registers (what's the HP-IB address of a BCD Interface??).

Most interfaces are capable of interrupting the HP-85 when a specified condition occurs, and if more than one condition has been enabled it will be necessary to determine which interrupt occurred and caused the end-of-line branch. The following service routine serves as an example of how STATUS would be so used:

```

10 ON INTR 7 GOSUB 1000
15 ! Enable interrupts for Active Controller/Talker/Listener.
20 ENABLE INTR 7:112
.
.
.
.
.
.
.
1000 ! End-of-line branch service for HP-IB interrupts
1010 STATUS 7,1 : S1 ! Read interrupt cause
1020 IF BIT(S1,4)=1 THEN GOTO 1100 ! Check for Active Talker
1030 IF BIT(S1,5)=1 THEN GOTO 1200 ! Check for Active Control
1040 IF BIT(S1,6)=1 THEN GOTO 1300 ! Check for Active Listener
1050 !
1060 ! The following lines provide for unpredictable errors.
1070 PRINT "Illegal interrupt condition on HP-IB"
1080 PRINT "HP-IB interrupt cause = ";S1
1090 STOP
  
```

```

1100 ! Service for Active Talker
.
.
1200 ! Service for Active Control
.
.
1300 ! Service for Active Listener
.
.

```

Line 1010 obtains the interrupt cause from SR1 of the interface, and lines 1020-1040 analyze the register bits for the specific cause of the end-of-line branch. Lines 1050-1090 are included to deal with possible program errors or illegal interrupt causes. No program or machine is perfect, so try to make provisions such as the one shown to deal with malfunctions that might occur.

Interface Control

The operating modes of an interface can generally be tailored to suit individual systems and their requirements. This may mean nothing more than selecting the type of interrupt conditions required to deal with events specific to your system. Or, it may mean selecting a handshake mode suitable for the device being connected to the interface. In any case, the CONTROL statement provides the means of programming the interface's control registers.

There are three primary items of interest in a general discussion about CONTROL and interface control registers. The first is the interface interrupt mask. This mask is CR1 for the HP-IB Interface. Therefore, the following two statements have **identical effects** for the HP-IB Interface (select code 7):

```

ENABLE INTR 7;8
and
CONTROL 7,1;8

```

Both statements enable an SRQ interrupt condition for the interface. They can be used interchangeably, however, the ENABLE INTR statement better documents the effects it will have on program execution.

The second item of interest is that of external I/O lines. In general, you can set or clear interface-specific control lines by writing to control register CR2 of the desired interface (however, consult the individual interface programming sections for exact details). Both of the following statements write to HP-IB control register CR2, the HP-IB Control Lines register (select code 7):

```

ASSERT 7;32
and
CONTROL 7,2;32

```

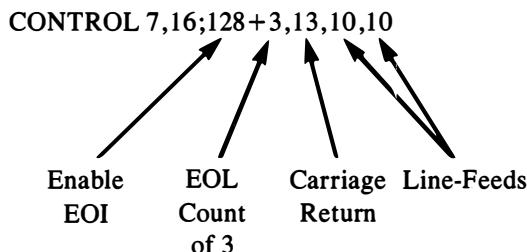
Both statements set the HP-IB SRQ (Service Request) control line TRUE, but there is a slight difference in the manner that they do so. The CONTROL statement is not immediately executed if there is an I/O operation already in progress (an interrupt TRANSFER, for example). The I/O in progress first completes, then the CONTROL operation is performed. This is in contrast to the ASSERT statement, which **immediately** sets the control line (SRQ in this case) regardless of any I/O in progress. In either case, exercise due caution when writing to the interface control line register: the consequences of an improper control operation may be an interface or device malfunction. (In fact, to assert SRQ you should really use the REQUEST statement: proper bus protocol is ensured.)

Note: Improper or invalid control line operation may cause loss of data or device malfunctions.

The third item of general interest in control register programming is the end-of-line (EOL) character sequence sent by the interface. This character sequence is essentially an end-of-record delimiter, which in the days of punched cards signalled that an entire card had been read or punched. For all interfaces, this EOL sequence defaults to carriage-return/line-feed, and is sent after every PRINT or OUTPUT operation (unless inhibited by the programmer). The EOL sequence is also sent for the “/” (slash) IMAGE specifier or for the “END” keyword of the SEND statement.

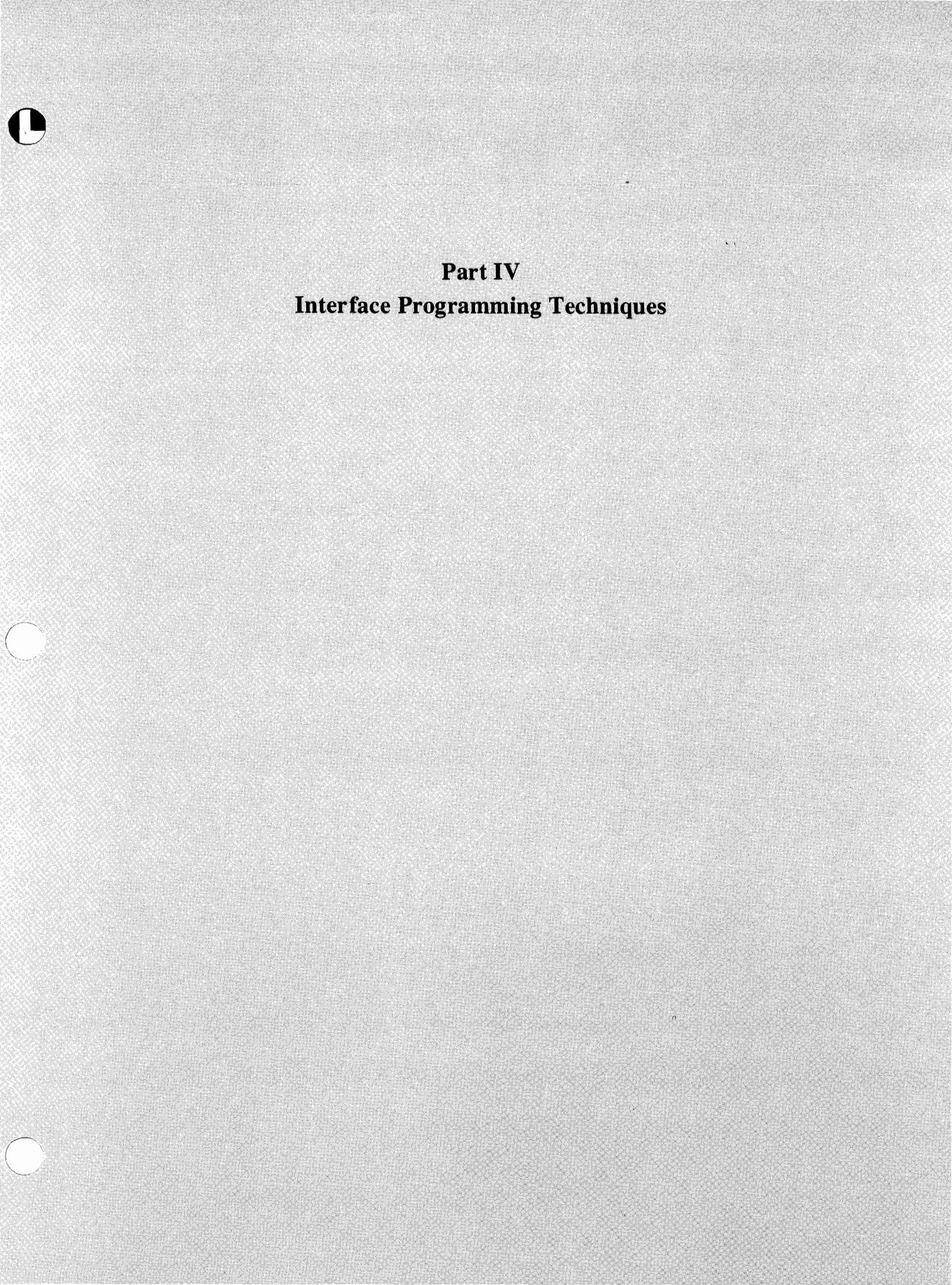
It is sometimes necessary to tailor the EOL sequence to suit the needs of a particular device. For instance, you might need double spacing on a printer (carriage-return/double line-feed) or carriage-return only for a CRT terminal that performs an automatic line-feed when a carriage-return is received.

The HP-IB Interface provides EOL sequence programmability with control registers CR16-CR23. Up to seven EOL characters can be sent if so desired, and are specified by writing the character values and EOL character count to those control registers. For example, to program a double line-feed EOL sequence with EOI (END or Identify) set on the last line-feed¹, the following CONTROL statement could be used:



In summary, the control registers provide the flexibility required to tailor interface operation to your specific system requirements. They should be used with caution, however, so that you don't get any unpleasant “surprises.” Study the appropriate interface programming section of this manual to determine which capabilities you need and how to implement them. It may be necessary to experiment a bit before the system works as you want it to, but even the “pros” have to do that. Good luck.

¹ You should note that it is not possible to send EOI with the last data byte of an OUTPUT or SEND operation, but it is possible with PRINT, DISP, or TRANSFER.



Part IV

Interface Programming Techniques



Section 12

Using the HP-IB Interface

Section Introduction

This part is organized into three major sections:

1. A Basic Introduction to the HP-IB (for the novice).
2. HP-IB Operations (for the intermediate).
3. HP-IB for the Specialist.

It is not necessary for you to read all sections to use the HP-IB interface. In fact, if you just wish to use a printer for program output and listings, you needn't read beyond the next paragraph.

Let's assume you have a printer and an interface. To use your printer, you must:

1. First install the interface **WITH THE POWER OFF!** [See the installation procedure in the Interface Installation Note].
2. Plug the cable connector into the connector on your printer.
3. Write down your printer's address (it's in your printer manual).
4. Then write down your interface select code (factory set to 7).

Assuming your printer address is 01, and your interface select code is 7, you turn power on and execute

PRINTER IS 701

Now PLIST a program, and the listing will be printed on your HP-IB printer rather than on the HP-85A printer. Any PRINT statements in your program will now be directed to your HP-IB printer.

The procedure is similar for other output devices as well. If this information is sufficient to make your device work, then go no further. However, if you have a more difficult problem, or if you wish to know more of the real power of the HP-IB interface, then read on.

A Basic Introduction to the HP-IB

This section is oriented to the new owner of an HP-IB interface. The depth of coverage is minimal, and is aimed primarily at giving the first-time user a means of understanding the "what" of HP-IB.

First Things First

HP-IB. The letters stand for **Hewlett-Packard Interface Bus**. It is Hewlett-Packard's implementation of the IEEE-488-1978¹ interface standard. The purpose of the HP-IB is to provide for mechanical, electrical, timing, and data compatibilities between all devices adhering to the standard². In essence, interfacing computers to other devices has been simplified by the HP-IB. Instead of worrying about **how** to hook up your devices so they can communicate, you merely have to consider **what** they are going to communicate. Given that all the details of compatibility have been worked out, we can move on to understanding the general structure of a **system** tied together with the HP-IB, commonly called "the bus."

General Structure of the HP-IB

Like most things one learns about, the HP-IB has a structure — a precise organization — that prevents chaos from becoming the general rule. 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 describe 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 the meetings and organizing the agenda. The chairman is responsible for the actions and results of the committee, and generally uses the "rules of order" to ensure the proper conduct of business. If, for some reason the committee chairman cannot attend a meeting, he designates some other member to be "acting chairman."

For the HP-IB, the committee chairman is the **system controller**. This is generally established by a switch setting on the interface, and cannot be changed under program control. However, it is possible to designate an "acting chairman" 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. These responsibilities may be subsequently passed to another device while the system controller tends to other business: more than one computer can be connected to the HP-IB at the same time.

On a committee, only one person at a time may speak (this alleviates confusion somewhat) and the chairman is responsible for "recognizing" appropriate members to speak, one at a time. For the HP-IB, the device designated to "speak" is called the **active talker**, and there may be only one active talker at any time. The act of "recognizing" or "giving the floor" to that device is called **addressing to talk**, and is performed by the active controller. The "message" delivered by the active talker is called a **data message**, and is the primary function of the HP-IB.

On a committee, those members present usually (?) listen, but this is not the case for the HP-IB. The active controller selects which devices are going to listen, and commands all other devices to ignore the present data message. A particular device is told to listen by being **addressed to listen** by the active controller. It is then an **active listener**. Devices are told to ignore a data message by being **unaddressed** with an **unlisten command** from the active controller.

¹ Institute of Electrical and Electronics Engineers 488-1978 standard for a general-purpose interfacing bus, commonly termed the GPIB.

² You should be aware that some devices claiming "IEEE-488 compatible" really are not fully compatible.

The concept of **unlisten** seems at first confusing, and one might wonder why do it? Imagine a slow note-taker on our committee, and a fast note-taker. Suppose also that the speaker is allowed to talk no faster than the slowest note-taker. This would guarantee that everybody gets the full set of notes and misses no important information. However, requiring all presentations to go at that slow pace certainly imposes a restriction on our committee. Now, if the chairman knows which presentations are not important for the slow note-taker, he can direct that person to put away his notes for those presentations. That way, the speaker and the fast note-taker can cover more items in less time. On the HP-IB, a similar situation may exist. A printer and a flexible disk (or "floppy") can both be listeners, but if all the data transfers must be slow enough for the printer to keep up, saving a program on the disk would take as long as listing the program on the printer. That would not be a very effective use of the speed of the disk drive, certainly. Instead, by unlistening the printer, the computer can save a program as fast as the disk can accept it.

Now that you have seen the general structure of the HP-IB, you are ready to learn about the primary function of the bus — that of data transfers.

Data Transfers on the HP-IB

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 listeners, and this transfer occurs 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 designated to listen.

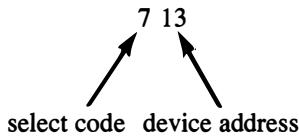
The technical term for the mechanism used to control the transfer rate is called **handshake**. Imagine, on the committee, that the speaker must pause every time a note-taker raises his hand. When the speaker presents one item, a slow note-taker can raise his hand to halt the speaker, while he (the note-taker) finishes jotting down that item. Then, when he is ready, the note-taker lowers his hand and the speaker continues with the next item.

A similar mechanism is used for the HP-IB. As you might expect, the HP-IB handshake is more complex in order to deal with many different devices and their different modes of operation. It is not necessary, however, to understand the details of this handshake in order to use the HP-IB.

How does one **address** a device? Remember, this is necessary in order to designate a device to be either a talker or a listener for a data transfer. The **select code** of the interface, 7 in the case of an HP-IB interface, is the first part of a **device selector**. This corresponds to the street number of an apartment house, say, 315 Exray Drive. The second part of a device selector is the **device address** (13, for our example) on the bus. This corresponds to the apartment number of the person in the apartment house. To address a letter to this person, it is necessary to give both the street address and the apartment number, like so:

315 Exray Drive]—street address
Apartment #27]—apartment number

The complete device selector looks like this:



Every device on the bus must have a different device address, or a great deal of confusion will result (as you might intuitively expect).

For a data transfer to occur, the active controller must:

1. Unlisten all devices (to ensure there are no "eavesdroppers").
2. Designate the talker (by **addressing** a device to talk).
3. Designate the listener(s) (by **addressing** the device(s) to listen).
4. Indicate to all devices involved that they may begin the data transfer.

Fortunately, this is normally done automatically by the computer. There are exceptions to this, but exceptions are topics for more advanced discussion and are not presented here.

The computer, as the active controller, is normally involved in any data transfers that occur. Take the case of the computer sending data to a printer. Suppose the message to be printed is "Now is the time for all good men." The statement

`OUTPUT 701;"Now is the time for all good men"`

does the following:

1. Unlistens all devices.
2. Designates the computer as talker (**My Talk Address** is sent).
3. Designates the printer (device #01) as listener.
4. Begins the data transfer.

The computer receives data as simply as it sends it. Assume that the computer is to take a reading from a voltmeter, which is reading + 1.075 volts. The device address of the voltmeter is 03, so the appropriate statement

`ENTER 703;V`

does the following:

1. Unlistens all devices.
2. Designates the computer as listener (**My Listen Address** is sent).
3. Designates the voltmeter (device #03) as talker.
4. Begins the data transfer.

Notice in both examples the computer only needs to be told what the address of the other device is — it already knows its own address.

If transferring data were all the HP-IB did, the simplicity of hooking up devices to the computer would be the only strong point of the bus. Its power goes far beyond that, however, as you will see in the next two sections.

Controlling the HP-IB

Just as the committee chairman must have certain measures of control over the committee, so must the system controller have methods of controlling the bus.

When a committee meeting has gotten out of hand for some reason (two or more members talking at once, or a member refusing to give up the floor) the chairman must call the meeting to order, usually by pounding a gavel. The possibility of a bus out of control exists, therefore, the system controller (and **only** the system controller) has the power to assert **Interface Clear**. This terminates all bus activity, **unaddresses** all talkers and listeners, and passes active controller responsibilities back to the system controller (if active control had been passed previously to some other device). Interface clear is not normally used in most bus operations, but has been provided to deal with any unusual situations that may occur.

Individual devices on the bus may be reset back to their power-on state by sending the **device clear** message. The active controller can send this message to any one device or all devices at once. **Device Clear** can be a quick means of re-programming a device back to its default (or power-on) parameters, and that device generally acts as if it had just been turned on.

Passing active control has been mentioned twice, and it refers to the ability of delegating active controller responsibilities to another device.

The analogy of designating an “acting chairman” in the absence of the committee chairman is appropriate. The system controller may need to direct the activities of another HP-IB system, or to perform calculations on data that has been gathered, or some other function that demands attention away from the bus. The technical term for passing active control is **Take Control**. It is a short message sent via the bus from the current active controller to the device receiving control which then assumes active control responsibilities. Obviously it doesn’t make sense to pass control to a device incapable of controlling bus activities, such as a printer.

The controller also has certain functions available that affect selected devices on the bus. It is sometimes desirable to have two or more instruments start their operations at the same time. Suppose two voltage readings have to be taken simultaneously at different points in a circuit under analysis. The two voltmeters involved have to be “triggered” at the same time, so the controller can send the **Group Execute Trigger** message to accomplish this.

Another type of control necessary in HP-IB systems is that of remote control of instruments. Normally, an instrument such as a voltmeter has switches, dials, and buttons on the front panel which the operator adjusts

to set range, polarity, and function. Requiring the presence of an operator to control these settings, however, would impose a severe restriction to the ability to automate a system. The **Remote Enable** message places an instrument under remote control so it can be set up by the active bus controller.

There is still a problem with the system described above. It is possible that a casual passer-by may come along and put an instrument back under local control of the front panel switches. This can be done with a switch generally provided on instruments called "return-to-local." If this happens, the computer could begin receiving erroneous readings, so a bus message is provided to "lock out" the return-to-local switch. The message is appropriately called **Local Lockout**. Its primary purpose is to prevent manual operation of bus instruments when they are under remote control.

Naturally, there needs to be some method of returning the instruments back to manual control. This is done by sending the **Go-to-Local** message. When a device receives the Go-To-Local message, any local lockout and remote messages in effect on that device are cancelled. The device then returns to local, front-panel control.

This completes the introduction to controller-directed activities, but one more aspect needs to be covered before this discussion of the HP-IB is complete. Devices on the bus need some mechanism of getting the attention of the controller when unusual circumstances develop. The next section deals with this mechanism.

Handling Requests for Service

When a device encounters a problem, or is ready with some other information for the controller, it needs to be able to signal that such a situation exists. It does this by a **Service Request**. An example of a need for a service request is that of a printer out of paper. The printer has to stop printing, but it needs to indicate to the controller that it has stopped, so that data will not be lost. The printer sends a service request to the controller, which must then find out

1. Who sent the request (if there are several devices on the bus)?
2. What is the problem?

The controller can determine who sent the service request by performing a **Parallel Poll**¹. This is like the committee chairman asking for those members with a problem to raise their hand. Once he determines which members have a problem, he can ask each in turn what their problem(s) are. On the HP-IB, this process of determining the problem is called a **Serial Poll** and, like the name implies, involves querying each device in turn as to its status. The information returned by each device is dependent upon the nature of the device: a printer may indicate out-of-paper, cover-off, tape low or whatever. A voltmeter may indicate overrange, illegal command and so on.

On one hand, the functions available with the HP-IB interface make possible a very sophisticated automated system. On the other hand, a very simple system is possible, because all the details of electrical, timing, mechanical, and data compatibilities have been already taken care of for you. The next section shows how an HP-IB system might be set up, and the sequence of operations typically used to operate it.

1. Note that response to Parallel Poll is device-dependent, so it may not be possible to use your device in this manner.

HP-IB Operations

Introduction

The HP-IB interface provides both simple operations and complex control functions for systems ranging from a single device plus controller to multiple devices with one or more controllers. This section is written for the user who understands the terms and structure of the HP-IB, but who needs information on how it is used. This is not an extensive treatment of statements and their syntax, but rather is intended to help the user understand the sequences of operations necessary to make an HP-IB system function. Use this section to design your program, then use the syntax reference in the back of this manual to code your program.

Turn-on and Check-out

The topic of device installation is covered in detail in the 82937A Interface Hardware and Theory manual, and is not covered here except in general terms.

Once the interface has been installed and the system devices have been connected via the HP-IB cables, a short routine should be run to check the status of the individual components. If you know the bus addresses of the devices in your system, a program similar to the one below can be used.

```

10 STATUS 7,0 ; A,B,C,D,E
20 PRINT "Interface card status = ";A,B,C,D,E
30 A=SPOLL(722)
40 B=SPOLL(703)
50 C=SPOLL(713)
60 PRINT "Device #22 status = ";A
70 PRINT "Device #03 status = ";B
80 PRINT "Device #13 status = ";C
90 END

```

In line 10, the status information of the 82937A interface itself is obtained, and that is printed in line 20. The status value as set at the factory should be 1, 0, 64, n, 53. Lines 30-50 obtain and print the **serial poll response byte** of the devices set to addresses 22, 03, and 13.

If you don't know, or aren't sure what the system device addresses are, the following program segment could be substituted:

```

10 S=7 @ ! VARIABLE S IS SELECT CODE
20 SET TIMEOUT S:500
30 ON TIMEOUT S GOTO 100
40 FOR I=0 TO 31
50 DISP "SPOOLL DEVICE # ";I
60 S1=SPOOLL(S*100+I)
70 PRINT "DEVICE ";I;" SPOOLL = ";S1
80 NEXT I
90 STOP
100 ABORTIO 7
110 PRINT "DEVICE ";I;" NOT PRESENT"
120 GOTO 80
130 END

```

Any device present should return some value for its serial poll response, and this will show up on the printout. Please note that the status information returned by each device is normally different. The meanings of the device status information printed by the above routine can be found in the operating manual for each device, and would typically be termed "Serial Poll Response Byte." Different devices may have this information in different areas of the manual, but the key to finding the information is Serial Poll Response.

Now that you have checked out the operation of your system (hopefully, all devices are operating properly), you can spend some time learning how to control it.

Controlling the Bus

All the operations in this section assume the HP85A and HP-IB interface are set up as system controller (as received from factory). Typically, the first operation necessary for HP-IB systems is to program all devices for **remote** operation via the bus. Most devices are capable of manual, front panel operation or of remote, bus operation. The remote mode of operation for a device is selected by setting the REN, or **Remote Enable** line, and addressing the device to listen. The REN line is set by the HP85A when power is turned on, the computer is RESET, or the REMOTE statement is executed. Addressing the device to listen, is performed by executing any statement which includes that device's listen address. To place devices 22 and 10 under remote control, the statement

```
10 REMOTE 722,710
```

could be used. To prevent any of the system devices from being returned to local operations from the front panel (by the Return-to-Local switch), a **local lockout**, or LLO message must be sent. The example cited above now looks like this:

```
10 REMOTE 713,722
20 LOCAL LOCKOUT 7
```

Now the system is set up for remote control, with the front panel controls disabled. The next step then is to program each device for the desired mode of operation. This is generally accomplished by means of simple OUTPUT statements directed to the device to be programmed, which is discussed further in the following section. For instance, suppose an HP 3455A DVM (digital voltmeter) device is to be set to the .1 volt DC range, continuous sample. The ASCII sequence of characters necessary to set it to this range is "F1R1T1." The following statement accomplishes this:

```
50 OUTPUT 722 ; "F1R1T1"
```

Output to device #22, select code 7,
the ASCII characters F1R1T1.

It is important to note that it is the **ASCII characters** sent by the OUTPUT statement which actually set the HP 3455A DVM to the .1V DC range. Selecting some other function merely involves changing the characters being sent to the device to the appropriate ones to select that function. The actual characters sent are "device-dependent," that is, their meaning depends upon the interpretation given them by the receiving device.¹

The example system consists of two devices, so both may need to be programmed, as shown below:

```
10 REMOTE 722,713
20 LOCAL LOCKOUT 7
30 ! Select .1vac range on device 22 ( 3455 voltmeter)
40 OUTPUT 722 ; "F1R1T1"
50 ! Select 1K hz resolution on device 13 (freq. counter)
60 OUTPUT 713 ; "PF4G3S1S3S5T"
```

Once the system devices are programmed for operation, it is possible to take readings from them. This is accomplished by means of the ENTER statement, which addresses the specified device and accepts data from it. For example, to take a voltage reading and a frequency reading, the following statements could be used:

```
10 REMOTE 713,722
20 LOCAL LOCKOUT 7
30 ! Select .1vac range on device 22 (3455A voltmeter)
40 OUTPUT 722 ; "F1R1T1"
50 ! Select 1K hz resolution on device 13 (5328A freq. counter)
60 OUTPUT 713 ; "PF4G3S1S3S5T"
70 ! Take reading from voltmeter and place into V
80 ENTER 722 ; V
90 ! Take reading from counter and place into F
100 ENTER 713 ; F
110 PRINT "Voltage = ";V
120 PRINT "Frequency = ";F
130 END
```

¹ For example, the word "vie" means "to compete" to an English-speaking person, but means "life" to a Frenchman.

The example as presented so far simply obtains the most recent readings taken by the voltmeter and counter. Suppose your particular application requires that the two readings occur **simultaneously**. An example of such an application is that of a voltage-to-frequency converter accepting a continuously varying voltage and generating an output frequency dependent upon the input voltage. To take both a voltage and a frequency reading simultaneously, the instruments can be triggered.

To accomplish this, it is necessary to change the operating modes of the devices from continual sample to triggered sample. The OUTPUT statements are changed accordingly, and a TRIGGER statement is inserted just before the ENTER statements. The TRIGGER and ENTER statements are put into a loop to provide a more complete picture of the converter's operation.

```

10 REMOTE 713,722
20 LOCAL LOCKOUT 7
30 ! Select 1vac range on device 22 (3455A voltmeter)
40 OUTPUT 722 ,>"F2R2T2T3"
50 ! Select 1K hz resolution on device 13 (5328A freq. counter)
60 OUTPUT 713 ,>"PF4G3R"
70 FOR I=1 TO 100
80 TRIGGER 713,722 @ RESUME 7 @ ! Drop ATN line.
90 ! Take reading from voltmeter and place into V
100 ENTER 722 ; V
110 ! Take reading from counter and place into F
120 ENTER 713 ; F
130 PRINT "Voltage =";V
140 PRINT "Frequency =";F
150 NEXT I
160 END

```

The ENTER and OUTPUT statements have been presented briefly as a means of programming instruments and taking readings. The next section deals with the more general topic of HP-IB data transfers.

HP-IB Data Transfers

Introduction

The data transfer¹, or the data message as it is known by HP-IB specialists, can be either the simplest or the most complex function available to the user. This section will show the simplest transfers first, then proceed to more complex transfers (this is the TRANSFER statement provided by the I/O rom).

¹ This is the general term for a data exchange — it does not relate specifically to TRANSFER, an advanced I/O capability.

Simple Output Operations

To send data from the HP-85A to a device on the HP-IB, the OUTPUT statement is generally used. It is also possible, however, to use the PRINT statement to send data to an HP-IB device. To do this requires executing a PRINTER IS statement to change the system printer over to the specified HP-IB device. The PRINTER IS statement can be inserted into the beginning of a standard BASIC program, then all the PRINT statements in the program will send their output to the specified device until a new PRINTER IS statement is executed. If only one HP-IB device is being sent data, this scheme works fine. For example,

```

10 ! The HP-IB Printer has a device address of 01
20 PRINTER IS 701
30 FOR I=0 TO 1 STEP .01
40 PRINT SIN(I)
50 NEXT I
60 END

```

This method becomes cumbersome when several devices are to be sent data, such as when programming a DVM (digital voltmeter) and a counter.

```

10 PRINTER IS 722
20 ! Address data output to DVM
30 PRINT "Programming sequence"
40 PRINTER IS 713
50 ! Address data output to Counter
60 PRINT "Programming sequence"
70 END

```

Obviously, this is not the method of choice. Compare the above program sequence to the following one, which does the same thing.

```

10 OUTPUT 722 ; "Programming sequence"
20 OUTPUT 713 ; "Programming sequence"

```

The OUTPUT statement data can be formatted with the OUTPUT USING statement, as shown in Section 1. The HP-IB version of this statement looks like this:

```
OUTPUT 722 USING "nA"; "prog seq"
```

In general, a simple HP-IB output looks very much like any other output, except that additional information in the form of the HP-IB device address must be specified with the select code. If you need more information about using the OUTPUT statement, you can find it in Part 1 of this manual. More advanced techniques of outputting data are covered in the Advanced HP-IB TRANSFERS section.

Simple Input Operations

The ENTER statement is the simplest method of inputting data from an HP-IB device to the HP-85A. To accomplish a data input from a device such as an HP 59309 Clock, the following statement can be used:

```
ENTER 716;A$
```

In this case, the statement is terminated by a line-feed sent by the clock as the last character. Other devices may send EOI (End or Identify) with the last character to terminate a data input. If this is the case for your device, the ENTER statement could be modified as shown below for device #06:

```
ENTER 706 USING "%,K";A$
```

The “%” image specifier allows the EOI message¹ sent by the device to terminate the ENTER statement. However, if a line-feed character is part of the data being sent, the line-feed will prematurely terminate the ENTER statement shown above. The following statement waits for the EOI message before terminating (does not terminate on line-feed):

```
ENTER 706 USING "#%,K";A$
```

If you need more information regarding input data formatting, please refer to Part 1, where the ENTER statement is covered in greater detail. For information pertaining to more flexible data input TRANSFERS, or data input to the HP-85A when it is a non-controller, refer to the following section on Advanced I/O Operations.

Advanced I/O Operations

Introduction

If you are reading this section, it is probably because you have a problem that can't be solved with a simple OUTPUT or ENTER statement. Basically, this section deals with four types of transfers:

1. Data transfers involving a non-controller HP-85A.
2. ENTERs and OUTPUTs that include multiple listeners.
3. Data transfers initiated by the HP-85A which do not include the HP-85A.
4. User-specified TRANSFER types, including interrupt and fast-handshake.

¹ The EOI message here is the END message, sent by some devices with the last character of a data exchange. It indicates the end of data.

Non-controller Addressing

This type of I/O is used when either the HP-85A has passed control to another device and is no longer active controller, or when the HP-IB interface has been set to non-system controller (set by a switch in the interface — see the 82937 Hardware and Theory manual for more detailed information) and has not received active control from the current active controller.

When the HP-85A (or **any** device, for that matter) is not the active controller, it must wait to be addressed to talk or listen before it may output or input data on the bus. There are three means of accomplishing this waiting to be addressed. The first is automatic and would be the method of choice for a simple system. Simply specify the interface select code with no device numbers in the ENTER or OUTPUT statement, and the HP-85A will wait to be addressed before transferring the data. The following statements show the conditions necessary to begin a transfer:

ENTER 7:A\$ } Wait to be addressed to **listen** by the controller, then read data into
or TRANSFER 7 TO A\$... } A\$.

OUTPUT 7:A\$ } Wait to be addressed to **talk** by the controller, then send A\$.
or TRANSFER A\$ TO 7... }

A second method of waiting to be addressed is to periodically check the interface status for the proper condition. The computer could be accomplishing some useful computation during the period of time it is not addressed, then when a status check does finally indicate that the HP85A has been addressed, it could transfer the requested data. For example, the following program increments and displays a counter then checks to see if it is addressed. If not, the program loops back to the increment and display statements. Otherwise, the value of the counter is transferred to the HP-IB.

```

10 ! Lines 50,60 check for addressed to talk
20 I=0
30 I=I+1
40 DISP I
50 STATUS 7,5 : S
60 IF BIT(S,5)=0 THEN GOTO 30 ! Is "TA" bit set?
70 OUTPUT 7 : I
80 GOTO 20
90 END

```

(See ‘‘HP-IB for the Specialist’’ section for a complete explanation of the Status Registers.)

The sequence shown above to read status and check for being addressed to talk is identical for an ENTER operation. The difference is in which status bit is to be checked, and is shown in the following statements:

```

10 STATUS 7,5 : S
20 IF BIT(S,3)=0 THEN DISP "Not addressed to listen"

```

Bit 3 of status register #5 indicates addressed to listen when it is set to a 1. Bit 5 of status register #5 indicates addressed to talk when it is set to a 1.

The third method of waiting to be addressed is to enable the interface to **interrupt** the HP-85A when it is addressed. This relieves the programmer of the necessity of designing his program around a periodic status check, and it allows the program to perform useful computation until the computer is addressed to talk or listen. An example serves to demonstrate this third method:

```

10 ! Set up "addressed-to-listen" interrupt and subsequent end
   -of-line branch
20 ENABLE INTR 7:64
30 ! Line 40 enables end-of-line branch service routine
40 ON INTR 7 GOSUB 120
50 I=0
60 I=I+1
70 DISP I
80 GOTO 60
90 ! Service routine for end-of-line branch
100 ! Always perform interrupt cause check
110 STATUS 7,1 ; S
120 ENTER 7 ; A$
130 PRINT "Data received was ";A$
140 ! Re-enable "addressed-to-listen" interrupt
150 ENABLE INTR 7:64
160 RETURN
170 END

```

Line 20 sets up an interrupt condition of listener active (or addressed-to-listen). Line 40 directs the program to line 100 when the listener active interrupt occurs. Lines 60-80 represent the computational portion of the program, purposefully kept simple here. Line 100 starts the end-of-line-branch service routine, which is executed when the HP85A becomes addressed to listen. The ENTER statement specifies only the interface select code (no device numbers!) which means “wait until addressed to listen before inputting data.” Line 130 re-enables the interface for an addressed-to-listen interrupt. Output works the same way, except a different value (16) is used for the ENABLE INTR mask, and an OUTPUT 7;⟨value⟩ statement is substituted for the ENTER statement. See ‘‘HP-IB for the Specialist’’ section for a complete explanation of the Control Registers, and interface interrupts as discussed under Status Register 1 of the same section.

Suppose our non-controller HP-85A is to both send and receive data? The basic sequence of operations remains the same but an additional status check is necessary to determine which operation to perform. The following program illustrates this status check. Line 100 obtains the status value and lines 110 and 120 determine if the cause of an end-of-line branch was either for being addressed to talk or for being addressed to listen.

```

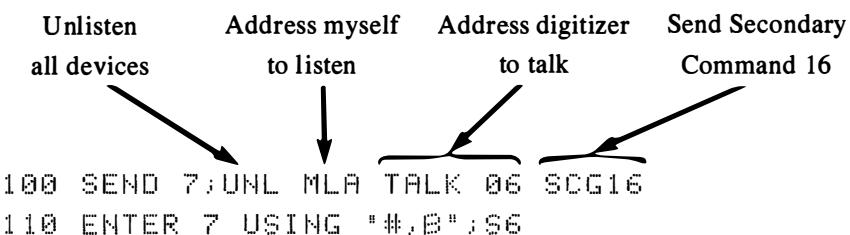
10 ! Enable talker active and listener active interrupts
20 ENABLE INTR 7:(64+16)
30 ON INTR 7 GOSUB 100
40 I=0
50 I=I+1
60 DISP I
70 GOTO 50
80 ! Service routine follows
90 ! Check status to determine the cause of the end-of-line branch
100 STATUS 7,1 : S ! Read interrupt cause
110 IF BIT(S,3)=1 THEN GOTO 150
120 IF BIT(S,5)=1 THEN GOTO 180
130 PRINT "Error in status code"
140 RETURN
150 ! Service for "addressed to talk"
160 OUTPUT 7, I
170 GOTO 210
180 ! Service for "addressed to listen"
190 ENTER 7 : A$
200 PRINT "Received data of ";A$
210 ENABLE INTR 7:(64+16)
220 RETURN

```

Custom Bus Sequences

Occassionally it may become necessary to send a custom bus sequence to a device under development or to one which requires a sequence different from those normally sent by the HP-85. The SEND statement makes such custom operations possible, and even gives a performance increase in the bargain. However, the price you pay is that you have to specify every character of the sequence yourself: its no longer done automatically.

As one example of a custom bus sequence, the following statements send a Secondary Command to a 9874 digitizer (device 06) on select code 7. The Secondary Command of 16 causes the digitizer to output its status which is read as one byte by the following ENTER statement (notice that ENTER does **not** specify addressing).



Other operations that could be performed include Parallel Poll Configure and Parallel Poll Unconfigure. You can utilize the tables included in "HP-IB for the Specialist" to determine the necessary codes to accomplish your task.

Multiple Listener Transfers

When more than one device needs to receive data being transferred, these additional devices need to be included in the **listen address group**. This is accomplished in different ways for ENTER and for OUTPUT transfers as described in the following paragraphs.

Both the OUTPUT and the output TRANSFER statements have a built-in provision for multiple listeners, as shown in this example: Assume that a string of characters (B1\$) is to be sent to a printer (device 04) and to a non-controller HP-85A (device 20) on the HP-IB. The OUTPUT statement would look like this:

```
OUTPUT 704, 720; B1$
```

The TRANSFER statement would look like this:

```
TRANSFER B1$ TO 704, 720 INTR
```

The only restriction to adding listeners to an output or TRANSFER is that they all be on the same select code (in this case 7).

Specifying other listeners for an ENTER data transfers is also possible, but it is not as straightforward as for the OUTPUT transfer. The ENTER statement specifies only the data source (ENTER 713, for example) and the HP-IB does not allow multiple talkers. The bus needs first to be configured for the transfer, then an ENTER statement can be executed with only the interface select code specified so the bus will not be reconfigured. For example, device 07 (a voltmeter) is to be the talker, devices 13 (a printer), our HP-85A (use MLA which means My Listen Address), and 04 (a tape cartridge unit) are to be listeners. The following sequence unlistens all previous listeners, sends the new talk and listen addresses, and finally enters the data:

```
10 SEND 7 ; UNL TALK 7 LISTEN 13,4 MLA
20 ENTER 7 ; V
```

A similar sequence could be used for an input TRANSFER, in this case an interrupt TRANSFER into T\$:

```
10 DIM T$[88]
20 IOBUFFER T$
30 SEND 7 ; UNL TALK 7 LISTEN 13,4 MLA
40 TRANSFER 7 TO T$ INTR
```

Another type of multiple listener transfer is one in which the HP-85A is neither sending or receiving the data: the transfer occurs between other devices on the bus. For example, to send a data file from device 11 to devices 23, 04, and 07, the following statements would be used:

```

10 SEND 7;UNL TALK 11 LISTEN 23,04,07
20 RESUME 7

```

The transfer begins automatically when the ATN (attention) line goes false (by executing RESUME), indicating that all commands have been sent. In this case, since the HP-85A has not addressed itself for the transfer, the operation does not have to wait for the HP-85A to execute an ENTER or OUTPUT statement in order to begin. Obviously, it is not necessary to specify more than one listener — the same statement could be used to start a transfer between one talker and one listener as well.

There is a problem with the above example: how does the HP-85 know when the data transfer is complete? The simplest manner is to address the HP-85 as a listener in line 10, execute an input interrupt TRANSFER to monitor the bus transfer, and wait for an end-of-line branch upon termination of the TRANSFER. The example below shows how this might be done.

```

5 DIM T$[1000]
6 IOBUFFER T$ @ ON EOT 7 GOSUB 1000
10 SEND 7 ; UNL TALK 11 LISTEN 23,4,7 MLA
20 ! For this example, EOI message ends the transfer.
30 TRANSFER 7 TO T$ INTR ; EOI
40 RESUME 7
50 ! Note that RESUME causes ATN to drop which starts the transfer.
r.

```

Alternate Transfer Types

As has been shown in examples cited in the previous sections, it is possible to transfer data under interrupt (which allows TRANSFERS and program execution to occur at the same time) or by fast handshake (maximum data transfer rate possible). The TRANSFER statement is used to specify which type (INTR or FHS) is to be used. This statement has been discussed in Part 3, in the Advanced TRANSFERS section, and details on its use are available there. The following program demonstrates how an HP-IB TRANSFER can be executed:

```

10 ! Transfer data under interrupt from device 04 to buffer T$
20 ! A line-feed terminates the transfer.
30 DIM T$[880]
40 I=0
50 IOBUFFER T$
60 ON EOT 7 GOSUB 120
70 ! Now configure the transfer
80 TRANSFER 704 TO T$ INTR ; DELIM 10 ! Line-feed terminates.
90 I=I+1

```

```

100 DISP "COUNTING",I
110 GOTO 90
120 PRINT "Received data",T$
130 ! Set up another transfer
140 IOBUFFER T$ ! Clear the buffer
150 TRANSFER 704 TO T$ INTR : DELIM 10 @ RETURN
160 END

```

Handling Service Requests

Introduction

This section deals with asynchronous requests for service. The cause for a service request is device-dependent, that is, different devices have different reasons for requesting service. For instance, a printer may request service because it has just run out of paper, or a digitizer may request service because an operator error has occurred. At any rate, once a request has been received, two actions must be taken:

1. Locating the device which requested service, and
2. Determining the reason for the device's request.

The following sections show you how to do this.

Sensing Service Requests

The HP-IB interface allows you to check the interface status to see if a service request (SRQ) has been received. This status check is performed by the following statement:

```
STATUS 7,1:S
```

Bit 3 of variable S now indicates if an SRQ has been received, and the program can make a decision of how to handle an SRQ if one has been received. (See "HP-IB for the Specialist" section for a complete description of the Status Registers.)

An alternative to the periodic status check is to program for an end-of-line branch condition of SRQ. The following sequence enables an end-of-line service routine, then establishes an interrupt mask for SRQ and enables interrupts from the interface:

```

10 ON INTR 7 GO8SUB 200
20 ! The value 8 sets bit 3 of the control register (SRQ)
30 ENABLE INTR 7:8

```

The program will execute a subroutine at line 200 whenever an SRQ is received. What to do at line 200 is the subject of the following sections.

Determining the Problem

It is the purpose of the HP-IB serial poll to provide the active controller with specific, device-dependent information about the device being polled. The bits of the device's serial poll response byte can have any meaning assigned to them and are generally used to indicate some problem or special condition within the device. Bit 6 however, is reserved to indicate that the device is currently requesting service.

Referring back to the example below, line 300 is executed when the program has determined that device 15 requested service. The first operation to perform in servicing device 15 is to obtain its status. Assume that the bits are assigned the following meanings:

- Bit 0: Out of paper.
- Bit 1: Cover off.
- Bit 2: Parameter out of range.
- Bit 3: Improper escape sequence.
- Bit 4: Always 0.
- Bit 5: Always 0.
- Bit 6: SRQ Active.
- Bit 7: Always 0.

Device 15's service routine might look like the following:

```

300 ! Service for device 15
310 S=SPOLL(715)
320 IF BIT(S,6)=1 THEN GOTO 350
330 ! Else SRQ was not this device.
340 RETURN
350 IF BIT(S,0)<>1 THEN GOTO 390
360 PRINT "HP-IB printer out of Paper"
370 PAUSE
380 RETURN
390 IF NOT BIT(S,1) THEN GOTO 420
400 ! And ETCetera
410 ! Re-enable interrupts if necessary.

```

In a larger system, the program would just perform sequential serial polls to determine which device requested service, and to determine that device's current status, as shown briefly below:

```

100 S=SPOLL(715)
110 IF NOT BIT(S,6) THEN GOTO 200
120 ! Lines 120-190 service device 15
200 S=SPOLL(723)
210 IF NOT BIT(S,6) THEN GOTO 300

```

Non-controller Operations

When the HP-85A is not the active controller, certain constraints are forced on the programmer to avoid violating bus protocols. That is, certain operations can only be performed by the system controller, others only by an active controller. A non-controller is allowed to only talk, listen, and request service.

Passing Control

The HP-85, as active controller, can pass controller responsibilities to another device by executing the PASS CONTROL statement. This allows the HP-85 to direct its attention to activities other than bus control, such as would be the case when it must direct and coordinate the activities of two or more HP-IB systems, or when it must dedicate all processing power to a high-speed block data transfer to a host computer. The following program segment passes control to device 20, another HP-85, which is currently a non-controller:

```
680 PASS CONTROL 720.
```

It may be necessary for our HP-85 to later assume control of the bus, in which case some provision must be made to determine that control has been passed to our HP-85. This is discussed in the following section.

Receiving Control

Although a simple check of interface status can tell the program that the HP-85 has received active control, it is more advantageous to use the end-of-line branch facility to do this automatically. The following program segment passes control to device 20, but also makes provision for an end-of-line branch to line 700 when active control is later received:

```
650 ! Pass control subroutine
660 ON INTR 7 GOSUB 710
670 ! Enable EOL branch on receiving active control
680 ENABLE INTR 7;32
690 PASS CONTROL 720
700 RETURN
710 ! Routine to accept active control goes here.
720 STATUS 7,1 ; S
730 IF BIT(S,5)<>1 THEN GOTO 760 ! Bit 5 is active control
740 !
750 RETURN
.
```

This end-of-line branch capability can serve to also indicate that the HP-85 has been addressed to talk or to listen by the active controller. This is discussed in the next section.

Non-controller Responses

In an earlier discussion it was shown how the HP-85 sends and receives data as a non-controller — only the interface select code is specified. No device addresses are allowed! The HP-85 can use the end-of-line branch facility to determine when it has been addressed to talk or listen, and this is shown in the following example. This example is an extension of the previous one, in that control is passed and interrupts are enabled for active control, addressed to talk, and addressed to listen.

```

:
:
650 ! Pass control subroutine
660 ON INTR 7 GOTO 700
670 PASS CONTROL 720
680 ENABLE INTR 7;(16+32+64) @ RETURN
690 !
700 ! Non-controller service routine
710 STATUS 7,1 ; S
720 IF BIT(S,5)=1 THEN GOTO 810
730 IF BIT(S,4)=1 THEN GOTO 780
740 IF NOT BIT(S,6) THEN PRINT "ERROR" @ STOP
750 ! Bit 6 = 1 so input data
760 ENTER 7 ; A$
770 GOTO 860
780 ! Bit 4 = 1 so output data
790 OUTPUT 7 ; X$
800 GOTO 860
810 ! Bit 5 = 1 so assume control
820 ! Do necessary controller things, then pass control again
:
:
850 PASS CONTROL 720
860 ENABLE INTR 7;(16+32+64) @ RETURN

```

Additional information on HP-IB interrupts can be found under Control Registers and Status Registers in the “HP-IB for the Specialist” section.

Sending Service Requests

Some condition in the HP-85 may require the attention of the active controller, so the HP-85 needs to be able to send a service request to the controller. The REQUEST statement allows the HP-85 to assert SRQ and to send a serial poll response byte to the controller when it finally conducts a serial poll on the HP-85. Bit 6 of this byte should be set to a 1 to indicate that the HP-85 indeed requested service, but the other bits may indicate anything you (as the system designer) deem important. So, to request service of device 720 (who we just passed control to) and to indicate that the HP-85 is ready with data, our example will set bits 0 and 6. (Bit 0 will mean ready-with-data in our example system.)

```
900 REQUEST 7;1 + 64
```

Handling Interface Problems

This section describes some of the tools available to you for avoiding and dealing with problems that may arise when using an HP-IB interface. This is not meant to say that you should **expect** problems, but good programming practice is to anticipate problems and deal with them in advance.

Avoiding Bus Hang-ups

Generally, when an HP-IB device develops a problem, either it holds up the data transfer that it is involved in, or it sends an SRQ (service request) to the controller, or it does both. We have seen how the controller might handle the service request (see the section called "Handling Service Requests"), but suppose the device stops handshaking in the middle of a data transfer and at the same time it sends an SRQ? This event presents a problem to the HP-85 because it cannot perform an end-of-line branch to service the SRQ. Why this is so becomes apparent when you consider the nature of an end-of-line branch: it does not occur until the current BASIC program line has been executed, and if an ENTER or OUTPUT still has not completed (the device halted the transfer, remember) the HP-85 is "hung". It cannot complete the transfer, and it cannot execute an end-of-line branch until the operation completes. The HP-85 **can** recover from an unsuccessful transfer however, by using the SET TIMEOUT capability provided for such a situation.

The following example shows the sequence of operations necessary to provide a program with the capability of recovering from a bus hang-up:

```

10 DIM S(6)
20 ! First set up the timeout service routine
30 ON TIMEOUT 7 GOSUB 170
40 ! Then set a handshake time limit (1200 msec here)
50 SET TIMEOUT 7:1200
60 ! Now program the HP3455 for triggered readiness.
70 OUTPUT 722 ; "FIR1T2T3"
80 ! Now trigger and read the DVM
90 TRIGGER 722
100 ENTER 722 ; X
110 DISP X;"Volts"
120 GOTO 90
130 !
140 ! Begin timeout service routine
150 ! First step is to check the interface status
160 ON TIMEOUT 7 GOTO 270 ! Make provision for bad interface.
170 FOR I=0 TO 6
180 STATUS 7,I ; S(I)
190 PRINT "Status byte #";I;" =";S(I)
200 NEXT I
210 ! The next step is to decide on the course of action
220 ! based upon the status info just obtained...
230 !
240 !
250 GOTO 290 ! End of timeout service.
260 !
270 PRINT "Interface failure"
280 RESET ? ! Try to help it.
290 ON TIMEOUT 7 GOSUB 170 ! Restore original service routine.
300 RETURN

```

It is up to the programmer to determine what actions the program is to take based upon the results of the status information obtained. In most instances it will be necessary to RESET the interface, avoid transfers with the device causing the hang-up, and signal to the system operator that a malfunction has occurred in that device. These actions are described in the next section.

Dealing with Problems

You are probably reading this section either because your HP-IB system is not working or because you need to make provisions in your program to deal with problems. This section first deals with the case of a non-working system, since it is then that you need this information in a hurry.

If the HP-IB system appears to be locked up (everything is running but nothing is happening) you can RESET the HP-85 to regain control of the computer. The exception to this is when a Fast-handshake TRANSFER is hung. The only possible recourse to such a case is to assert IFC — by whatever means¹ — or to power down the HP-85 and then power it back up.

You may now wish to perform a serial poll of the device in question to determine its condition. This, however, requires that you know how to obtain the correct information from the device. It may be sufficient to simply turn the device off then back on to get it back into operation.

The following section shows the alternatives (and reasons for each) available to the programmer writing a routine to handle error conditions encountered during HP-IB operations.

Action	Reason/Result
FOR I = 1 TO 6 STATUS 7,I;S(I) NEXT I	Obtain the current state of the interface to determine conditions and possible causes.
S=SPOLL (<each device>)	Obtain current status of devices in the system. (Maybe a printer is out of paper.)
CLEAR(<selected device>)	Return the desired device to its particular device-dependent "clear" state. (Like a reset.)
CLEAR 7	Return all devices to their device-dependent states.
ABORTIO 7	Terminates all bus activity and returns control to the HP-85 as system controller. This would normally be used only by an HP-85 that is set to system controller (the original factory setting is that of system controller).

It is a good practice to preserve a hard copy record of the interface and device status as the information is obtained. Also printed should be a record of any actions taken to rectify the situation. This information is vital for analyzing the cause of a system failure, and can be used by the system operator to make necessary adjustments or corrections (such as loading paper in the printer!).

¹ A bus analyzer could be used to assert IFC.

HP-IB for the Specialist

This section is intended for use by the specialist who is familiar with the IEE-488-1978 interface standard and who requires detailed information about programming the 82937A Interface. It is merely a description of the tools available for use by the expert; it is not a discussion of how to use them.

HP-IB I/O Statements

ABORTIO:	If the Interface is System Controller, it pulses Interface Clear (IFC) and sets Remote-Enable (REN) true. If the Interface is not System Controller but is Controller Active, it sources its Talk Address which "untalks" any other Talkers on the bus. If the Interface is neither System Controller nor Controller Active, it leaves the interface bus in the present state and becomes ready for the next I/O operation (see HALT).
ASSERT:	ASSERT does an immediate write to Control Register 2, the HP-IB control lines, regardless of whether the Interface is "busy" with I/O. Writing to Register 2 via CONTROL is identical except that CONTROL execution waits until the Interface is no longer busy. The user must be aware of the Interface logic to correctly write to Register 2 or the Interface may become inoperative and have to be reset.
CLEAR:	Must be Controller Active; if bus addressing is specified, does the addressing and sends the Selected Device Clear command. If no addressing is specified, sends the universal Device Clear command. Upon completion of CLEAR, the ATN uni-line message remains true; it can be set false by doing a RESUME.
CONTROL:	Writes to Control Registers within the interface. Error 111 is generated if writing occurs to a non-existent register. The valid Control Registers are 0-3 and 16-23.
ENABLE INTR:	Same as Write Control to Register 1, provides the End-of-Line Interrupt Enable Mask.
ENTER:	If addressing is specified, the Interface must be Controller Active; if no addressing is specified, the Interface begins inputting when it becomes Listener Active. The examples below show use of the ENTER statement with and without addressing.
	ENTER 305 ; A\$ Must be Controller Active, performs bus addressing prior to entering into the string variable A\$.
	ENTER 3 ; A\$ If not Controller Active, the Interface waits until it is addressed to Listen. If the Interface is Controller Active, it must have already addressed itself to Listen or error 116 is generated.

HALT:	Causes the Interface to "break away" from its current I/O operation and become ready for the next I/O operation. HALT leaves the bus signals unaffected. Note that ABORTIO has the same affect as HALT if the Interface is neither System Controller nor Controller Active.
LOCAL:	If no addressing is specified, then the Interface must be System Controller and REN is set false. If addressing is specified, the Interface must be Controller Active, then addressing is performed and the Go to Local (GTL) command is sent. After GTL is sent, the ATN uni-line message remains true; it can be set false by doing a RESUME.
LOCAL LOCKOUT:	Must be Controller Active, causes the Interface to source the Local Lockout (LLO) command. After LLO is sent, the ATN uni-line message remains true; it can be set false by doing a RESUME.
OUTPUT:	If addressing is specified, the Interface must be Controller Active; if no addressing is specified, the Interface begins outputting when it becomes Talker Active. The examples below show use of the OUTPUT statement with and without addressing.
	OUTPUT 305,306;A\$ Must be Controller Active, performs bus addressing prior to outputting A\$.
	OUTPUT 3 ; A\$ If not Controller Active, the Interface waits until it is addressed to Talk. If the Interface is Controller Active, it must have already addressed itself to Talk or error 115 is generated.
PASS CONTROL:	Must be Controller Active. Addressing, if specified, precedes the Take Control command. The Active controller can pass control to any device capable of controlling the bus (including itself).
PPOLL:	Must be Controller Active, returns a Parallel Poll response byte from the Interface.
REMOTE:	Must be System Controller. Sets REN and sources addresses, if specified, to put bus devices in the remote state. If addresses are specified, ATN remains true; RESUME can be used to set ATN false.
REQUEST:	Must be non-Controller Active, sets SRQ on the interface if bit 6 (based on bits 0-7) is equal to one in the specified byte. This byte is provided to the Controller Active device upon being Serial Polled (which causes SRQ to be cleared). SRQ can also be cleared by providing a REQUEST byte with bit 6 = 0.
RESET:	Provides a hardware reset to the Interface, returning it unconditionally to its power-on state. This causes the Interface to perform self-test. If self-test fails, Error 110 is displayed; if self-test passes, no display occurs.

RESUME:	Must be Controller Active, sets ATN = 0. This is useful, for example, after doing a CLEAR, which leaves ATN = 1. Note that normally RESUME is not required since data I/O statements (OUTPUT, ENTER, TRANSFER, etc.) ensure that ATN is set to 0.
SEND:	Used to send commands (in which case the interface must be Controller Active) or data (in which case the interface must be Talker Active). While sending data, ATN is false. While sending commands, ATN is true. Upon completion, ATN can be set false by using the RESUME instruction.
SPOLL:	Must be Controller Active, used to conduct a Serial Poll of a device on the bus to obtain its Status Byte. The Status Byte indicates whether the device is requesting service and provides additional device-dependent information.
STATUS:	Used to read Status Registers in the Interface. STATUS is immediately executed regardless of the Interface state (unless IFC is true). The valid Status Registers are 0-6; attempting to read registers outside of this range generates Error 111.
TRANSFER:	Used for Interrupt or Fast Handshake I/O. TRANSFER relies on the use of declared BUFFERS as discussed in the I/O ROM ERS; refer to this document for information on buffer control; i.e. Fill and Empty pointers, etc. Associated with TRANSFER are three termination specifiers: (1) A delimiter numeric expression DELIM, (2) a termination count (COUNT) and (3) an EOI enable specifier (EOI). Shown below are the termination specifiers which are permitted for each type of TRANSFER.

TRANSFER Type	DELIM	COUNT	EOI
TRANSFER (in) FHS	Not applicable	Used if supplied, otherwise Transfer proceeds from Fill Pointer to End of Buffer.	Enables EOI termination.
TRANSFER (out) FHS	Not applicable	Not applicable — Transfer proceeds from Empty Pointer to Fill Pointer.	Not applicable — EOL sequence sent.
TRANSFER (in) INTR	Yes	Used if supplied, otherwise input terminates with DELIM, with EOI or when Fill Pointer is at End of Buffer.	Enables EOI termination.
TRANSFER (out) INTR	Not applicable	Not applicable — Transfer proceeds from Empty Pointer to the Buffer Fill Pointer.	Not applicable — EOL sequence sent.

TRIGGER:	Must be Controller Active, does addressing (if specified) followed by the Group Execute Trigger command. Upon completion, ATN remains true; it can be set false by the RESUME instruction.
-----------------	--

Typical HP-IB Output Sequence

The table below illustrates the command bus sequence produced by the following statement (commands are in black, data bytes are in color):

```
10 OUTPUT 705;"HEWLETT-PACKARD INTERFACE BUS"
```

Command	Binary	Decimal	Mnemonics
U	01010101	85	ATN,MTA
?	00111111	63	UNL
%	00100101	37	LAG,ATN
H	01001000	72	(DATA)
E	01000101	69	(DATA)
W	01010111	87	(DATA)
•	•	•	•
•	•	•	•
•	•	•	•
•	•	•	•
•	•	•	•
•	•	•	•
B	01000010	66	(DATA)
U	01010101	85	(DATA)
S	01010011	83	(DATA)
	00001101	13	(CR)
	00001010	10	(LF)

Typical HP-IB Enter Sequence

The table below illustrates the command bus sequence produced by the following statement (commands are in black, data bytes are in color):

```
10 ENTER 705 ;A$
```

Command	Binary	Decimal	Mnemonics
?	00111111	63	ATN • UNL
5	00110101	53	MLA
E	01000101	69	TAG,ATN
H	01001000	72	(DATA)
E	01000101	69	(DATA)
W	01010111	87	(DATA)
•	•	•	•
•	•	•	•
•	•	•	•
B	01000010	66	(DATA)
U	01010101	85	(DATA)
S	01010011	83	(DATA)
	00001101	13	(CR)
	00001010	10	(LF)

HP-IB Control Registers

These registers are set by executing the CONTROL statement. For example, to set the value of control register 0 to a 1, execute CONTROL7,0;1. Execution of the CONTROL statement is delayed until any operation currently in progress has been completed. The ASSERT statement, however, provides immediate access to CR2, the HP-IB Control Lines Register, regardless of any operations in progress.

CAUTION

Control Registers 2 and 3 provide direct access to the HP-IB control and data lines. They must be used with care, and used only by persons aware of HP-IB protocols! It is possible to cause bus malfunctions or device damage by improper use of these registers.

A complete control register table is given, followed by explanations of the individual registers.

HP-IB Control Registers

Register Number	Bit Number								Default Value	Register Function
	7	6	5	4	3	2	1	0		
CR0	X	X	X	X	Odd	Even	Always One	Always Zero	0	Parity Control
CR1	IFC	LA	CA	TA	SRQ	DCL or SDC	GET	SCG	0	Interrupt Mask
CR2	X	REN	SRQ	ATN	EOI	DAV	NDAC	NRFD	Not Applicable	HP-IB Control Lines
CR3	DI08	DI07	DI06	DI05	DI04	DI03	DI02	DI01	Not Applicable	HP-IB Data Lines

CR0: Parity Control

This register controls the parity mode for input and output data: no parity is used for commands. The right-most non-zero bit controls the parity selection, with an all-zero value (CR0 = 0) meaning no parity (default at power-on). For example, CONTROL 7,0;4 selects even parity. CONTROL 7,0;6 selects always one parity (right-most bit set is Bit 1 = always one).

CR1: Interrupt Mask

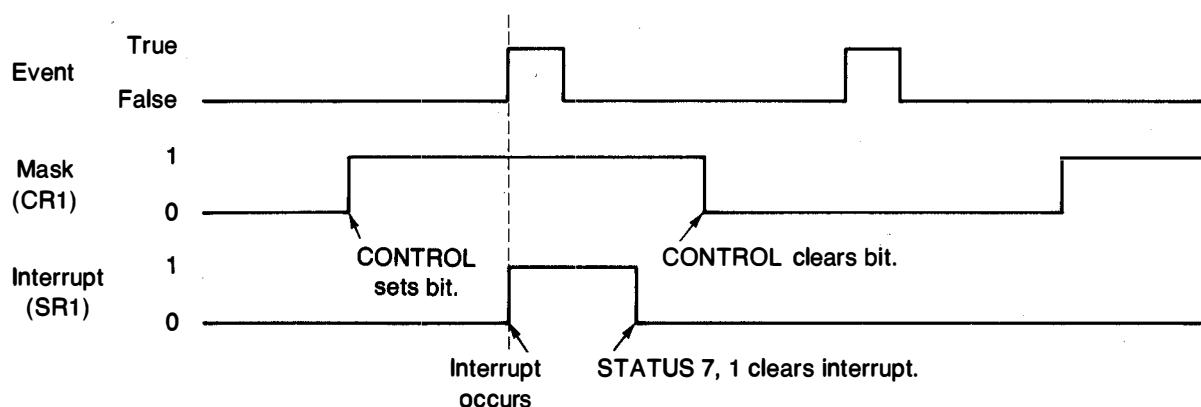
A bit when set enables the corresponding interrupt condition to cause an end-of-line branch.

- Bit 0 when set enables interrupt when a secondary command is received immediately following receipt of MTA or MLA. The value of the secondary command that was received is stored in SR6, the secondary command register. This is an event-initiated interrupt.
- Bit 1 when set enables interrupt when a GET (Group Execute Trigger) is received while addressed to listen (LA). This is an event-initiated interrupt.
- Bit 2 when set enables interrupt either when DCL (Device Clear) is received, or when SDC (Selected Device Clear) is received while addressed to listen (LA). This is an event-initiated interrupt.
- Bit 3 when set enables interrupt when SRQ (Service Request) is True.

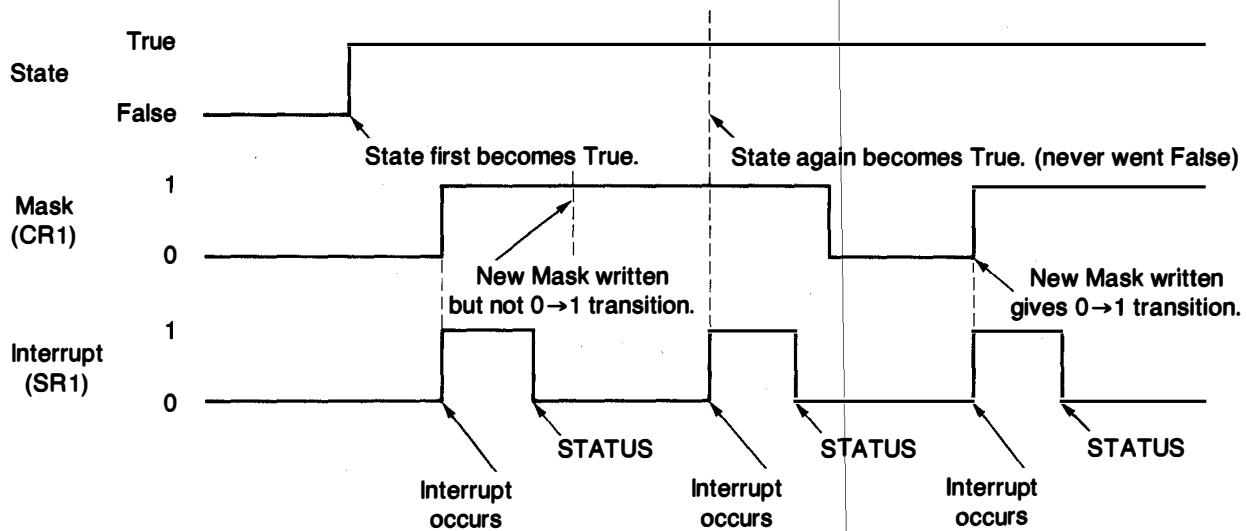
- Bit 4 when set enables interrupt when TA (talker active:addressed to talk) becomes true. If TA is **already** true, then a 0-to-1 transition of Bit 4 of CR1 causes an interrupt. This is a state-initiated interrupt.
- Bit 5 when set enables interrupt when CA (controller active) becomes true (by receiving control). If CA is **already** true, then a 0-to-1 transition of Bit 5 of CR1 causes an interrupt. This is a state-initiated interrupt.
- Bit 6 when set enables interrupt when LA (listener active:addressed-to-listen) becomes true. If LA is **already** true, then a 0-to-1 transition of Bit 6 of CR1 causes an interrupt. This is a state-initiated interrupt.
- Bit 7 when set enables interrupt when an IFC (Interface Clear) occurs. An **externally** caused IFC can cause an interrupt even when the interface is the system controller. This is an event-initiated interrupt.

The following diagrams illustrate the interface response to state-initiated, event-initiated, and SRQ interrupts. The interrupt-cause status registers (SR1) bits are set when an interrupt occurs and cleared when the status of SR1 is read.

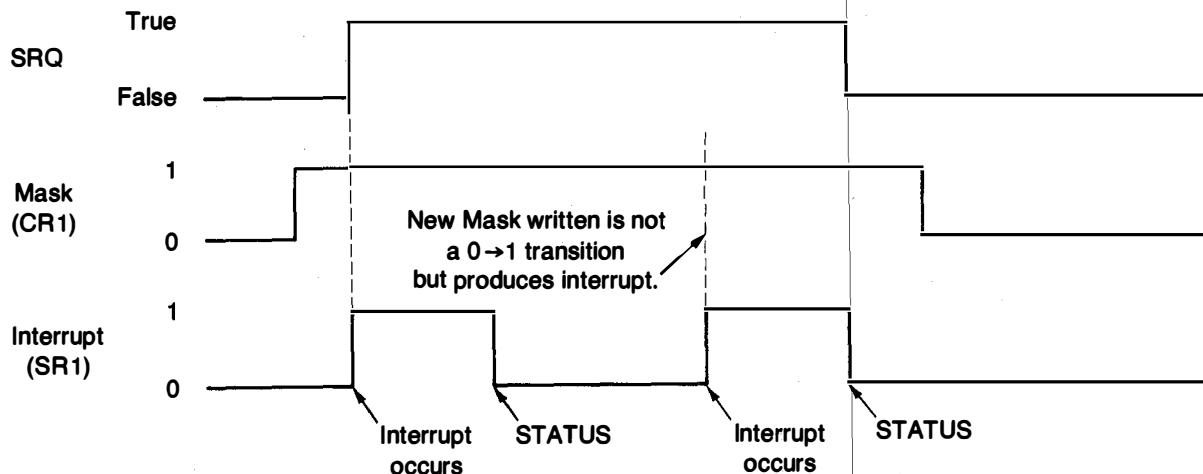
Event-Initiated Interrupt (SCG,GET,SDC/DCL,IFC)



**State-Initiated Interrupt
(TA,CA,LA)**



SRQ interrupt



CR2: HP-IB Control Lines

This register gives direct access to the 8 HP-IB control lines. A bit, when set, causes the corresponding HP-IB control line to be set TRUE for as long as the CR2 bit is set. The user is cautioned to be aware of bus protocols when using this register. For example, a non-controller may not set the ATN line true, and a non-system controller is not supposed to assert REN line.

CR3: HP-IB Data Lines

This register gives direct access to the 8 HP-IB data lines, DI01 through DI08. Setting a bit in this register causes the corresponding HP-IB data line to be set TRUE. (The user should note; however, that HP-IB data lines are numbered 1 through 8 while the control register data lines are numbered 0 through 7.) The user should exercise caution when writing to CR3, the HP-IB data lines register. For example, by writing to CR3 while the interface is addressed to listen (LA state), a hardware conflict will occur in the interface that could cause erratic operation and damage the interface!

User-defined End-of-Line Sequence Registers

Control registers CR16 through CR23 provide the user with the capability of customizing the end-of-line output sequence that is sent at the end of a data transfer and with the “/” OUTPUT/PRINT image specifier. Also provided is the capability of automatically asserting EOI (End or Identify) with the last character of a data transfer.

The following table shows these registers, and their meanings are explained in the following paragraphs.

HP-IB End-of-Line Sequence Registers

Control Register Number	Bit Number								Default Value	Register Function
	EOI					EOL2	EOL1	EOL0		
CR16	EOI Enable	X	X	X	X	EOL2	EOL1	EOL0	2	EOL Control
CR17	Default value = 13 (Carriage Return)								13	Character 1
CR18	Default value = 10 (Line Feed)								10	Character 2
CR19									0	Character 3
CR20									0	Character 4
CR21									0	Character 5
CR22									0	Character 6
CR23									0	Character 7

CR16: EOL Control

This register controls the end-of-line character sequence that is normally sent after a line of output and for the “/” image specifier.

- Bits 0 through 2 specify the number of characters sent as the EOL (end-of-line) sequence. The default count is 2, which causes 2 characters (CR and LF) of registers CR17-CR23 to be sent. A count of 0 specifies that no EOL sequence is to be sent.
- Bits 3 through 6 are not used. (X=don't care)
- Bit 7 when set causes the HP-IB control line EOI to be asserted with the last byte of a data transfer. If the EOL count is non-zero, EOI is asserted with the last character of the EOL sequence. If the EOL count is zero, EOI is asserted with the last character of the data list being sent (PRINT, SEND and TRANSFER only).

CR17-CR23: EOL Sequence

These registers contain the characters sent as the end-of-line sequence. Default values are a carriage-return (decimal 13) for CR17 and a line-feed (decimal 10) for CR18.

To set up an EOL sequence for double-spaced printing for example, set the EOL count to 3 and the EOL sequence to CR, LF, LF.

CONTROL 7,16;3,13,10,10

HP-IB Status Registers

These registers are read by executing the STATUS statement. For example, to return the value of status register 3 in variable S3, execute **STATUS 7,3;S3**. A complete status register table is given, followed by explanations of the individual registers.

HP-IB Status Registers

Status Register Number	Bit Number								Default Value	Register Function
	7	6	5	4	3	2	1	0		
SR0	0	0	0	0	0	0	0	1	1	Interface Identification
SR1	IFC	LA	CA	TA	SRQ	DCL or SDC	GET	SCG	0	Interrupt Cause
SR2	0	REN	SRQ	ATN	EOI	DAV	NDAC	NRFD	64	HP-IB Control Lines
SR3	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	Not Applicable	HP-IB Data Lines
SR4	0	0	SC	A4	A3	A2	A1	A0	53	HP-IB Address/ System Controller
SR5	SC	LA	CA	TA	SPE	Parity Error	REN	LLO	160	State Register
SR6	0	0	0	SC5	SC4	SC3	SC2	SC1	0	Secondary Commands

SR0: Interface Identification

Always returns a value of 1, meaning an HP-IB type interface.

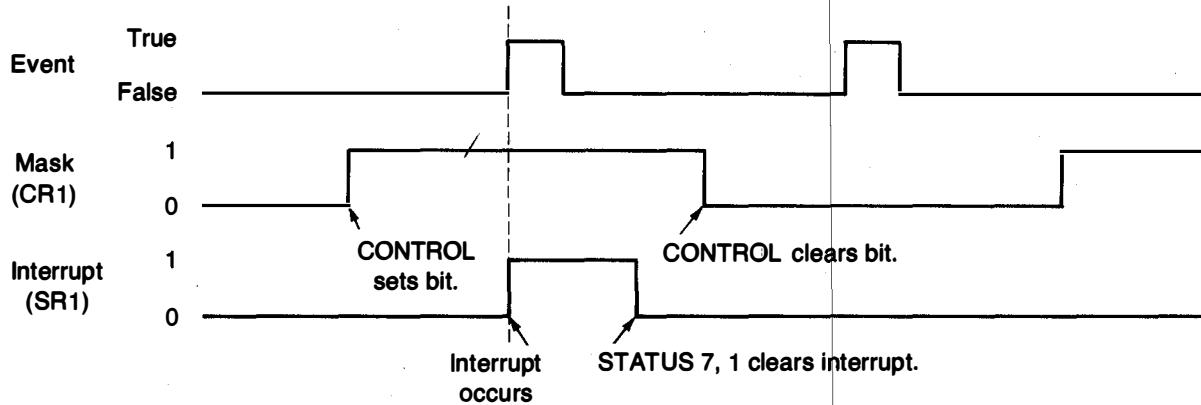
SR1: Interrupt Cause

A bit when set indicates the interrupt condition that caused an end-of-line branch. SR1 is reset to 0 when it is read by a STATUS statement.

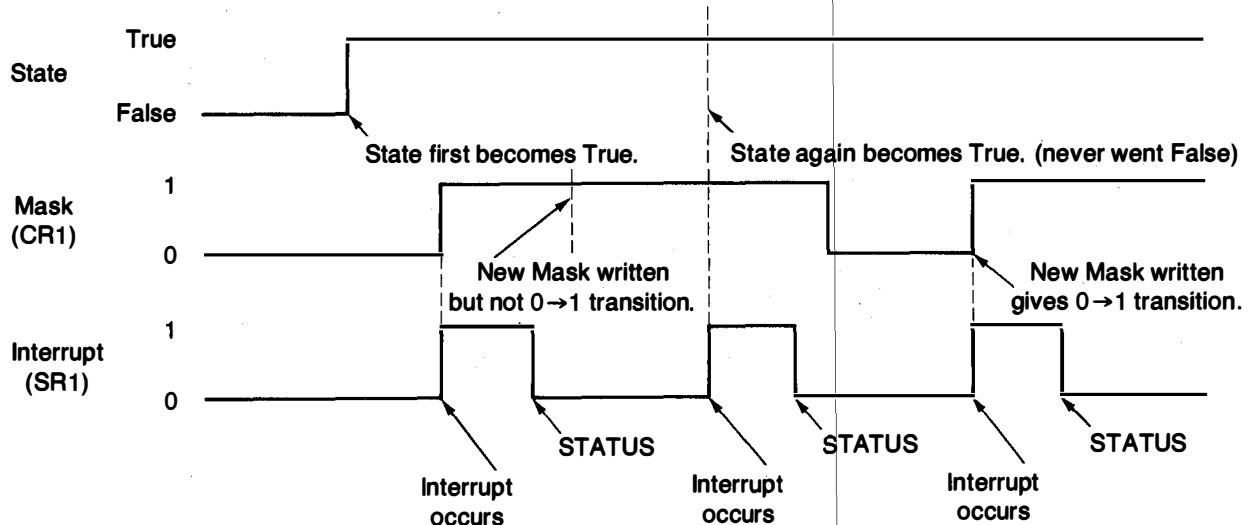
- Bit 0 when set indicates that an SCG (secondary command) was received. The value of the secondary command received is available in SR6. This is an **event** type interrupt: the interrupt, if enabled, will occur when a secondary command is received.
- Bit 1 when set indicates that a GET (group execute trigger) was received while addressed to listen (LA). This is an **event** type interrupt: if enabled, the interrupt will occur when trigger is received.
- Bit 2 when set indicates that either (1) a DCL (device clear) was received or that (2) an SDC (selected device clear) was received while addressed to listen (LA). This is an **event** type interrupt: if enabled, the interrupt will occur when device clear is received.
- Bit 3 when set indicates that an SRQ (service request) was received. The interrupt will occur as long as the SRQ line is TRUE. In general, this means that your service routine must make the SRQ "go away," which is usually accomplished by satisfying the requesting device's need for service.
- Bit 4 when set indicates that either (1) the talker active (TA) bit of CR1 underwent a 0-to-1 transition while the interface was addressed to talk, or (2) the interface became addressed to talk while the talker active (TA) bit of CR1 was set. This is a state-enable interrupt.
- Bit 5 when set indicates that either (1) the controller active (CA) bit of CR1 underwent a 0-to-1 transition while the interface was active controller, or (2) the interface received control while the controller active (CA) bit of CR1 was set. This is a state-enable interrupt.
- Bit 6 when set indicates that either (1) the listener active (LA) bit of CR1 underwent a 0-to-1 transition while the interface was addressed to listen, or (2) the interface become addressed to listen while the listener active (LA) bit of CR1 was set. This is a state-enable interrupt.
- Bit 7 when set indicates that an IFC (interface clear) has occurred on the bus. This is an **event** type interrupt: the interrupt, if enabled, will occur when interface clear is received.

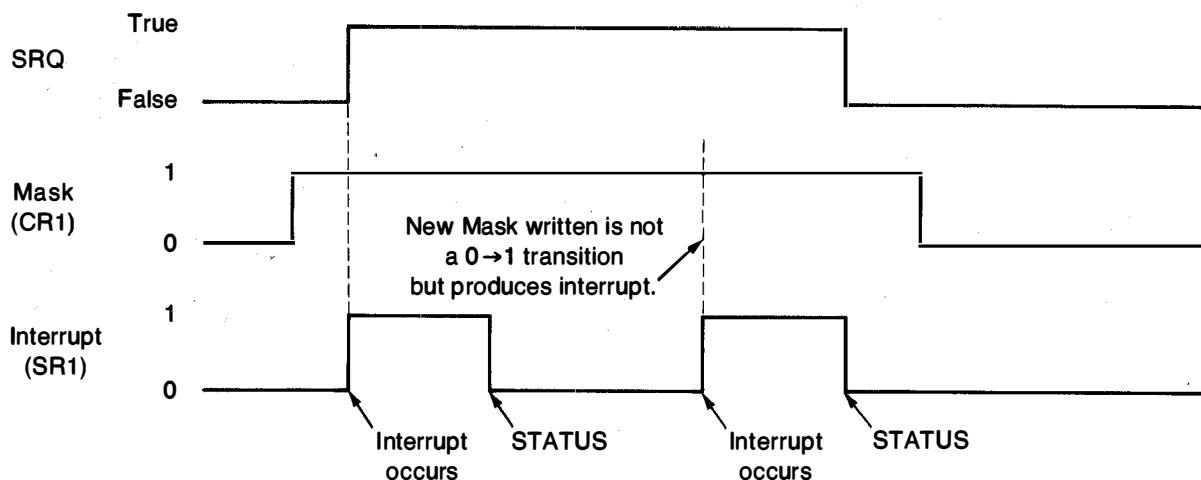
The following diagrams illustrate the interface response to state-initiated, event-initiated, and SRQ interrupts. The interrupt-cause status register (SR1) bits are set when an interrupt occurs and cleared when the status of SR1 is read.

Event-Initiated Interrupt (SCG,GET,SDC/DCL,IFC)



State-Initiated Interrupt (TA,CA,LA)



SRQ interrupt**SR2: HP-IB Control Lines**

A bit when set indicates that the corresponding HP-IB control line is TRUE.

SR3: HP-IB Data Lines

A bit when set indicates that the corresponding HP-IB data line is TRUE.

SR4: HP-IB Address/System Controller Switches

- Bits 0 through 4 indicate the current setting of the HP-IB address switches of the interface. These are factory set to 21.
- Bit 5 indicates the setting of the system controller switch of the interface. A 1 indicates system controller.
- Bits 6 and 7 are always 0.

SR5: HP-IB State Register

This register indicates current HP-IB status of the 82937A Interface.

- Bit 0 when set indicates that the interface is in a local lockout state (LLO).
- Bit 1 when set indicates that the interface is in a remote state (REN).
- Bit 2 when set indicates that a parity error occurred on input while parity was enabled. It is cleared when R5 is read.
- Bit 3 when set indicates that SPE (serial poll enable) has been received. It is cleared when SPD (serial poll disable) is received.
- Bit 4 when set indicates that the interface is addressed to talk (TA or talker active).
- Bit 5 when set indicates that the interface is active controller (CA or controller active).
- Bit 6 when set indicates that the interface is addressed to listen (LA or listener active).
- Bit 7 when set indicates that the interface is system controller (same as SR4 bit 5).

SR6: Secondary Command Register

- Bits 0 through 4 indicate the last secondary command received when the secondary command bit (Bit 0) of CR1 is set. The SR6 register contains any secondary command that follows the interfaces talk or listen address.
- Bits 5 through 7 are always 0.

Mnemonic Conventions

The following conventions are used to document the HP-IB control sequences that are used to implement bus functions.

ATN: Attention (ATN) TRUE
ATN: Attention (ATN) FALSE
(CA): Controller Active State
(CR): Carriage Return
(data): One or more data bytes.
(DCL): Device Clear
(GET): Group Execute Trigger
(GTL): Go To Local
(LA): Listener Active State
(LAG): Listen Address Group (listen addresses of specified devices)
(LF): Line Feed
(LLO): Local Lockout
(MLA): My Listen Address (listen address of HP85)
(MTA): My Talk Address (talk address of HP85)
(PPC): Parallel Poll Configure
(PPU): Parallel Poll Unconfigure
(SC): System Controller
(SCG): Secondary Command Group
(SDC): Selected Device Clear
(SPD): Serial Poll Disable
(SPE): Serial Poll Enable
(TA): Talker Active State
(TAD): Talk Address (talk address of specified device)
(TCT): Take Control
(UNT): Untalk
($\cong 6 \mu s$): Time span slightly greater than 6 microseconds

Message Concepts

Devices which communicate along the interface bus are transferring quantities of information. The transfer of information can be from one device to another device, or from one device to more than one device. These quantities of information can easily be thought of as "messages."

In turn, the messages can be classified into twelve types. The HP-85A Desktop Computer is capable of implementing all twelve of the interface messages. The list below gives the twelve message types for the HP-IB.

1. **The Data Message.** This is the actual information which is sent from one talker to one or more listeners along the interface bus.
2. **The Trigger Message.** This message causes the listening device(s) to perform a device-dependent action when addressed.
3. **The Clear Message.** This message causes either the listening device(s) or all of the devices on the bus to return to their predefined device-dependent states.
4. **The Remote Message.** This message causes listening devices to switch from local front-panel control to remote program control when addressed to listen.
5. **The Local Message.** This 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.** This message prevents a device operator from manually inhibiting remote program control.
7. **The Clear Lockout/Local Message.** This message causes all devices on the bus to be removed from Local Lockout and revert to Local. This message also clears the Remote Message for all devices on the bus.
8. **The Require Service Message.** A device can send this message at any time to signify that the device needs some type of interaction with the controller. This message is cleared by sending the device's Status Byte Message if the device no longer requires service.
9. **The Status Byte Message.** A byte that represents the status of a single device on the bus. Bit 6 indicates whether the device sent a Require Service Message, and the remaining bits indicate operational conditions defined by the device. This byte is sent from a talking device in response to a serial poll operation performed by a controller.
10. **The Status Bit Message.** A byte that represents the operational conditions of a group of devices on the bus. Each device responds on a particular bit of the byte thus identifying a device-dependent condition. This bit is typically sent by devices in response to a parallel poll operation.
The Status Bit Message can also be used by a controller to specify the particular bit and logic level that a device will respond with when a parallel poll operation is performed. Thus more than one device can respond on the same bit.
11. **The Pass Control Message.** This transfers the bus management responsibilities from the active controller to another controller.
12. **The Abort Message.** The system controller sends this message to unconditionally assume control of the bus from the active controller. This message terminates all bus communications (but does not implement a Clear Message).

These messages represent the full implementation of all HP-IB system capabilities. 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 the operational compatibility of the system.

HP-IB Control Lines

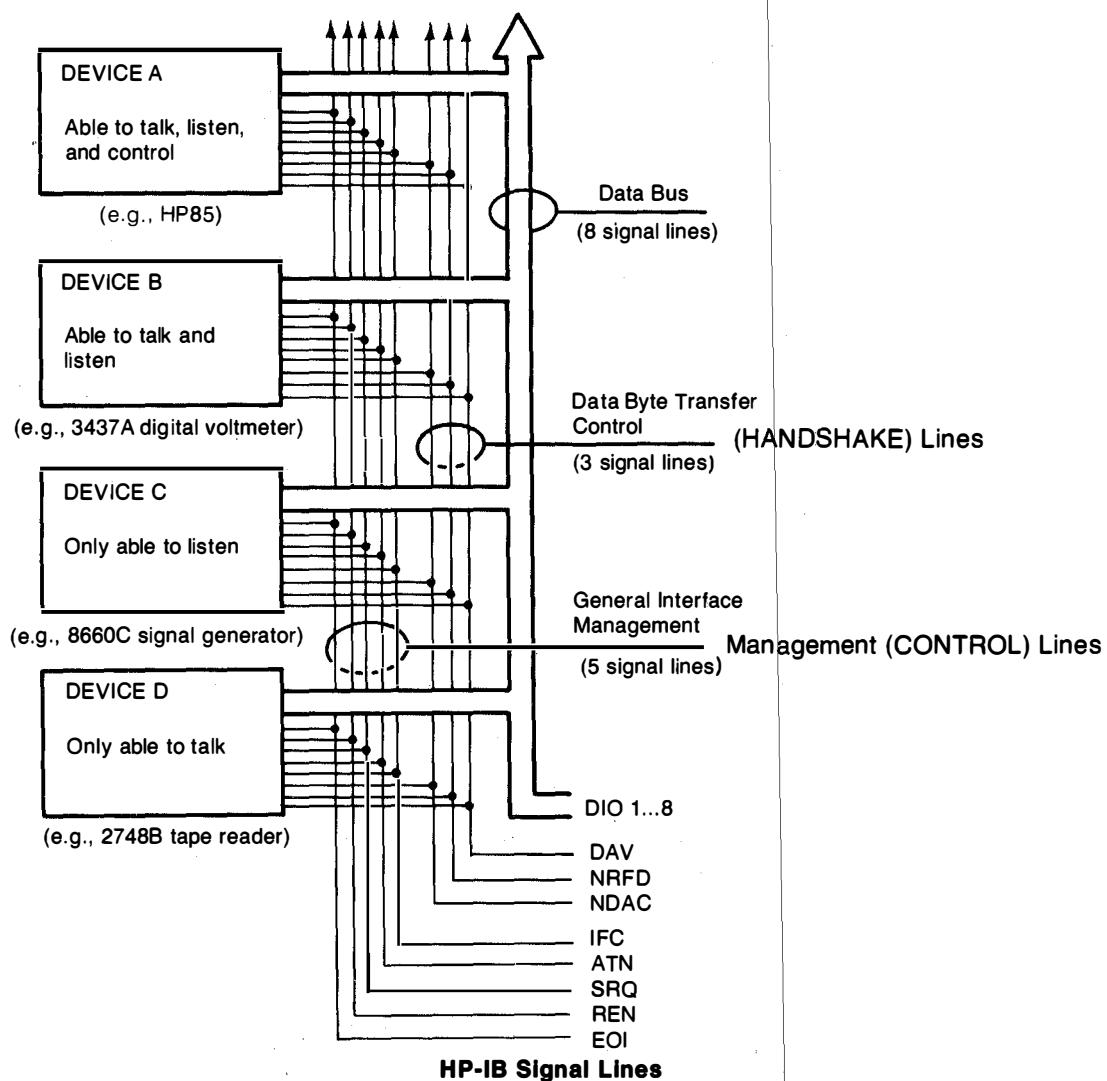
The figure below shows the meanings given to the eight control lines that make up the HP-IB. Three of these lines are designated as the "handshake" lines and are used to control the timing of data byte exchanges so that the talker does not get ahead of the listener(s). The three handshake lines are:

DAV Data Valid

NRFD Not Ready for Data

NDAC Not Data Accepted

Using these lines, a typical data exchange would proceed as follows. All devices currently designated as active listeners would indicate (via the NRFD line) when they are ready for data. A device not ready would pull this line low (ground), while a device that is ready would let the line float high. Since a low overrides a passive high, this line will stay low until all active listeners are ready for data. When the talker senses this, it places the next data byte on the data lines and then pulls DAV low. This tells the listeners that the information on the data lines is valid and that they may read it. Each listener (at its own speed) then takes the data and lets the NDAC line go high. Again, only when all listeners have let NDAC go high will the talker sense that all listeners have read the data. It can then remove DAV (let it go high) and start the entire sequence over again for the next byte of data.



ATN (Attention)

Command messages are encoded on the data lines as 7-bit ASCII characters, and are distinguished from normal data characters by the setting of the attention (ATN) line. That is, when the ATN line is false, bytes on the data lines are interpreted as simple data characters. But when the ATN line is true, the data lines become the carriers of command information. The set of 128 ASCII characters that can be placed on the data lines during this ATN-true mode are divided into four classes as shown on the inside back cover.

IFC (Interface Clear)

Only the hardwired system controller can set the IFC line True. By asserting IFC, all bus activity is unconditionally terminated, the system controller regains (if it has been passed to another device) the status of active controller, and any current talker and listeners become unaddressed. Normally, this line is only used to abort an unwanted operation, or to allow the system controller to regain control of a bus where something has gone wrong. It overrides any other activity that is currently taking place on the bus.

REN (Remote Enable)

This line is used to allow instruments on the bus to be programmed remotely by another device on the bus, usually (but not necessarily) the active controller. Any device that is addressed to listen while REN is True is placed in the REMOTE mode of operation.

EOI (End or Identify)

Normally, data messages sent over the HP-IB are sent using the standard ASCII code and are terminated by the ASCII line-feed character (LF = decimal 10). A device (e.g., a disk) may wish to send blocks of information in 8-bit bytes which represent general binary patterns; and no specific 8-bit pattern can be designated as a terminating character since it could occur anywhere in the data stream. In this case, the EOI line is used to mark the end of the data message. When the listeners detect that the EOI line is True, they recognize that the byte on the data lines is the last one of the data message.

The EOI line is also used during an identify (parallel poll) sequence.

SRQ (Service Request)

The active controller is always in charge of the order of events on the HP-IB. If a device on the bus has some information of which the controller should be aware, it can use the service request line to ask for the controller's attention. For example, a printer might request service to inform the controller that it is out of paper. Or a digitizer might assert service request to tell the controller that its sample button was pressed by the operator and a reading is ready to be taken. This represents a request (NOT a demand), and it is up to the controller 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 each individual device and will be contained in the operating manual for that device.

SC (System Controller)

There is one and only one special device on the bus known as the system controller. This capability is established by the hardware of the device itself (usually by the setting of a slide switch or a jumper) so that when power is turned on or the bus is reset, the device set to be the system controller will also assume the role of the active controller. At any time, the current active controller may pass control to any other device on the bus that is capable of performing the functions of a controller. (All devices are not required to have this capability.) The role of system controller, however, stays with the device which is physically set for that function and cannot be passed off. At any time when the system controller determines that something has gone wrong with the normal bus operations, it can reset the bus (by asserting IFC) and get back active control.

HP-IB Limitations

Before leaving this brief overview of the HP-IB, some of the limitations of the HP-IB should be considered. The first limitation is that a maximum of 15 devices may be connected together by one HP-IB. This limitation arises from electrical specifications for the line driver and receiver circuits, and how much current they can provide or sink. Another limitation is that the total cable length connecting all of the instruments on one bus cannot exceed 20 meters in length. Voltage levels on the various lines do not change instantaneously, but require a certain amount of time proportional to the length of the cable. A limit is placed on the cable length to insure that the bus will operate properly at its rated maximum speed. In general, then, the HP-IB is intended to provide a simple means of interconnecting local instrumentation clusters. Other means of interfacing (such as serial I/O) are better suited to long distance communications.

HP-IB Control Responses

The following table shows the responses of the 82937A Interface when it receives the various bus control messages.

ATN:	Interprets bus data as commands. The Interface can still interact with computer while receiving commands.
IFC:	Clears all talker and listener states to the power-on state. May release Active Control unless also System Controller.
REN:	Sets bit 1 of SR5.
EOI:	Terminates data input transfer to a buffer. Can terminate ENTER or TRANSFER statement.
SRQ:	Sets the service request bit (bit 3 of SR1) and interrupt if bit 3 of interrupt mask is set.
DCL, SDC:	Can interrupt the computer if bit 2 of CR1 is set.
GTL, LLO:	Set appropriate bits in SR5.
GET:	Can interrupt the computer if bit 1 of CR1 is set.
Serial poll:	Delivers the currently set serial poll response byte (REQUEST) without computer intervention.
Parallel poll:	Responds to a parallel poll using the line and sense set by the switches on the card if asserting SRQ.
PPU, PPC:	Parallel poll response is switch settable and not programmable. No response.
TCT:	Assumes active control of the HP-IB.

HP-IB Universal Commands (ATN true)

The table below documents the decimal value of the HP-IB interface messages. Also shown are the numeric ranges for Address and Command Groups.

Primary Command Group (PCG)

Decimal Value	ASCII Character	Interface Message	Description
0	NUL		
1	SOH	GTL	Got To Local
2	STX		
3	ETX		
4	EOT	SDC	Selected Device Clear
5	ENQ	PPC	Parallel Poll Configure
6	ACK		
7	BEL		
8	BS	GET	Group Execute Trigger
9	HT	TCT	Take Control
10	LF		
11	VT		
12	FF		
13	CR		
14	SO		
15	SI		
16	DLE		
17	DC1	LLO	Local Lockout
18	DC2		
19	DC3		
20	DC4	DCL	Device Clear
21	NAK	PPU	Parallel Poll Unconfigure
22	SYN		
23	ETB		
24	CAN	SPE	Serial Poll Enable
25	EM	SPD	Serial Poll Disable
26	SUB		
27	ESC		
28	FS		
29	GS		
30	RS		
31	US		
32-62	SP to > (Numbers, special char)	LAG	Listen Address Group
63	?	UNL	Unlisten
64-94	@ to > (Upper case ASCII)	TAG	Talk Address Group
95	—	UNT	Untalk
96-126	(lowercase ASCII)	SCG	Secondary Command Group
127	DEL		

Available Bus Addresses and Codes

Address Characters		Address Switch Settings					Address Codes	
Listen	Talk	(5)	(4)	(3)	(2)	(1)	Decimal	Octal
SP	@	0	0	0	0	0	0	0
!	A	0	0	0	0	1	1	1
"	B	0	0	0	1	0	2	2
#	C	0	0	0	1	1	3	3
\$	D	0	0	1	0	0	4	4
%	E	0	0	1	0	1	5	5
&	F	0	0	1	1	0	6	6
,	G	0	0	1	1	1	7	7
(H	0	1	0	0	0	8	10
)	I	0	1	0	0	1	9	11
*	J	0	1	0	1	0	10	12
+	K	0	1	0	1	1	11	13
.	L	0	1	1	0	0	12	14
-	M	0	1	1	0	1	13	15
.	N	0	1	1	1	0	14	16
/	O	0	1	1	1	1	15	17
0	P	1	0	0	0	0	16	20
1	Q	1	0	0	0	1	17	21
2	R	1	0	0	1	0	18	22
3	S	1	0	0	1	1	19	23
4	T	1	0	1	0	0	20	24
5	U	1	0	1	0	1	21	25
6	V	1	0	1	1	0	22	26
7	W	1	0	1	1	1	23	27
8	X	1	1	0	0	0	24	30
9	Y	1	1	0	0	1	25	31
:	Z	1	1	0	1	0	26	32
;	[1	1	0	1	1	27	33
<	/	1	1	1	0	0	28	34
=]	1	1	1	0	1	29	35
>	^	1	1	1	1	0	30	36

← Preset

HP-IB Statement Summary

The following table summarizes the Interface conditions that must be met for a given Interface state and Program statement, and the resultant actions that are performed.

Statement	Given current state(s) of:	Additional required conditions are:	Actions Performed (Bus sequences are in color)
ABORTIO 7	SC	none	IFC, assumes active control. [IFC]
	SC & CA	none	MTA, leaves ATN true. [ATN • MTA]
	SC & CA	none	Terminates I/O operation. [No sequence generated]
ASSERT 7;X	any	none	Immediate write to CR2. IFC is not asserted.
CLEAR 701	CA	none	Addressing performed, then send SDC to device 01. [ATN • UNL, MTA, LAG, SDC]
CLEAR 7	CA	none	No addressing. Send DCL (Note: ATN remains true; user may use RESUME 7 to set ATN false.) [ATN • DCL]
CONTROL 7,n;X	any	none	Writes X to CRn when interface becomes non-busy.
ENABLE INTR 7;X	any	none	Writes X to CR1 when interface becomes non-busy.
ENTER 705;X	CA	none	Device 05 is addressed to talk HP-85 is addressed to listen, data is input to X. [See ENTER sequence table]
ENTER 7;X	CA	LA	Inputs data to X. [No sequence generated]
	CA	wait for LA	Waits until addressed to listen then inputs data to X. [No sequence generated]
HALT 7	any	none	Terminates I/O operation. [No sequence generated]
LOCAL 7	SC	none	REN is set false. [REN]
LOCAL 701	CA	none	Addressing is performed, then GTL is sent. Note: ATN remains true. User may use RESUME 7 to set ATN false. [ATN • UNL, MTA, LAG, GTL]
LOCAL LOCKOUT 7	CA	none	LLO is sent. [ATN • LLO]

OUTPUT 705;X	CA	none	HP-85 is addressed to talk, device 05 is addressed to listen, data X is sent. [See OUTPUT sequence table]
OUTPUT 7;X	CA	TA	Outputs data X. [No sequence generated]
	CA	wait for TA	Waits until addressed to talk, then outputs data X. [No sequence generated]
PASS CONTROL 715	CA	none	Device 15 is addressed to talk then TCT is sent. [ATN • UNL,MLA,TAG,UNL,TCT,ATN]
PASS CONTROL 7	CA	none	No addressing. Send TCT. [ATN • UNL,TCT,ATN]
PPOLL (7)	CA	none	Sends IDY (Identify). [ATN • EOI,(>6μs),ATN • EOI]
REMOTE 7	SC	none	REN is set true. [REN]
REMOTE 701	SC	none	REN is set true, then device 01 is addressed. Note: ATN is left true. [REN,ATN • UNL,MTA,LAG]
REQUEST 7;X	CA	none	If bit 6 of X is = 1 then SRQ is set true. The HP-85 then sends X in response to a serial poll and drops SRQ if set.
RESET 7	any	none	Sets the HP-IB interface to its power-on state. If system controller, then IFC is asserted and REN is turned off then on again.
RESUME 7	CA	none	Sets ATN false. [ATN]
SEND 7;commands	CA	none	Sends specified commands with ATN true. ATN is left true.
SEND 7;data	any	TA	Sends specified data with ATN false. ATN is left false.
SPOLL (7)	CA	LA	Conducts a serial poll. [ATN • SPE,ATN,<data>,ATN • SPD,UNT]
SPOLL (724)	CA	none	Addresses device 724 to talk then conducts serial poll. [ATN • UNL,MLA,TAG,SPE,ATN,<data>,ATN • SPD,UNT]
STATUS 7,n;X	any	none	Sets X to the value of SRn.
TRIGGER 7	CA	none	Sends GET. [ATN • GET]
TRIGGER 701	CA	none	Addresses device 01, sends GET. Note ATN is left true. [ATN • UNL,MTA,LAG,GET]

HP-IB Errors

Error #	Meaning
111	A STATUS or CONTROL statement was executed with a non-valid register number specified. Only status registers SR0-SR6, control registers CR0-CR3, and control registers CR16-CR23 may be specified.
113	The statement executed requires that the HP-85 be the system controller.
114	The statement executed requires that the HP-85 be the current active controller.
115	The statement executed requires that the HP-85 be addressed-to-talk. A possible cause for this error is that an OUTPUT <sc> statement was executed while the HP-85 was active controller but not addressed to talk. (You executed OUTPUT 7;X but the HP-85 was not previously addressed to talk.)
116	The statement executed requires that the HP-85 be addressed-to-listen. A possible cause for this error is that an ENTER <sc> statement was executed while the HP-85 was active controller but not addressed to listen. (You executed ENTER 7;X but the HP-85 was not previously addressed to listen.)
117	The statement executed requires that the HP-85 be non-controller. A probable cause of this error is that a REQUEST statement was executed by the HP-85 while active controller.

Section 13

Using the Serial Interface

Section Introduction

The HP 82939A Serial I/O Interface enables your HP-85 computer to communicate with a variety of devices that are configured for serial communication. These devices can range from simple serial printers to large-scale computers.

This section of the manual explains how to program the Serial I/O Interface to communicate with these devices. The subjects presented are:

- Introduction to Serial Interfacing
- Printer and Terminal Interfacing
- Advanced Serial Interfacing
- Registers
- Troubleshooting Hints

All discussions in this section assume the following factory preset defaults:

- Interface select code = 10
- Character specification = 7 bits/character, odd parity, 1 stop bit.
- Transfer Rate = 300 Baud.
- Auto-handshake = disabled (off).

The 82939A Interface Installation and Theory of Operation Manual explains:

- How to set the interface select code.
- How to set the interface default switches.
- How to install the interface in your HP-85.

You should read and understand these parts of the Installation and Theory of Operation Manual before proceeding with this section.

Introduction to Serial Interfacing

This topic explains serial I/O concepts and provides simple descriptions of the terms that are used in the printer and teletype discussions.

Serial I/O simply means the transmission of data, one bit after another, over a line. Contrast this with parallel I/O, which transfers eight or more data bits simultaneously. Each method of data transmission has unique advantages and disadvantages. Parallel I/O can transfer eight or more data bits at a time but requires one wire (or line) for each bit and one ground (or common) wire. Serial I/O transfers data one bit at a time but only requires one wire for the data and one wire for the ground. The cost and logistics of parallel I/O become prohibitive when considering communication over distances greater than 50 feet. Serial I/O allows for inexpensive long-distance communication through use of an existing telephone system.

Equipment Configuration

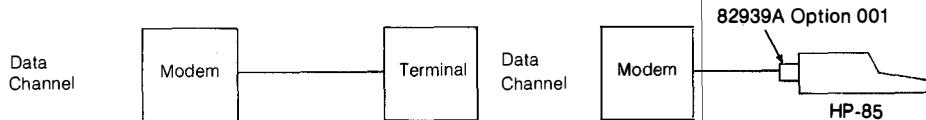
Serial I/O devices are described according to the functions that they perform. These functional descriptions are: Data Terminal Equipment, and Data Communications Equipment.

Data Terminal Equipment

Data Terminal Equipment (DTE) is any location in a network where information can enter or exit. Items included as DTE are:

- The remote terminal.
- The remote terminal interface.
- The host computer.
- The host computer interface.

The 82939A Option 001 interface configures the HP-85 as a DTE device. The following drawings show a typical DTE configuration for the HP-85.

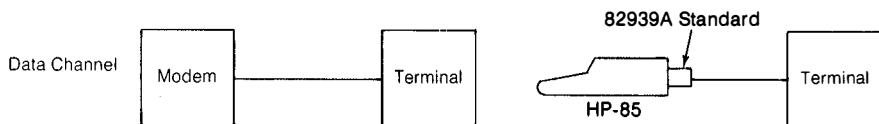


Data Communications Equipment

Data Communications Equipment (DCE) is the equipment used to convey information between locations. Items included as DCE are:

- The modems.
- The modem interfaces.
- The link (e.g., telephone lines).

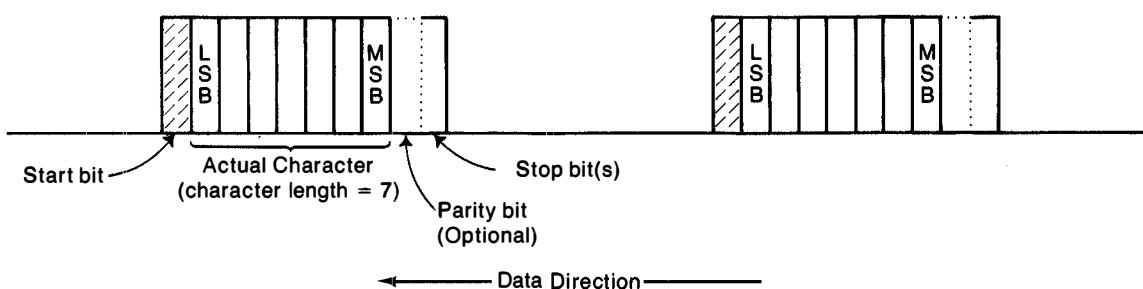
The 82939A Standard interface configures the HP-85 as a DCE device. The following drawings show a typical DCE configuration for the HP-85.



Asynchronous Data Transmission

The HP 82939A Interface communicates asynchronously with external devices. Asynchronous communication means simply that each character is sent over the line (or link) with synchronization built into the character. The next drawing shows typical asynchronous transmission of characters.

Character Definition

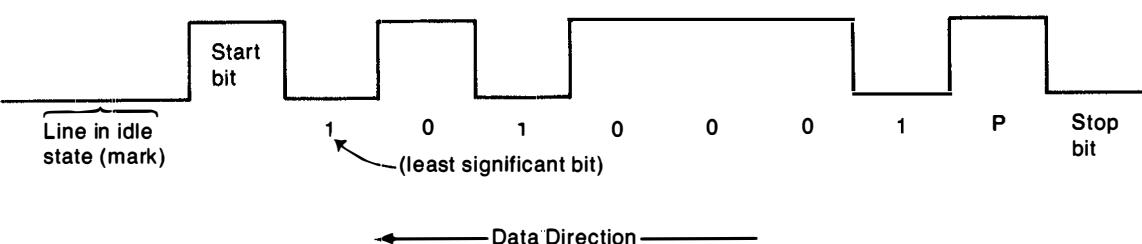


The actual data is transmitted over the line using two voltage levels to represent the two possible states of a binary digit. The following table shows the two binary states and the voltage levels assigned to each state.

Binary State:	logic 0	logic 1
Voltage Range: Level Name: Line State:	+3V to +25V SPACE high	-3V to -25V MARK low (idle)

When data is not being transmitted, the line is held in the low (or mark) state. When the transmitting device has data to send, it places the line in the high (space) state for one bit time. This change to the high state for one bit time is called the start bit. The remaining digital data is then transmitted to the receiving device. The following drawing shows how a typical character (an ASCII "E") is transmitted over the link.

Character Transmission



The previous drawings show characters that consist of a start bit, the actual ASCII character, a parity bit, and a stop bit. The meaning of each of these parts of the character is explained next.

Start Bit

The start bit is inserted at the beginning of the character by the interface. The start bit is used to signal the receiving device that the transmission of a character is starting. When the start bit is detected, the receiver starts its internal clock to synchronize the receiver to the input data.

Data Character

The character is the binary bit-pattern of the actual transmitted character. For example, the bit pattern 0 110 010 is transmitted for the ASCII “2” character. Assuming that the receiver expects ASCII characters, it interprets the bit pattern as a “2”.

Note that the ASCII code is made up of seven bits. Other codes may be made up of five, six, or eight bits. The interface provides a “character length” specification to define the number of bits that make up a character. This “character length” specification does NOT include start, stop, or parity bits. The factory setting for the reset default switches specifies seven bit character length. See Register 4 for further discussion about character length.

Parity

Parity provides a method of error checking. The parity bit (if specified) always follows the character. No parity bit is added when parity is not specified (parity = none). The parity bit is always a “1” when parity = 1 is specified, and the parity bit is always a “0” when parity = 0 is specified. Parity = even and parity = odd specify that the parity bit is determined as shown next.

Number of “1” Bits In Character	Parity Specified	Parity Bit
Odd	Odd	0
Even	Odd	1
Odd	Even	1
Even	Even	0

For example, the bit pattern for the ASCII “2” character is 0 110 010. There are three “1” bits in this pattern. If odd parity is specified, the parity bit is “0”. If even parity is specified, the parity bit is “1”. The factory setting for the reset default switches specifies odd parity. See Register 4 for further discussion of parity.

Stop Bits

Stop bits are added following the parity bit by the interface. The stop bits are not really bits. The transmitter holds the line in the “idle” state for the amount of bit times specified by the stop bits parameter. This amount of bit times is referred to as stop bits. Allowable stop bits parameters are 1 and 2. The factory setting for the reset default switches specifies one stop bit. See Register 4 for further discussion of stop bits.

Transfer Rates (or Baud)

When two devices are communicating, they must transfer and receive information at compatible data rates. If the transmitter sends data at a faster rate than the receiver is expecting the data, information will be lost. Most devices (such as printers) provide a switch to select the data rate. The serial I/O interface provides programmable data rates and a switch selectable default data rate (see Control Register 3).

Handshakes

Handshakes are used to communicate status information from one device to another. The handshakes are used to indicate a buffer full condition, received data errors, and modem status. Some devices use modem lines to indicate an input buffer full condition (see Printer Interfacing). Other types of handshake protocols (DC1/DC3 and ENQ/ACK) are explained in the Advanced Serial Interfacing topic.

Printer and Terminal Interfacing

This topic explains how to connect the HP-85 to a printer or teletype in order to produce hard-copy output and to a terminal to input data. Example programs are shown for typical applications.

Printer Interfacing

Interfacing to a printer is not difficult if you first determine the printer's requirements. Interfacing information can be found in the owner's or operator's manual supplied with the printer. Look for key information that may be listed as:

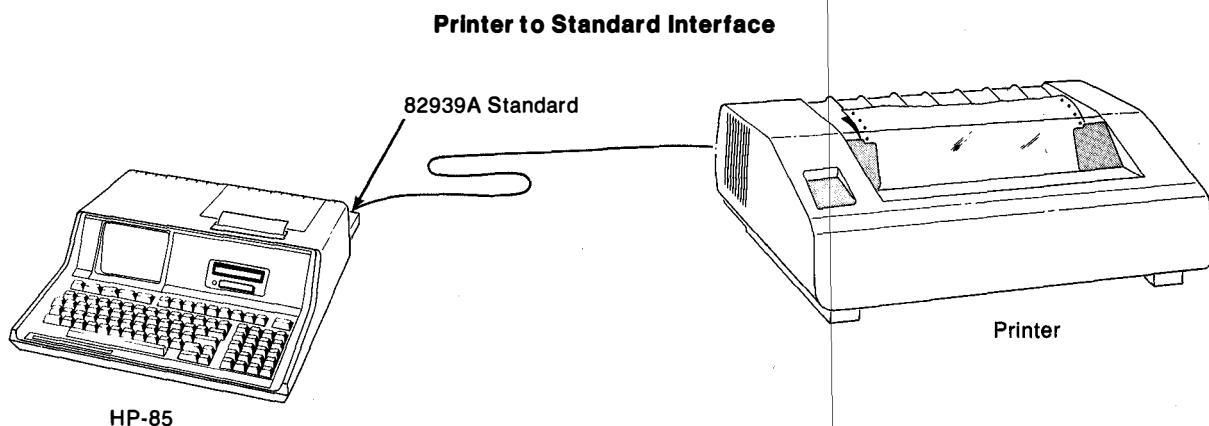
- Technical Data
- Technical Specifications
- Performance Specifications
- Transfer Rates
- Baud Rates
- Character Set
- Interfacing Diagrams
- Handshake

Example 1 (Printer)

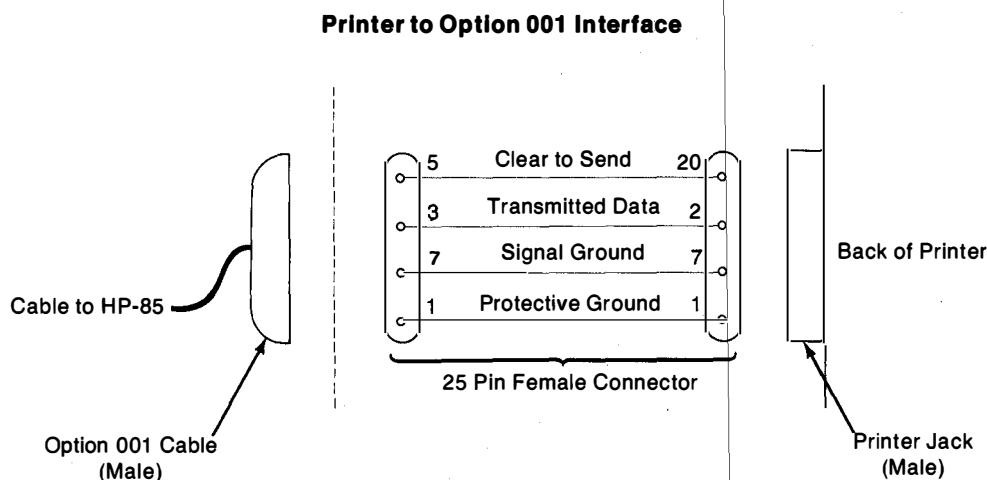
As example, let's assume that the following information is obtained from the owner's manual supplied with your printer:

- Character Set - Asynchronous bit serial. Seven data bits and one parity bit or eight data bits and no parity. The eighth bit is ignored. The printer accepts one or two stop bits. Full 96-character ASCII coding.
- Serial Baud Rate - Switch selectable rates of 110, 300, 600, and 1200 bits per second.
- Handshake - The printer generates the Data Terminal Ready signal to regulate Received Data input and prevent input buffer overflow. When the input buffer is full, the Data Terminal Ready signal is dropped. This prevents further data transfer until the printer input buffer can accept more data from the computer.
- Interfacing - The following diagrams show the connections for Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) configurations.

Here's how the printer is interfaced with the HP-85:



When connecting this printer to the 82939A Option 001 interface, the following adapter cable must be used.



Let's use the data that we have collected thus far. The interface is preset at the factory to the following default values:

- Interface select code = 10
- Transfer rate = 300 baud
- Autohandshake = Off
- Character length = 7
- Parity = Odd
- Stop bits = 1

If we compare the requirements of the printer with the defaults provided by the interface, we see that the printer baud rate must be set to 300, and the interface autohandshake feature must be enabled.

This example shows the simple programming steps that enable the HP-85 to send data to the printer.

Set the printer Baud Rate Switch to the 300¹ baud position.

Key the following program into your HP-85

```
10 CONTROL 10,5 ; 48 ! ENABLE AUTO HANDSHAKE
20 FOR X=1 TO 10
30 OUTPUT 10 "NUMBER=";X
40 NEXT X
50 STOP
```

The printer should print the data as shown next.

```
NUMBER= 1
NUMBER= 1
NUMBER= 2
NUMBER= 3
NUMBER= 4
NUMBER= 5
NUMBER= 6
NUMBER= 7
NUMBER= 8
NUMBER= 9
NUMBER= 10
```

¹ The example shown here uses default values provided by the serial I/O interface whenever possible. Since the printer uses a handshake to prevent buffer overrun, the baud rate can be set to maximum rate allowed by the printer (in this case 1200) to obtain maximum throughput.

You should note that many commercial printers have a female output connector, but the connector is wired as a DTE (male) device. This configuration must be changed when the 82939A option 001 cable is used. Pins 2 and 3 on the interface cable or on the printer connector must be interchanged. A cable is available that implements these wiring changes. Order Hewlett Packard P/N 8120-3097.

Example 1 implements the auto-handshake feature provided by the interface. Other handshakes such as "Enquire/Acknowledge" and XON/XOFF may be used. See Advanced Serial Interfacing for a explanation of these handshakes.

Terminal Interface

The 82939A Standard interface is supplied with a 25-pin female EIA connector. The standard interface is used to connect to a terminal. Check the owner's manual supplied with the terminal for key information such as baud rate, character set, interfacing diagrams, and technical specifications. Most terminals provide a half/full duplex switch. The serial I/O interface can operate in either half or full duplex mode. Full duplex terminals usually require that their transmitted data be echoed by the host computer. The host computer simply retransmits each received character back to the terminal. This retransmission (or echo) of characters provides the terminal operator with a visual indication of the data that was transmitted to the computer. See Register 9, bit 1 for a description of the auto-echo feature.

Example 2 (Terminal)

Here's how to interface with a terminal. Let's assume that the following information is obtained from the owner's manual supplied with your terminal:

- Character Set - Asynchronous bit serial; 1 start bit with 7 data bits, odd parity and 1 or 2 stop bits. Full 96-character ASCII coding.
- Serial Baud Rate - Switch selectable rates of 110, 300, 600, and 1200 bits per second.
- Duplex - Switch selectable half and full-duplex operation.
- Handshake - The terminal uses XON/XOFF handshake (see Advanced Serial Interfacing for XON/XOFF handshake details).
- The terminal requires a DC1, CHR\$(17), prompt from the remote device in order to begin transmitting.
- The terminal transmits a line-feed character as its output end-of-line terminator.
- The terminal requires a carriage-return/line-feed as its input end-of-line terminator.

Here's how to interface the terminal with the HP-85:

Set the terminal switches as follows. Baud Rate = 300, Duplex = Full. The reset defaults for the serial I/O interface specify 300 Baud, and seven bit ASCII characters with odd parity. The only interface card specifications that must be programmed are: enable the auto-echo feature, and implement the XON/XOFF handshake protocol. The following program shows how to set up the interface to communicate with the terminal.

Key the following program into your HP-85.

```

10 DIM A$[150]
20 CONTROL 10,9 : 139 ! ADD AUTO ECHO FEATURE TO REG 9
30 CONTROL 10,11 : 194,13 ! ENABLE XON/XOFF-REG 12 = LF
40 CONTROL 10,14 : 17,19 ! REG 14 = DC1, REG 15 = DC3
50 OUTPUT 10 :"START TERMINAL";CHR$(17)
60 ENTER 10 USING "%,%K" ; A$
70 OUTPUT 10 USING "#,A" ; CHR$(10)
80 P=POS(A$,CHR$(8))
90 IF NOT P THEN 120
100 A$[CP-1]=A$[CP+1]
110 GOTO 80
120 DISP A$
130 GOTO 60
140 END

```

Line 30 - Activates Transmitter Flag Enable/Disable feature and specifies Control Register 12 character as an input termination character. Line 30 also sets the value 13 (for Line-Feed) to register 12.

Line 40 - Sets the value 17 (DC1) to register 14 and the value 19 (DC3) to register 15. These characters control the Transmitter Flag Enable/Disable feature.

Line 60 - Enters the data from the terminal. The “%,%K” image specifier allows the entry into A\$ to be terminated by an “EOI”. This interface generates an “EOI” type signal when the character defined by register 12 is detected in the incoming data.

Line 80 through Line 110 - Interprets the Backspace character.

You can now enter data from the terminal.

Teletype Interface

The 82939A Option 002 interface is supplied with an unterminated cable for use with a teletype device. See the Installation and Theory of Operation Manual for a description of the Option 002 cable. Most teletypes use a 20 milliamp current loop-mode for data transfer. The interface provides 20 milliamp drivers for current loop operation. Most teletypes are configured as shown next:

- Data Rate - 110 baud
- Character - 7 bits
- Parity - Even
- Stop bits - 2 stop bits

Example 3 (Teletype)

The following program transmits data to a teletype that is configured as shown:

```

10 DIM A$[C50]
20 CONTROL 10,3 ; 2 ! 110 BAUD
30 CONTROL 10,4 ; 29 ! 7 BITS, EVEN PARITY, 2 STOP BITS
40 ! THE 20 NULL SEQUENCE ALLOWS 200 MILLISECONDS @ 110
50 ! BAUD FOR TELETYPE TO EXECUTE THE CR/LF SEQUENCE
60 CONTROL 10,16 ; 22 ! EOL = CR/LF AND 20 NULLS
70 INPUT A$
80 OUTPUT 10 ;A$
90 GOTO 70
100 END

```

Advanced Serial Interfacing

This topic explains advanced interfacing techniques. Items discussed include handshakes, modems, and long distance communication over telephone links.

Handshakes

In the printer discussion, the printer used the Data Terminal Ready signal to indicate a buffer-full condition to the HP-85. Other types of handshakes exist, as discussed here.

ENQuire/ACKnowledge

Some peripherals and host computer systems use the ENQuire/ACKnowledge protocol for buffer-full or not-ready indication. When the transmitting device sends a line of text to a receiver, an ENQuire character, e.g., CHR\$(5), is also transmitted following the text. The transmitter waits for an ACKnowledge character, e.g., CHR\$(6), from the receiver before sending another line of text. The receiver responds with the ACKnowledge response if the input data contained no errors and there is sufficient space in the input buffers for at least one more line of text. This discussion uses the ASCII ENQ and ACK characters for the handshake sequences. Other ASCII characters such as ENQ/ESC may be used for these sequences. The following program lines show how to set Control Registers 11, 15, 16, and 19 to implement the ENQuire/ACKnowledge handshakes when the HP-85 is defined as the host.

```

10 CONTROL 10,11 ; 128 ! ALLOW XMIT FLAG ENABLE
20 CONTROL 10,15 ; 6 ! ACK ENABLES XMIT FLAG
30 CONTROL 10,16 ; 67 ! 3 CHAR EOL AND DISABLE XMIT
40 CONTROL 10,19 ; 5 ! END IS 3rd EOL CHARACTER

```

XON/XOFF

Another handshake protocol used by some peripherals and host computer systems is the XON/XOFF handshake (commonly referred to as the DC1/DC3 handshake). During data transfers, the receiver monitors its input buffers to ensure that sufficient space remains for at least one more line of data. When there is not sufficient space remaining in the input buffers, the receiver sends an XOFF (normally a DC3) to the transmitter. The transmitter then suspends further transmission until the receiver sends an XON (normally a DC1) to indicate that transmission may resume. The receiver sends the XON when sufficient buffer space

becomes available for at least one more line of text. The following program lines show how set Control Registers 11, 14, and 15 to implement XON/XOFF handshakes when the HP-85 is defined as the host.

```
10 CONTROL 10,11 ; 192 ! SET XMIT FLAG FOR XON/XOFF
20 CONTROL 10,14 ; 19 ! DC3 IS XON
30 CONTROL 10,15 ; 17 ! DC1 IS XOFF
```

Note: When the HP-85 is configured as an input device (not as a host), you must implement the ENQ/ACK or XON/XOFF handshake sequences to prevent overflowing the HP-85 buffers. The handshakes must be implemented from the BASIC program.

Modems

The word MODEM is a contraction of the words MODulator and DEModulator. A modem is a device that changes digital information into audio tones for transmission over the existing telephone system and converts received audio tones into digital information for input to the receiving device. A modem is used to establish a communication link using telephone lines, since digital data can only be transmitted over short distances with direct point-to-point wiring. Also, the bit format of digital data is incompatible with long distance communication.

Modems utilize handshake signals to communicate with the device to which they are connected. The serial I/O interface implements the following RS-232-C modem control signals:

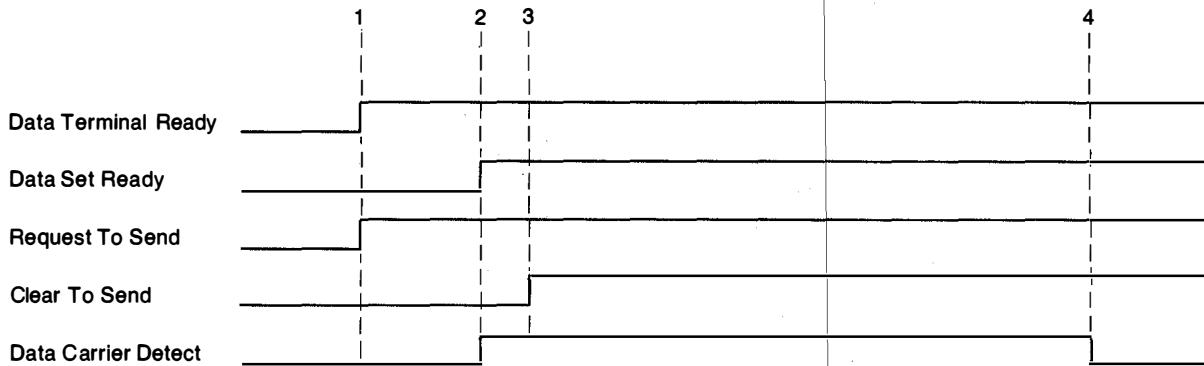
Cable Pin	Signal Name	Function
1	Protective Ground	Frame and ac power ground
2	Transmitted Data	Output data sent by the DTE device to the DCE device.
3	Received Data	Input data detected by the DCE device.
4	Request to Send	Indicates to the DCE device that the DTE device is ready to transmit data.
5	Clear to Send	Indicates to the DTE device that the DCE device is ready to accept data.
6	Data Set Ready	Indicates to the DTE device that the DCE device is not in test mode and power is on.
7	Signal Ground	Establishes reference point between the remote device and the terminal.
8	Data Carrier Detect	Indicates to the DTE that the DCE device is receiving carrier signals.
20	Data Terminal Ready	Indicates to the DCE that the DTE device is ready to transfer data.
23	Data Signal Rate Select	Selects one of two rates available on two speed modems.

You can control and monitor these modem signals by accessing interface control and status registers. See Registers, later in this section.

The next drawings show typical handshake sequences that occur between a modem and a terminal. Half and full-duplex handshakes are shown.

Establishing Full Duplex Connection

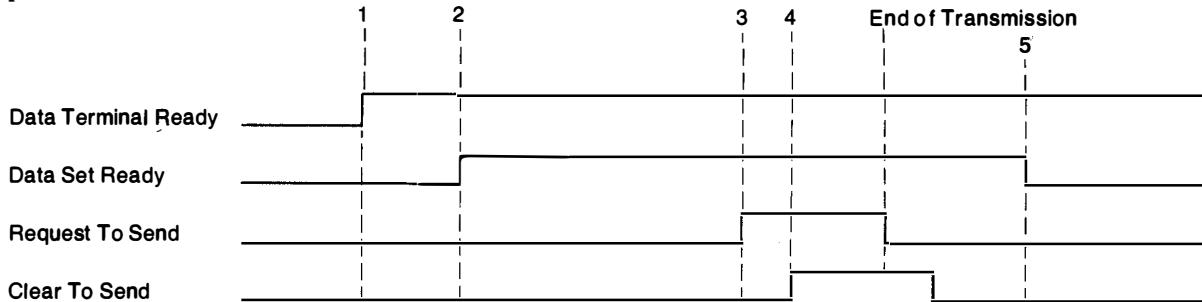
This drawing shows typical handshake sequences that occur when full duplex data transfers are implemented:



1. Interface sets Data Terminal Ready and Request To Send (Register 2, bits 0 and 1).
2. Interface monitors Data Set Ready and Data Carrier Detect responses. These signals are monitored from the program (Status Register 3, bits 1 and 3), or can be automatically monitored by the interface (Register 5, bits 1 and 3).
3. Interface monitors Clear To Send response. When Clear To Send becomes active (true), the interface transmits the output data to the remote. This signal is monitored from the program (Status Register 3, bit 0), or can be automatically monitored by the interface (Register 5, bits 0 and/or 4). The automatic monitoring feature enables you to suspend transmission or execute an auto-disconnect if Clear To Send, Data Set Ready, or Data Carrier Detect signals are lost. (goes false).
4. Auto disconnect for lost carrier.

Establishing Half Duplex Connection

This drawing shows typical handshake sequences that occur when half duplex data transfers are implemented:



1. Interface sets Data Terminal Ready (Register 2, bit 0).
2. Interface monitors Data Set Ready response (Status Register 3, bit 1).
3. Interface sets Request To Send. Automatic control of this signal can be specified (Control Register 16, bit 7).
4. Interface monitors Clear To Send response. When Clear To Send is active (true), the interface transmits the output data. Automatic control of this handshake sequence can be specified (Register 10, bit 4).
5. Auto disconnect when Data Set Ready signal is lost.

The modem handshake sequences described here are for the Option 001 interface. Similar automatic signal monitoring, auto-handshake and auto-disconnect features are provided for the Standard interface. See the appropriate registers for an explanation of these features.

Note: Full duplex transfers can not be implemented using ENTER and OUTPUT statements. Use interrupt TRANSFERs instead. This allows the serial I/O interface to interrupt the HP-85 when there is input data present, or to request data for transmission.

Auto-originate and Auto-answer Routines

This discussion shows typical auto-originate and auto-answer programming routines that are used when connecting to a remote computer. These routines should be added to the main program used to communicate with the remote computer. Contact your local HP-85 dealer or Sales and Service Office for information about terminal emulator programs.

Example 4 (Auto-originate)

```

10 ! =====
20 ! This routine controls a
30 ! modem for originating a
40 ! call.
50 !
60 ! Modem Lines:
70 !   DTR: Data Terminal Ready
80 !   RTS: Request to Send
90 !   DSR: Data Set Ready
100 !  DCD: Data Carrier Detect
110 !  CTS: Clear to Send
120 !
130 ! S=I/O Card Select Code
140 ! =====
150 !
160 ! Turn ON DTR & RTS.
170 !
180 ASSERT S>3
190 !
200 ! Establish Connection.
210 !
220 DISP "Dial Number - then 'CO
NT!''"
230 PAUSE
240 !
250 ! Enable autodisconnect.
260 ! If DSR, DCD or CTS drop
270 ! then autodisconnect will
280 ! turn OFF DTR & RTS and
290 ! generate an error.
300 !
310 CONTROL S,5 , 11
320 !
330 ! Transfer Data
340 !
350 ! ****
360 ! Add Data I/O routine here
370 ! ****
380 END

```

Example 5 (Auto-answer)

```

10 ! =====
20 ! This routine controls a
30 ! modem for autoanswer of an
40 ! incoming call.
50 !
60 ! Modem Lines:
70 !   DTR: Data Terminal Ready
80 !   RTS: Request to Send
90 !   DSR: Data Set Ready
100 !  DCD: Data Carrier Detect
110 !  CTS: Clear to Send
120 !
130 ! S=I/O Card Select Code
140 ! =====

```

```
150 !  
160 ! Turn ON DTR & RTS.  
170 !  
180 ASSERT S:3  
190 !  
200 ! Wait for incoming call  
210 ! (DSR 'ON')  
220 !  
230 STATUS S:3 ; M  
240 IF NOT BIT(M,1) THEN 230  
250 !  
260 ! Start 25 second connect  
270 ! timer and enable auto-  
280 ! disconnect. If DSR, OCD  
290 ! or CTS drop then auto-  
300 ! disconnect will turn OFF  
310 ! DTR & RTS and branch to  
320 ! line 590.  
330 !  
340 ON TIMER# 1,25000 GOTO 580  
350 ON INTR S GOTO 590  
360 ENABLE INTR S:4  
370 CONTROL S:5 ; 11  
380 !  
390 ! Check DCD line.  
400 !  
410 STATUS S:3 ; M  
420 IF NOT BIT(M,3) THEN 410  
430 !  
440 ! If DCD 'ON' THEN disable  
450 ! timer and transfer data.  
460 !  
470 OFF TIMER# 1  
480 ! *****  
490 ! Add Data I/O routine here  
500 ! *****  
510 !  
520 ! If timer times out before  
530 ! OCD turns ON then turn  
540 ! OFF DTR & RTS, disable  
550 ! autodisconnect, wait 5  
560 ! sec & setup for next call  
570 !  
580 ASSERT S:0  
590 CONTROL S:5 ; 0  
600 OFF TIMER# 1  
610 STATUS S:1 ; M  
620 WAIT 5000  
630 GOTO 180  
640 END
```

Registers

The serial I/O interface contains 24 distinct registers. These registers are accessed with STATUS or CONTROL statements. This discussion explains the contents of each register and shows the access procedures.

Register 0

To access this register execute:

```
STATUS 10,0;S
```

Register 0 is a read-only register. The value returned (always 2) indicates that this is the Serial I/O Interface.

Register 1

Interrupt Mask

Interrupt Mask							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Break Received	Framing Error	Parity Error	Received Data Available	DCD (Opt. 001) RTS (Standard)	Auto-disconnect	DSR (Opt. 001) DRS (Standard)	CTS (Opt. 001) DTR Standard
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 10,1;S1  
CONTROL 10,1,<value>
```

Register 1 is a read/write register. The value contained in this register determines the conditions that cause an interrupt to the HP-85. To specify interrupt conditions, execute the above CONTROL statement and specify the bit value (or values) for the desired interrupts. The value remains in this register until the interface is reset or until a new value is written to this register.

Reset default: 0 (all bits clear)

The meanings of the various bits are described next.

- Bit 7 set indicates that an interrupt is generated when a BREAK is detected in the input data. A BREAK is used to signal an interrupt to the remote device. A BREAK appears to the serial interface as a null character, CHR\$(0), and may cause parity and framing errors. Detecting a break may indicate that an erroneous null character has been input along with other valid data.
- Bit 6 set indicates that an interrupt is generated when a framing error is detected in the input data. A framing error may be caused by line transients or incorrectly specified stop bits or character length. The character that caused the framing error can be converted to a specific character if you desire (see Register 9).
- Bit 5 set indicates that an interrupt is generated when a parity error is detected in the input data. A parity error may be caused by line transients or when parity is incorrectly specified (see Register 4). The character that caused the parity error can be converted to a specific character if you so desire (see Register 9).
- Bit 4 set indicates that an interrupt is generated when there is received data in the interface input buffer. The data is then entered with an ENTER or TRANSFER statement.
- Bit 3 set indicates that the Data Carrier Detect (Option 001 cable) or Request To Send (Standard cable) modem signal has changed state. Note that the interrupt is generated when the modem signal changes from false to true or from true to false.
- Bit 2 set indicates that an interrupt is generated when an auto-disconnect is activated. An auto-disconnect is generated when specified modem signals are lost (see Register 5). Detecting an auto-disconnect when bit 2 is clear (0) generates error 115.
- Bit 1 set indicates that the Data Set Ready (Option 001 cable) or Data Rate Select (Standard cable) modem signal has changed state. Note that the interrupt is generated when the modem signal changes from false to true or from true to false.
- Bit 0 set indicates that the Clear To Send (Option 001 cable) or Data Terminal Ready (Standard cable) modem signal has changed state. Note that the interrupt is generated when the modem signal changes from false to true or from true to false.

Register 2

Modem Control Signals

Modem Control Signals							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Not Used	Not Used	Not Used	Not Used	DRS (Opt. 001)	RTS (Opt. 001)	DTR (Opt. 001)
					DSR (Standard)	DCD (Standard)	CTS (Standard)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 10,2;S2
CONTROL 10,2;<value>
ASSERT 10;<value>
```

Register 2 is a read/write register. The value contained in this register controls the state of the interface control signals shown here. The state of these control signals is returned when the STATUS statement is executed. To change these signals, execute the above CONTROL or ASSERT statement and specify the bit value (or values) to set the desired lines. Executing the ASSERT statement immediately sets the modem signals as specified, regardless of any I/O operations that may be occurring. The value specified remains in this register until the interface is reset or until a new value is written to this register.

Reset default: 0 (all bits clear)

The meanings of the various bits are described next.

- Bits 7 through 3 are not used.
- Bit 2 set activates the Data Rate Select (Option 001 cable) or Data Set Ready (Standard cable) modem signal. To deactivate this signal, clear this bit (set to 0).
- Bit 1 set activates the Request To Send (Option 001 cable) or Data Carrier Detect (Standard cable) modem signal. To deactivate this signal, clear this bit (set to 0).
- Bit 0 set activates the Data Terminal Ready (Option 001 cable) or Clear To Send (Standard cable) modem signal. To deactivate this signal clear this bit (set to 0).

Control Register 3

To access this register execute:

```
CONTROL 10,3;<value>
```

Control register 3 is a write-only register. The value contained in this register determines the baud rate for transmitted and received data. This register selects STANDARD baud rates only. Non-standard baud rates are specified by registers 6 and 7. To specify a standard baud rate, execute the above CONTROL statement and specify the desired value as shown in the table below. The value remains in this register until the interface is reset or until a new value is written to this register.

Reset default: 6 (300 baud)

Value	Rate Specified	Value	Rate Specified
0	50	8	1200
1	75	9	1800
2	110	10	2000
3	134.5	11	2400
4	150	12	2600
5	200	13	4800
6	300	14	7200
7	600	15	9600

Status Register 3

Modem Status and Cable Option

Modem Status and Cable Option							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Not Used	Not Used	Not Used	DCD (Opt. 001) RTS (Standard)	Cable Type	DSR (Opt. 001) DRS (Standard)	CTS (Opt. 001) DTR (Standard)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

STATUS 10,3;63

Status register 3 is a read-only register. The value returned from this register indicates the status of the specified modem signals and the type of cable installed on the interface.

Reset Default: None

The meanings of the various bits are described next.

- Bits 7 through 4 are not used.
- Bit 3 set indicates that the Data Carrier Detect (Option 001 cable) or Request To Send (Standard cable) modem signal is active (true). Bit 3 clear (0) indicates an inactive modem signal.
- Bit 2 set indicates that the interface has the standard cable installed. Bit 2 (0) clear indicates an Option 001 cable.
- Bit 1 set indicates that the Data Set Ready (Option 001 cable) or Data Rate Select (Standard cable) modem signal is active (true). Bit 1 clear (0) indicates an inactive modem signal.
- Bit 0 set indicates that the Clear To Send (Option 001 cable) or Data Terminal Ready (Standard cable) modem signal is active (true). Bit 0 clear (0) indicates inactive modem signal.

Register 4

Line Characteristics

Line Characteristics							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Set Break	Force Parity	Odd/Even Parity	Enable Parity	Stop Bits	Character Length	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 10,4:64
CONTROL 10,4:<value>
```

Register 4 is a read/write register. The value contained in this register specifies character length, stop bits, parity, and forces a BREAK to be transmitted. To change these character definitions, execute the above CONTROL statement and specify the bit value (or values) as desired. The value remains in this register until the interface is reset or until a new value is written to this register. A BREAK is sent by setting bit 6 with a CONTROL statement or by executing the REQUEST statement. The REQUEST statement causes the transmit line to be held in the space condition for the amount of character times specified for the value parameter, followed by holding the transmit line in the mark condition for at least 5 character times. When using the CONTROL statement to set a BREAK, the line remains in the space condition until bit 6 is cleared.

Reset default: 10 (seven bits, odd parity, one stop bit)

The meanings of the various bits are described next.

- Bit 7 is not used.
- Bit 6 set causes a BREAK signal to be output to the remote device. A BREAK forces the transmit line to the space condition. The transmit line remains in the space condition until bit 6 is cleared.
- Bits 5, 4, and 3 are defined in the following table.

Value	Bits			Parity Specified
	5	4	3	
0	0	0	0	No parity bit
8	0	0	1	Odd parity
24	0	1	1	Even parity
40	1	0	1	Always 1
56	1	1	1	Always 0

- Bit 2 set specifies two stop bits. Bit two clear (0) specifies one stop bit.

- Bits 1 and 0 specify character length as shown in the following table:

Value	Bits		Character Length
	1	0	
0	0	0	5
1	0	1	6
2	1	0	7
3	1	1	8

The next table shows how to select the value for the CONTROL statement to specify combinations of character length, parity, and stop bits.

Bits/Character	Parity Specifier				
	None	Odd	Even	1	0
5	0	8	24	40	56
6	1	9	25	41	57
7	2	10	26	42	58
8	3	11	27	43	59

For example, you want to select 7 bits per character, odd parity. From the table, you find the value 10 specifies this configuration. The statement required to specify this configuration is:

CONTROL 10,5;10

Note: The table shown above specifies 1 stop bit. To specify 2 stop bits, add 4 to the values found in the table.

Register 5

Modem Features

Modem Features							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Not Used	Receive Handshake	Transmit Handshake	DCD (Opt. 001) RTS (Standard)	Not Used	DSR (Opt. 001) DRS (Standard)	CTS (Opt. 001) DTR (Standard)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 10,5;S5
CONTROL 10,5; <value>
```

Register 5 is a read/write register. The value contained in this register determines the auto-handshake and auto-disconnect features that are specified. To change the specifications, execute the above CONTROL statement and specify the bit value (or values) as desired. The value remains in this register until the interface is reset or until a new value is written to this register. Note that bits 5 and 4 can also be controlled by the reset defaults (see the Installation and Theory of Operation Manual).

Reset default: 0 (all bits clear)

The meanings of the various bits are described next.

- Bits 7 and 6 are not used.
- Bit 5 set enables the receive auto-handshake feature. This feature discards received data and character status information if the Data Carrier Detect (Option 001 cable) or Request To Send (Standard cable) modem signals are inactive. Bit 5 clear (0) disables this feature.
- Bit 4 set enables the transmit auto-handshake feature. This feature suspends data transmission when the Clear to Send (Option 001 cable) or Data Terminal Ready (Standard cable) modem signals are inactive. Data transmission resumes when this bit is clear (0). See Printer Interfacing for an example use of this feature.
- Bit 3 set enables the auto-disconnect feature. An auto-disconnect is generated when the Data Carrier Detect (Option 001 cable) or Request To Send (Standard cable) modem signal becomes inactive. Auto-disconnect generates error 115 unless an interrupt is specified (see register 2, bit 2). Bit 3 clear (0) disables this feature.
- Bit 2 is not used.
- Bit 1 set enables the auto-disconnect feature. An auto-disconnect is generated when the Data Set Ready (Option 001 cable) or Data Rate Select (Standard cable) modem signal becomes inactive. Auto-disconnect generates error 115 unless an interrupt is specified (see register 2, bit 2). Bit 1 clear (0) disables this feature.

- Bit 0 set enables the auto-disconnect feature. An auto-disconnect is generated when the Clear To Send (Option 001 cable) or Data Terminal Ready (Standard cable) modem signal becomes inactive. Auto-disconnect generates error 115 unless an interrupt is specified (see register 2, bit 2). Bit 0 clear (0) disables this feature.

Register 6

To access this register execute:

```
STATUS 10,6;S6
CONTROL 10,6;<value>
```

Register 7

To access this register execute:

```
STATUS 10,7;S7
CONTROL 10,7;<value>
```

Registers 6 and 7 are read/write registers. The value contained in these registers indicates the value specified for the transmit and receive data transfer rates. These registers are used to specify non-standard baud rates (see register 3 for standard baud rate specifiers). The values for these registers are determined as follows:

1. Rate = 115 200/Divisor
2. Divisor = ((Register 6 value × 256) + (Register 7 value))

The default values for these two registers specify 300 baud (see register 3). These default values are 1 for register 6 and 128 for register 7. Entering these values in to step 2 we obtain the following results:

$$\begin{aligned} \text{Divisor} &= ((1 \times 256) + (128)) = 384 \\ \text{Rate} &= 115\,200/384 = 300 \end{aligned}$$

To specify a non-standard baud rate, the values for registers 6 and 7 are determined as follows:

1. Divisor = 115 200/Rate
2. Register 7 value = (Divisor) MOD 256
3. Register 6 value = (Divisor) DIV 256

For example, you want to specify a baud rate of 275 bits per second. Calculate the register values as follows:

1. Divisor = 115 200/275 = 418
2. Register 7 value = (418) MOD 256 = 162
3. Register 6 value = (418) DIV 256 = 1

To specify a baud rate of 275 execute the following CONTROL statement to registers 6 and 7:

```
CONTROL 10,6;1,162
```

Reset defaults:

```
Register 6 = 1
Register 7 = 128
```

Register 8

To access this register execute:

```
STATUS 10,8;S8
CONTROL 10,8;<value>
```

Register 8 is a read/write register. The value contained in this register is the decimal value of the ASCII character that is specified as the parity and framing error replacement character. When a parity or framing error is detected in the input data, the character that contained the error can be replaced with the character in this register. This feature is used to flag errors and simplify troubleshooting. This feature is enabled by Bit 4, Register 9.

Reset default : 0 (all bits clear)

Register 9

Transmitter/Receiver Control

Transmitter/Receiver Control							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Transmitter	Strip Received Rubouts	Strip Received Nulls	Change Character if Error	Set Bit 7 of Character if Error	Reset Receive Queue	Auto-echo Enable	Enable Receiver
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 10,9;S9
CONTROL 10,9;<value>
```

Register 9 is a read/write register. This register is the primary control register for the interface transmitter and receiver. The value contained in this register indicates the state of the various control signals. To change these control signals, execute the above CONTROL statement and specify the bit value (or values) to activate the desired control signals. The value remains in this register until the interface is reset or until a new value is written to this register. Executing the RESUME statement immediately enables the transmitter (sets bit 7). The RESUME statement may be executed concurrent with active transfers.

Reset default: 137 (bits 0, 3, and 7 set)

The meanings of the various bits are described next.

- Bit 7 set enables the data transmitter. When bit 7 is clear, only echo data is transmitted (see bit 1). Bit 7 may be controlled by the XON/XOFF feature provided by Control Register 11.
- Bit 6 set allows received rubout characters, CHR\$(127), to be stripped from incoming data. This feature is used when entering data from a device that sends extra rubout characters to avoid buffer overruns. Bit 6 clear (0) disables this feature.
- Bit 5 set allows received null characters, CHR\$(0), to be stripped from incoming data. This feature is used when entering data from a device that sends extra null characters to avoid buffer overruns. Bit 5 clear (0) disables this feature.
- Bit 4 set enables replacement of characters received with parity or framing errors. The replacement character is defined by register 8. Bit 4 clear (0) disables this feature.
- Bit 3 set enables the underline feature for parity and framing errors. This feature is only valid when no replacement character is specified for register 8. This feature sets bit 7 of register 8. This bit set causes all characters with parity or framing errors to be underlined when they are displayed on the internal CRT or are printed on the internal printer. Bit 3 clear (0) disables this feature.
- Bit 2 set clears the receive data queue. When the receive data queue is cleared, this bit is automatically reset by the interface. All data in the receive queue is lost when this feature is activated. Ensure that values written to this register are correct before you execute the CONTROL statement.
- Bit 1 set enables the auto-echo feature. This feature causes all received characters to be retransmitted to the sending device. Bit 1 clear (0) disables this feature.
- Bit 0 set enables the receiver. All input data, including status, is entered into the interface input buffer. Bit 0 clear (0) disables the receiver. All input data, including status and BREAK is ignored.

Control Register 10

Control register 10 is not implemented. Executing a CONTROL statement to this register generates error 111.

Status Register 10

Line Status

Line Status							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Not Used	Transmit Register Empty	Break Received	Framing Error	Parity Error	Not Used	Received Data Available
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 10,10;S0
```

Status register 10 is a read-only register. The value contained in this register provides information about each character as it is received. This status information is entered into the receive queue in synchronization with each character.

Reset default: 0 (all bits clear)

The meanings of the various bits are described next.

- Bits 7 and 6 are not used.
- Bit 5 set indicates that the transmitter is ready to accept another character from the HP-85. This does NOT mean that all data has been transmitted to the remote device. The data may still be in the interface serial output register.
- Bit 4 set indicates that a BREAK from the remote has been detected.
- Bit 3 set indicates that a framing error has been detected for at least one input character. Framing errors usually indicate a line transient, an incorrect number of stop bits, or an incorrect number of bits per character.
- Bit 2 set indicates that a parity error has been detected for at least one input character. Parity errors indicate a line transient, an incorrect number of bits per character, or an incorrect parity specifier.
- Bit 1 is not used.
- Bit 0 set indicates that received data is available in the input queue. The received data should be entered into the HP-85.

Control Register 11

Input Data Control

Input Data Control							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Transmit Flag XON	Disable Transmit Flag XOFF	Not Used	Terminate if CR15 (see CR15)	Terminate if CR14 (see CR14)	Terminate if CR13 (see CR13)	Terminate if CR12 (see CR12)	Not Used
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
CONTROL 10,11,:<value>
```

This register specifies the input terminator character location. This register also provides control of the transmitter enable flag when input terminator characters are detected. To specify the input terminator character location, or to control the transmitter enable flag, execute the above CONTROL statement and specify the bit values for the desired characters or flag state. The value remains in this register until the interface is reset or until a new value is written to this register.

Reset default : 0 (all bits clear)

The meanings of the various bits are described next.

- Bit 7 set specifies that the transmitter enable flag is set when the character defined by Control Register 15 is detected in the input data stream (XON).
- Bit 6 set clears the transmitter enable when the character defined by Control Register 14 is detected in the input data stream (XOFF).
- Bit 5 is not used.
- Bit 4 set specifies that input operations are terminated when the character defined by Control Register 15 is detected in the input data stream.¹
- Bit 3 set specifies that input operations are terminated when the character defined by Control Register 14 is detected in the input data stream.¹
- Bit 2 set specifies that input operations are terminated when the character defined by Control Register 13 is detected in the input data stream.¹
- Bit 1 set indicates that input operations are terminated when the character defined by Control Register 12 is detected in the input data stream.¹
- Bit 0 is not used.

¹ Input operations can ONLY be terminated by these characters if the % image specifier is used.

Status Register 11

I/O Termination Cause

I/O Termination Cause							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
End of Output Data List	End of Input Data List	Transfer Count Expired	CR15 Character Received	CR14 Character Received	CR13 Character Received	CR12 Character Received	DELIM Character Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 10,11;R1
```

Status register 11 is a read-only register. The value returned from this register indicates the reason for terminating an I/O operation. When multiple termination conditions are specified, this register is read to determine the I/O termination cause.

Reset default : 0 (all bits clear)

The meanings of the various bits are explained next.

- Bit 7 set indicates that the I/O operation was terminated by the HP-85 at the end of the output list.
- Bit 6 set indicates that the I/O operation was terminated by the HP-85 when the input list was filled.
- Bit 5 set indicates that the input TRANSFER operation was terminated by satisfying the COUNT parameter.
- Bit 4 set indicates that the input operation was terminated when the character defined by Control Register 15 was detected in the input data stream.
- Bit 3 set indicates that the input operation was terminated when the character defined by Control Register 14 was detected in the input data stream.
- Bit 2 set indicates that the input operation was terminated when the character defined by Control Register 13 was detected in the input data stream.
- Bit 1 set indicates that the input operation was terminated when the character defined by Control Register 12 was detected in the input data stream.
- Bit 0 set indicates that an input TRANSFER operation was terminated when the DELIM character was detected in the input data stream.

Control Registers 12 through 15

Control Registers 12, 13, 14, and 15 contain the termination characters that are specified by Control Register 11. The transmit enable flag is also controlled by the characters specified by Control Registers 14 and 15. To define termination characters for these registers, execute a CONTROL statement to the desired register and specify the decimal value of the ASCII character defined as the termination character.

To access these registers execute:

```
CONTROL 10,12; <value>
CONTROL 10,13; <value>
CONTROL 10,14; <value>
CONTROL 10,15; <value>
```

Control Register 16

Output EOL Sequence

Output EOL Sequence							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Auto RTS Enable	EOL Transmit Disable	Six Bit EOL Character Count					
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
CONTROL 10,16; <value>
```

Control register 16 is a write-only register. This register contains the EOL sequence character count, the EOL transmitter control, and the auto Request To Send enable. The EOL is a character or sequence of characters that is output at the end of each line of data (e.g., CR/LF). To change the EOL character count or to enable the auto Request To Send or EOL transmit disable, execute the above CONTROL and specify the bit value as desired. The new value remains in this register until the interface is reset or until a new value is written to this register.

Reset default: 2 (bit 1 set)

The meanings of the various bits are described next.

- Bit 7 set enables the auto Request To Send feature. This feature activates the Request To Send modem signal at the beginning of each transmission. When the EOL sequence is transmitted (end of TRANSFER or OUTPUT statement) the Request To Send signal is set to the inactive state. Bit 7 clear disables this feature.
- Bit 6 set clears the transmitter enable flag after all output data (including EOL sequences) has been transmitted. This flag must be reenabled from the program or by detecting a special received character (see Control Register 11) before transmission can resume. This feature is used with ENQuire/ACKnowledge handshakes. Bit 6 clear (0) disables this feature.
- Bits 5 through 0 contain the EOL character count value. This value specifies how many characters are defined for the EOL sequence. This value ranges from 0 through 63. Long EOL sequences are normally used with slow printing devices such as teletypes to allow the print mechanism sufficient time to execute a Carriage-Return/Line-Feed before sending more data from the interface. The characters that make up the actual End Of Line sequence are contained in Control Registers 17 through 23. The EOL character count parameter and the actual character sequences are explained following Control Registers 17 through 23.

Control Registers 17 through 23

Control Registers 17 through 23 contain the user defined output EOL sequence. You define the EOL sequence by specifying a maximum of seven separate characters. The EOL characters are defined by executing a CONTROL statement to the desired register and specifying the decimal value of the desired ASCII character.

To access these registers execute:

```
CONTROL 10,17; <value>
CONTROL 10,18; <value>
CONTROL 10,19; <value>
CONTROL 10,20; <value>
CONTROL 10,21; <value>
CONTROL 10,22; <value>
CONTROL 10,23; <value>
```

Reset defaults:

- Register 17 - CR (CHR\$(13))
- Register 18 - LF (CHR\$(10))
- Registers 19 through 23 - Null (CHR\$(0))

The default value for Control Register 16 is 2. This specifies a two character EOL sequence. This default and the default for Control Registers 17 and 18 mean that the default EOL sequence is a CR/LF. The EOL sequence definition is changed as shown next.

Sequence	Control Reg 16	Control Regs 17 through 23
CR-LF-Null	3	Reg 17 = 13 Reg 18 = 10 Reg 19 = 0
CR-LF-Null Null-RO-RO	6	Reg 17 = 13 Reg 18 = 10 Reg 19 = 0 Reg 20 = 0 Reg 21 = 255 Reg 22 = 255
CR-LF-Null-Null- RO-RO-followed by 15 Nulls	21 ¹	Reg 17 = 13 Reg 18 = 10 Reg 19 = 0 Reg 20 = 0 Reg 21 = 255 Reg 22 = 255 Reg 23 = 0

¹ When the value for Register 16 exceeds 7, the characters defined by Registers 17 through 23 are output first, followed by continuous Register 23 characters until the count value is satisfied.

Troubleshooting Hints

Problem	Probable Cause
OUTPUT and ENTER operations hang.	Ensure auto-handshake feature is disabled if not required.
Random unintelligible data. OUTPUT completes, but ENTER does not.	Incorrect Baud rate specified, or mismatched data codes. The internal data code of the HP-85 is ASCII. Use conversion tables if other codes are used. May also be caused by incorrect stop bits or bits per character specified.
Input data has underlines when printed or displayed.	Underlining is the default method used to indicate parity or framing errors.
All input characters are underlined.	Incorrect parity specified.
Approximately half of input is underlined.	Check for correct bits per character and stop bits specified. May also be caused when parity is specified for the interface and is not used by the remote.
Overwritten data on slow printers.	Increase the number of EOL "Null" characters that are output. This allows the print mechanism time to execute a CR/LF sequence.
Remote loses data.	Remote may require handshakes to prevent buffer overrun.
Single Character OK, but multiple characters in succession produce garbage.	Incorrect stop bits or bits per character specified.

Serial I/O Statements

Statement	Description
ABORTIO	Aborts any transfers in progress, deactivates modem lines.
ASSERT	Executes an immediate write to Control Register 2. May be executed while active data transfers are in process.
CONTROL	Writes to the Control Registers in the interface.
ENABLE INTR	Writes interrupt mask to Control Register 1.
ENTER	Enters data from the interface into the BASIC program.
HALT	Aborts any transfers in progress, does not deactivate modem lines.
OUTPUT	Transfers data from the BASIC program to the interface for transmission to the remote.
REQUEST	Transmits a BREAK to the remote. Optional parameter specifies length of BREAK signal.
RESET	Resets the interface, disconnects the modem lines, sets the interface to the reset defaults, and runs the internal self test.
RESUME	Enables the interface transmitter.
SEND	Outputs data as bytes, and sends EOL sequence if specified.
STATUS	Enters the values contained in the interface status registers.
TRANSFER	Enters received data into a buffer. Outputs data from a buffer to the interface.

Serial Interface Errors

Error No.	Meaning	Possible Cause
113	UART receiver overrun; data has been lost.	Data rate too high.
114	Receiver buffer overrun; data has been lost.	Data rate too high. Handshake needed.
115	Automatic disconnect forced.	Loss of modem signals.

Section 14
Using the BCD Interface

Section Introduction

The 82941A Interface enables your HP-85 to communicate with a variety of instruments that present data in Binary Coded Decimal (BCD) format. This section explains how to program the interface.

The 82941A Interface Installation and Theory of Operation Manual explains:

- How to set the interface select code.
- How to set the interface default switches.
- How to wire the interface cables.
- How to install the interface in your HP-85.

You should read and understand the Installation and Theory of Operation Manual before proceeding with the topics covered in this section.

Binary Coded Decimal

Binary Coded Decimal is, as the name implies, a method of encoding the decimal digits (0 through 9) in a four bit format. The following table shows the binary coding of the ten decimal digits and six additional ASCII characters that are allowed. The interface can be programmed through default switches or program statements to recognize either positive-true or negative-true logic. The table shows the allowable ASCII characters and the associated BCD codes for each logic sense.

BCD Coding

ASCII	Positive True	Negative True
0	0000	1111
1	0001	1110
2	0010	1101
3	0011	1100
4	0100	1011
5	0101	1010
6	0110	1001
7	0111	1000
8	1000	0111
9	1001	0110
:	1010	0101
:	1011	0100
<	1100	0011
=	1101	0010
>	1110	0001
?	1111	0000

Data and Handshake Lines

Data and handshake lines are provided as follows:

- Data - 44 bi-directional data lines organized as 11 four-bit ports (P0 through P10).
- Sign Bits - four sign bits (S1 through S4) through port P11.
- Handshake - four output handshake lines (I/OA, I/OB, CTLA, CTLB).
 - two input handshake lines (FLGA, FLGB).

Operating Modes

The operating mode refers to selecting the fields as shown in the next table.

Operating Mode

Field	Range
Number of channels	A, B, or both.
Channel Direction	Input or output.
Number of mantissa digits in each channel ¹	0 through 11.
Number of exponent digits for each channel ¹	0 through 3.
Number of function digits in each channel ¹	0 through 11.
Decimal point placement for each mantissa	May be placed before any mantissa digit.
Logic sense of all signals	Positive or negative true.
Handshake triggering	Leading or trailing edge.

Default Formats

The BCD interface provides two standard default formats: single and dual channel. These formats are selected by changing the reset default switch (S2) switch 1 or by program control. The default formats and their associated port assignments are shown in the next tables.

Single Channel Format (S2 switch 1 = "0" position)

Number of Ports	Ports (or bits) Used	Data	Handshake
8	P0 through P7	Mantissa	Channel A
1	P8	Exponent	Channel A
1	P9	Function	Channel A
1/2	P11(bits S1 and S2)	Sign Bits	Channel A

¹ The combined sum of mantissa, function, and exponent digits specified cannot exceed 11.

Dual Channel Format (S2 switch 1= "1" position)

Number of Ports	Ports (or bits) Used	Data	Handshake
4	P0 through P3	Mantissa	Channel A
1	P4	Function	Channel A
4	P5 through P8	Mantissa	Channel B
1	P9	Function	Channel B
1/2	P11 (bits S1 and S2)	Sign Bits	Channel A
1/2	P11 (bits S3 and S4)	Sign Bits	Channel B

Examples are provided following the discussion of the interface registers to show how to select these and other formats with program statements.

Data Rates

The data transfer rate depends upon the direction of the transfer, the type of transfer (normal, fast handshake, or interrupt), concurrent operations taking place and peripheral timing. The maximum data rates that the interface can handle are as follows:

Transfer Rates

Transfer Type	Input	Output
Fast Handshake	20k bytes/sec	22k bytes/sec
Normal ENTER	7k bytes/sec	4k bytes/sec
Interrupt	400 bytes/sec	400 bytes/sec

The data rates shown for the fast handshake mode are achieved by placing the following restrictions on the fast handshake mode:

1. Standard Format
 - Channel A only (one peripheral).
 - Eight mantissa digits only.
 - One exponent digit only.
 - Function digit may be selected but is ignored.
 - Mantissa and exponent signs will be transferred.
2. Byte count must be specified on an input.
3. Positive true logic on data lines.
4. Trailing edge handshake only.
5. Decimal points may be selected but will not be transferred.
6. ASCII codes 0 through 32 and ASCII code 44 (comma) cannot be used for output.

If the peripheral fails to complete a handshake during the fast handshake mode, the CPU in the HP-85 halts and the system hangs up. To recover, the FLGB line, unused in the fast handshake mode format, may be used to indicate errors by the peripheral. If FLGB goes true while the interface is waiting for a handshake, the fast handshake mode is terminated and Error 115 is displayed.

Program Statements

The following program statements are implemented by the BCD interface card. See the syntax reference for complete description of these statements.

ABORTIO ¹	ENABLE INTR	REMOTE	SEND TALK
ASSERT ¹	ENTER	RESET	SEND LISTEN
HALT ¹	OUTPUT	SEND CMD	STATUS ¹
CONTROL		SEND DATA	TRANSFER

Using the Interface

This discussion assumes that you have read the BCD Installation and Theory of Operation Manual.

Programming With Default Formats

This discussion shows how to program the interface with the single or dual channel defaults formats that are provided. Factory settings are assumed for the interface select code and the default switches. The factory settings are:

- Interface Select Code (S1) = 3 (all set to the "0" position).
- Reset Default Switches (S2) = all set to the "0" position.

Example 1

Run the following program:

```

10 RESET 3 ! RESET INTERFACE
20 STATUS 3,0 ; R0,R1,R2,C3,C4,C5,C6,C7,C8,C9,C0
30 IMAGE K,X,00,/ ! PRINT FORMAT
40 PRINT USING 30 ; "READ REG 0=";R0;"READ REG 1=";R1
50 PRINT USING 30 ; "READ REG 2=";R2
60 PRINT USING 30 ; "REG 3=";C3;"REG 4=";C4;"REG 5=";C5
70 PRINT USING 30 ; "REG 6=";C6;"REG 7=";C7;"REG 8=";C8
80 PRINT USING 30 ; "REG 9=";C9;"REG 10=";C0
90 END

```

¹ Executing this statement interrupts any I/O operation that may be in progress.

Here is the printout:

```
READ REG 0= 3
READ REG 1= 0
READ REG 2= 0
REG 3= 8
REG 4= 1
REG 5= 1
REG 6= 0
REG 7= 0
REG 8= 0
REG 9= 0
REG 10= 0
```

- Interface I.D.
- Interrupt Mask
- Handshake Lines
- Mantissa Digits Specified
- Exponent Digits Specified
- Function Digits Specified
- Decimal Point Location
- Handshake Logic Sense
- Data Logic Sense
- Function Logic Sense
- Signs and Port 10 Logic Sense

Note that read register 0 and registers 3, 4, and 5 are not zero. Read register 0 identifies the interface as the BCD interface. Register 3 indicates the number of mantissa digits to be input (in this case eight). Register 4 indicates the number of exponent digits (one) to be input and register 5 indicates the number of function digits (one) to be input. Registers 7 through 10 indicate the logic sense of the various data and handshake lines (the value of 0 indicates positive true logic sense). Read registers 1 and 2 and register 6 are not used for this example and are discussed later in this section.

Example 2

Assume that you are connected to a digital multimeter. The multimeter outputs eight significant data digits, a one-digit exponent, and a one-digit function value. All logic senses (data, handshake, etc.) are defined as positive true.

The following program reads the data from the multimeter and prints the results on the HP-85 printer.

```
10 RESET 3 ! RESET INTERFACE
20 ENTER 3 ; D1,F1 ! GET CHANNEL A DATA
30 PRINT "READING=",D1
40 PRINT "FUNCTION=",F1
50 STOP
```

Here is a typical printout:

```
READING= 1250.524
FUNCTION= 1
```

Example 3

The function value is used to indicate the position of the function switch (ohms, volts, amps, etc.) located on the multimeter. If the function values are defined as:

- 1 = Ohms
- 2 = Volts AC
- 3 = Amps
- 4 = Volts DC

Run the following program:

```

10 RESET 3 ! RESET INTERFACE
20 ENTER 3, D1,F1 ! GET CHANNEL A DATA
30 ON F1 GOSUB 70,90,110,130 ! CHANNEL A BRANCHES
40 PRINT "READING=";D1,Z$
50 PRINT "FUNCTION=";F1
60 STOP
70 Z$="OHMS" ! FUNCTION =1
80 RETURN
90 Z$="VOLTS-AC" ! FUNCTION =2
100 RETURN
110 Z$="AMPS" ! FUNCTION =3
120 RETURN
130 Z$="VOLTS-DC" ! FUNCTION =4
140 RETURN

```

Here are typical results:

```

READING= 1250.524 OHMS
FUNCTION= 1

```

Example 4

The BCD interface can be connected to two BCD devices simultaneously. Set the reset default switch (S2) switch 1 to the "1" position. This changes the default mode of the interface to two channel operation. Run the following program:

```

10 RESET 3 ! RESET INTERFACE
20 STATUS 3,0 ; R0,R1,R2,C3,C4,C5,C6,C7,C8,C9,C0
30 IMAGE K,X,DD,/ ! PRINT FORMAT
40 PRINT USING 30 ; "READ REG 0=";R0,"READ REG 1=";R1
50 PRINT USING 30 ; "READ REG 2=";R2
60 PRINT USING 30 ; "REG 3=";C3,"REG 4=";C4,"REG 5=";C5
70 PRINT USING 30 ; "REG 6=";C6,"REG 7=";C7,"REG 8=";C8
80 PRINT USING 30 ; "REG 9=";C9,"REG 10=";C0
90 END

```

Here are the results:

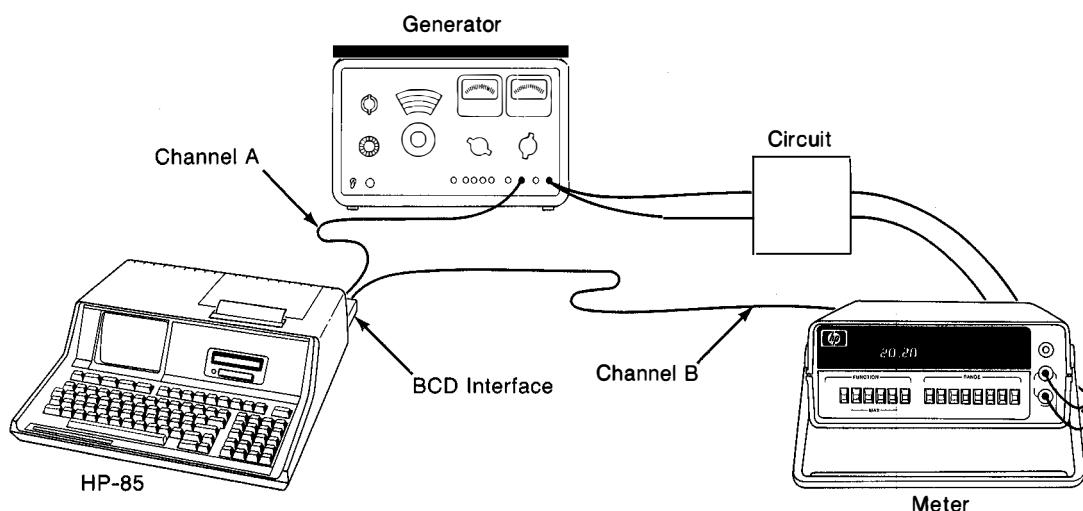
```
READ REG 0= 3
READ REG 1= 0
READ REG 2= 0
REG 3= 68
REG 4= 0
REG 5= 17
REG 6= 0
REG 7= 0
REG 8= 0
REG 9= 0
REG 10= 0
```

- Interface I.D.
- Interrupt Mask
- Handshake Lines
- Mantissa Digits Specified
- Exponent Digits Specified
- Function Digits Specified
- Decimal Point Location
- Handshake Logic Sense
- Data Logic Sense
- Function Logic Sense
- Signs and Port 10 Logic Sense

Note that registers 3, 4, and 5 values are different than those obtained for example 1.

The value (68) for register 3 indicates that four mantissa digits are reserved for channel A input and four mantissa digits are reserved for channel B input. The value (0) for register 4 indicates that neither channel is using an exponent digit. The value (17) for register 5 indicates that each channel has one digit reserved for input function data.

For example, the interface might be connected to a signal generator and a digital voltmeter:



Example 5

The signal generator is driving a circuit under test for frequency response. The voltmeter takes readings of the circuit output. By comparing the voltage vs. frequency readings, the bandpass of the circuit is determined.

Here is an example program. Typical frequency and voltage parameters are used.

```

10 ENTER 3 : A1,F1,B2,F2 ! GET CHANNEL A AND B DATA
20 DISP A1,F1,B2,F2 ! DISPLAY DATA
30 ON F1 GOSUB 80,120,160 ! CHANNEL A FUNCTION BRANCHES
40 ON F2 GOSUB 100,140,180 ! CHANNEL B FUNCTION BRANCHES
50 PRINT A1;Z$
60 PRINT B2;Y$
70 STOP
80 Z$$="Hz" ! A FUNCTION =1
90 RETURN
100 Y$$="Microvolts" ! B FUNCTION =1
110 RETURN
120 Z$$="KHz" ! A FUNCTION =2
130 RETURN
140 Y$$="Millivolts" ! B FUNCTION =2
150 RETURN
160 Z$$="MHz" ! A FUNCTION =3
170 RETURN
180 Y$$="Volts" ! B FUNCTION =3
190 RETURN

```

Run the program. Here is a typical result:

```

1000 Hz
2020 Microvolts

```

Partial Fields

The examples shown thus far enter the input data from the interface in the following order: mantissa A, function A, mantissa B, function B. You cannot enter a portion (or field) of data (e.g., mantissa B) without first entering all preceding data. Partial field addressing is provided to enable you to access only the data that you desire. Partial field addressing is accomplished by specifying primary addresses 00 through 06. The partial field specifiers and the data that they access are shown in the next table. Interface select code 3 is assumed.

Partial Field Specification

Device Selector	Data Entered
300	All data (Default)
301	Channel A mantissa, exponent, and function.
302	Channel B mantissa, exponent, and function.
303	Channel A mantissa and exponent.
304	Channel B mantissa and exponent.
305	Channel A function.
306	Channel B function.

Example 6

Run the previous program (example 5) and observe the results. Execute the following statements and observe the results shown here.

ENTER 300;A1,F1,B2,F2	
PRINT A1;F1;B2;F2	1000 1 2020 1
ENTER 301;A1,F1	
PRINT A1;F1	1000 1
ENTER 302;B2,F2	
PRINT B2;F2	2020 1
ENTER 303;A1	
PRINT A1	1000
ENTER 304;B2	
PRINT B2	2020
ENTER 305;F1	
PRINT F1	1
ENTER 306;F2	
PRINT F2	1

Once a partial field has been specified, that field remains in effect until a new field is selected or the interface is reset.

Partial fields are addressed by ENTER, OUTPUT, SEND TALK, SEND LISTEN and SEND CMD statements.

Port 10

The previous examples show how to take readings from external devices. The operator is required to manually set the instrument functions (frequency, function, etc.). The BCD interface provides a special port to allow control of external instruments from the program. This discussion explains how to use port 10 for output.

In the two channel example (example 5), the BCD interface ports are dedicated as follows:

- Ports P0 through P3 - Signal generator frequency reading.
- Port P4 - Signal generator function (range switch).
- Ports P5 through P8 - Voltmeter reading.
- Port P9 - Voltmeter function (voltage range switch).
- Port 10 - Unused.

Port 10 is a special purpose port that can be used as an input or output port. Port 10 is the only port that can be used as an output port without setting the reset default switch to the output enable position. Port 10 is accessed by performing a control operation to the four least significant bits of write register 2. For example: CONTROL 3,2;15 sets all four bits of port 10. The next example assumes the same instrument configuration and function values as those used for example 5.

The function values returned are the same as defined for that example.

The following assumptions about the signal generator are made:

- The generator sweep values range from 1 through 1000.
- A bit can be set to begin the sweep at 1.
- The generator has three ranges.
- Each range can be externally selected.

Here is the bit mask for the generator:

Bit 3	Bit 2	Bit 1	Bit 0
1 = MHz Range	1 = kHz Range	1 = Hz Range	1 = Start Sweep
Value = 8	Value = 4	Value = 2	Value = 1

Example 7

The program initially sets the generator to the 1 Hz range and starts the sweep. When the sweep reaches 1000 Hz, the program switches the generator to the kHz range and restarts the sweep. When the sweep reaches 1000 kHz, the program switches the generator to the MHz range and again restarts the sweep. When the sweep reaches 1000 MHz, the entire program is restarted. Here is the program:

```

10 CONTROL 3,2 ; 3 ! SET PORT 10 OUTPUT VALUE TO 3
20 ENTER 3 ; A1,F1,B2,F2 ! ENTER CHANNEL A AND B DATA
30 IF A1>999 THEN GOTO 230 ! CHECK TO CHANGE FREQ
40 DISP A1;F1;B2;F2
50 ON F1 GOSUB 110,150,190 ! CHANNEL A FUNCTION BRANCHES
60 ON F2 GOSUB 130,170,210 ! CHANNEL B FUNCTION BRANCHES
70 DISP A1,Z$
80 DISP B2,Y$
90 GOTO 20
100 END
110 Z$="Hz" ! A FUNCTION =1
120 RETURN
130 Y$="Microvolts" ! B FUNCTION =1
140 RETURN

```

```

150 Z$="KHz" ! A FUNCTION =2
160 RETURN
170 Y$="Millivolts" ! B FUNCTION =2
180 RETURN
190 Z$="MHz" ! A FUNCTION =3
200 RETURN
210 Y$="Volts" ! B FUNCTION =3
220 RETURN
230 IF F1<3 THEN 250 ! CHECK FOR MAX FREQ RANGE
240 GOTO 10
250 CONTROL 3,2 ; 2^(F1+1)+1 ! CHANGE GENERATOR FUNCTION
260 GOTO 20

```

Interrupts

The BCD interface provides program interrupt. Interrupts are generated only from the most-significant digit of the function. If you are only inputting one digit of function information (see the previous example), then this digit generates the interrupt. You enable the interrupt mask to write register 1 with an ENABLE INTR and specify a BASIC service routine with an ON INTR statement.

Example 8

Let's change the previous example to generate an interrupt if the voltmeter detects an overrange condition. Assume that the voltmeter returns a function value of 8 when an overrange is detected. Add the following program lines to the program used for the previous example:

```

1 ON INTR 3 GOSUB 300
2 ENABLE INTR 3,128 ! INTERRUPT FOR BIT 3 OF FUNCTION B
300 STATUS 3,1 ; S1
310 IF S1<128 THEN ENABLE INTR 3,128 @ RETURN ! INT. CAUSE
320 DISP "METER OVERRANGE"
330 STOP

```

Line 1 - defines the program branch.

Line 2 - defines the bit in the function digit that can cause an interrupt (see write register 1) and enables the interrupt.

Line 300 - enters the value of read register 1 into variable S1. The status of read register 1 must ALWAYS be checked following an interrupt. Subsequent interrupts are prevented until this interrupt is serviced.

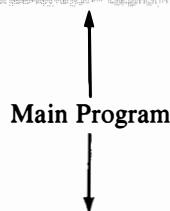
Line 310 - checks to see which bit of the function digit caused the interrupt. This check is used to prioritize the interrupts when more than one bit generates the interrupt (see multiple interrupts). This line also re-enables the interrupt with a new control word (128) to write register 1 if the function A value returned is less than 8 (bit 4 not set). Notice that the RETURN statement is placed on the same line as the ENABLE INTR statement.

Multiple Interrupts

More than 1 bit of the function value can be specified to generate an interrupt. This feature allows you to define the priorities of the interrupts. Here is an example routine to service multiple interrupts. Note that the bit 3 routine is serviced prior to the bit 0 routine if they both occur.

Example 9

```
10 ON INTR 3 GOSUB 200
20 ENABLE INTR 3:9 ! INTERRUPT BITS
```



```
200 STATUS 3,1 ; S1
210 IF BIT(S1,3) THEN 240
220 PRINT "BIT 0 DETECTED"
230 ENABLE INTR 3:9 @ RETURN
240 PRINT "BIT 3 DETECTED"
250 IF BIT(S1,0) THEN 220
260 ENABLE INTR 3:9 @ RETURN
```

Line 20 - specifies bits 0 and 3 to generate an interrupt.

Line 200 - enters the status byte from read register 1 (interrupt cause).

Line 210 - checks for a bit 3 interrupt (first priority). If bit 3 is true (set), the program branches to the bit 3 service routine (lines 240 through 260).

Line 220 - is the bit 0 service routine.

Line 230 - re-enables the interrupt and returns to the main program.

Line 240 - is the bit 3 service routine.

Line 250 - checks for a bit 0 interrupt (second priority). A program branch to the bit 0 routine (line 220) occurs only if bit 0 is true (set).

Line 260 - re-enables the interrupt and returns to the main program.

You determine the priorities for the interrupts. The bit 0 interrupt can be defined as having the higher priority by exchanging the arguments of the bit functions (lines 210 and 250).

Registers

The interface contains 13 registers that can be accessed from the HP-85. These registers are divided into three groups: read registers, write registers, and bi-directional registers. Read registers are accessed with the STATUS statement. Write registers are accessed with the CONTROL statement. The bi-directional registers are accessed with either the STATUS (input) statement or the CONTROL (output) statement. Other statements (ASSERT, ENABLE INTR) access specific registers only and are explained with the register description. This discussion explains each register and shows the access procedure.

Note: The default values shown for all registers assume that the reset default switch (S2) is set to the factory setting (all "0's). Positive-true logic sense is assumed for all descriptions.

Read Register 0

Interface ID

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Always 0	Always 0	Always 0	Always 0	Always 0	Always 0	Always 1	Always 1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

STATUS 3,0;S0

The value returned (always 3) from this register is entered into variable S0. The value 3 indicates that this is the BCD interface.

Read Register 1

Interrupt Cause

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Generated from Function B (Most Significant Digit)				Generated from Function A (Most Significant Digit)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

STATUS 3,1;S1

The value returned indicates the bit (or bits) that generated the interrupt. See write register 1 for a complete discussion about the BCD card interrupts.

Default value: 0

Read Register 2

Uniline Messages

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I/O A 0 = Input 1 = Output	I/O B 0 = Input 1 = Output	Cntrl A 0 = Ready 1 = Busy	Cntrl B 0 = Ready 1 = Busy	Flag A 0 = Ready 1 = Busy	Flag B 0 = Ready 1 = Busy	0	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

STATUS 3,2,S2

The value returned indicates the status of the various handshake signals.

Default value: 0

The meanings of the various bits are described next.

- Bit 7 set (1) indicates that channel A ports are enabled for output operation. Bit 7 reset (0) indicates that channel A ports are set to the input mode.
- Bit 6 set (1) indicates that channel B ports are enabled for output operation. Bit 6 reset (0) indicates that channel B ports are set to the input mode.
- Bit 5 set (1) indicates that the channel A “Control” handshake line is active (busy). Bit 5 reset (0) indicates that the channel A “Control” handshake line is ready.
- Bit 4 set (1) indicates that the channel B “Control” handshake line is active (busy). Bit 4 reset (0) indicates that the channel B “Control” handshake line is ready.
- Bit 3 set (1) indicates that the channel A “Flag” handshake line is busy. The peripheral device connected to channel A uses the “Flag” line to indicate its status to the HP-85. Bit 3 reset (0) indicates that the channel A peripheral is ready.
- Bit 2 set (1) indicates that the channel B “Flag” handshake line is busy. The peripheral device connected to channel B uses the “Flag” line to indicate its status to the HP-85. Bit 2 reset (0) indicates that the channel B peripheral is ready.
- Bits 1 and 0 are not used.

Write Register 0

Write register 0 is not implemented. Attempting to access write register 0 generates error 111.

Write Register 1

Interrupt Mask

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Function B (Most Significant Digit)				Function A (Most Significant Digit)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

ENABLE INTR 3 : <mask value>

CONTROL 3,1 : <mask value>

Write register 1 contains the interrupt mask for channel A and channel B. The interrupts are generated from the most significant digit of the function value of each channel. For example, assume that channel A is configured for two digit function values from an external device. These values range from 00 through 99. You can specify an interrupt only for values of the tens digit. The units digit can not generate an interrupt. If a single digit function value is specified (default) then that single digit is the most significant digit and can generate an interrupt (see example 8).

Write Register 2

Handshakes and Port 10

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I/O Channel A	I/O Channel B	CTL Channel A	CTL Channel B	Port 10 Output (when available)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

CONTROL 3,2 : <value>

ASSERT 3,2 : <value>

Write register 2 enables you to control the interface handshake lines from a program. The output data for port 10 (if port 10 is available for output) is also written to this register.

The meanings of the various bits are described next.

- Bit 7 set (1) enables channel A ports for output. The reset default switch (S2) switch 8 must be set to the "1" position. Attempting to set bit 7 with S2 switch 8 in the "0" position (default) generates error 113. Bit 7 clear (0) enables channel A ports for input.
- Bit 6 set (1) enables channel B ports for output. The reset default switch (S2) switch 8 must be set to the "1" position. Attempting to set bit 6 with S2 switch 8 in the "0" position (default) generates error 113. Bit 6 clear (0) enables channel B ports for input.
- Bit 5 enables you to control the CTL handshake line for channel A. This line remains active until the interface is reset or bit 5 is cleared (set to 0). If this bit is not reset, subsequent ENTER or OUTPUT operations generate error 118.
- Bit 4 enables you to control the CTL handshake line for channel B. This line remains active until the interface is reset or bit 4 is cleared (set to 0). If this bit is not reset, subsequent ENTER or OUTPUT operations generate error 118.
- Bits 3 through 0 are port 10. When port 10 is used as an output port, the data is written to these four bits (see example 7). The input/output switch (S2) switch 8 does not affect port 10. When port 10 is enabled as an input port or configured as part of a channel, any attempt to output to port 10 generates error 114.

Register 3

Mantissa Digits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of Digits Assigned for Channel B Mantissa (0 – 11)				Number of Digits Assigned for Channel A Mantissa (0 – 11)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 3,3;S3
CONTROL 3,3;<value>
```

The value contained in this register indicates how many mantissa digits are specified for each channel. To change this specification, execute the appropriate CONTROL statement to this register. The new specification remains in this register until changed by another CONTROL statement or until the interface is reset.

Reset default: 8 (Channel A only)

Note: The reset default value for two channel operation is 68. This indicates that four mantissa digits are specified for each channel (see example 4).

Register 4

Exponent Digits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of Digits Assigned for Channel B Exponent (0 – 3)				Number of Digits Assigned for Channel A Exponent (0 – 3)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 3,4;34
CONTROL 3,4;<value>
```

The value contained in this register indicates how many exponent digits are specified for each channel. To change this specification, execute the appropriate CONTROL statement to this register. The new specification remains in this register until changed by another CONTROL statement or until the interface is reset.

Reset default: 1 (Channel A only)

Note: The reset default value for two channel operation is 0. This indicates that no exponent digits are specified for either channel.

Register 5

Function Digits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of Digits Assigned for Channel B Function (0 – 11)				Number of Digits Assigned for Channel A Function (0 – 11)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 3,5;55
CONTROL 3,5;<value>
```

The value contained in this register indicates how many function digits are specified for each channel. To change this specification, execute the appropriate CONTROL statement to this register. The new specification remains in this register until changed by another CONTROL statement or until the interface is reset.

Reset default: 1 (Channel A only)

Note: The reset default value for two channel operation is 17. This indicates that one function digit is specified for each channel.

Register 6

Decimal Point Placement

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of Mantissa Digits Assigned to the Right of the Decimal Point. (Channel B)				Number of Mantissa Digits Assigned to the Right of the Decimal Point. (Channel A)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 3,6;S6
CONTROL 3,6;<value>
```

The value contained in this register indicates how many mantissa digits to the right of the decimal point are specified. To change this specification, execute the appropriate CONTROL statement to this register. The new specification remains in this register until changed by another CONTROL statement or until the interface is reset. Note that the specification cannot exceed the number of mantissa digits specified for that channel. For example, if eight mantissa digits are specified for channel A, then a maximum of eight digits can be specified to the right of the decimal point for this channel. Decimal point placement can only be specified for input operations.

Reset default: 0

Register 7

Control and Handshake Sense

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Logic Sense I/O A	Logic Sense I/O B	Logic Sense CTL A	Logic Sense CTL B	Logic Sense Flag A	Logic Sense Flag B	Handshake Mode A	Handshake Mode B
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 3,7;S7
CONTROL 3,7;<value>
```

The value contained in this register indicates the logic sense of the control and handshake lines. To change this specification, execute the appropriate CONTROL statement to this register. The new specification remains in this register until changed by another CONTROL statement or until the interface is reset.

Reset default: 0

The meanings of the various bits are described next.

- Bit 7 set (1) indicates that the logic sense for the channel A I/O control line is negative-true. Bit 7 reset (0) indicates positive-true logic sense.
- Bit 6 set (1) indicates that the logic sense for the channel B I/O control line is negative-true. Bit 6 reset (0) indicates positive-true logic sense.
- Bit 5 set (1) indicates that the logic sense for the channel A CTL handshake line is negative-true. Bit 5 reset (0) indicates positive-true logic sense.
- Bit 4 set (1) indicates that the logic sense for the channel A CTL handshake line is negative-true. Bit 4 reset (0) indicates positive-true logic sense.
- Bit 3 set (1) indicates that the logic sense for the channel A Flag handshake line is negative-true. Bit 3 reset (0) indicates positive-true logic sense.
- Bit 2 set (1) indicates that the logic sense for the channel B Flag handshake line is negative-true. Bit 2 reset (0) indicates positive-true logic sense.
- Bit 1 set (1) indicates leading-edge trigger sense for the interface CTL A handshake line. This specifies that the CTL line goes false on the leading edge of the channel A Flag line. Bit 1 reset (0) indicates trailing-edge trigger sense for this handshake sequence.
- Bit 0 set (1) indicates leading-edge trigger sense for the interface CTL B handshake line. This specifies that the CTL line goes false on the leading edge of the channel B Flag line. Bit 0 reset (0) indicates trailing-edge trigger sense for this handshake sequence.

Register 8

Data Sense

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Logic Sense for Channel B Input Data.				Logic Sense for Channel A Input Data.			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 3,8:$8
CONTROL 3,8:<value>
```

The value contained in this register indicates the logic sense of each bit of the data digits in each channel. To change this specification, execute the appropriate CONTROL statement to this register. The new specification remains in this register until changed by another CONTROL statement or until the interface is reset.

Reset default: 0

The meanings of the various bits are described next.

- Bit 7 (1) set indicates that the logic sense of the most significant bit (bit 3) of channel B data is negative-true. Positive-true logic sense is indicated by bit 7 clear (0).
- Bit 6 set (1) indicates that the logic sense of bit 2 of channel B data is negative-true. Positive-true logic sense is indicated by bit 6 clear (0).
- Bit 5 set (1) indicates that the logic sense of bit 1 of channel B data is negative-true. Positive-true logic sense is indicated by bit 5 clear (0)
- Bit 4 set (1) indicates that the logic sense of the least significant bit (bit 0) of channel B data is negative-true. Positive-true logic sense is indicated by bit 4 clear (0).
- Bit 3 (1) set indicates that the logic sense of the most significant bit (bit 3) of channel A data is negative-true. Positive-true logic sense is indicated by bit 3 clear (0).
- Bit 2 set (1) indicates that the logic sense of bit 2 of channel A data is negative-true. Positive true-logic sense is indicated by bit 2 clear (0).
- Bit 1 set (1) indicates that the logic sense of bit 1 of channel A data is negative-true. Positive-true logic sense is indicated by bit 1 clear (0)
- Bit 0 set (1) indicates that the logic sense of the least significant bit (bit 0) of channel A data is negative-true. Positive-true logic sense is indicated by bit 0 clear (0).

Register 9

Function Data Sense

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Logic Sense for Channel B Function Data.				Logic Sense for Channel A Function Data.			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 3,9;S9
CONTROL 3,9;<value>
```

The value contained in this register indicates the logic sense of each bit of the function digits in each channel. To change this specification, execute the appropriate CONTROL statement to this register. The new specification remains in this register until changed by another CONTROL statement or until the interface is reset.

Reset default: 0

The meanings of the various bits are described next.

- Bit 7 (1) set indicates that the logic sense of the most significant bit (bit 3) of channel B function is negative true. Bit 7 clear (0) indicates positive-true logic sense.
- Bit 6 set (1) indicates that the logic sense of bit 2 of channel B function is negative-true. Positive-true logic sense is indicated by bit 6 clear (0).
- Bit 5 set (1) indicates that the logic sense of bit 1 of channel B function is negative-true. Positive-true logic sense is indicated by bit 5 clear (0)
- Bit 4 set (1) indicates that the logic sense of the least significant bit (bit 0) of channel B function is negative true. Bit 4 clear (0) indicates positive-true logic sense.
- Bit 3 (1) set indicates that the logic sense of the most significant bit (bit 3) of channel A function is negative true. Bit 3 clear (0) indicates positive-true logic sense.
- Bit 2 set (1) indicates that the logic sense of bit 2 of channel A function is negative-true. Positive-true logic sense is indicated by bit 2 clear (0).
- Bit 1 set (1) indicates that the logic sense of bit 1 of channel A function is negative-true. Positive-true logic sense is indicated by bit 1 clear (0)
- Bit 0 set (1) indicates that the logic sense of the least significant bit (bit 0) of channel A function is negative true. Bit 0 clear (0) indicates positive-true logic sense.

Register 10

Sign and Port 10 Sense

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Logic Sense Exponent B	Logic Sense Mantissa B	Logic Sense Exponent A	Logic Sense Mantissa A	Logic Sense for Port 10 Data.			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

To access this register execute:

```
STATUS 3,10;S0
CONTROL 3,10;<value>
```

The value contained in this register indicates the logic sense of the sign bits and the port 10 data bits. To change this specification, execute the appropriate CONTROL statement to this register. The new specification remains in this register until changed by another CONTROL statement or until the interface is reset.

Reset default: 0

The meanings of the various bits are described next.

- Bit 7 set (1) indicates negative-true logic sense for the sign bit of the channel B exponent. Bit 7 clear (0) indicates positive-true logic sense for this bit.
- Bit 6 set (1) indicates negative-true logic sense for the sign bit of the channel B mantissa. Bit 6 clear (0) indicates positive-true logic sense for this bit.
- Bit 5 set (1) indicates negative-true logic sense for the sign bit of the channel A exponent. Bit 5 clear (0) indicates positive-true logic sense for this bit.
- Bit 4 set (1) indicates negative-true logic sense for the sign bit of the channel A mantissa. Bit 4 clear (0) indicates positive-true logic sense for this bit.
- Bit 3 (1) set indicates that the logic sense of the most significant bit (bit 3) of port 10 data is negative-true. Bit 3 clear (0) indicates positive-true logic sense.
- Bit 2 set (1) indicates that the logic sense of bit 2 of port 10 data is negative-true. Positive-true logic sense is indicated by bit 2 clear (0).
- Bit 1 set (1) indicates that the logic sense of bit 1 of port 10 data is negative-true. Positive-true logic sense is indicated by bit 1 clear (0)
- Bit 0 set (1) indicates that the logic sense of the least significant bit (bit 0) of port 10 data is negative-true. Bit 0 clear (0) indicates positive-true logic sense.

Non-Standard Formatting

You can specify formats other than the single or dual channel default formats that are provided. This discussion explains how to select formats by executing CONTROL statements to the appropriate registers.

Single Channel Formatting

Assume that you want to specify the following single channel format for channel A:

Format Selection

Format Desired	Register Access	Value
5 mantissa digits	Register 3	5
2 exponent digits	Register 4	2
2 function digits	Register 5	2
4 digits to right of decimal point	Register 6	4
Negative true data	Register 8	15

Example 10

Run the following program and observe the results as shown:

```

10 RESET 3 ! RESET INTERFACE
20 STATUS 3,0 ; R0,R1,R2,C3,C4,C5,C6,C7,C8,C9,C0
30 IMAGE K,X,DDD,/
40 PRINT USING 30 ; "READ REG 0=";R0,"READ REG 1=";R1
50 PRINT USING 30 ; "READ REG 2=";R2
60 PRINT USING 30 ; "REG 3=";C3,"REG 4=";C4,"REG 5=";C5
70 PRINT USING 30 ; "REG 6=";C6,"REG 7=";C7,"REG 8=";C8
80 PRINT USING 30 ; "REG 9=";C9,"REG 10=";C0
90 DISP "PAUSE"
100 PAUSE
110 CONTROL 3,3 ; 5,2,2,4@ CONTROL 3,8 ; 15 ! SET FORMAT
120 STATUS 3,0 ; R0,R1,R2,C3,C4,C5,C6,C7,C8,C9,C0
130 PRINT USING 30 ; "READ REG 0=";R0,"READ REG 1=";R1
140 PRINT USING 30 ; "READ REG 2=";R2
150 PRINT USING 30 ; "REG 3=";C3,"REG 4=";C4,"REG 5=";C5
160 PRINT USING 30 ; "REG 6=";C6,"REG 7=";C7,"REG 8=";C8
170 PRINT USING 30 ; "REG 9=";C9,"REG 10=";C0
180 DISP "PAUSE2"
190 PAUSE

```

READ REG 0=	3
READ REG 1=	0
READ REG 2=	0
REG 3=	8
REG 4=	1
REG 5=	1
REG 6=	0
REG 7=	0
REG 8=	0
REG 9=	0
REG 10=	0

- Interface I.D.
- Interrupt Mask
- Handshake Lines
- Mantissa Digits Specified
- Exponent Digits Specified
- Function Digits Specified
- Decimal Point Location
- Handshake Logic Sense
- Data Logic Sense
- Function Logic Sense
- Signs and Port 10 Logic Sense

When PAUSE appears on the display, press continue and observe the results as shown.

```
READ REG 0= 3
READ REG 1= 0
READ REG 2= 0
REG 3= 5
REG 4= 2
REG 5= 2
REG 6= 4
REG 7= 0
REG 8= 15
REG 9= 0
REG 10= 0
```

- Interface I.D.
- Interrupt Mask
- Handshake Lines
- Mantissa Digits Specified
- Exponent Digits Specified
- Function Digits Specified
- Decimal Point Location
- Handshake Logic Sense
- Data Logic Sense
- Function Logic Sense
- Signs and Port 10 Logic Sense

The interface is now configured to the desired format. This configuration remains until it is changed or until the interface is reset.

Dual Channel Formatting

Assume that you want to specify the following dual channel format:

Format Selection

Channel A		Channel B		Register Access
Format Desired	Value	Format Desired	Value	
2 mantissa digits	2	2 mantissa digits	32	Register 3
1 exponent digit	1	2 exponent digits	32	Register 4
1 function digit	1	1 function digit	16	Register 5
No decimal point	0	No decimal point	0	Register 6
Positive true data	0	Negative true data	240	Register 8

Example 11

Add the following lines to the previous program:

```
175 ! SET 2 CHANNEL FORMAT
180 CONTROL 3,3 : 34,33,17,0@ CONTROL 3,8 : 240 ! SET FMT
190 STATUS 3,0 : R0,R1,R2,C3,C4,C5,C6,C7,C8,C9,C0
200 PRINT USING 30 : "READ REG 0=";R0;"READ REG 1=";R1
210 PRINT USING 30 : "READ REG 2=";R2
220 PRINT USING 30 : "REG 3=";C3;"REG 4=";C4;"REG 5=";C5
230 PRINT USING 30 : "REG 6=";C6;"REG 7=";C7;"REG 8=";C8
240 STOP
```

When PAUSE2 is displayed, press CONT and observe the results as shown here. The interface is now configured for the dual channel format as specified. This configuration remains until it is changed or until the interface is reset.

READ REG 0= 3	— Interface I.D.
READ REG 1= 0	— Interrupt Mask
READ REG 2= 0	— Handshake Lines
REG 3= 34	— Mantissa Digits Specified
REG 4= 33	— Exponent Digits Specified
REG 5= 17	— Function Digits Specified
REG 6= 0	— Decimal Point Location
REG 7= 0	— Handshake Logic Sense
REG 8= 240	— Data Logic Sense

Data Output

The interface can output data via the I/O ports. The reset default switch (S2) switch 1 must be placed in the "1" position.

Example 12

The next program assumes single channel default format. Here is the program:

```

10 RESET 3 ! RESET INTERFACE
20 CONTROL 3,2 ; 128 ! SET CHANNEL A TO OUTPUT MODE
30 STATUS 3,2 ; S2
40 DISP S2
50 IMAGE SBZE,Z
60 OUTPUT 3 USING 50 ; 12.345678,2
70 STOP

```

The following data is output to the peripheral:

12345678E-6

2

The output image specification must correspond exactly with the mode of the interface card. The following image statements can be used for single and dual channel default formats:

SBZE,Z - Single channel default format.

4Z,Z,M4Z,Z - Dual channel default format.

The interface can be configured for non-standard output formats in the identical manner as shown for non-standard inputs (see Non Standard Formats).

Transfers

When entering data with the TRANSFER statement, you can calculate the buffer sizes and count parameters as follows:

1. Count one byte for each mantissa and function digit.
2. Add one byte for mantissa sign (if applicable).
3. Count one byte for each exponent digit plus two bytes for the "E" and the exponent sign.

BCD I/O Statements

Statement	Description
ABORTIO	Aborts any transfer in progress, sets all control lines false.
ASSERT	Immediately writes a value to register 2, placing I/OA, I/OB, CTLA, CTLB, and port 10 lines in the specified state.
CONTROL	Writes values to the interface control registers.
ENABLE INTR	Writes enable mask to control register 1. Used to select event interrupts.
ENTER	Enters data from the interface into the BASIC program.
HALT	Aborts any transfer in progress. Does not change external lines and does not reset the interface.
OUTPUT	Outputs data from the BASIC program to the interface.
REMOTE	Sets the partial field specifier if the listen address ranges from 00 to 06, otherwise ignored.
RESET	Resets the interface to the default switch settings, sets I/O lines to the input state, and sets all handshake lines false.
SEND	CMD - Commands 32 through 38, and 64 through 70 are valid. DATA - transfers data to the peripheral. Values 0 through 32, and 44 are ignored. TALK and LISTEN - Addresses whose lower 5 bits evaluate from 0 to 6 set the partial field.
STATUS	Inputs values from the interface status registers.
TRANSFER	Moves data from a port to a buffer, or from a buffer to a port. INT and FHS transfers are allowed.

BCD Interface Errors

Error No.	Meaning	Possible Cause
110	Self test failure in I/O card.	Hardware failure
111	Illegal I/O operation.	Instruction not recognized by the interface. Illegal parameter in instruction. Exceeded the available number of registers.
113	Illegal mode of card.	Exceeded 11 digits total. More than 3 exponent digits per channel. Channel B has mantissa or exponent digits when direction of channel B does not equal the direction of A. Not standard format for FHS. Tried to set channel to output when the output enable switch was open.
114	Port 10 not available.	Tried to write data to port 10 when it was selected by a channel.
115	Fast handshake transfer aborted by the peripheral.	Flag B went true during fast handshake.
116	Direction mismatch.	Tried to OUTPUT to an input channel or ENTER from an output channel.
117	Instruction directed to nonexistent field.	Field was allocated 0 digits. Partial field specifies a field of 0 digits.
118	Handshake not ready.	Control was true at start of handshake.
119	Data format error.	Output data did not match card's mode in number of digits.

Section 15

Using the GPIO Interface

Section Introduction

The 82940A interface allows your HP-85 to communicate with a wide variety of devices through the use of parallel data exchanges. A **parallel** interface sends or receives an entire byte or word of data in one operation. This is the most basic, and most versatile, method of I/O. However, it is the inherent versatility of this interface that makes it appear somewhat confusing at first. Don't be overwhelmed. Consider each interface characteristic of your peripheral device and deal with these characteristics one at a time. For example, there are 16 primary addresses to choose from on this interface. But if you know that your only requirement is the input of 8-bit data, you can eliminate 14 of the 16 choices. By using this "process of elimination" approach, you can master a parallel interfacing task in short order.

This section explains the use of the 82940A interface from a programming point of view. The emphasis is on accessing the capabilities of the interface using program statements. Unlike the HP-IB interface however, a basic parallel interface does not isolate you from the hardware. Many references to the characteristics of the hardware are necessary to properly explain the various features available to you. If your background is solely in software, you will probably want to solicit the help of a person with some hardware background. In fact, a technician or "hardware type" is practically a necessity during the installation of a parallel interface because there aren't any connectors wired to the 82940A when you receive it. Most of the hardware information is presented in the 82940A Interface Installation and Theory of Operation manual. Please refer to that manual for hardware details such as:

- How to set the interface select code
- How to set the default configuration switches
- How to connect the interface cables
- Recommended driver and receiver circuits

Throughout this section, the abbreviation "GPIO" is used when referring to the 82940A parallel interface. This stands for "General Purpose Input and Output" and reflects the flexible nature of the interface.

Essentials of a Parallel Interface

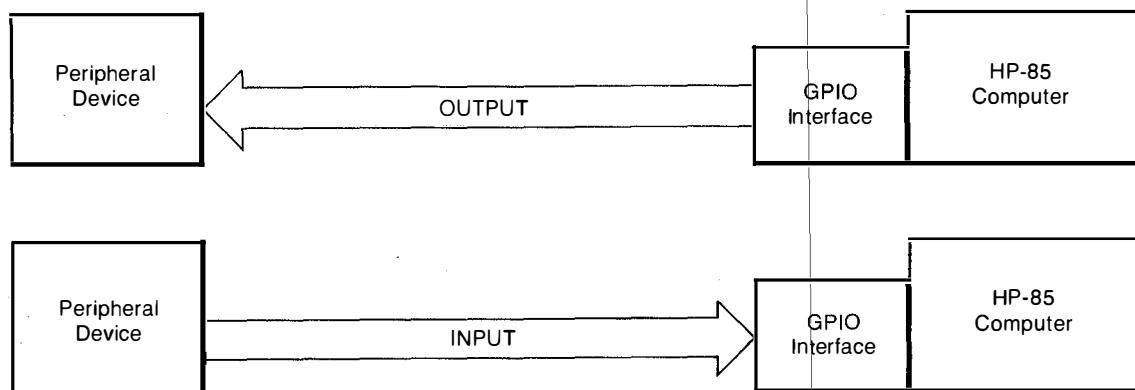
The Section Introduction recommended that you consider each interface characteristic individually whenever possible. What are these essential characteristics? In most cases, a parallel interface will be successful if each of the following characteristics has been properly determined:

- Direction of data flow
- Number of bits in a unit of data
- Method and timing of handshake
- Logical polarity of data and control lines
- Type of I/O statement used in the program

Note that these five categories represent only the essentials of a parallel interface. There may be other factors to consider in individual applications, such as parity, end-of-line sequence, and creative use of interrupts. But no amount of attention to parity or end-of-line sequence will get an interface working if the handshake or polarity is wrong! Therefore, deal first with the five factors listed above. Extra features and special capabilities can be added after the GPIO is properly handling the basic communication task.

Direction of Data Flow

Because an interface connects to both the computer and the peripheral device, it is important to avoid confusion about the direction of data flow. The output of the computer is the input to the peripheral device. All references to data direction in this section are given with respect to the computer. This is shown in the following diagrams.



There are four basic choices when selecting data direction with the GPIO interface. They involve direction of data flow and drive capability. Two kinds of output ports are available. One kind has a small drive capability of about 2 standard TTL loads. The other kind has a larger drive capability of about 12 standard TTL loads. A list of the data direction choices available is shown below. A detailed description of each choice is given in the following paragraphs.

1. Bidirectional - small output drive (choose this for **input-only** applications)
2. Bidirectional - large output drive
3. Input and output on separate lines - large output drive
4. Output only - large output drive

Choice #1 is a bidirectional port with a small output drive capability. This type of port is recommended for input-only operations and for bidirectional interface to light loads. A “light” load is a circuit that sources less than 4.5 mA. Some examples are NMOS interface chips, one TTL gate with a 2.2 k Ω pull-up resistor, or CMOS gates with a 10 k Ω pull-up resistor.

Choice #2 is a bidirectional port with increased output drive capability. This type of port is recommended for bidirectional interface to heavier loads. The output drivers on this port type are open-collector transistors rated to sink 20 mA. Any bidirectional load that sources more than 4.5 mA must be connected to this port type.

Choice #3 is similar to choice #2, but there is a significant difference. The bidirectional port (choice #2) uses a common data bus for input and output. The port type of choice #3 uses one data bus for input and a separate data bus for output. This type of port is useful when interfacing to a device that has separate input and output lines which cannot be connected together for electrical reasons.

Choice #4 is for output-only applications. This port type uses open-collector drivers rated to sink up to 20 mA.

Number of Bits and Ports

The GPIO interface allows the selection of either 8-bit or 16-bit ports. The number of ports available depends upon the size you choose and the data direction requirements. If you are using 8-bit ports, there can be a maximum of four independent ports. This is two bidirectional ports with small output drive and two output-only ports. Note that other configurations yield less ports. For example, if you need bidirectional ports with large output drive, there can only be two. The reason for this will become apparent as you read the next topic on addressing and configuration.

A similar situation exists with 16-bit ports. You can have two of them if one is output only and the other is bidirectional with small output drive. However, any other configuration is limited to one 16-bit port.

Using Primary Addresses

You select the type of port by specifying a primary address in your OUTPUT, ENTER, or TRANSFER statements. In essence, each type of port is treated as a separate device and is accessed by using a device selector. There are other ways to access a port, but they are all related to primary addresses. The primary address can be written directly into register 5, and the default configuration switches allow any primary address for an 8-bit port to be selected automatically at power-on or reset (refer to the Installation and Theory of Operation manual). However, the simplest way to avoid surprises and confusion is to include the desired primary address in your device selector when performing I/O operations.

If you do not specify a primary address in the device selector (e.g. OUTPUT 4 ; X), the last primary address specified is used. If no primary address has been specified since the last power-on or reset, the address set by the default configuration switches is used.

The following tables summarize the port options available. The tables also indicates which lines are used for handshake and direction indication with each port type. The handshake lines are discussed at length in "Handshake Methods" (covered next). The direction indicator is a line used with a bidirectional port to indicate in which direction the data is currently flowing. This line is often used for the control of tri-state gates or selector circuits. It presents a logic low when the interface is outputting and a logic high when the interface is inputting.

8-Bit Ports

Data Direction	Primary Address	Port Description	Handshake Lines	Direction Indicator
Bidirectional; small output drive	00	Port A	CTLA/FLGA	$\overline{\text{OUTA}}$
	01	Port B	CTLB/FLGB	$\overline{\text{OUTB}}$
Input and output on separate lines; large output drive	02	Input to Port A Output from Port C	CTLA/FLGA	$\overline{\text{OUTA}}$
	03	Input to Port B Output from Port D	CTLB/FLGB	$\overline{\text{OUTB}}$
Output only; large output drive	04	Port C	CTL0/ST0	none
	05	Port D	CTL1/ST1	none
Bidirectional; large output drive	06	Port A or Port C wired together	CTLA/FLGA	$\overline{\text{OUTA}}$
	07	Port B and Port D wired together	CTLB/FLGB	$\overline{\text{OUTB}}$

16-Bit Ports

Data Direction	Primary Address	Port Description	Handshake Lines	Direction Indicator
Bidirectional; small output drive	08	LSB ¹ on Port A MSB ¹ on Port B	CTA/FLGA	<u>OUTA</u>
	09	same	CTLB/FLGB	<u>OUTB</u>
Input and output on separate lines; large output drive	10	LSB input on Port A MSB input on Port B LSB output on Port C MSB output on Port D	CTLA/FLGA	<u>OUTA</u>
	11	same	CTLB/FLGB	<u>OUTB</u>
Output only; large output drive	12	LSB on Port C MSB on Port D	CTL0/ST0	none
	13	same	CTL1/ST1	none
Bidirectional; large output drive	14	Port A and Port C wired together (LSB) Port B and Port D wired together (MSB)	CTLA/FLGA	<u>OUTA</u>
	15	same	CTLB/FLGB	<u>OUTB</u>

Handshake Methods

A "handshake" is a sequence of electrical events used to synchronize a transfer of data. There is a brief overview of the handshake process in Section 1. With the GPIO interface, you have four basic methods of handshake to choose from (with some variations, of course). These basic choices are:

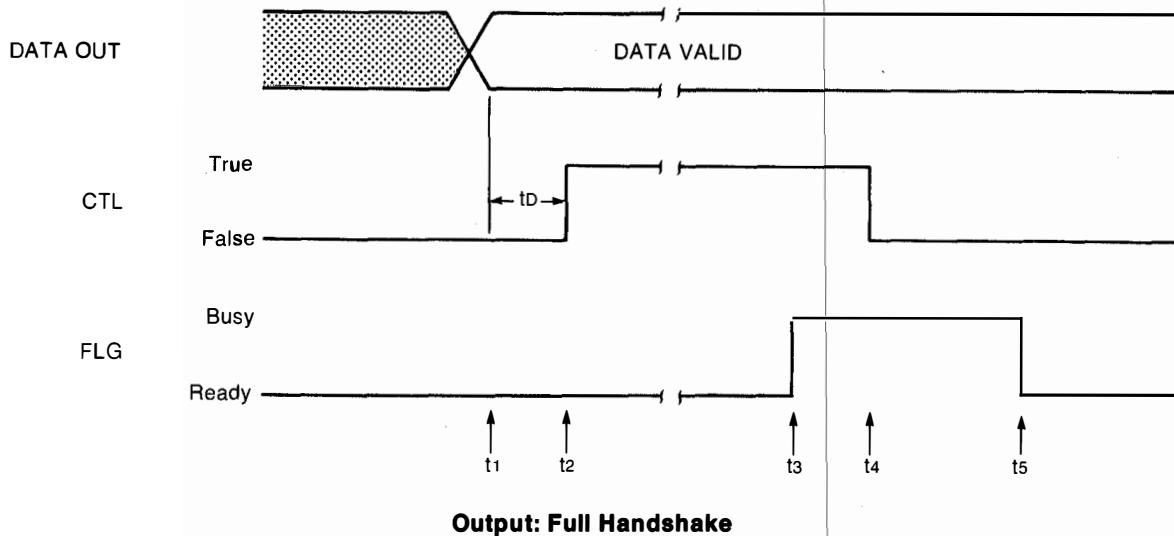
- Full handshake
- Strobe handshake
- Partial handshake
- No handshake

The handshake lines on the 82940A are called **FLAG** (FLG) and **CONTROL** (CTL). The FLAG line is used to sense the handshake signal coming from the peripheral device, and the CONTROL line is used to send a handshake signal from the interface to the peripheral device. (The output-only ports use a line called **STATUS** (ST) to perform the same function as the FLAG line.) Exactly what signals are sent and received depends upon the handshake mode that you select. Let's look at the details of each method.

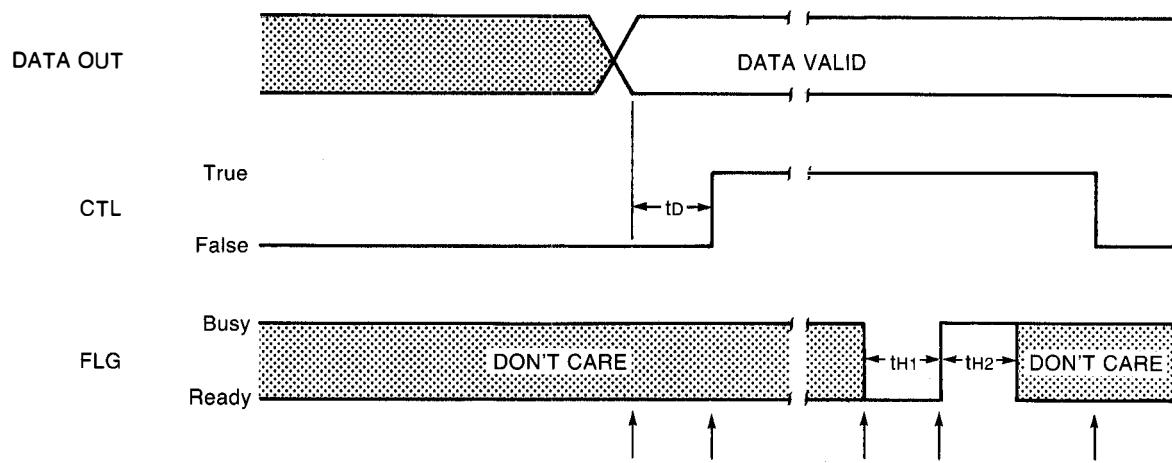
¹ As it is used here, the abbreviation "LSB" stands for "Least Significant Bits". These are bit 0 thru bit 7 of the 16-bit word. Likewise, "MSB" stands for "Most Significant Bits". These are bit 8 thru bit 15 of the 16-bit word.

Output Handshakes

Output handshakes are somewhat simpler than input handshakes, so they are presented first. The following timing diagrams show only the essential action of the DATA and handshake lines. The line used to indicate data direction has been left out for the sake of simplicity, and not all timing relationships have been given numeric values. More complete timing information is available in the Installation and Theory of Operation manual. These diagrams are intended to clarify the concept of the handshake methods. The important factors to note are the order of events and the causal relationship of events.

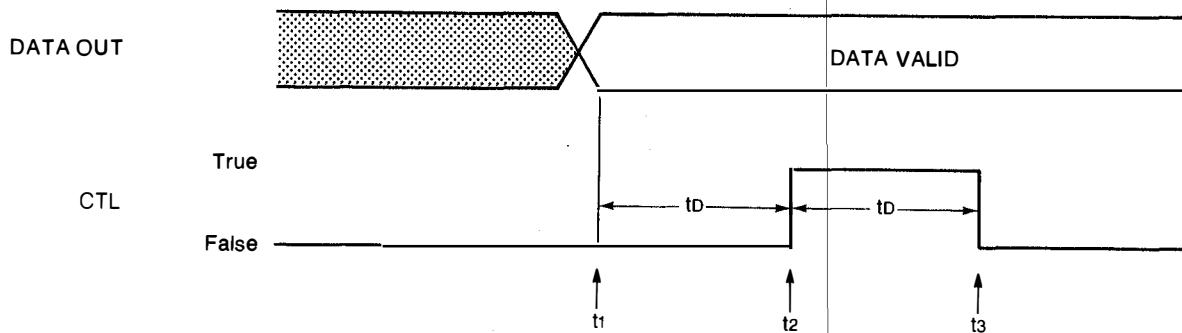


When the full handshake cycle starts, the interface checks for a READY indication on the FLG line. If the line is READY, the interface places a new word of data on the DATA lines (t_1). After a programmable delay time (tD), the interface places the CTL line in the TRUE state (t_2). This signals the peripheral device that the data is valid. The delay time is provided to ensure that the DATA lines are stable and valid before CTL is asserted. This delay time is set by register 6, which is explained later. When the peripheral device sees the TRUE state of the CTL line, it does whatever is necessary to input the data presented to it. The peripheral device indicates that it is busy inputting data by placing FLG in the BUSY state (t_3). This serves as an acknowledgment to the interface that the CTL signal was received. Therefore, when the interface sees FLG go BUSY, it can return the CTL signal to the FALSE state (t_4). When the peripheral device has finished inputting data, it returns the FLG line to the READY state to indicate that it is ready for the next cycle (t_5).



Output: Partial Handshake

The key difference between full and partial handshake is that partial handshake does not check the FLG line before it outputs the data. The output cycle can begin with the FLG line BUSY or READY. As in the full handshake, the interface outputs the data (t_1) and sets CTL to the TRUE state (t_2) after a programmable delay (t_D). This signals the peripheral device that the data is valid. The interface then waits for the peripheral device to indicate that it has received the data. This is the reason for the name “partial handshake”. The interface does not require a READY signal to start the transfer, but it does require an acknowledgment to complete it. The peripheral device inputs the data and supplies the “data accepted” signal by holding FLG in the READY state (t_3) for at least $30 \mu s$ (t_{H1}) and then in the BUSY state (t_4) for at least $35 \mu s$ (t_{H2}). Note that although this action greatly resembles an edge-triggered event, it really is not. The minimum state times mentioned are necessary for the interface to sense the READY to BUSY transition. Once the interface senses the FLG signal from the peripheral device, it can return the CTL signal to the FALSE state (t_5).

**Output: Strobe Handshake**

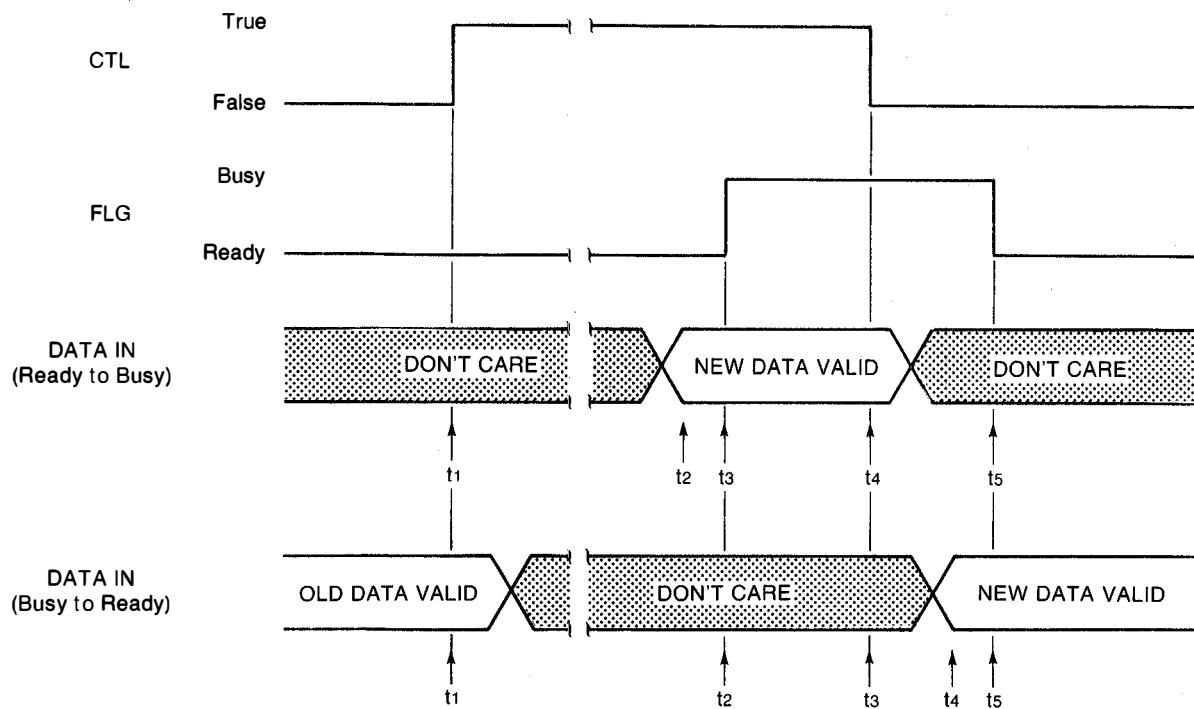
The strobe handshake for output is a very simple sequence. It is probably the most common handshake method used in devices that do not implement full handshake. This method assumes that the peripheral device is always ready and the FLG line is not used. (If your device is not always ready, then the “Output Inhibit” feature can be used. This is explained in “Selecting the Handshake Method”.) The cycle starts with the output of data (t1). After a programmable delay (tD), the interface sets CTL to the TRUE state (t2). This state is held for the delay time, then CTL is returned to the FALSE state (t3). In other words, the interface supplies data, followed by a strobe pulse to indicate that the data is valid.

The “no handshake” option does not need a timing diagram. The interface simply places new data on the DATA lines when it becomes available. The FLG and CTL lines are not used. The ASSERT and STATUS statements can be used to supply your own handshake in this mode (see “Direct Use of Control Lines”).

Input Handshakes

One reason that the input handshakes are more complex than the output handshakes is that each input handshake has two options for the timing of the interface’s read operation. These options are called “READY to BUSY” and “BUSY to READY”.¹ In the following diagrams, both options are shown on the same drawing. The upper part of each diagram shows the timing that is common to both options and the READY to BUSY timing. The lower part of each diagram shows the timing changes that pertain to the BUSY to READY option.

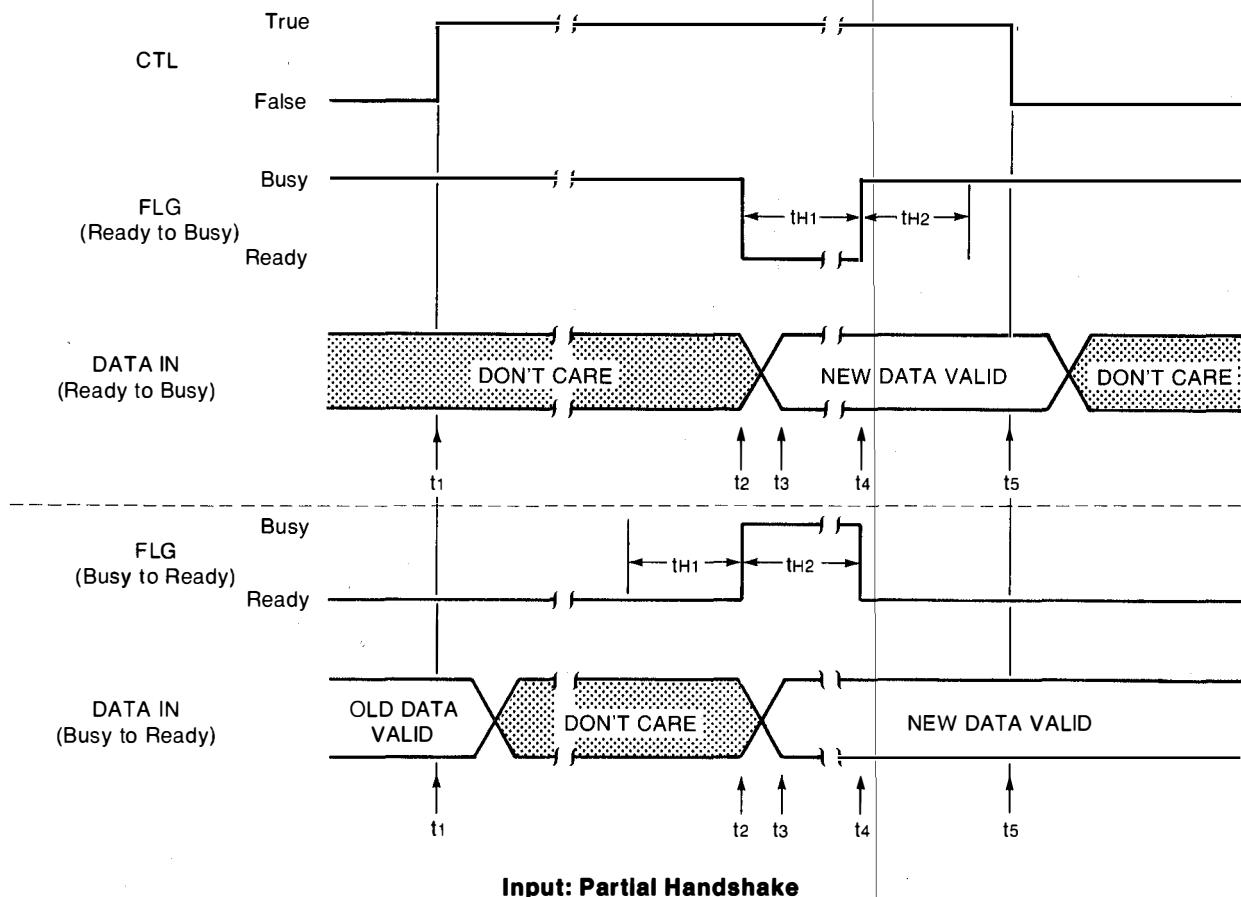
¹ These names were derived from the state change on the FLG line that triggers the read operation in full handshake mode. However, the terms are not meant to imply that FLG lines are edge-triggered. They are not. Also, these names are used to describe the timing choices for all input handshakes, even though strobe handshake does not use a FLG line.



Input: Full Handshake

READY to BUSY: When the full handshake cycle starts, the computer checks for a READY indication on the FLG line. If the FLG line is READY, the interface requests data by setting CTL to the TRUE state (**t1**). The peripheral device sees this request and places data on the DATA lines (**t2**). The peripheral device then signals that the data is valid by placing the FLG line in the BUSY state (**t3**). When the interface sees this signal, it inputs the data (sometime between **t3** and **t4**). The interface then signals that it has received the data by returning CTL to the FALSE state (**t4**). When the peripheral device sees that the data has been received, it returns FLG to the READY state to prepare for the next cycle (**t5**).

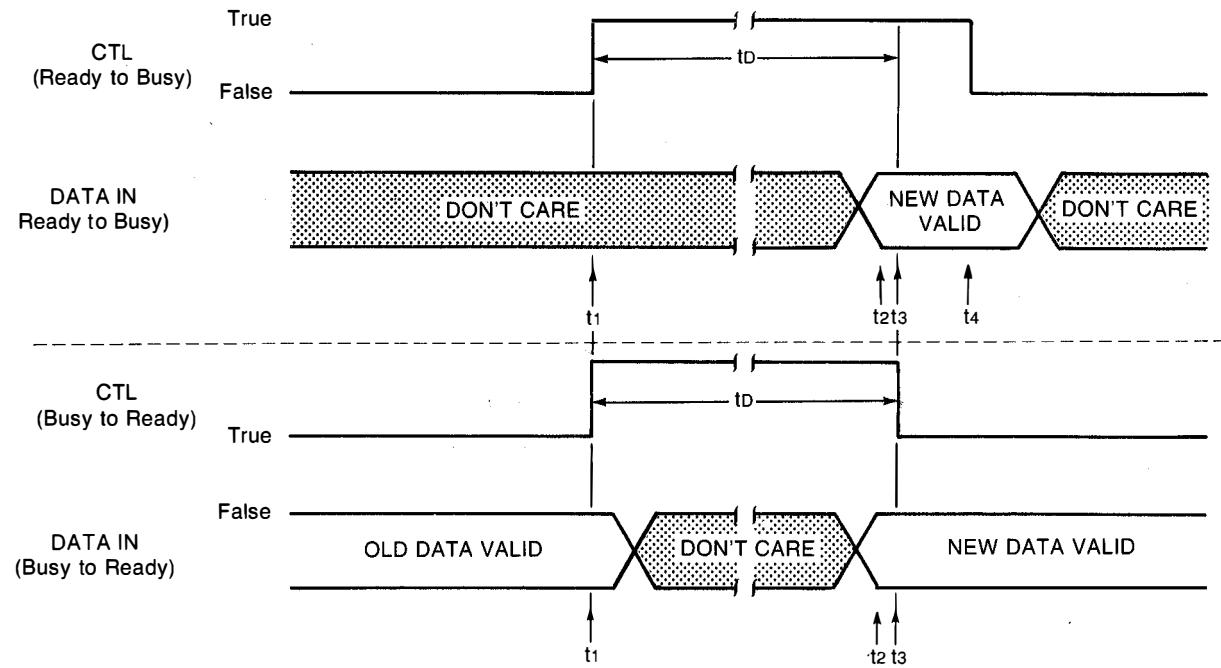
BUSY to READY: When the full handshake cycle starts, the computer checks for a READY indication on the FLG line. If the FLG line is READY, the interface requests data by setting CTL to the TRUE state (**t1**). This signal tells the peripheral device that it can place new data on the DATA lines. The peripheral acknowledges the CTL signal by placing FLG in the BUSY state (**t2**). The interface then acknowledges the FLG signal by returning CTL to the FALSE state (**t3**). After all these acknowledgments are taken care of, the peripheral device places new data on the DATA lines (**t4**). The peripheral device then signals that the data is valid by returning FLG to the READY state (**t5**). After the interface sees this signal, it inputs the data (sometime between **t5** of this cycle and **t1** of the next cycle).



READY to BUSY: The primary use of this handshake method is to input data that is being sent with a strobe handshake from the peripheral device. Partial handshake does not wait for the FLG line to be READY before starting the cycle. Regardless of the state of the FLG line, the request for data is made by setting CTL to the TRUE state (t_1). It does not matter if the peripheral device outputs the data first or starts the strobe pulse first. The important thing is that the data should be valid before the end of the strobe pulse. This diagram shows the strobe pulse starting first as the peripheral device places the FLG line in the READY state (t_2). The data becomes valid before the end of the pulse (t_3). Then the peripheral signals that the data is valid by placing the FLG line in the BUSY state (t_4). The minimum state times of $30 \mu s$ (t_{H1}) and $35 \mu s$ (t_{H2}) are necessary for the interface to detect this READY to BUSY transition. When the interface sees this transition, it inputs the data (sometime between t_4 and t_5). The interface then indicates receipt of the data by returning CTL to the FALSE state (t_5). Note that the difference between this option and the "BUSY to READY" option is timing of the input operation with respect to the end of the CTL pulse, not the polarity of the FLG pulse. Either option can be used with any polarity of FLG pulse (see "Setting the Logic Polarity").

BUSY to READY: This is a variation of the previous method. Regardless of the state of the FLG line, the request for data is made by setting CTL to the TRUE state (t_1). It does not matter if the peripheral device outputs the data first or starts the strobe pulse first. The important thing is that the data should be valid before the end of the strobe pulse. This diagram shows the strobe pulse starting first as the peripheral device places the FLG line in the BUSY state (t_2). The data becomes valid before the end of the pulse (t_3). Then the peripheral signals that the data is valid by placing the FLG line in the READY state (t_4). The minimum state times of $30 \mu s$ (t_{H1}) and $35 \mu s$ (t_{H2}) are necessary for the interface to detect the READY to BUSY transition.

After the pulse on the FLG line is finished, the interface returns CTL to the FALSE state (t5). The interface then inputs the data (sometime between t5 of this cycle and t1 of the next cycle). Note that the difference between this option and the “READY to BUSY” option is timing of the input operation with respect to the end of the CTL pulse, not the polarity of the FLG pulse. Either option can be used with any polarity of FLG pulse (see “Setting the Logic Polarity”).



Input: Strobe Handshake

READY to BUSY: This is a simple handshake method that can be used when you are sure that your peripheral device can provide valid data in a fixed amount of time after a request signal. The peripheral device is not given an opportunity to acknowledge any signals or control the handshake timing in any way. Therefore, the FLG line is not used. The cycle starts when the interface requests data by setting CTL to the TRUE state (t1). The peripheral device then places new data on the DATA lines (t2). After a programmable delay (tD), the interface inputs the data (sometime between t3 and t4). The interface completes the cycle by returning CTL to the FALSE state (t4).

BUSY to READY: This is a variation of the previous method. The cycle starts when the interface requests data by setting CTL to the TRUE state (t1). The peripheral device then places new data on the DATA lines (t2). After a programmable delay (tD), the interface returns CTL to the FALSE state (t3). Then the interface inputs the data (sometime between t3 of this cycle and t1 of the next cycle).

The “no handshake” option does not need a timing diagram. The interface simply inputs new data whenever a data input statement is executed. The FLG and CTL lines are not used. The ASSERT and STATUS statements can be used to supply your own handshake in this mode (see “Direct Use of Control Lines”).

Selecting the Handshake Method

The handshake characteristics of the GPIO are selected by writing various codes to interface control registers. The registers of interest are control registers 4, 6, and 9. These are accessed by using the CONTROL and STATUS statements.

Register 4 - Data Normalization and Handshake Control

Most Significant Bit								Least Significant Bit							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0								
Handshake Method		0 = Ready to Busy 1 = Busy to Ready	Not Used	Data Polarity (see "Selecting the Logic Polarity")											
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1								

Register 4 has two primary functions. The lower four bits are used to select either positive-true or negative-true data for each 8-bit port. This is explained in "Selecting the Logic Polarity". The top three bits are used to select the handshake method. The primary selection of handshake method is done with bit 6 and bit 7, as follows:

Bit 7	Bit 6	Handshake
0	0	Full
0	1	Partial
1	0	Strobe
1	1	None

Bit 5 of this register is used to select the input timing option. Its states are defined as follows:

Bit 5	Data Input Timing
0	READY to BUSY
1	BUSY to READY

The meaning of all these options is discussed at length in "Handshake Methods". The following is a summary of all the choices, listed with the decimal value of the control byte used to select each choice.

Decimal Value of Bit 5 thru Bit 7	Handshake Method
0	Output: Full Handshake
64	Output: Partial Handshake
128	Output: Strobe Handshake
0	Input: Full Handshake; READY to BUSY
32	Input: Full Handshake; BUSY to READY
64	Input: Partial Handshake; READY to BUSY
96	Input: Partial Handshake; BUSY to READY
128	Input: Strobe Handshake; READY to BUSY
160	Input: Strobe Handshake; BUSY to READY
192	Input or Output: No Handshake

It is possible to write the values shown directly into register 4. However, if you do that, you will also clear all the data normalization bits. This gives all data ports a positive-true logic sense. If that does not cause any problems, statements like the following can be used. These, and all other example statements in this section, assume that the interface select code is 4.

```
CONTROL 4,4 ; 128 ! Set strobe handshake
    isc      reg#   control byte
```

```
CONTROL 4,4 ; 96 ! Partial hndsk, input BUSY to READY
CONTROL 4,4 ; 192 ! Turn off handshake
```

You are encouraged to get into the habit of using comments on statements like these. The word "CONTROL" followed by a bunch of numbers can be very mysterious when you look at a program some months after it was written. Anyone who needs to support a program will be very thankful for a little bit of information about the action of a cryptic CONTROL statement. Remember, that support person just might be you!

Now suppose that you are concerned about affecting the normalization bits. There are two approaches to this problem. First, and most common, is to set all the options in register 4 with the same statement. This simply means that you determine the value of the bits used for handshake control, determine the value of the bits used for normalization, add those two values together, and use the sum as your control byte.

If for some reason you need to change the handshake bits after the normalization bits have been set, that's OK. The value of bits previously in the register can easily be maintained by using a couple extra statements. The general technique is to read the current register contents, mask out the old handshake bits, "OR" in the new handshake bits, then place the result back into the register. The following example shows the details of this process.

100 STATUS 4,4 ; C ! Read current value
110 C=BINAND(15,C) ! Clear B4 thru B7
120 C=BIMIOR(64,C) ! Set Partial hndshk mode
130 CONTROL 4,4 ; C ! Write new value

Another register affecting handshake is control register 6. This register establishes the delay time between data output and the setting of CTL, and it establishes the width of a strobe pulse. These times are shown in “Handshake Methods” as “tD”. The value in register 6 is used to establish an **additional** delay time which is added to a minimum time that is always present. The minimum times are generally around 60 μ s, but there are exceptions. Refer to the Installation and Theory of Operation manual for more specific details.

Register 6 - CTL Delay and Strobe Pulse Duration

Register 6 - CTL Delay and Strobe Pulse Duration							
Most Significant Bit	Least Significant Bit						
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Increment 0 = 10 μ s 1 = 1ms	Delay: Number of Increments						
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

The control byte in this register contains a delay value (bit 0 thru bit 6) and a range selection bit (bit 7). When bit 7 is clear, the value of the other bits is multiplied times 10 μ s to determine the additional delay time. When bit 7 is set, the value of the other bits is multiplied times 1 ms to determine the additional delay time. In other words, the lower seven bits specify how many time intervals to use, and bit 7 defines the size of each interval. This system yields two overlapping ranges that include times from 10 μ s to 127 ms. Here are some examples:

```
CONTROL 4,6 ; 20 ! Add 200 us of CTL delay
CONTROL 4,6 ; 130 ! Set 2 ms CTL delay
CONTROL 4,6 ; 128+50 ! Set 50 ms strobe pulse
```

The final handshake-related register is register 9. This register has only one active bit. It is used to select the “Output Inhibit” function. Although this feature is most often used with certain strobe handshake devices, it can be used with any handshake method. When the Output Inhibit function is disabled, all output handshakes work exactly as described in “Handshake Methods”.

If the Output Inhibit function is enabled, the output sequence is slightly modified. Enabling this function causes an additional handshake line to be assigned as an “inhibit” line. If the output port is using CTLA as a handshake line, then ST0 becomes the inhibit line. If the output port is using CTLB as a handshake line, then ST1 becomes the inhibit line. **If the output port is already using ST0 or ST1 as a normal handshake line, then the Output Inhibit function cannot be used.** The action of the Output Inhibit function is simple. Before starting an output cycle, the interface first checks the inhibit line. If the inhibit line is FALSE, the handshake proceeds in the normal manner. If the inhibit line is TRUE, the interface waits until inhibit returns to the FALSE state before proceeding with the output operation.

This function makes it easy to interface to devices that have a general “Busy” line that is not part of the normal handshake sequence. An example is a printer with an internal buffer. This type of device often uses a strobe handshake, since all the internal buffer needs is a pulse to latch the valid data. However, once the buffer is full, or a line ending is received, the printer “goes busy” and prints the entire buffer. Buffers of this type usually cannot print and receive data at the same time, so the GPIO interface must halt its output while the printer is busy. Hence the use of the Output Inhibit function.

Register 9 - Output Inhibit Function

Most Significant Bit								Least Significant Bit							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0								
Not Used								Enable Output Inhibit							
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1								

Bit 0 is the only bit used in this register. When bit 0 is clear (value=0), the Output Inhibit function is disabled and no inhibit line is used. When bit 0 is set (value=1), the Output Inhibit function is enabled and the inhibit line is assigned and monitored as described two paragraphs ago. Access to this register is shown in the following example statements:

```
CONTROL 4,9 : 1 ! Enable Output Inhibit
CONTROL 4,9 : 0 ! Don't use Output Inhibit
```

Setting the Logic Polarity

Register 3 allows you to individually determine the logic sense of each handshake line. Register 4 allows you to individually determine the logic sense of each data port. This is a tremendous amount of flexibility. The term “logic sense”, or “logic polarity” means whether the lines are treated as positive true or negative true. A positive-true line interprets a logic low as a “0” and a logic high as a “1”. A negative-true line interprets a logic low as a “1” and a logic high as a “0”. Note that if you change the normalization of any CTL line, the line changes states immediately after the normalization bit is changed in the control register.

Register 3 - Handshake Line Normalization

Most Significant Bit								Least Significant Bit							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Invert ST1	Invert ST0	Invert FLGB	Invert FLGA	Invert CTL1	Invert CTLB	Invert CTL0	Invert CTLA
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1								

Each bit in this register corresponds to one handshake line. When a bit is “0”, its corresponding line is positive true. When a bit is “1”, its corresponding line is negative true. In other words, each bit is used to enable or disable an inversion for its corresponding line. The following table shows the polarity definitions of the handshake lines.

Line Type	Normalization Bit	Logic Sense
FLG (or ST)	0	Logic HI = BUSY Logic LO = READY
	1	Logic HI = READY Logic LO = BUSY
CTL	0	Logic HI = TRUE Logic LO = FALSE
	1	Logic HI = FALSE Logic LO = TRUE

Here are some example statements:

```
CONTROL 4,3 ; 15 ! Invert CTL lines
CONTROL 4,3 ; 128+8 ! Invert Port D hndsk lines
CONTROL 4,3 ; 16+32 ! Invert FLGA and FLGB
```

Register 4 - Data Normalization and Handshake Control

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Handshake Control (see "Selecting the Handshake Method")	Not Used	Invert Port D Data	Invert Port C Data	Invert Port B Data	Invert Port A Data		
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This is the same register 4 discussed in "Selecting the Handshake Method". This time we are interested in the lower four bits. Each of these bits corresponds to one of the data ports. When a normalization bit is "0", all the data lines on the corresponding port are positive true. When a normalization bit is "1", all the data lines on the corresponding port are negative true.

Because this register contains normalization bits and handshake control bits, you should pay particular attention to what you are doing when you write to it. As was mentioned in "Selecting the Handshake Method", the cleanest approach is to set all the options with one statement. This simply means that you determine the value of the bits used for handshake control, determine the value of the bits used for port normalization, add those two values together, and use the sum as your control byte. For example, assume that you wanted to output negative true data from Port A using strobe handshake. The handshake control value is 128. To invert the data lines on Port A, a value of "1" is used. The total of these two values is 129. Therefore, the following statement would be used:

```
CONTROL 4,4 ; 129 ! Strobe hndsk; invert Port A data
```

isc reg# control byte

If for some reason you need to change the normalization bits after the handshake bits have been set, you can use the masking technique discussed in “Selecting the Handshake Method”. The following examples show two methods of isolating normalization bits. The first example sets Port A and Port C to negative true, set Port B and Port D to positive true, and leaves the handshake control bits unchanged.

```

100 STATUS 4,4 ; C ! Read current value
110 C=BINAND(240,C) ! Clear B0 thru B3
120 C=BINIOR(5,C) ! Invert Port A & C
130 CONTROL 4,4 ; C ! Write new value

```

The second example shows how the normalization of a single port can be changed without effecting any other bits in the register.

```

250 STATUS 4,4 ; X ! Read current value
260 X=BINAND(X,BTO("11111101")) ! Port B = POS. true
270 CONTROL 4,4 ; X ! Write new value

```

Why Won't This Thing Output?

So far you have seen how to set the method, timing, and polarity of handshake, how to set the logic polarity of the data, and how to address a port of the proper size, direction, and drive capability. This is enough information to get most of the ports working, but there is an extra little “trick” needed to activate the output drivers on Port A and Port B.

The GPIO interface has protection mechanisms built in that must be satisfied before Port A or Port B will output. The reason for this is to ensure that the output drivers are not accidentally activated while they are grounded or connected to current-sourcing circuitry. If the GPIO is trying to drive a device that is also trying to drive the GPIO, an electrical conflict results that will only be resolved by the death of one set of drivers. In other words, don't try an output operation on an input port. Without safeguards, this could happen as the result of a simple typing error when entering a device selector.

Although it is impossible to completely prevent human error, it is unlikely that you will accidentally generate an output from Port A or Port B. To enable the output drivers of Port A or Port B, you must set an enable bit in register 8. To set any enable bits in register 8, switch 4 of the default configuration switches must be on. The details on accessing and setting this switch are in the Installation and Theory of Operation manual. Trying to write to register 8 without first setting this switch results in Error 115.

Register 8 - Output Enable for Port A and Port B

Most Significant Bit								Least Significant Bit	
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Enable Port B Outputs	Enable Port A Outputs
Not Used									
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1		

Assuming that switch 4 has been properly set, the following example shows an output enable operation for Port B.

```
CONTROL 4,8 : 2 ! Enable Port B output
```

Choosing the Method of Transfer

The basic software link with the GPIO interface is the program statement used to perform the I/O operation. Assuming that you have studied Parts I, II, and III of this manual, choosing the proper statement is usually a simple matter. Some of the factors to be considered are reviewed here to help refresh your memory.

If you are using an 8-bit port and dealing with ASCII data, most of your needs can be met with simple OUTPUT and ENTER statements. For example, string data is output to Port B by the following statement:

```
OUTPUT 401 : A$
```

The following statement inputs a number that is being sent to Port A as an ASCII representation:

```
ENTER 400 : N
```

If straight binary numbers are being transferred, you will probably want to suppress the end-of-line sequence and use binary formatting. An 8-bit binary number can be output to Port C as follows:

```
OUTPUT 404 USING "#,B" : X
```

The following is an example of inputting a 16-bit binary number from Ports A&B using Port A handshake lines:

```
ENTER 408 USING "#,W" : K
```

If you are using 16-bit ports without the "W" image, be careful to move an even number of bytes. Attempting to move an odd number of bytes through a 16-bit port generates Error 113. A common oversight is trying to output data using free-field format. If your 16-bit data is stored as a string, the string must contain an even number of bytes and a "K" image should be used.

All I/O formatting capabilities are available for use with the GPIO interface. An image is used in the following example to output a column of numbers using the output side of the Port B/Port D combination:

```
OUTPUT 403 USING "4Z.00,/"; A,B,C
```

The details of alternate transfer methods, such as fast handshake or interrupt transfer, are covered next.

Advanced Capabilities

The first half of this section dealt with the basic characteristics of a parallel interface. This half covers the extensions to those capabilities that are provided by the 82940A interface. These extra features are:

- Specialized transfers: fast handshake and interrupt
- User-defined end-of-line sequences
- Sensing and controlling individual handshake lines
- Parity generating and checking
- Using external events to generate interrupts
- A trigger-byte function for transfer control

FHS and INTR Transfers

In addition to the normal OUTPUT and ENTER statements, the GPIO interface can be used with TRANSFER...FHS and TRANSFER...INTR statements. No special configuration is necessary to use INTR transfers. Simply set the control registers as you would for a normal OUTPUT or ENTER. The GPIO interface accepts DELIM and COUNT terminating conditions for an input transfer, but EOI is not defined. Transfer speeds are limited to about 400 bytes/sec using this method. The following example program shows the use of an interrupt transfer to input string data using full handshake and positive-true logic on Port A.

```

10 DIM A$(100),B$(100)
20 IOBUFFER B$
30 ON EOT 4 GOSUB 140
40 TRANSFER 400 TO B$ INTR ; DELIM 10
50 !
60 ! Dummy loop; indicates other processing
70 !
80 K=1
90 DISP K;@ K=K+1
100 GOTO 90
110 !
120 ! EOT Routine
130 !
140 ENTER B$ ; A$ ! Empty buffer
150 PRINT A$
160 TRANSFER 400 TO B$ INTR ; DELIM 10 @ RETURN

```

Notice that the TRANSFER statements have a “DELIM 10” parameter. Ten is the value of a line-feed character. This is a common delimiter when dealing with string data because it allows the TRANSFER to terminate at the end of each line of incoming text. Without the DELIM parameter, the TRANSFER would not terminate until the buffer was full. Another point to remember about buffers is shown in line 140. The incoming data could be viewed by simply printing B\$. However, this approach would leave data in the buffer. When more data came in, it would be appended to the existing data until an ERROR 126 was generated. Since this is usually not the desired action, the buffer should be emptied after each TRANSFER. ENTERing from the buffer is one way to empty it (as shown in the example). Another way to empty a buffer is to re-execute the IOBUFFER statement. For example:

```
140 PRINT B$  
150 IOBUFFER B$ ! Empty buffer
```

Performing an IOBUFFER is faster and simpler than performing an ENTER, but it also clears any CONVERT values you may have established. If this is not acceptable, the third choice is to perform a CONTROL to the buffer’s pointers. This is the fastest way to “empty” a buffer, although the statement is somewhat cryptic. The following is an example:

```
140 PRINT B$  
150 CONTROL B$,0 ; 1,0 ! Empty buffer
```

ABORTIO, ASSERT, HALT, RESET, and STATUS statements can be executed during an interrupt transfer. These statements take affect immediately and are the only statements that do so. All other interface-controlling statements (such as CLEAR, CONTROL, ENTER, OUTPUT, or another TRANSFER) wait until the end of the current transfer before taking effect. Therefore, the GPIO interface can only do one TRANSFER at a time. (ENTER from a buffer or OUTPUT to a buffer are not interface-controlling statements and can be performed at any time.)

The fast handshake transfer is a special case. In order to achieve maximum speed, certain constraints are placed on this transfer method. A FHS transfer is allowed for 8-bit data only. This TRANSFER type must use the Port A/Port C bidirectional configuration (primary address 06). Only full handshake is allowed, with FLGA and CTLA as the handshake lines. Normalization changes for FLGA, CTLA, Port A and Port C are allowed, but alternate handshake methods or port configurations cannot be used. A COUNT terminating condition can be specified for an input TRANSFER, but EOI is not defined. Transfer speeds of about 18 000 bytes/sec can be achieved using this method.

Because a FHS transfer locks out all interrupts (including the RESET key) and a full handshake is used, the computer can be completely “hung” or “locked up” if the peripheral device stops handshaking before the TRANSFER is complete. To help deal with this problem, a special feature has been added to the GPIO interface. A FHS transfer can be aborted by placing the ST0 line in the TRUE state. This can be done by an operator with a pushbutton or by a control line from the peripheral device. If ST0 is asserted during a FHS transfer, the transfer is aborted, ERROR 114 is generated, and the program stops. If you wish to trap this event and keep the program running, use an ON ERROR statement.

EOL Sequence

In its default state, the GPIO interface outputs a carriage-return/line-feed as the end-of-line (EOL) sequence. This sequence is sent at the end of each OUTPUT statement and whenever it is called for by an IMAGE statement. If you desire a different EOL sequence, any sequence up to seven characters long can be programmed by using the CONTROL statement. The EOL sequence is controlled by registers 16 thru 23. Register 16 holds a number 0 thru 7. This is the number of EOL characters in the sequence. The characters themselves are stored starting in register 17. For example, the default sequence has the value 2 (number of characters) in register 16 and the character values 13 (carriage-return) and 10 (line-feed) in registers 17 and 18, respectively. Although these registers can be changed using the CONTROL statement, they cannot be examined with the STATUS statement. In other words, they are write-only registers.

The following program statement redefines the EOL sequence to be four control characters: bell, carriage-return, line-feed, DC3.

```
CONTROL 4,16 ; 4,7,13,10,19
```

It is important to note that the EOL sequence is also sent at the end of an outgoing TRANSFER operation. If you don't pay close attention to the contents of the buffer, this can produce an extra, unwanted carriage-return/line-feed. For example:

```
10 IOBUFFER A$  
20 OUTPUT A$ ; "Hello"  
30 TRANSFER A$ TO 406 INTR
```

This program sequence outputs the word "Hello" followed by a carriage-return/line-feed **and** an EOL sequence. The carriage-return/line-feed is placed in the buffer by the OUTPUT statement. The EOL sequence is added by the GPIO after it sends the buffer contents. To avoid this duplication, one of the EOL sequences must be suppressed. To suppress the carriage-return/line-feed in the buffer, the following change can be made. This change uses an OUTPUT image to place the string into the buffer without any EOL characters.

```
10 IOBUFFER A$  
20 OUTPUT A$ USING "#,K" ; "Hello"  
30 TRANSFER A$ TO 406 INTR
```

To suppress the EOL sequence sent by the GPIO, the following change can be used. This change writes a value of zero to register 16, which tells the interface to output zero EOL characters. Unless you need an EOL sequence different from carriage-return/line-feed, this method is the most convenient of the two. It is especially handy when transferring strings and text, since blank lines (null strings) can be output without worrying about transferring an empty buffer.

```
10 IOBUFFER A$  
20 CONTROL 4,16 ; 0 ! Use no EOL characters  
30 OUTPUT A$ ; "Hello"  
40 TRANSFER A$ TO 406 INTR
```

Note that the programmable EOL sequence is sent to 8-bit ports only. When 16-bit ports are used (primary addresses 08 thru 15), no EOL bit patterns are sent. You do not need to suppress the EOL sequence when using 16-bit data, just be aware that it is not sent.

Direct Use of Control Lines

The state of all the GPIO handshake lines can be monitored and controlled by using the STATUS and ASSERT statements. An ASSERT statement can set the state of any CTL or RESET (RES) line. The STATUS statement can read the state of any FLG, ST, or CTL line. The following registers are associated with these lines.

Register 2 (read) - Line Status

Most Significant Bit								Least Significant Bit
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
ST1	ST0	FLGB	FLGA	CTL1	CTLB	CTL0	CTLA	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1	

Assertion Control and Register 2 (write)

Most Significant Bit								Least Significant Bit	
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
<u>RESB</u>	<u>RESA</u>	Not Used		CTL1	CTLB	CTL0	CTLA		
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1		

The following simple example shows a method of checking a handshake line. This program reads register 2 and checks the state of ST1. If ST1 is FALSE, the program loops back and continues checking. When the ST1 line is TRUE, the program takes an alternate path. This example just beeps and goes back to the main loop. In actual application, the BEEP statement would be replaced by whatever action is appropriate as a response to the line being monitored.

```

10 STATUS 4,2 ; X
20 IF NOT BIT(X,7) THEN GOTO 10
30 !
40 ! Your program goes here
50 !
60 BEEP
70 GOTO 10

```

Register 2 returns the logical state of the lines, not necessarily their electrical state. Therefore, a line state shows as "1" if that line is TRUE. It does not matter if a logic HI or a logic LO has been defined as the TRUE state (see "Setting the Logic Polarity").

The ASSERT statement has a similar definition. If you set a bit to "1", the corresponding line is placed in the TRUE state. That TRUE state might be a logic HI or a logic LO, depending upon the normalization bits in register 3. The RES lines do not have normalization bits and are always negative true. If you use a bit value of "1" to ASSERT a RES line, that line is placed in the logic LO state.

The following example shows the use of an ASSERT statement to produce a custom handshake sequence. This example sets the CTLA line TRUE, outputs a byte, waits 500 ms, then sets CTLA FALSE. Notice that to control the handshake line directly, the interface handshake options must be turned off.

```

10 CONTROL 4,4 ; 192 ! Turn off handshake
20 ASSERT 4;1 ! CTLA True
30 OUTPUT 406 USING "#,B" ; X
40 WAIT 500
50 ASSERT 4;0 ! CTLA False

```

The following example pulses RESA and RESB.

```

10 ASSERT 4;192
20 ASSERT 4;0

```

The outgoing control lines can also be set by using a CONTROL statement directed to register 2. However, the action of a CONTROL statement does not take effect until the current process of the interface is completed. The ASSERT statement, like the STATUS statement, takes effect immediately.

Parity

The GPIO interface provides both parity generation and parity checking. **Parity** is a simple method of detecting erroneous transmissions of data. It uses one bit of data as an indicator which is set or cleared according to known rules. If the sender and receiver are both using the same set of rules, any disagreement in the state of the **parity bit** indicates a possible transmission error. The GPIO can only use parity with 8-bit ports and data that has 7 bits (or less). The most common data type that fits these requirements is the ASCII character set.

There are five parity choices on the GPIO.

- **ZERO** parity
- **ONE** parity
- **EVEN** parity
- **ODD** parity
- **NO** parity

If **ZERO** parity is selected, all outgoing characters have “0” in their eighth bit. All incoming characters are checked to make sure that the eighth bit is “0”. If any incoming character has “1” in the top bit, a parity error is generated. **ONE** parity is similar to **ZERO** parity, except that the eighth bit is set to “1” for outgoing data and checked for “1” on the incoming data.

EVEN parity and **ODD** parity are more sophisticated than that. The object of these methods is to ensure that all data bytes have an even or odd number of “1” bits in them. For example, consider the letter “A” being output with odd parity. The bit pattern for this character is “01000001”. There are two bits set to “1”. When odd parity is applied to the character, the top bit is set so that there are an odd number of 1’s. The resultant pattern is “11000001”. **EVEN** parity operates in a like manner, except that the top bit is used to guarantee an even number of “1” bits. If an incoming byte does not have the proper number of 1’s in it, a parity error is generated.

A parity error can be detected in two ways. If an incoming character generates a parity error, its eighth bit is set to “1”. This can be detected by the **BIT** function. Also, if the character is printed or displayed on the HP-85, the top bit shows up as an underline. The second way to detect parity is to use an **ON INTR** statement. This is explained in “Event Interrupts”.

The parity function is selected by control register 0. The desired parity mode is specified by using a **CONTROL** statement to set one of the lower four bits. The current parity mode cannot be read by using a **STATUS** statement. Reading the status of register 0 always results in the value “4”, which is the interface ID code.

Register 0 (write) - Parity Control

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Enable ODD Parity	Enable EVEN Parity	Enable ONE Parity	Enable ZERO Parity
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Here are some examples of parity selection:

```
CONTROL 4,0 ; 8 ! Use ODD Parity
CONTROL 4,0 ; 0 ! Turn OFF Parity
```

If more than one parity mode is selected, the one occupying the lowest bit position has preference. For example, if you try to select both EVEN and ODD parity, EVEN parity will be used. If you try to use parity with 8-bit data, the top bit of the data will be replaced by the parity bit.

Event Interrupts

There are two kinds of interrupts available with the GPIO interface. One kind is used for **INTR** transfers. This kind is handled automatically by the computer and interface. All you need to do is specify "INTR" in your TRANSFER statement and provide an ON EOT statement. The other kind of interrupt is used for **ON INTR** programming. This kind requires more involvement from the programmer and is referred to as an "event interrupt".

The following is a summary of the steps necessary to use event interrupts with the GPIO.

1. Have an interrupt service routine in your program that performs the desired tasks as a result of the interrupt.
2. Provide an ON INTR statement to direct the program to the service routine in the event of an interrupt.
3. Use an ENABLE INTR statement to select the event(s) that you want to cause an interrupt.
4. When an interrupt takes the program to the service routine, read the STATUS of register 1. This lets you determine the cause of the interrupt and is a necessary part of the interrupt-handling protocol.
5. If you are expecting further interrupts, do another ENABLE INTR on the same line as the RETURN statement in the interrupt service routine.

Events that can cause an interrupt are selected and detected by using register 1. These events and their corresponding bit positions are shown in the following diagram.

Register 1 - Interrupt Enable/Cause

Register 1 - Interrupt Enable/Cause							
Most Significant Bit	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Least Significant Bit
ST1 Interrupt	ST0 Interrupt	FLGB Interrupt	FLGA Interrupt	Not Used		Parity Error Interrupt	Not Used
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

The first example shows the simplest form of interrupt handling. This program counts pulses on the FLGA line by using interrupts. Each time FLGA goes TRUE, an end-of-line branch directs the program to the interrupt service routine (line 120). This routine performs the necessary STATUS 4,1 statement, waits for the end of the pulse, increments a counter, enables the interface for the next interrupt, and then returns to the main program. The main program simply monitors the counter and beeps when a count of 10 is reached. In actual applications, an action more significant than beeping would be included. For example, the interrupt might be generated by a photocell detector that is counting items moving on a conveyor into a packing carton. When the proper number of items is in the carton, the main program might send bit patterns to an output port that controls the folding and sealing of the carton.

```

10 ON INTR 4 GOSUB 120
20 ENABLE INTR 4;16 ! FLGA interrupt
30 !
40 ! Main Program goes here
50 !
60 N=0
70 IF N<10 THEN GOTO 70 ! Check counter
80 BEEP @ GOTO 60
90 !
100 ! Interrupt Service Routine
110 !
120 STATUS 4,1 ; S
130 ! Wait for end of FLGA pulse
140 STATUS 4,2 ; X
150 IF BIT(X,4) THEN 140
160 N=N+1 ! Increment counter
170 ENABLE INTR 4;16 @ RETURN

```

When dealing with interrupts in real-life applications, it is very important to consider the timing of the interrupt pulse. Both the width and the frequency of the pulse are important factors. The number of interrupts per second that can be handled depends greatly upon the amount of processing being done in the interrupt service routine and the amount of time spent on the lines in the main program. To get the fastest response time, keep the processing in the service routine to an absolute minimum. It is also helpful to keep the main program moving from line to line as fast as possible. This can be done by coding efficiently and by not using the “@” symbol to combine statements on a line. In any event, it is unlikely that you will be able to handle interrupts that are more closely spaced than 25 ms or so.

Interrupt processing can be adversely affected by pulse widths that are too long or too short. An interrupt pulse must be over 100 μ s long for the interface to “see” it under ideal conditions. If the interface is busy

with other tasks, such as handling data or communicating with the computer, even longer pulses are necessary. However, if the interrupt pulse is too long, the same pulse can cause more than one interrupt. This happens if the computer makes it through the interrupt service routine and re-enables interrupts before the pulse has ended. This problem does have a solution, as shown by lines 140 and 150 in the preceding example. If you expect a relatively long pulse, you can use the STATUS statement to ensure that the pulse has ended before re-enabling for more interrupts. When this technique is used, the only disadvantage of long interrupt pulses is that they will slow down the program.

Not all event interrupts are based on pulses. A handshake action is sometimes employed. In these applications, the interrupting device holds its line TRUE until the interface sends a signal that clears it. When this is the case, the interrupt service routine can use ASSERT statements to cause the peripheral device to "drop" its request before interrupts are re-enabled.

The next example shows the detection of more than one interrupt cause. Assume for this example that the HP-85 is controlling a test station. This is a hypothetical test for circuit reliability at certain temperatures. The example program does two things. The main loop sends data to the circuit under test and reads the response. If the response is equal to the data sent, the circuit is OK. This operation uses full handshake on the Port A/Port C combination.

The interrupt service routine runs the temperature controller. An interrupt from ST0 indicates that the heater must be turned on, while an interrupt from ST1 tells the computer to turn the heater off. The heater is controlled by bit 0 in Port D. This operation uses strobe handshake.

Admittedly, a simple thermostat could be set up without involving the computer. This example merely demonstrates some principles of handling interrupt-driven events. In actual applications, you will probably be doing something more complex than turning a heater on and off.

```

10 IMAGE #,B
20 CONTROL 4,4 ; 0 ! Set full hndshk
30 ON INTR 4 GOSUB 170
40 ENABLE INTR 4:(128+64) ! ST0 & ST1 interrupt
50 !
60 E=0 ! Reset error counter
70 FOR K=0 TO 255
80 OUTPUT 402 USING 10 ; K ! Test value to Port C
90 ENTER 402 USING 10 ; X ! Response from Port A
100 IF X<>K THEN E=E+1 ! Count errors
110 NEXT K
120 PRINT E;"Errors for this test"
130 GOTO 60 ! Repeat continuously
140 !
150 ! Interrupt service routine
160 !
170 STATUS 4,1 ; S
180 IF BIT(S,6) THEN H=1 ! Heater on
190 IF BIT(S,7) THEN H=0 ! Heater off
200 CONTROL 4,4 ; 128 ! Go to strobe hndshk
210 OUTPUT 403 USING 10 ; H ! Heater byte to Port D
220 CONTROL 4,4 ; 0 ! Return to full hndshk
230 ENABLE INTR 4:(128+64) @ RETURN

```

Notice that a binary image is used for all I/O statements because actual binary values are being handled, not ASCII representations. Primary address 03 is used to access Port D so that FLGB and CTLB are used for handshake. If primary address 05 were used, there would be a conflict between the use of ST1 as an interrupt line and as a handshake line. Notice also that register 4 defines the handshake mode for all ports on the interface. Line 200 sets up a strobe handshake for the Port D operation. Then line 220 re-establishes the full handshake mode that is needed by Port A and Port C in the main routine.

The previous example assumed that the two interrupts would never occur at the same time. This is not always the case. Many applications need to handle multiple interrupts that may occur in any order or in any combination. To deal with this situation, it is recommended that you employ a carefully structured polling routine. The need for proper polling is especially critical since the HP-85 does not have a priority system for interrupts (see "Interactions and Permutations" in Section 9). The recommended polling technique is shown in the following example.

This program uses interrupts from ST0, ST1, and FLGB to demonstrate the concept of polling. When an interrupt occurs, the program branches to line 100. The required STATUS statement reads the interrupt cause register. Note that only one STATUS 4,1 statement is used. The interrupt cause register is automatically cleared when it is read. Therefore, attempts to identify multiple causes by repeatedly reading register 1 would be futile. This is not a problem since the register contents are placed in variable "S", where they can be inspected as often as necessary. Variable "S" is tested by the BIT function to isolate the interrupt causes. The IF...GOSUB structure is the simplest and least confusing method of dealing with multiple causes. This technique allows all causes to be tested before returning to the main program and yields independent subroutines for handling each cause. If two or more interrupts occur simultaneously, the one that is tested first in the polling routine will be serviced first.

```

10 ON INTR 4 GOSUB 100
20 E=128+64+32 ! IRQ conditions
30 ENABLE INTR 4,E
40 !
50 ! Main program goes here
60 GOTO 60
70 !
80 ! Interrupt Polling routine
90 !
100 STATUS 4,1 , S
110 IF BIT(S,5) THEN GOSUB 180
120 IF BIT(S,6) THEN GOSUB 210
130 IF BIT(S,7) THEN GOSUB 240
140 ENABLE INTR 4,E @ RETURN
150 !
160 ! Interrupt service routines
170 !
180 DISP "FLGB Interrupt"
190 RETURN
200 !
210 DISP "ST0 Interrupt"
220 RETURN
230 !
240 DISP "ST1 Interrupt"
250 RETURN

```

Note: Do not use FLG or ST interrupts and the TRANSFER statement at the same time. During a TRANSFER, the interface's resources are dedicated to the data movement operation. An external interrupt is not recognized during a TRANSFER. If a FLG or ST interrupt occurs at the end of an incoming TRANSFER, the computer and interface might "lock up".

Parity error is the only event interrupt that is recognized during an interrupt TRANSFER. Interrupts for FLG and ST lines should be disabled before a TRANSFER starts and re-enabled (if desired) after the TRANSFER completes. Any form of event interrupt may be used in conjunction with OUTPUT and ENTER statements, but remember that the end-of-line branch is not taken until the OUTPUT or ENTER statement completes.

The Trigger Function

An elegant feature of the 82940A is its ability to initiate actions upon the detection of a trigger byte. The GPIO interface can input and inspect data without interacting with the computer, thereby freeing computer time for other operations. The trigger byte is defined by a CONTROL statement. Incoming data can be tested for "less than", "greater than", or "equal to" the trigger byte, or any combination of these conditions. When a trigger condition is detected, the interface can initiate an interrupt transfer or signal the peripheral device with a CTL line. The registers involved are register 5 and register 7.

The trigger byte itself is written into or read from register 7. The following statement defines the binary value 127 as the trigger byte:

```
CONTROL 4,7 ; 127
```

The trigger actions are established by the top four bits in register 5. The lower four bits of this register hold the currently-assigned primary address. As was the case for register 4, writing only high-order bit values will clear the lower bits. However, if primary addresses are included in the OUTPUT, ENTER, or TRANSFER statements, register 5 will be updated automatically for every operation. This means that you don't have to worry about those lower four bits if the primary addresses are included in your program. If you are concerned about preserving the primary address in register 5 for some special reason, use a masking technique like the one explained in "Selecting the Handshake Method".

Register 5 - Primary Address and Trigger Action

Register 5 - Primary Address and Trigger Action							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger If Data <R7	Trigger If Data = R7	Trigger If Data >R7	If Trigger, Pulse CTL	Primary Address			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

The trigger compare bits can be summarized as follows:

Comparison Test	Register 5 Value
Data > Register 7	32
Data ≥ Register 7	96
Data < Register 7	128
Data ≤ Register 7	192
Data = Register 7	64
Data ≠ Register 7	160
No Trigger	0

The primary trigger function is to initiate an interrupt transfer. The following example establishes an interrupt transfer that begins when the interface detects a DC2 control character (decimal value 18) and ends when the buffer is full. Port A of a Port A/Port C combination is used for the input.

```

10 DIM B$[40]
20 IOBUFFER B$
30 CONTROL 4,5 : 64 ! Start if == trigger
40 CONTROL 4,7 : 18 ! DC2 is trigger
50 ON EOT 4 GOTO 160
60 TRANSFER 402 TO B$ INTR : DELIM 10
70 !
80 ! Dummy loop: indicates other processing
90 !
100 K=1
110 DISP K:@ K=K+1
120 GOTO 110
130 !
140 ! EOT Routine
150 !
160 ENTER B$ : X,Y
170 PRINT X,Y
180 END

```

In addition to initiating a TRANSFER, the trigger function can also signal a peripheral device that the trigger byte has been detected. The signal is a short pulse on a CTL line. If FLGA/CTLA are being used as the handshake lines, the pulse is sent on CTL0. If FLGB/CTLB are being used as the handshake lines, the pulse is sent on CTL1. The line is pulsed from FALSE to TRUE and back to FALSE, with normalization changes allowed. The pulse width is roughly 40 μ s.

This function, called "auto response", is enabled in register 5. If bit 4 of register 5 is set to "1", the appropriate CTL line will be pulsed when an incoming byte meets the trigger byte requirements. If bit 4 is "0", the auto response function is disabled. To enable this function, add 16 to the values shown in the preceding summary table.

Note that this feature is available **in addition to** the automatic start of a TRANSFER. Auto response cannot be selected without setting up a TRANSFER of at least one byte. The following example shows the minimum software required to produce an auto response pulse. This example is inputting data from Port A and using CTL0 as the auto-response line. The pulse will be sent when the first control character (value<32) is input.

```

10 IOBUFFER Z$           ; Set up buffer
20 CONTROL 4,7 ; 32 ! Trigger byte
30 CONTROL 4,5 ; 128+16 ! If < R7, pulse CTL
40 ON EOT 4 GOSUB 120   ; Set up interrupt
50 TRANSFER 400 TO Z$ INTR ; COUNT 1
60 !
70 ! Main program goes here
80 GOTO 80
90 !
100 ! EOT routine does nothing
110 !
120 RETURN

```

This example can be enhanced to produce a CTL0 pulse for **every trigger condition detected**, instead of just the first. Assume that you want an auto response pulse for each form-feed character (value=12). The EOT routine is changed to empty the buffer and start a new TRANSFER.

```

10 IOBUFFER Z$           ; Set up buffer
20 CONTROL 4,7 ; 12 ! FF is trigger
30 CONTROL 4,5 ; 64+16 ! If = R7, pulse CTL
40 ON EOT 4 GOSUB 120   ; Set up interrupt
50 TRANSFER 400 TO Z$ INTR ; COUNT 1
60 !
70 ! Main program goes here
80 GOTO 80
90 !
100 ! EOT routine
110 !
120 IOBUFFER Z$ ! Empty buffer
130 TRANSFER 400 TO Z$ INTR ; COUNT 1 @ RETURN

```

GPIO I/O Statements

Statement	Description
ABORTIO	Aborts any interrupt transfer in progress, sets all CTL lines to the FALSE state, places ports A and B in the high-impedance state, and sets ports C and D to the OFF state.
ASSERT	Immediately writes a value to control register 2, placing the CTL and RES lines in the specified states.
CLEAR	Pulses RES line(s). CLEAR 4 pulses both RESA and RESB. CLEAR 400 pulses RESA; CLEAR 401 pulses RESB (assuming that 4 is the interface select code).
CONTROL	Writes values to the interface control registers.
ENABLE INTR	Writes enable mask to control register 1. Used to select event interrupts.
ENTER	Enters data from a port to the BASIC program.
HALT	Stops any interrupt transfer in progress, leaving all handshake lines in their current state. Thus, STATUS can be used to troubleshoot a faulty handshake sequence.
OUTPUT	Outputs data from the BASIC program to a port.
RESET	Places interface in the power-on state. CTL lines are FALSE, OUTA and OUTB indicate output, ports A and B are put in high-impedance state, ports C and D are OFF.
SEND	SEND...CMD can set primary address or pulse RES line(s) with Device Clear. SEND...DATA outputs data bytes and EOL if specified. SEND...LISTEN or SEND...TALK can set primary address. UNL, UNT, MLA, MTA are ignored; SCG generates ERROR 111.
STATUS	Reads values from interface status registers.
TRANSFER	Moves data from a port to a buffer or from a buffer to a port. Interrupt or fast handshake may be used.

GPIO Interface Errors

Error No.	Meaning	Possible Cause
113	Word cut in half during a 16-bit operation.	Odd byte count specified in an image or COUNT parameter. Also beware of odd-length buffers and free-field format.
114	FHS transfer was aborted by ST0.	TRUE state on ST0. See "FHS and INTR Transfers."
115	Output to Port A or Port B is not allowed.	Using the wrong primary address in an output operation. Not enabling the proper bit in register 8. Attempting to write to register 8 when switch 4 is not properly set. Switch 4 is explained in the Installation and Theory of Operation Manual
116	CTL line not in proper state to start handshake.	A previous ASSERT or CONTROL statement that left CTL in the TRUE state. CTL must be in the FALSE state at the start of an handshake operation.

Part V
Appendix



Syntax Reference

Conventions Used to Represent Syntax

This reference section uses two methods of representing the syntax of I/O ROM statements. The conventions of each form are as follows.

Pictorial Representation

All items enclosed by a rounded envelope must be entered exactly as shown. Items enclosed by a rectangular box are names of parameters used in the statement. A description of each parameter is given in the text following the drawing. Statement elements are connected by lines. Each line can only be followed in one direction, as indicated by the arrow at the end of the line. Any combination of statement elements that can be generated by following the lines in the proper direction is syntactically correct. A statement element is optional if there is a valid path around it. This form of syntax representation is easy to use, and in some cases, more formally correct than the alternate form described next.

Linear Representation

This form of syntax representation is included to be compatible with previous HP-85 manuals. Many users are accustomed to seeing this form. If both forms are new to you, it is recommended that you concentrate on the Pictorial form.

DOT MATRIX : All items shown in dot matrix must be entered exactly as shown.

[] : Items within square brackets are optional.

| : A vertical line between two items reads as "or"; only one of the items may be included.

... : Three dots indicate that successive parameters are allowed.

{ } : Braces are used to indicate a group of items from which one item must be chosen.

ABORTIO

Syntax



ABORTIO interface select code

Example Statements

```

100 ABORTIO 7
250 IF S<128 THEN ABORTIO S0

```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

All interfaces: Terminates any interrupt TRANSFER in progress. Performing an ABORTIO on an interface with an active transfer and EOT branching enabled causes the branch to be taken.

HP-IB:

- System Controller: Sends Interface Clear (IFC) and Remote Enable (REN).
- Active Controller (but not System Controller): Sends Attention (ATN) and My Talk Address (MTA).
- Non-controller: Stops handshaking data and becomes ready for next operation.

Serial: Turns off all modem control lines (control register 2).

BCD: Stops handshaking data, sets CTL lines false, and places external data lines in high-impedance state.

GPIO: Stops handshaking data, sets control lines false, places ports A and B in high-impedance state, and sets lines from ports C and D to false state.

Related Statements

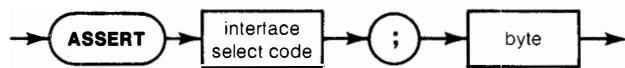
```

HALT
ON EOT
RESET

```

ASSERT

Syntax



ASSERT interface select code ; byte

Example Statements

```

100 ASSERT 7 ; 12
210 IF A1=128 THEN ASSERT S ; X
  
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

byte — a numeric expression that evaluates to an integer 0 thru 255. Binary value of the byte is used to set or clear the lines to be asserted.

Actions Taken

HP-IB: Immediately writes the value of the byte to control register 2. IFC bit (2^7 , decimal 128) is ignored (use ABORTIO).

Serial, BCD, GPIO: Immediately writes the value of the byte to control register 2.

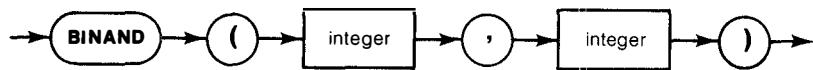
Related Statements

ABORTIO

CONTROL

BINAND

Syntax



`BINAND (integer , integer)`

Example Statements

```

10 B1=BINAND(X1,15)
100 PRINT BINAND(I,N*2^3)
  
```

Parameters

integer — a numeric expression that evaluates to an integer –32 768 thru 32 767.

Action Taken

BINAND is a function that returns the 16-bit binary AND of two integer values. Each bit of the result is calculated using the corresponding bit of each argument, according to the following truth table:

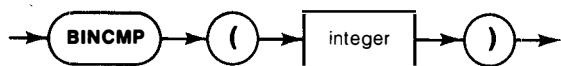
Arg.1	Arg.2	Result
0	0	0
0	1	0
1	0	0
1	1	1

Related Statements

BINCMP
BINEOR
BINIOR
BIT

BINCMP

Syntax



`BINCMP <integer>`

Example Statements

```
100 C=BINCMP(X1)  
120 PRINT BINCMP(N*2^3)
```

Parameters

integer — a numeric expression that evaluates to an integer -32 768 thru 32 767.

Action Taken

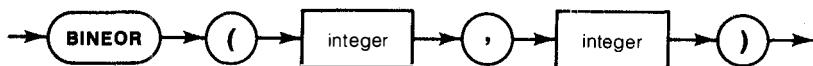
BINCMP is a function that returns the 16-bit binary complement of an integer value. Each bit of the result is the inverse of the corresponding bit in the argument. If the argument has less than 16 bits, leading zeros are assumed.

Related Statements

BINAND
BINEOR
BINIOR
BIT

BINEOR

Syntax



`BINEOR (integer , integer)`

Example Statements

```

20 B1=BINEOR(X1,15)
140 PRINT BINEOR(I,2^N)
  
```

Parameters

integer — a numeric expression that evaluates to an integer -32 768 thru 32 767.

Action Taken

BINEOR is a function that returns the 16-bit binary exclusive OR of two integer values. Each bit of the result is calculated using the corresponding bit of each argument, according to the following truth table:

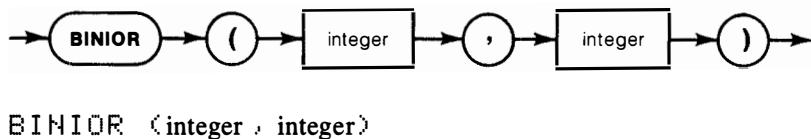
Arg.1	Arg.2	Result
0	0	0
0	1	1
1	0	1
1	1	0

Related Statements

BINAND
BINCMP
BINIOR
BIT

BINIOR

Syntax



Example Statements

```

30 Y=BINIOR(X1,255)
160 DISP BINIOR(I,2^N)

```

Parameters

integer — a numeric expression that evaluates to an integer –32 768 thru 32 767.

Action Taken

BINIOR is a function that returns the 16-bit binary inclusive OR of two integer values. Each bit of the result is calculated using the corresponding bit of each argument, according to the following truth table:

Arg.1	Arg.2	Result
0	0	0
0	1	1
1	0	1
1	1	1

Related Statements

BINAND
BINCMP
BINEOR
BIT

BIT

Syntax



`BIT (integer , bit position)`

Example Statements

```

40 Y=BIT(X3,7)
180 IF BIT(N,2^I) THEN GOTO 220
  
```

Parameters

integer — a numeric expression that evaluates to an integer -32 768 thru 32 767.

bit position — a numeric expression that evaluates to an integer 0 thru 15. Least-significant bit is in position 0, most-significant in position 15.

Action Taken

BIT is a function that returns the value of one bit in an integer argument. Result of the function is TRUE if bit is set, FALSE if bit is clear.

Related Statements

BINAND

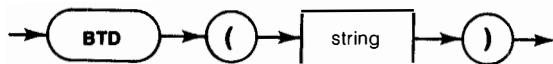
BINCMP

BINEOR

BINIOR

BTD

Syntax



BTD (string)

Example Statements

```
20 X=BTD(H$&L$)+A1  
130 DISP BTD("11000001")
```

Parameters

string — a string expression that contains the base 2 representation of an integer. Limited to 16 significant characters that must be “1” or “0”.

Action Taken

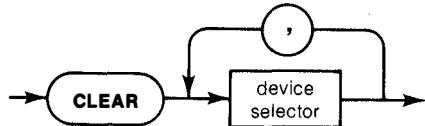
BTD is a function that returns the value of a base 2 representation contained in the string argument. The argument is a character representation and the result is a numeric quantity.

Related Statements

DTB\$
DTH\$
DTO\$
HTD
OTD

CLEAR

Syntax



`CLEAR device selector [, device selector]...`

Example Statements

`60 CLEAR 3`
`250 CLEAR S*100+D1,S*100+D2`

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”). If multiple device selectors are specified, they must all be on the same interface select code.

Actions Taken

HP-IB: Must be Active Controller. Leaves ATN true; use RESUME if you wish to set ATN false.

- If device selector is only an interface select code, interface sends a Device Clear (DCL).
- If device selector contains a primary address, interface sends a Selected Device Clear (SDC).

Serial, BCD: Error

GPIO:

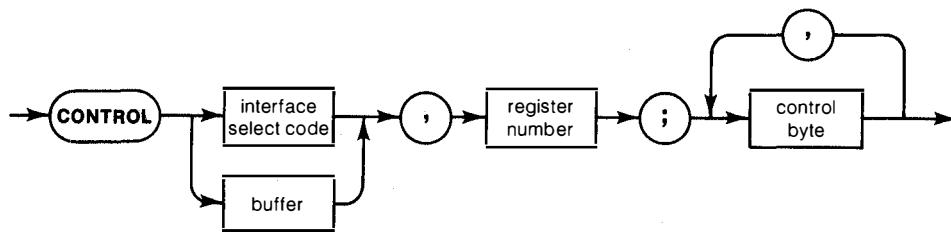
- If device selector is only an interface select code, interface pulses RESA and RESB.
- If device selector contains an even primary address, interface pulses RESA.
- If device selector contains an odd primary address, interface pulses RESB.

Related Statements

CONTROL
SEND

CONTROL

Syntax



`CONTROL {interface select code| buffer}, register number ; control byte[, control byte] ...`

Example Statements

```
10 CONTROL 9,1 ; C1,C2,C3,C4,C5,C6
50 CONTROL S,R ; C(R)
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

buffer — the name of a string variable that has been declared as an IOBUFFER.

register number — a numeric expression that evaluates to an integer 0 thru 23. Must specify a valid control register for the selected interface.

control byte — a numeric expression that evaluates to an integer 0 thru 255. Binary value of byte is used set and clear bits in the control register.

Action Taken

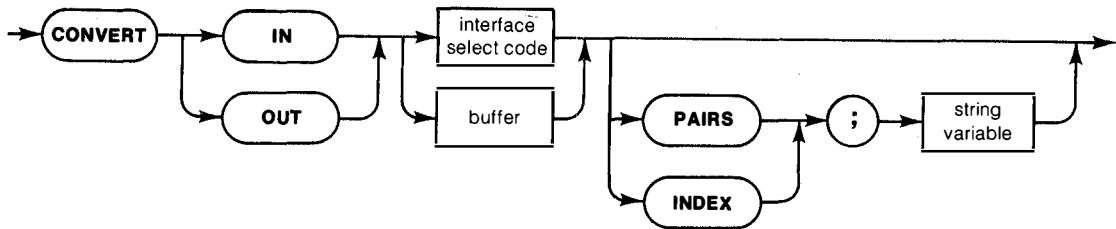
CONTROL writes one or more control bytes to interface or buffer control registers. The register number specifies the first register to be used. If multiple control bytes are specified, they are stored in consecutive control registers, beginning with the specified register number.

Related Statements

- ABORTIO
- ASSERT
- ENABLE INTR
- IOBUFFER
- STATUS

CONVERT

Syntax



`CONVERT{ IN| OUT }{ interface select code| buffer }[{ PAIRS| INDEX } ; string variable]`

Example Statements

```

20 CONVERT OUT 4 PAIRS ; A$
50 CONVERT IN 10 INDEX ; C$
110 CONVERT IN 7 ! Turn off conversion
  
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

buffer — the name of a string variable that has been declared as an IOBUFFER.

string variable — the name of a string variable in which the conversion table has been previously stored.

Actions Taken

Enables or disables a character conversion process for a specified interface or buffer and a specified direction. Although you can CONVERT using either an interface or a buffer, conversions can only be performed with OUTPUT and ENTER statements. Conversions are **not** performed during SEND or TRANSFER.

If the optional parameters are not included (as in the 3rd example statement), a previously selected conversion is turned off for the specified interface and direction.

If direction is specified as "IN", all bytes being input from the specified source are processed through a conversion table immediately after they are received from the source. If direction is specified as "OUT", all bytes being output to the specified destination are processed through a conversion table immediately before they are sent to the destination. "IN" and "OUT" conversions may both be specified for a given interface select code or buffer.

If conversion method “PAIRS” is specified, the conversion table is treated as a sequential list of character pairs, the second character in each pair being substituted for the first character. If the byte to be converted is not found as one of the first characters in a pair, it is passed through unchanged. Recommended when only a few characters need to be converted.

If the conversion method “INDEX” is specified, the numeric value of the byte to be converted is used as an index into the conversion table. The byte found as a result of this indexed lookup is substituted for the original byte. If the index value is greater than the length of the table, no conversion is performed. The first character in the string corresponds to the index value of 0. Recommended when a large number of characters need to be converted.

Related Statements

ENTER
IOBUFFER
OUTPUT

DTB\$

Syntax



`DTB$(integer)`

Example Statements

```

100 A$=DTB$(16+2*N)
200 PRINT DTB$(X1)
  
```

Parameters

integer — a numeric expression that evaluates to an integer –32 768 thru 32 767.

Action Taken

DTB\$ is a function that returns the base 2 representation of an integer argument. The result is a 16-character string and the argument is a numeric quantity.

Related Statements

- BTD
- DTH\$
- DTO\$
- HTD
- OTD

DTH\$

Syntax



DTH\$ (integer)

Example Statements

```
110 B$=DTH$ (32+2^N)  
210 PRINT DTH$ (X2)
```

Parameters

integer — a numeric expression that evaluates to an integer –32 768 thru 32 767.

Action Taken

DTH\$ is a function that returns the base 16 representation of an integer argument. The result is a 4-character string and the argument is a numeric quantity.

Related Statements

BTD
DTB\$
DTO\$
HTD
OTD

DTO\$

Syntax



DTO\$(integer)

Example Statements

```

120 C$=DTO$(64+2^N)
220 PRINT DTO$(X3)
  
```

Parameters

integer — a numeric expression that evaluates to an integer -32 768 thru 32 767.

Action taken

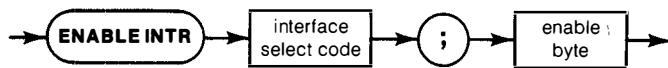
DTO\$ is a function that returns the base 8 representation of an integer argument. The result is a 6-character string and the argument is a numeric quantity.

Related Statements

- BTD
- HTD
- OTD
- DTB\$
- DTH\$

ENABLE INTR

Syntax



ENABLE INTR interface select code ; enable byte

Example Statements

```

10 ENABLE INTR 7 ; 8 ! SRQ interrupt, HP-IB
50 IF S>2 AND S<11 THEN ENABLE INTR S ; X
  
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

enable byte — a numeric expression that evaluates to an integer 0 thru 255. Binary value of byte is used to set and clear bits in the control register.

Action Taken

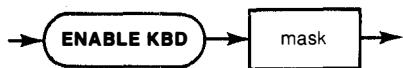
Enables the specified interface for interrupts according to the bits set in the enable byte. The enable byte is placed in control register CR1. The meaning of each bit in CR1 is interface dependent; refer to the appropriate interface programming section or the interface register summary in this appendix for details. This statement is identical to performing a CONTROL statement to Control Register 1.

Related Statements

- CONTROL
- ON INTR
- STATUS

ENABLE KBD

Syntax



ENABLE KBD mask

Example Statements

```

30 ENABLE KBD 33
180 IF X THEN ENABLE KBD K1
  
```

Parameters

mask — a numeric expression that evaluates to an integer 0 thru 255. Binary value of byte determines which keyboard modes are enabled and disabled.

Action Taken

Bits in the mask byte correspond to various keyboard areas and program modes as shown in the following table. If a bit is set in the mask, its feature is enabled. If a bit is clear, its feature is disabled.

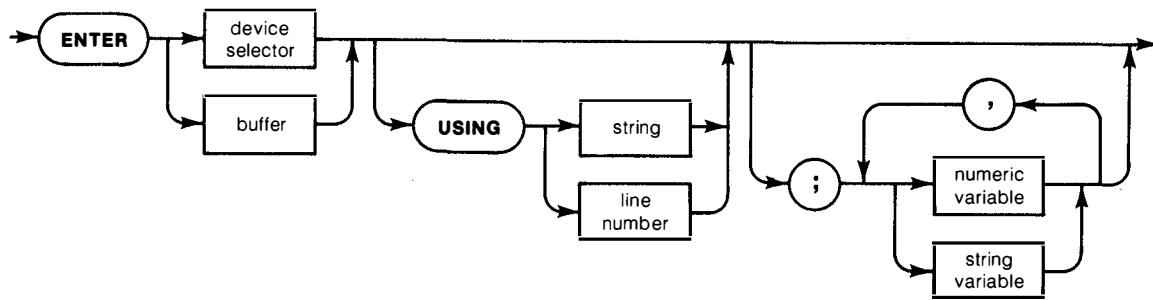
Bit	Mode	Keys Affected
7	RUN	RESET
6	RUN	PAUSE
5	RUN	SFKs and KEYLABEL
4	RUN	All other keys
3	INPUT	RESET
2	INPUT	PAUSE
1	INPUT	SFKs and KEYLABEL
0	INPUT	All other keys

Related Statements

INPUT
ON KEY#

ENTER

Syntax



`ENTER {device selector| buffer}[USING {string| line number}][: [variable] [, variable] ...]`

Example Statements

```

70 ENTER 701 USING A$ ; X,Y,Z
90 ENTER C$ ; N(I),Z$
120 ENTER 3 USING 30 ; A$
250 ENTER 100*S+A USING "#,B" ; N
  
```

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”).

buffer — the name of a string variable that has been declared as an IOBUFFER.

string — a string expression that contains a valid set of image specifiers.

line number — the line number of an IMAGE statement that contains a valid set of image specifiers.

variable (numeric or string) — the name of a variable intended as a destination of the ENTER operation.

Action Taken

Inputs bytes from the specified buffer or device; uses those bytes to build a number or string; places the result into a BASIC variable. If a CONVERT is in effect, the conversion occurs immediately after the character is taken from the interface or buffer.

When USING is not specified, free-field format is used. A free-field entry into a string places incoming bytes into the variable until either a line-feed is received, a carriage-return/line-feed sequence is received, or the string is full. Terminating sequences are not placed into the destination string. A free-field entry into a numeric variable ignores up to 256 leading non-numeric characters. Blanks are ignored during number building. Entry into a numeric variable is terminated by the first trailing character that is non-blank and non-numeric.

When USING is specified, input operations are formatted according to the image specifiers used. Image specifiers may be enclosed in quotes and placed in the ENTER statement, contained in a string variable named in the ENTER statement, or placed in an IMAGE statement referenced by the ENTER statement. For detailed information on image specifiers, refer to the IMAGE statement in this appendix or see “Formatted ENTER”.

ENTER requires a line-feed character to satisfy the statement after the variable list has been satisfied. This can be the same line-feed that satisfied the last variable in the list. If the source is a device selector and no line-feed is detected, the computer will be “hung” on the ENTER statement. If the source is a buffer and no line-feed is detected, a NO TERM error is generated. This requirement can be removed by using “#” as the first image specifier. For more detailed information on statement terminators, see “Formatted ENTER”. A “hung” condition can be trapped by use of the SET TIMEOUT and ON TIMEOUT statements.

Related Statements

CONVERT
IMAGE
IOBUFFER
ON TIMEOUT
SET TIMEOUT
TRANSFER

ERROM

Syntax



ERROM

Example Statements

```
30 X=ERROM  
70 IF ERROM=192 THEN GOTO 100
```

Parameters

None

Action Taken

ERROM is a function that returns the ID number of the option ROM associated with the last error generated by an option ROM. All option ROMs use error numbers greater than 100. The ID number of the I/O ROM is 192. Note that ERROM is modified only by the occurrence of another option ROM error.

Related Statements

ERRL
ERRN
ERRSC

ERRSC

Syntax



ERRSC

Example Statements

```
40 Y=ERRSC  
90 IF ERRSC=7 THEN GOSUB 200
```

Parameters

None

Action Taken

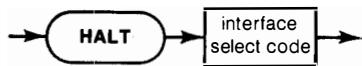
ERRSC is a function that returns the interface select code responsible for the most recent I/O error. Note that ERRSC is not cleared by a system Reset and is modified only by the occurrence of another interface-dependent I/O error.

Related Statements

ERRL
ERRN
ERROM

HALT

Syntax



HALT interface select code

Example Statements

```
100 HALT 7  
200 HALT S1
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

All interfaces: Stops current I/O operation. If an interface is HALTed with a TRANSFER active and an EOT branch enabled, the branch will be taken.

HP-IB: Leaves bus in present state.

Serial, BCD, GPIO: Does not affect external lines, so STATUS can be used to inspect line states. RESET or ABORTIO may be necessary after a halt to return handshake lines to the proper state for the next operation.

Related Statements

ABORTIO
ON EOT
RESET

HTD

Syntax



HTD (string)

Example Statements

```

20 Y=HTD(H$&L$)+A2
40 DISP HTD("F73A")
  
```

Parameters

string — a string expression that contains the base 16 representation of an integer. Limited to 4 significant characters that must be “0” thru “9” or “A” thru “F”.

Action Taken

HTD is a function that returns the value of a base 16 representation contained in the string argument. The argument is a character representation and the result is a numeric quantity.

Related Statements

- BTD
- DTB\$
- DTH\$
- DTO\$
- OTD

IMAGE

Syntax

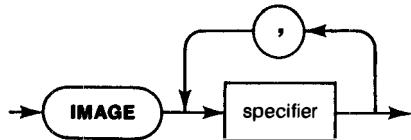


IMAGE specifier [, specifier]...

Example Statements

```
10 IMAGE "Total =",4D.DD
100 IMAGE #,%K,2X,#K
```

Summary of OUTPUT Image Specifiers

Image	Meaning
A	Output one string character
B	Output number as one 8-bit byte
C	Output a comma separator in a number
D	Output one digit character; blank for leading zero
E	Output exponent information; five characters
e	Output exponent information; four characters
K	Output a variable in free-field format
M	Output number's sign if negative, blank if positive
P	Output a period separator in a number
R	Output a European radix point (comma)
S	Output number's sign, plus or minus
W	Output number as two 8-bit bytes (16-bit word)
X	Output one blank
Z	Output one digit character, including leading zeros
"..."	Output a literal
#	Suppress end-of-line sequence at end of statement
*	Output one digit character; asterisk for leading zero
.	Output an American radix point (decimal point)
/	Output an end-of-line sequence

Summary of ENTER Image Specifiers

Image	Meaning
A	Demands one string character
B	Enter number as one 8-bit byte
C	Demand one character for a numeric field; allows commas to be skipped over
D	Demand one character for a numeric field
E	Demand five characters for a numeric field
e	Demand four characters for a numeric field
K	Enter a variable in free-field format
M	Demand one character for a numeric field
S	Demand one character for a numeric field
W	Enter number as two 8-bit bytes (16-bit word)
X	Skip one character
Z	Demand one character for a numeric field
#	Suppress requirement for a line-feed to terminate statement or field
%	Allow EOI to terminate statement or field
*	Demand one character for a numeric field
•	Demand one character for a numeric field
/	Demand a line-feed

Related Statements

CONVERT
 ENTER ... USING
 OUTPUT ... USING

IOBUFFER

Syntax



IOBUFFER string variable

Example Statements

```
10 DIM A$[88]
20 IOBUFFER A$
```

Parameters

string variable — the name of a string variable with a dimensioned length 8 characters longer than the desired size of the buffer.

Actions Taken

Eight characters of the string variable are reserved for control of buffer activity.

Buffer empty pointer:

- Initial value = 1. Accessed by Control/Status registers CR0, SR0. Characters are taken from the buffer (by ENTER or TRANSFER) using the following sequence:
 1. Read character
 2. Increment empty pointer

Buffer fill pointer:

- Initial value = 0. Accessed by Control/Status registers CR1, SR1. Characters are put into the buffer (by OUTPUT, TRANSFER, or string assignment) using the following sequence:
 1. Increment fill pointer
 2. Store character

Active-out select code:

- Initial value = 0. Accessed by Status register SR3. When active-out select code equals 0, there is no output TRANSFER operation active for this buffer. When an output TRANSFER is active for this buffer, the active-out select code is set equal to the interface select code that is the destination of the TRANSFER.

Active-in select code:

- Initial value = 0. Accessed by Status register SR2. When active-in select code equals 0, there is no input TRANSFER operation active for this buffer. When an input TRANSFER is active for this buffer, the active-in select code is set equal to the interface select code that is the source of the TRANSFER.

Conversion pointers:

- These pointers cannot be accessed from BASIC. When a CONVERT statement to the IOBUFFER is executed, pointers to the appropriate conversion tables are established. **These pointers are initialized by the IOBUFFER statement.** Therefore, execute CONVERT after executing IOBUFFER.

Full buffer:

- A buffer is full when the fill pointer equals the dimensioned length of the string minus eight. Attempting to store data into a full buffer generates a BUFFER error.

Empty buffer:

- A buffer is empty when the empty pointer equals the fill pointer plus one. When the buffer becomes empty, the fill pointer is reset to zero, and the empty pointer is reset to one. Active-out and active-in select codes are not affected by the buffer becoming empty; neither are the conversion pointers affected. Old data in the buffer is NOT lost, but the buffer fill pointer must be modified if you wish to re-access the data in the buffer (so the buffer will "look" full).

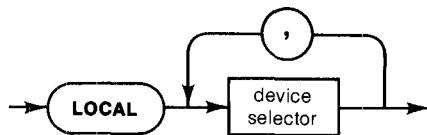
Buffer Status Registers		Buffer Control Registers	
Empty pointer Fill pointer Active in select code Active out select code	SR0 SR1 SR2 SR3	Empty pointer Fill pointer	CR0 CR1

Related Statements

CONTROL
 CONVERT
 ENTER
 OUTPUT
 STATUS
 TRANSFER

LOCAL

Syntax



LOCAL device selector [, device selector]...

Example Statements

220 LOCAL 7
330 LOCAL 100*S+D1 @ RESUME 7

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”). If multiple device selectors are specified, they must all be on the same interface select code.

Actions Taken

HP-IB:

- If device selector is only an interface select code, Remote Enable (REN) is set false. Must be System Controller.
- If device selector contains a primary address, interface addresses specified device(s) and sends Go To Local (GTL) message. Leaves ATN true; use RESUME if you wish to set ATN false. Must be Active Controller.
- If device is in REMOTE with LOCAL LOCKOUT set, the device must receive the GTL message to be returned to local (front panel) control.

Serial, BCD, GPIO: Error

Related Statements

LOCAL LOCKOUT
REMOTE
RESUME

LOCAL LOCKOUT

Syntax



LOCAL LOCKOUT interface select code

Example Statements

```
50 LOCAL LOCKOUT S0 @ RESUME S0  
100 REMOTE 706,712 @ LOCAL LOCKOUT 7
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Action Taken

HP-IB:

- Must be Active Controller. Sends Local Lockout (LLO) command. Leaves ATN true; use RESUME if you wish to set ATN false.

Serial, BCD, GPIO: Error

Related Statements

LOCAL
REMOTE

OFF EOT

Syntax



OFF EOT interface select code

Example Statements

```
40 OFF EOT 3  
120 IF S>128 THEN OFF EOT S1
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Action Taken

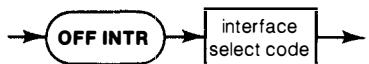
Disables end-of-line branching for termination of a TRANSFER on the specified interface. OFF EOT does NOT cancel a branch permanently. For example, if the TRANSFER has terminated and an ON EOT statement is re-executed, the branch will be taken at that time.

Related Statements

- OFF INTR
- OFF TIMEOUT
- ON EOT
- ON INTR
- ON TIMEOUT

OFF INTR

Syntax



OFF INTR interface select code

Example Statements

```
110 OFF INTR 7  
180 IF X THEN OFF INTR S2
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Action taken

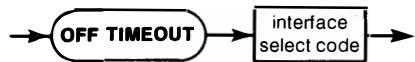
Disables end-of-line branching for interrupts from the specified interface. OFF INTR does **NOT** cancel a branch permanently. For example, if the interface has interrupted and an ON INTR statement is re-executed, the branch will be taken at that time.

Related Statements

CONTROL
ENABLE INTR
OFF EOT
OFF TIMEOUT
ON EOT
ON INTR
ON TIMEOUT

OFF TIMEOUT

Syntax



OFF TIMEOUT *interface select_code*

Example Statements

```
50 OFF TIMEOUT S DIV 100  
120 OFF TIMEOUT 7
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Action Taken

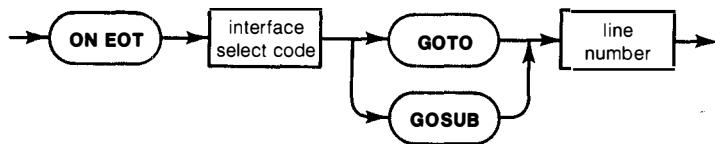
Disables end-of-line branching for occurrence of a timeout on the specified interface. OFF TIMEOUT does NOT cancel a branch permanently. For example, if the interface has timed out and an ON TIMEOUT statement is re-executed, the branch will be taken at that time.

Related Statements

- OFF EOT
- OFF INTR
- ON EOT
- ON INTR
- ON TIMEOUT
- SET TIMEOUT

ON EOT

Syntax



`ON EOT interface select code {GOTO|GOSUB} line number`

Example Statements

```
20 ON EOT 7 GOTO 150
120 ON EOT S4 GOSUB 1000
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

line number — an integer constant from 1 thru 9999 that specifies a valid line number within the program.

Actions Taken

Enables end-of-line branches to the specified line number when a TRANSFER to or from the specified interface is terminated. A pending end-of-line branch from a previous, unserviced TRANSFER termination (for the specified interface select code) is taken immediately. Only one TRANSFER termination per interface select code is retained by the system.

Each interface may have alternate causes for TRANSFER terminations that are user-programmable. Refer to the appropriate Interface Programming section for details about this capability.

Overrides any previous ON EOT statement for the same interface select code.

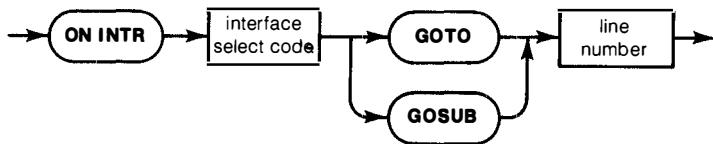
Related Statements

- CONTROL
- OFF EOT
- OFF INTR
- OFF TIMEOUT
- ON INTR
- ON TIMEOUT
- STATUS
- TRANSFER

Also see Branch Precedence Table

ON INTR

Syntax



`ON INTR interface select code {GOTO|GOSUB} line number`

Example Statements

```
30 ON INTR S1 GOSUB 2000
60 ON INTR 3 GOTO 185
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

line number — an integer constant from 1 thru 9999 that specifies a valid line number within the program.

Actions Taken

Enables end-of-line branches to the specified line number when an interface interrupt occurs (see ENABLE INTR). A pending end-of-line branch from a previous, unserviced interface interrupt (for the specified interface select code) is taken immediately. Only one interrupt per interface select code is retained by the system.

Interrupt causes are specified by either ENABLE INTR or CONTROL statements. Interrupt causes are interface-dependent; refer to the appropriate Interface Programming Section for details.

Overrides any previous ON INTR statement for the same interface select code.

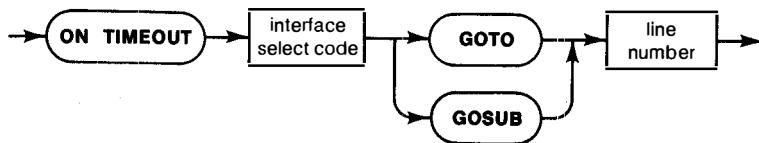
Related Statements

- CONTROL
- ENABLE INTR
- OFF EOT
- OFF INTR
- OFF TIMEOUT
- ON EOT
- ON TIMEOUT

Also see Branch Precedence Table

ON TIMEOUT

Syntax



ON TIMEOUT interface select code {GOTO|GOSUB} line number

Example Statements

```
20 ON TIMEOUT S3 GOSUB 2500
50 ON TIMEOUT 7 GOTO 320
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

line number — an integer constant from 1 thru 9999 that specifies a valid line number within the program.

Actions Taken

Enables end-of-line branches to the specified line number when an interface timeout occurs (see SET TIMEOUT). A pending end-of-line branch from a previous, unserviced interface timeout (for the specified interface select code) is taken immediately. Only one timeout per interface select code is retained by the system.

End-of-line branching for TIMEOUT is **not** applicable to the actual data movement portion of a TRANSFER (INTR or FHS) operation. A TRANSFER can timeout if the interface or device cannot be addressed to start the TRANSFER, but there will be no ON TIMEOUT branch if the peripheral device stops handshaking in the middle of the TRANSFER.

Overrides any previous ON TIMEOUT statement for the same interface select code.

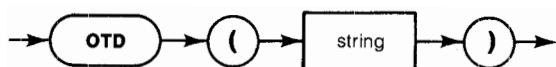
Related Statements

- OFF EOT
- OFF INTR
- OFF TIMEOUT
- ON EOT
- ON INTR
- SET TIMEOUT

Also see Branch Precedence Table

OTD

Syntax



OTD (string)

Example Statements

```
80 X=OTD(H$&L$)+A3  
110 DISP OTD("177345")
```

Parameters

string — a string expression that contains the base 8 representation of an integer. Limited to 6 significant characters that must be “0” thru “7” (except most significant character must be “0” or “1”).

Action Taken

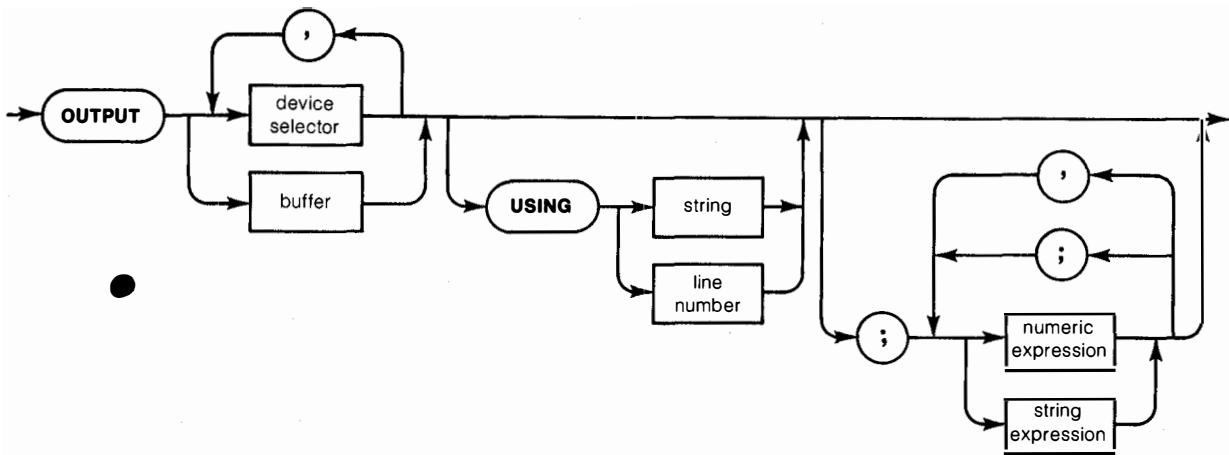
OTD is a function that returns the value of a base 8 representation contained in the string argument. The argument is a character representation and the result is a numeric quantity.

Related Statements

DTB\$
DTH\$
DTO\$
BTD
HTD

OUTPUT

Syntax



**OUTPUT {device selector [, device selector ...] | buffer} [USING {string| line number}]
[#: expression [, expression] [#: expression] ...]**

Example Statements

```

70 OUTPUT 701 USING A$ ; X,Y,Z
90 OUTPUT C$ ; N(I);Z$
120 OUTPUT 3 USING 30 ; A$
250 OUTPUT 100*S+A USING "#,B" ; N
  
```

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”). If multiple device selectors are specified, they must all be on the same interface select code. OUTPUT allows device selectors 1 and 2 for addressing the internal CRT and printer.

buffer — the name of a string variable that has been declared as an IOBUFFER.

string — a string expression that contains a valid set of image specifiers.

line number — the line number of an IMAGE statement that contains a valid set of image specifiers.

expression (string or numeric) — any string expression or numeric expression intended to be OUTPUT. Expressions may be constants or variables and may be separated by commas or semicolons.

Actions Taken

Outputs bytes to the specified buffer or device(s); bytes may be string or numeric. If a CONVERT operation is specified, the conversion is performed immediately before the byte is sent to the interface or buffer.

When USING is not specified, and output items are separated by commas, free-field format is used. A free-field output of a string item causes it to be left-justified in a field with no more than 20 trailing blanks. A free-field output of a numeric item causes it to be left-justified in a field of 11, 21, or 32 characters.

When USING is not specified, and output items are separated by semicolons, compact format is used. A compact output of a string variable causes it to be sent with no leading or trailing blanks. A compact output of a numeric variable causes it to be sent with one trailing blank and one leading sign character (blank if positive, minus sign if negative).

When USING is specified, output operations are formatted according to the image specifiers used. Image specifiers may be enclosed in quotes and placed in the OUTPUT statement, contained in a string variable named in the OUTPUT statement, or placed in an IMAGE statement referenced by the OUTPUT statement. For detailed information on image specifiers, refer to the IMAGE statement in this appendix or see “Formatted OUTPUT”.

OUTPUT sends an end-of-line sequence after the last item in the OUTPUT list. This sequence is interface-dependent; can be changed by the CONTROL statement; and defaults to carriage-return/line-feed. This sequence can be suppressed by using “#” as the first image specifier. For more detailed information on statement terminators, see “Formatted OUTPUT”. If the OUTPUT is to a buffer, a carriage-return/line-feed is placed in the buffer after the last data byte unless the “#” image is used.

Related Statements

CONTROL
CONVERT
IMAGE
IOBUFFER
TRANSFER

PASS CONTROL

The PASS CONTROL statement passes Active Controller responsibility to the specified device. This clears the CA bit (bit 5 of SR5) when control is passed out of the interface.

Syntax



PASS CONTROL device selector

Example Statements

```
100 PASS CONTROL 100*S+D
250 PASS CONTROL 721 @ ENABLE INTR 7;32
```

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see "Choosing the Source or Destination").

Actions Taken

HP-IB:

- Must be Active Controller. Passes Active Controller responsibility to the specified device. This clears the CA bit (bit 5 of SR5) when control is passed out of the interface.
- If device selector is only an interface select code, interface sends the Take Control (TCT) message, then sets ATN false. Be sure that the device receiving control has been addressed to talk before using this form of PASS CONTROL.
- If device selector contains a primary address, interface sends the specified device's talk address, sends the TCT message, then sets ATN false.

Serial, BCD, GPIO: Error

Related Statements

ABORTIO
ENABLE INTR
ON INTR
REQUEST
RESET

PPOLL

Syntax



PPOLL (interface select code)

Example Statements

```
310 X=PPOLL(7)
620 P9=PPOLL(S0)
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

HP-IB:

- Must be Active Controller. PPOLL is a function that returns the results of a Parallel Poll operation. Sends Identify (IDY) message. Devices capable of responding each assert one bit of the parallel poll response byte.

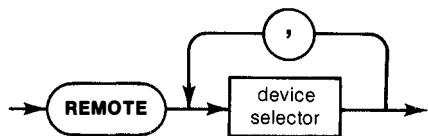
Serial, BCD, GPIO: Error

Related Statements

SPOLL

REMOTE

Syntax



REMOTE device selector [, device selector]...

Example Statements

```
50 REMOTE 720
130 REMOTE 100*S+D @ RESUME S
```

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”). If multiple device selectors are specified, they must all be on the same interface select code.

Actions Taken

HP-IB: Must be System Controller. Puts the bus into remote operation.

- If device selector is only an interface select code, interface sets Remote Enable (REN) true. Devices do not go into remote state until they are addressed to listen.
- If device selector contains a primary address, interface sets REN true, sends Unlisten (UNL) message, then sends the listen address of the specified device(s). REMOTE leaves ATN true; use RESUME if you wish to set ATN false.

Serial: Error

BCD: Sets partial field separator. Refer to “Using the BCD Interface” for details.

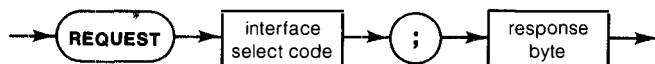
GPIO: Error

Related Statements

LOCAL
LOCAL LOCKOUT
RESUME

REQUEST

Syntax



REQUEST interface select code ; response byte

Example Statements

```

50 REQUEST 7 ; 64+4
260 IF T>40 THEN REQUEST S ; 64+X

```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

response byte — a numeric expression that evaluates to an integer 0 thru 255.

Actions Taken

HP-IB: Must be non-controller. Sets up a Serial Poll response byte. Sets Service Request (SRQ) true if bit 6 (decimal value 64) of the response byte is set. The response byte is sent to the Active Controller in response to an incoming Serial Poll operation. The Active Controller's Serial Poll operation clears SRQ, which can also be cleared by executing REQUEST with bit 6 of the response byte equal to zero.

Serial: Sends a BREAK. The BREAK is defined by the response byte. A space (0-state) condition is held for the number of character times specified in the response byte. It is then followed by a mark (1-state) condition for five character times.

BCD, GPIO: Error

Related Statements

PASS CONTROL

SPOLL

RESET

Syntax



RESET interface select code

Example Statements

```

30 RESET 7
300 IF S>128 THEN RESET S3
  
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

All interfaces: Performs a hardware reset of the interface, returning it unconditionally to its power-on state. The interface performs a self test (failure causes ERROR 110), and the control registers are set according to the configuration switches on the interface circuit assembly. RESETing an interface with a TRANSFER active and EOT branching enabled causes the branch to be taken.

HP-IB: If System Controller, sends Interface Clear (IFC), then Remote Enable (REN).

Serial: Modem control lines are turned off.

BCD: Data lines are set to high-impedance state, handshake lines are set false, and I/O lines are set to input state.

GPIO: Ports A and B are set to high-impedance state, Ports C and D are set to off state, CTL lines are set false, and OUTA and OUTB are set to indicate output.

Related Statements

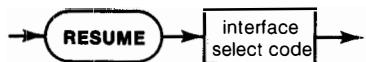
ABORTIO

HALT

ON EOT

RESUME

Syntax



RESUME interface select code

Example Statements

```
110 RESUME 7  
190 RESUME S DIV 100
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

Actions Taken

HP-IB: Must be Active Controller (CA=1). Sets the Attention (ATN) line false. Statements that can leave the ATN line true are: CLEAR, LOCAL, LOCAL LOCKOUT, REMOTE, SEND, TRIGGER.

Serial: The transmitter is enabled. Refer to "Using the Serial Interface" for details.

BCD, GPIO: Error

Related Statements

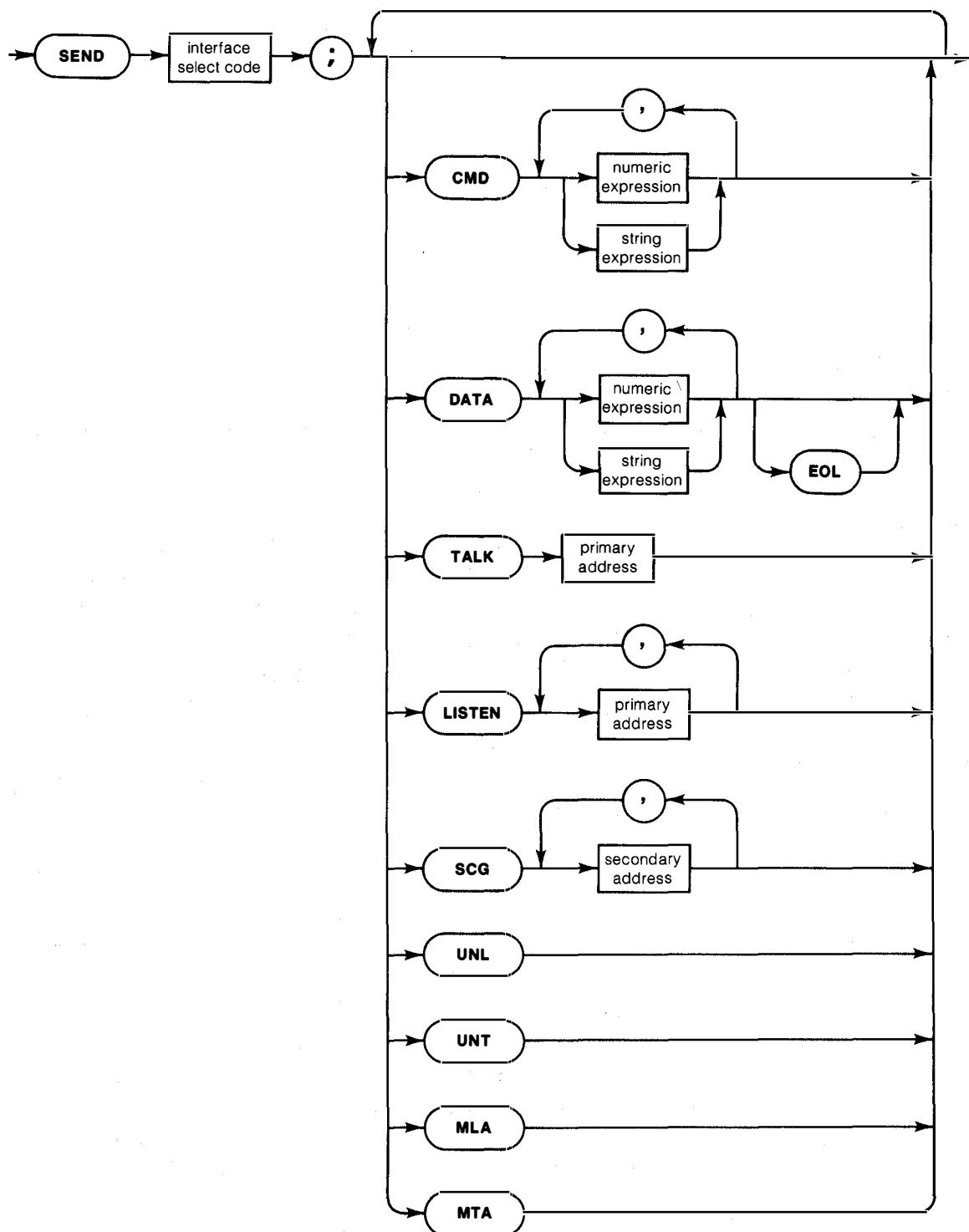
CONTROL

HALT

SEND

SEND

Syntax



```
SEND interface select code ; [[CMD list][DATA list[EOL]][TALK primary address]
[LISTEN primary address[, primary address] ... ][SCG secondary address[, secondary address] ... ]
[UNL][UNT][MLA][MTA] ... ]
```

Example Statements

```
100 SEND 7 ; CMD "U?%" DATA "Hello"
200 SEND 7 ; CMD A$ SCG 14,18 DATA X$
300 SEND S ; MTA UNL LISTEN 6,14 CMD P,R SCG 6
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

list — a list of numeric or string expressions, separated by commas.

primary address — a numeric expression that evaluates to an integer 0 thru 31.

secondary address — a numeric expression that evaluates to an integer 0 thru 31.

Actions Taken

HP-IB: If sending any commands (CMD, TALK, LISTEN, SCG, UNL, UNT, MLA, MTA) must be Active Controller. The ATN line is set true while sending commands. The ATN line is set false while sending DATA, even if no actual data is sent (i.e. DATA "").

- **CMD:** Commands: send list of 8-bit expressions with ATN true. Primary commands have a bit pattern = X00CCCC, where X=don't care, C=bits of command (decimal 0 thru 31). SEND CMD can be used to create odd parity on commands, if necessary.
- **DATA:** Send list of numeric or string expressions with ATN false. Any 8-bit pattern may be sent. If EOL is specified, the interface's end-of-line character sequence is sent following data (Control registers 17 thru 23).
- **TALK:** Send device's Talk Address (TAD). Bit pattern = X10TTTT, where X=don't care, T=bits of talk address (decimal 0 thru 31).
- **LISTEN:** Send device's Listen Address (LAD). Bit pattern = X01LLLL, where X=don't care, L=bits of listen address (decimal 0 thru 31).
- **SCG:** Secondary Command Group: Send secondary address to device. Bit pattern = X11SSSS, where X=don't care, S=bits of secondary address (decimal 0 thru 31).
- **UNL:** Send Unlisten command (UNL). Numeric value sent is 63; ATN is true.
- **UNT:** Send Untalk command (UNT). Numeric value sent is 95; ATN is true.
- **MLA:** Send My Listen Address (MLA). This is the listen address of the interface. Factory setting = 53.
- **MTA:** Send My Talk Address (MTA). This is the talk address of the interface. Factory setting = 85.

Serial: Only form that can be sent is DATA.

- DATA: Sends list of numeric or string expressions. If EOL is specified, the interface's end-of-line character sequence is sent (control registers 17 thru 23).

BCD: See "Using the BCD Interface" details.

- CMD: Primary addresses 0 thru 6 set partial field specifier.
- DATA: Lower 4 bits of data bytes are sent; control characters, spaces, and commas are ignored. If EOL is specified, data format checking is enabled.
- LISTEN, TALK: Primary addresses 0 thru 6 set partial field specifier.
- SCG: Error
- UNL, UNT, MLA, MTA: Ignored

GPIO: See "Using the GPIO Interface" details.

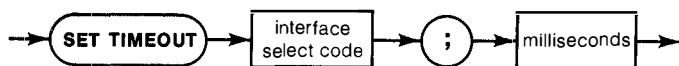
- CMD: Primary addresses 0 thru 15 select port configuration. Device Clear command pulses RESA and RESB. Selected Device Clear pulses RESA or RESB according to the most recent primary address.
- DATA: Send list of numeric or string expressions. Data is sent as 8-bit bytes. If EOL is specified, the interface's end-of-line character sequence is sent (control registers 17 thru 23).
- LISTEN, TALK: Primary addresses 0 thru 15 select port configuration.
- SCG: Error
- UNL, UNT, MLA, MTA: Ignored

Related Statements

OUTPUT

SET TIMEOUT

Syntax



SET TIMEOUT interface select code ; milliseconds

Example Statements

```

100 SET TIMEOUT S0 ; X*1000
280 ON TIMEOUT 7 GOTO 550 @ SET TIMEOUT 7 ; 4500
  
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

milliseconds — a numeric expression that evaluates to an integer 0 thru 32 767.

Action Taken

Establishes an approximate time limit (in milliseconds) that the interface will wait to complete a handshake with its peripheral device. If the specified time limit is exceeded and ON TIMEOUT end-of-line branching is enabled, the branch is taken. If no ON TIMEOUT is currently in effect, there is no indication that a timeout has occurred until an ON TIMEOUT is subsequently executed.

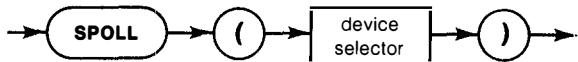
Related Statements

OFF TIMEOUT

ON TIMEOUT

SPOLL

Syntax



SPOLL (device selector)

Example Statements

```

50 P=SPOLL(S4)
250 IF SPOLL(701)>63 THEN GOTO 750

```

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”).

Actions Taken

HP-IB:

- Conducts a Serial Poll of a device on the bus and returns the device’s status byte. If bit 6 of the status byte is set (decimal value 64), it indicates that the device is requesting service (asserting SRQ).
- If device selector is only an interface select code, interface sends Serial Poll Enable (SPE), sets ATN false, receives the status byte, sends Serial Poll Disable (SPD), then sends Untalk (UNT).
- If device selector contains a primary address, interface sends Unlisten (UNL), My Listen Address (MLA), devices Talk Address (TAD), Serial Poll Enable (SPE), then sets ATN false. It receives the status byte, sends Serial Poll Disable (SPD), then sends Untalk (UNT).

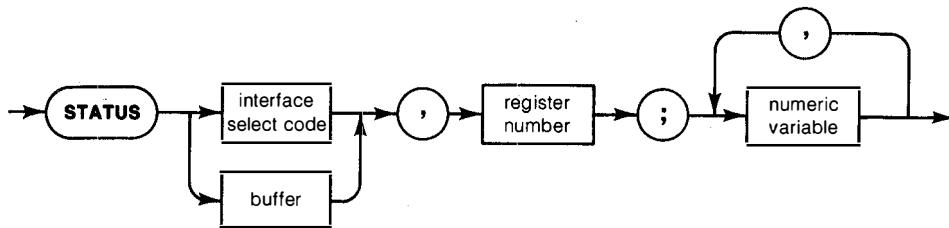
Serial, BCD, GPIO: Error

Related Statements

PPOLL

STATUS

Syntax



STATUS {interface select code| buffer} , register number ; numeric variable [, numeric variable] ...

Example Statements

```
20 STATUS 7,0 ; C0,C1,C2,C3,C4
70 STATUS S1,5 ; X
```

Parameters

interface select code — a numeric expression that evaluates to an integer 3 thru 10.

buffer — the name of a string variable that has been declared as an IOBUFFER.

register number — a numeric expression that evaluates to an integer 0 thru 15. Must specify a valid status register for the selected interface.

numeric variable — any numeric variable intended as a destination for the status information.

Actions Taken

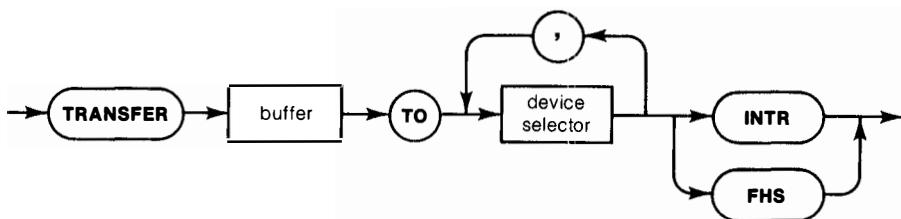
Reads one or more status register(s) and assigns the value(s) to the specified variable(s). When more than one variable is specified, consecutive status registers are read, starting at the specified register number. Status values returned are integers 0 thru 255.

Related Statements

- ASSERT
- CONTROL
- ENABLE INTR
- IOBUFFER

TRANSFER (out)

Syntax



TRANSFER buffer TO device selector [, device selector [, device selector ...]]{INTR|FHS}

Example Statements

```

150 TRANSFER B$ TO 721 INTR
280 OUTPUT B$ USING "#,K" ; D$ @ TRANSFER B$ TO 9 FHS
400 ON EOT S1 GOSUB 660 @ TRANSFER X$ TO S1 INTR
  
```

Parameters

buffer — the name of a string variable that has been declared as an IOBUFFER.

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”). If multiple device selectors are specified, they must all be on the same interface select code.

Actions Taken

Takes data bytes from the specified buffer and sends them to the specified device(s). Data is taken from the buffer according to the buffer empty pointer. If the device selector contains a primary address, addressing is performed prior to sending the first byte. The interface’s programmable end-of-line sequence is sent after the last byte from the buffer has been sent. Note that the buffer may contain an additional carriage-return/line-feed placed there by an OUTPUT statement. The TRANSFER terminates when the buffer is empty. If ON EOT branching is enabled, the branch is taken when the TRANSFER terminates.

If INTR (Interrupt) is specified, the interface is automatically enabled to interrupt the computer each time it is ready for a new character. The TRANSFER continues to completion even though program execution may have stopped. A WARNING 101 is issued if the program stops with a TRANSFER still active. Be certain that the TRANSFER has terminated (use STATUS, RESET, HALT, or ABORTIO) before attempting to modify the program. This TRANSFER type (under ideal conditions) is capable of a maximum data transfer rate of about 400 bytes per second.

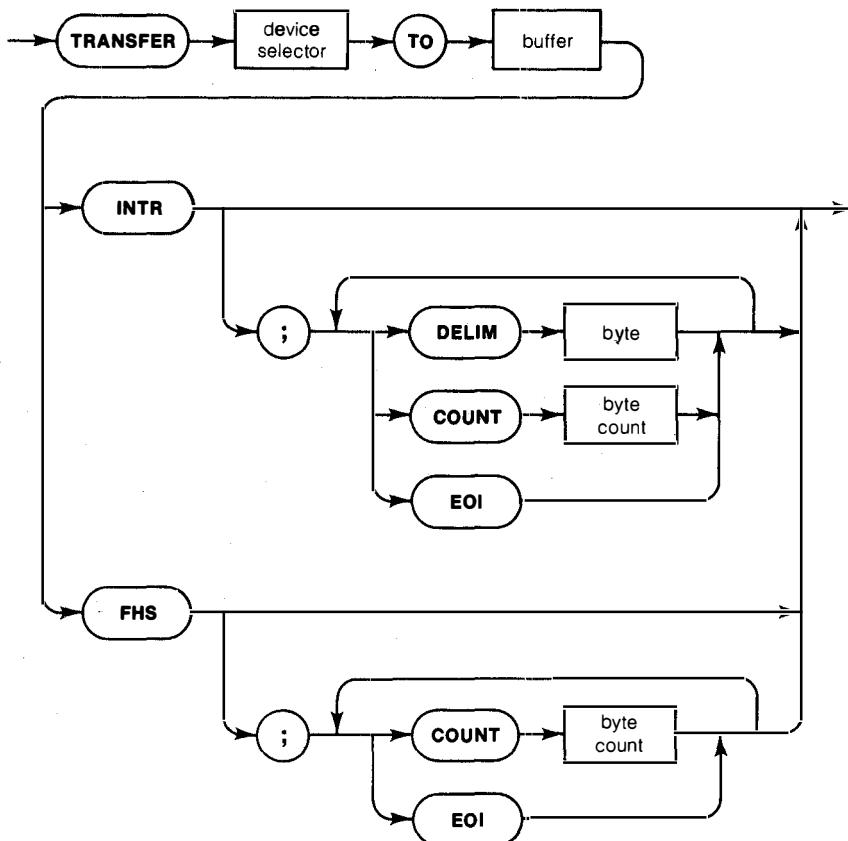
If FHS (Fast Handshake) is specified, the interface and computer are dedicated to the TRANSFER until it is complete. No interrupts or keypresses (not even the RESET key) are detected until the TRANSFER terminates. If the computer "locks up" on a FHS TRANSFER, only a power-on or special interface-specific termination (i.e. Interface Clear on HP-IB) can return the computer to its "normal" state. This TRANSFER type (under ideal conditions) is capable of data transfer rates in excess of 20 000 bytes per second.

Related Statements

ABORTIO
CONTROL
HALT
IOBUFFER
ON EOT
OUTPUT
RESET
STATUS

TRANSFER (in)

Syntax



TRANSFER device selector TO buffer [INTR [; [COUNT byte count][DELIM byte][EOI]]]

or

TRANSFER device selector TO buffer [FHS [; [COUNT byte count][EOI]]]

Example Statements

```

100 TRANSFER 706 TO B$ INTR
200 TRANSFER 100*S+D TO B$ INTR ; COUNT 80 DELIM 10 EOI
300 TRANSFER 3 TO A$ FHS ; COUNT 16
  
```

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”).

buffer — the name of a string variable that has been declared as an IOBUFFER.

byte count — a numeric expression that evaluates to an integer 0 thru 32 767. Specifies maximum number of bytes to be input.

byte — a numeric expression that evaluates to an integer 0 thru 255. Specifies the ASCII value of a character which can terminate the transfer.

Actions Taken

Takes data bytes from the specified device and places them into the specified buffer. Characters are placed into the buffer according to the buffer fill pointer. The TRANSFER terminates when the buffer is full or when the first one of the specified terminating conditions is met. The interface may also have a programmable terminating condition; refer to the appropriate Interface Programming section for details. Specifying COUNT sets a maximum limit on the number of characters to be TRANSFERed. DELIM specifies the numeric value of a character that can terminate the transfer. Specifying EOI allows the TRANSFER to terminate when an interface-dependent “END” signal is detected (such as the EOI line on HP-IB). The terminating condition for buffer full is always in effect. If an ON EOT branch is enabled, the branch is taken when the TRANSFER terminates.

If INTR (Interrupt) is specified, the interface is automatically enabled to interrupt the computer each time it is ready with a new character. The TRANSFER continues to completion even though program execution may have stopped. A WARNING 101 is issued if the program stops with a TRANSFER still active. Be certain that the TRANSFER has terminated (use RESET, HALT, or ABORTIO) before attempting to modify the program. This TRANSFER type (under ideal conditions) is capable of a maximum data transfer rate of about 400 bytes per second.

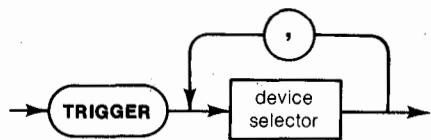
If FHS (Fast Handshake) is specified, the interface and computer are dedicated to the TRANSFER until it is complete. No interrupts or keypresses (not even the RESET key) are detected until the TRANSFER terminates. If the computer “locks up” on a FHS TRANSFER, only a power-on or special interface-specific termination (i.e. Interface Clear on HP-IB) can return the computer to its “normal” state. This TRANSFER type (under ideal conditions) is capable of data transfer rates in excess of 20 000 bytes per second.

Related Statements

- ABORTIO
- CONTROL
- ENTER
- HALT
- IOBUFFER
- ON EOT
- RESET
- STATUS

TRIGGER

Syntax



TRIGGER device selector [, device selector]...

Example Statements

```
70 TRIGGER 706,715 @ RESUME 7
190 TRIGGER S1
```

Parameters

device selector — a valid interface select code or a valid combination of interface select code and primary address (see “Choosing the Source or Destination”). If multiple devices are selected, they must all be on the same interface select code.

Actions Taken

HP-IB:

- Must be Active Controller. Sends the Group Execute Trigger command (GET).
- If device selector is only an interface select code, interface sends the GET command and leaves ATN true; use RESUME if you wish to set ATN false. Those devices already addressed to listen receive the GET command.
- If device selector contains a primary address, interface sends Unlisten (UNL), then the Listen Address (LAD) of the specified device(s). Sends the GET command and leaves ATN true; use RESUME if you wish to set ATN false.

Serial, BCD, GPIO: Error

Related Statements

RESUME

SEND

Interface Register Maps

HP-IB Interface Registers

Status Register 0 - Interface ID

Status Register 0 - Interface ID							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 0 - Parity Control

Control Register 0 - Parity Control							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Enable ODD Parity	Enable EVEN Parity	Enable ONE Parity	Enable ZERO Parity
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 1 - Interrupt Cause/Enable

Status/Control Register 1 - Interrupt Cause/Enable							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IFC Interrupt	LA Interrupt	CA Interrupt	TA Interrupt	SRQ Interrupt	DCL or SDC Interrupt	GET Interrupt	SCG Interrupt
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 2 - Control Lines

Status/Control Register 2 - Control Lines							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	REN	SRQ	ATN	EOI	DAV	NDAC	NRFD
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 3 - Data Lines

Status/Control Register 3 - Data Lines							
Most Significant Bit				Least Significant Bit			
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

Status Register 4 - Primary Address and System Controller

Status Register 4 - Primary Address and System Controller								
Most Significant Bit				Least Significant Bit				
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
Not Used		System Controller	Primary Address (HP-IB Address)					
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1	

Status Register 5 - Interface State

Status Register 5 - Interface State							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Listener Active	Controller Active	Talker Active	Serial Poll Enable	Parity Error	Remote Enable	Local Lockout
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 6 - Secondary Command

Status Register 6 - Secondary Command								
Most Significant Bit				Least Significant Bit				
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
Not Used			Secondary Address (Secondary Command)					
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1	

Control Register 16 - EOL Control

Control Register 16 - EOL Control							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable EOI	Not Used				Number of Characters in EOL Sequence		
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 17 thru Control Register 23 - EOL Characters**Serial Interface Registers****Status Register 0 - Interface ID**

Status Register 0 - Interface ID							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 1 - Interrupt Cause/Enable

Status/Control Register 1 - Interrupt Cause/Enable							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Break Received	Framing Error	Parity Error	Received Data Available	DCD (Opt. 001) RTS (Standard)	Auto-disconnect	DSR (Opt. 001) DRS (Standard)	CTS (Opt. 001) DTR Standard
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 2 - Modem Control

Status/Control Register 2 - Modem Control							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					DRS (Opt. 001)	RTS (Opt. 001)	DTR (Opt. 001)
					DSR (Standard)	DCD (Standard)	CTS (Standard)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 3 - Modem Signals and Cable Type

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				DCD (Opt. 001)	Cable Type	DSR (Opt. 001)	CTS (Opt. 001)
				RTS (Standard)	0 = Opt. 001 1 = Std.	DRS (Standard)	DTR (Standard)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 3 - Standard Baud Rates

Value	Rate Specified	Value	Rate Specified
0	50	8	1200
1	75	9	1800
2	110	10	2000
3	134.5	11	2400
4	150	12	2600
5	200	13	4800
6	300	14	7200
7	600	15	9600

Status/Control Register 4 - Character Format

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Set Break	Force Parity	Odd/Even Parity	Enable Parity	Stop Bits	Character Length	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bits/Character	Parity Specifier				
	None	Odd	Even	1	0
5	0	8	24	40	56
6	1	9	25	41	57
7	2	10	26	42	58
8	3	11	27	43	59

The table above assumes one stop bit. For two stop bits, add 4 to the value shown.

Status/Control Register 5 - Modem Features

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Enable Receive Handshake	Enable Transmit Handshake	Auto-discon. if not DCD (001) RTS (Std.)	Not Used	Auto-discon. if not DSR (001) DRS (Std.)	Auto-discon. if not CTS (001) DTR (Std.)	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 6 - Least Significant Byte of Baud Rate Divisor**Status/Control Register 7 - Most Significant Byte of Baud Rate Divisor****Status/Control Register 8 - Parity and Framing Error Replacement Character****Status/Control Register 9 - Transmitter and Receiver Control**

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Transmitter	Strip Received Rubouts	Strip Received Nulls	Change Character if Error	Set Bit 7 of Character if Error	Reset Receive Queue	Auto-echo Enable	Enable Receiver
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 10 - Transmitter and Receiver Status

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used	Not Used	Transmit Register Empty	Break Received	Framing Error	Parity Error	Not Used	Received Data Available
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 11 - I/O Termination Cause

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
End of Output Data List	End of Input Data List	Transfer Count Expired	CR15 Character Received	CR14 Character Received	CR13 Character Received	CR12 Character Received	DELIM Character Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 11 - Input Termination Control

Control Register 11 - Input Termination Control							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable Transmit Flag XON	Disable Transmit Flag XOFF	Not Used	Terminate if CR15	Terminate if CR14	Terminate if CR13	Terminate if CR12	Not Used
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 12 - Input Termination Character**Control Register 13 - Input Termination Character****Control Register 14 - Input Termination and XOFF Character****Control Register 15 - Input Termination and XON Character****Control Register 16 - EOL Control**

Control Register 16 - EOL Control							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Auto RTS Enable	EOL Transmit Disable	Number of characters in EOL sequence					
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 17 thru Control Register 23 - EOL Characters**BCD Interface Registers****Status Register 0 - Interface ID**

Status Register 0 - Interface ID							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 1 - Interrupt Cause/Enable

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Function B (Most Significant Digit)				Function A (Most Significant Digit)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 2 - Line Status

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I/OA 0 = Input 1 = Output	I/OB 0 = Input 1 = Output	CTLA 0 = Ready 1 = Busy	CTLB 0 = Ready 1 = Busy	FLGA 0 = Ready 1 = Busy	FLGB 0 = Ready 1 = Busy	0	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 - Line Assertion and Port 10

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I/OA	I/OB	CTLA	CTLB	Port 10 Output (when available)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 3 - Mantissa Digits

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of Digits Assigned for Channel B Mantissa (0 – 11)				Number of Digits Assigned for Channel A Mantissa (0 – 11)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 4 - Exponent Digits

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of Digits Assigned for Channel B Exponent (0 – 3)				Number of Digits Assigned for Channel A Exponent (0 – 3)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 5 - Function Digits

Status/Control Register 5 - Function Digits							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of Digits Assigned for Channel B Function (0–11)				Number of Digits Assigned for Channel A Function (0–11)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 6 - Decimal Point Placement

Status/Control Register 6 - Decimal Point Placement							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of Mantissa Digits Assigned to the Right of the Decimal Point. (Channel B)				Number of Mantissa Digits Assigned to the Right of the Decimal Point (Channel A)			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 7 - Handshake Normalization

Status/Control Register 7 - Handshake Normalization							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Invert I/OA	Invert I/OB	Invert CTLA	Invert CTLB	Invert FLGA	Invert FLGB	Channel A Handshake 0 = Trailing 1 = Leading	Channel B Handshake 0 = Trailing 1 = Leading
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 8 - Data Normalization

Status/Control Register 8 - Data Normalization							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Invert Channel B Data				Invert Channel A Data			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 9 - Function Normalization

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Invert Channel B Function Bits				Invert Channel A Function Bits			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 10 - Sign Bit and Port 10 Normalization

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Invert Exponent B Sign Bit	Invert Mantissa B Sign Bit	Invert Exponent A Sign Bit	Invert Mantissa A Sign Bit	Invert Port 10 Data			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

GPIO Interface Registers**Status Register 0 - Interface ID**

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	1	0	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 0 - Parity Control

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Enable ODD Parity	Enable EVEN Parity	Enable ONE Parity	Enable ZERO Parity
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 1 - Interrupt Cause/Enable

Status/Control Register 1 - Interrupt Cause/Enable							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ST1 Interrupt	ST0 Interrupt	FLGB Interrupt	FLGA Interrupt	Not Used		Parity Error Interrupt	Not Used
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 2 - Line Status

Status Register 2 - Line Status							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ST1	ST0	FLGB	FLGA	CTL1	CTLB	CTL0	CTLA
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 - Line Assertion

Control Register 2 - Line Assertion							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESB	RESA	Not Used		CTL1	CTLB	CTL0	CTLA
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 3 - Handshake Line Normalization

Status/Control Register 3 - Handshake Line Normalization							
Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Invert ST1	Invert ST0	Invert FLGB	Invert FLGA	Invert CTL1	Invert CTLB	Invert CTL0	Invert CTLA
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 4 - Data Normalization and Handshake Control

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Handshake Method		0 = Ready to Busy 1 = Busy to Ready	Not Used	Invert Port D Data	Invert Port C Data	Invert Port B Data	Invert Port A Data
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Decimal Value of Bit 5 thru Bit 7	Handshake Method
0	Output: Full Handshake
64	Output: Partial Handshake
128	Output: Strobe Handshake
0	Input: Full Handshake; READY to BUSY
32	Input: Full Handshake; BUSY to READY
64	Input: Partial Handshake; READY to BUSY
96	Input: Partial Handshake; BUSY to READY
128	Input: Strobe Handshake; READY to BUSY
160	Input: Strobe Handshake; BUSY to READY
192	Input or Output: No Handshake

Status/Control Register 5 - Primary Address and Trigger Action

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger If Data < R7	Trigger If Data = R7	Trigger If Data > R7	If Trigger, Pulse CTL	Primary Address			
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 6 - CTL Delay and Strobe Pulse Width

Most Significant Bit				Least Significant Bit			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Increment 0 = 10µs 1 = 1ms	Delay: Number of Increments						
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 7 - Trigger Character**Status/Control Register 8 - Output Enable for Port A and Port B**

Status/Control Register 8 - Output Enable for Port A and Port B							
Most Significant Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Least Significant Bit
	Not Used						Enable Port B Outputs
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status/Control Register 9 - Output Inhibit Function

Status/Control Register 9 - Output Inhibit Function							
Most Significant Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Least Significant Bit
	Not Used						Enable Output Inhibit
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 16 - EOL Control

Control Register 16 - EOL Control							
Most Significant Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Least Significant Bit
	Not Used				Number of Characters in EOL Sequence		
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 17 thru Control Register 23 - EOL Characters

SALES OFFICES (cont.)

EUROPE, NORTH AFRICA, MIDDLE EAST

Hewlett-Packard GmbH
Technisches Büro München
Eschenstrasse 5
D-8021 **Taufkirchen**
Tel: (089) 6117-1
Telex: 0524985

Hewlett-Packard GmbH
Technisches Büro Berlin
Kaihstrasse 2-4
D-1000 **Berlin** 30
Tel: (030) 24 90 86
Telex: 018 3405 hpbln d

GREECE
Kostas Karayannis
8 Omirou Street
Athens 133
Tel: 32 30 303/32 37 731
Telex: 21 59 62 RAKAR GR
Cable: RAKAR ATHENS

ICELAND
Medical Only
Elding Trading Company Inc.
Hafnarfjölli - Tryggvagötu
P.O. Box 895
IS-Reykjavík
Tel: 1 58 20 1 63 03
Cable: ELDING Reykjavík

IRELAND
Hewlett-Packard Ltd.
King Street Lane
Winnersh, Wokingham
Berkshire, RG11 5AR
GB-England
Tel: (0734) 78 47 74
Telex: 847178
Cable: Hewpie London
Hewlett-Packard Ltd.
Kestrel House
Clanwilliam Place
Lower Mount Street
Dublin 2, Eire

Hewlett-Packard Ltd.
2C Avonberg Ind. Est.
Long Mile Road
Dublin 12
Tel: 514322/514224
Telex: 30439

Medical Only
Cardiac Services (Ireland) Ltd.
Kilmore Road
Artane
Dublin 5, Eire
Tel: (01) 315820

Medical Only
Cardiac Services Co.
95A Finaghy Rd, South
Belfast BT10 0BY
GB-Northern Ireland
Tel: (0232) 625566
Telex: 747626

ISRAEL
Electronics Engineering Div.
of Motorola Israel Ltd.
16, Kremenetski Street
P.O. Box 25016
Tel-Aviv
Tel: 38973
Telex: 33569, 34164
Cable: BASTEL Tel-Aviv

ITALY

Hewlett-Packard Italiana S.p.A.
Via G. Di Vittorio, 9
20063 **Cernusco Sul Naviglio** (MI)
Tel: (02) 903691
Telex: 334632 HEWPACKIT
Hewlett-Packard Italiana S.p.A.
Via Turazza, 14
35100 **Padova**
Tel: (49) 664888
Telex: 430315 HEWPACKI
Hewlett-Packard Italiana S.p.A.
Via G. Armellini 10
1-00143 **Roma**
Tel: (06) 54 69 61
Telex: 610514
Cable: HEWPACKIT Roma
Hewlett-Packard Italiana S.p.A.
Corso Giovanni Lanza 94
I-00133 **Torino**
Tel: (011) 659308
Telex: 221079

Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43 G/C
I-95126 **Catania**
Tel: (095) 37 05 04
Telex: 970291

Hewlett-Packard Italiana S.p.A.
Via Nuova San Rocco A
Capadimonte, 62A
80131 **Napoli**
Tel: (081) 710698

Hewlett-Packard Italiana S.p.A.
Via Martin Luther King, 36/111
I-40132 **Bologna**
Tel: (051) 402394
Telex: 511630

JORDAN
Mouasher Cousins Co.
P.O. Box 1387
Amman
Tel: 24907/39907
Telex: SABCO JO 1456
Cable: MOUASHERCO

KUWAIT
Al-Khalidiya Trading & Contracting
P.O. Box 830-Safat
Kuwait
Tel: 42 4910/41 1726
Telex: 2481 Areeg kt
Cable: VISCOUNT

LUXEMBURG
Hewlett-Packard Benelux S.A./N.V.
Avenue du Col-Vert, 1
(Groenkragegaan)
B-1170 **Brussels**
Tel: (02) 660 5050
Cable: PALOBEN Brussels
Telex: 23 494

MOROCCO
Dolbeau
61 rue Karatchi
Casablanca
Tel: 3041 62
Telex: 23051/22822
Cable: MATERIO

Gerep
2, rue d'Agadir
Boite Postal 156
Casablanca
Tel: 272093/5
Telex: 23 739
Cable: GEREP-CASA

NETHERLANDS

Hewlett-Packard Benelux N.V.
Van Heuven Goedhartlaan 121
P.O. Box 667
1181KK **Amstelveen**
Tel: (20) 47 20 21
Cable: PALOBEN Amsterdam
Telex: 13 216

NORWAY
Hewlett-Packard Norge A/S
Ostendalen 18
P.O. Box 34
1345 **Osteraas**
Tel: (02) 17 11 80
Telex: 16621 hpnas n

Hewlett-Packard Norge A/S
Nygaardsgaten 114
P.O. Box 4210
5013 Nygaardsgaten,
Bergen

Tel: (05) 21 97 33
POLAND

Biuro Informacji Technicznej
Hewlett-Packard
Ul Stawki 2, 6P
PL00-950 **Warszawa**
Tel: 39 59 62, 39 51 87
Telex: 81 24 53

PORTUGAL
Telecra-Empresa Técnica de
Equipamentos Eléctricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531

P-Lisbon 1
Tel: (19) 68 60 72
Cable: TELECTRA Lisbon
Telex: 12598

Medical Only
Mundinter
Intercambio Mundial de Comercio
S.a.r.l.
P.O. Box 2761

Avenida Antonio Augusto
de Aguiar 138
P-Lisbon
Tel: (19) 53 21 31/7
Telex: 16691 munter p
Cable: INTERCAMBIO Lisbon

QATAR
Nasser Trading & Contracting
P.O. Box 1563

Doha
Tel: 22170
Telex: 4439 NASSER
Cable: NASSER

ROMANIA
Hewlett-Packard Reprezentanta
Bd.n. Balcescu 16
Bucuresti
Tel: 15 80 23/13 88 85
Telex: 10440

SAUDI ARABIA
Modern Electronic
Establishment (Head Office)
P.O. Box 1226, Baghdadia Street
Jeddah
Tel: 27 796
Telex: 40035

Cable: ELECTA JEDDAH
Modern Electronic Establishment
(Branch)
P.O. Box 2726

Riyadh
Tel: 62596/66232

Telex: 202049

MODERN ELECTRONIC ESTABLISHMENT

P.O. Box 193
Al-Khobar
Tel: 44678-44813
Telex: 670136

Cable: ELECTA AL-KHOBAR

SPAIN
Hewlett-Packard Española, S.A.
Calle Jerez 3
E-Madrid 16
Tel: (1) 458 26 00 (10 lines)

Telex: 23515 hpe
Hewlett-Packard Española S.A.
Colonia Mirasierra

Edificio Juban

c/o Costa Brava, 13
Madrid 34
Hewlett-Packard Española, S.A.
Milanesado 21-23

E-Barcelona 17
Tel: (3) 203 6200 (5 lines)

Telex: 52603 hpbe e
Hewlett-Packard Española, S.A.
Av Ramón y Cajal, 1

Edificio Sevilla, planta 9°
E-Sevilla 5
Tel: 64 44 54/58

Hewlett-Packard Española S.A.
Edificio Albia II 7° B
E-Bilbao 1
Tel: 23 83 06/23 82 06

Hewlett-Packard Española S.A.
C/Ramon Gordillo 1
(Entlo.)
E-Valencia 10
Tel: 96-361.13.54/361.13.58

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvägen 3, Fack
S-161 **Bromma** 20
Tel: (08) 730 05 50

Telex: 10721
Cable: MEASUREMENTS
Stockholm

Hewlett-Packard Sverige AB
Frötalsgatan 30
S-421 32 **Västra Frölunda**

Tel: (031) 49 09 50
Telex: 10721 via Bromma office

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Zürcherstrasse 20
P.O. Box 307

CH-8952 Schlieren-Zürich
Tel: (01) 730 5240
Telex: 53933 hpag ch
Cable: HPAG CH

Hewlett-Packard (Schweiz) AG
Château Bloc 19
CH-1219 Le Lignon-Geneva

Tel: (022) 96 03 22
Telex: 27333 hpag ch

Cable: HEWPACKAG Geneva
SYRIA
General Electronic Inc.
Nur Basha-Annah Ebn Kays Street
P.O. Box 5761

Damascus
Tel: 33 24 87
Telex: 11215 ITIKAL
Cable: ELECTROBOR DAMASCUS

MODERN ELECTRONIC ESTABLISHMENT

Medical only
Sawah & Co.
Place Azmé
B.P. 2308

Damascus
Tel: 16 367-19 697-14 268

Telex: 11304 SATACO SY
Cable: SAWAH, DAMASCUS

Suleiman Hilal El Miawi

P.O. Box 2528
Mamoun Bitar Street, 56-58

Damascus
Tel: 11 4663

Telex: 11270
Cable: HILAL DAMASCUS

TUNISIA
Tunisie Electronique
31 Avenue de la Liberte

Tunis
Tel: 280 144
Corema

1 ter. Av. de Carthage
Tunis
Tel: 253 821

Telex: 12319 CABAM TN
TURKEY
TEKNIM Company Ltd.

Riza Sah Pehlevi
Cadde No. 7
Kavaklıdere, **Ankara**
Tel: 275800

Telex: 42155
Teknim Com., Ltd.
Barberos Bulvari 55/12
Besiktaş, **Istanbul**
Tel: 613 546

Telex: 23540
E.M.A.
Muhandislik Kollektif Sirketi
Medha Eldem Sokak 4/6
Yüksek Caddesi

Ankara
Tel: 17 56 22
Cable: EMATRADE/Ankara

Yilmaz Ozyurek
Milli Mudafaa Cad 16/6
Kizilay

Ankara
Tel: 25 03 09 - 17 80 26
Telex: 42576 OZEK TR

Cable: OZYUREK ANKARA
UNITED ARAB EMIRATES
Emilac Ltd. (Head Office)
P.O. Box 1641
Sharjah
Tel: 354121/3

Telex: 6136
Emilac Ltd. (Branch Office)
P.O. Box 2711

Abu Dhabi
Tel: 331370/1
UNITED KINGDOM
Hewlett-Packard Ltd.
King Street Lane

Winnersh, Wokingham
Berkshire RG11 5AR
GB-England
Tel: (0344) 784774
Telex: 84 71 76/9
Hewlett-Packard Ltd.
Fourier House,
257-263 High Street
London Colney

St. Albans, Herts
GB-England
Tel: (0727) 24400
Telex: 1-8952716

HEWLETT-PACKARD LTD.

Trafalgar House
Navigation Road
Altringham
Cheshire WA14 1NU

GB-England
Tel: (061) 928 6422

Telex: 668088
Hewlett-Packard Ltd.
Lygon Court

Hereward Rise
Dudley Road

Halesowen,
West Midlands, B62 8SD
GB-England
Tel: (021) 501 1221

Telex: 339105

Hewlett-Packard Ltd.
Wedge House
799, London Road

Thornton Heath
Surrey, CR4 6XL
GB-England
Tel: (01) 684-0103/8

Telex: 946825
Hewlett-Packard Ltd.
14 Wesby St

Castleford
Yorks WF10 1AE
Tel: (0977) 550016
TWX: 5557335

Hewlett-Packard Ltd.
Traxda House
St. Mary's Walk

Maidenhead
Berkshire, SL6 1ST
GB-England
Hewlett-Packard Ltd.
Morley Road

Staplehill
Bristol, BS16 4QT
GB-England

Hewlett-Packard Ltd.
South Queensferry
West Lothian, EH30 9TG
GB-Scotland

Tel: (031) 331 1188
Telex: 72682

Hewlett-Packard Ltd.
Kestrel House
Clanwilliam Place

Lower Mount Street
Dublin 2, Eire

Hewlett-Packard Ltd.
2C Avonberg Ind. Est.
Long Mile Road

Dublin 12
Tel: 514322/514224
Telex: 30439

USSR
Hewlett-Packard

Representative Office
USSR
Pokrovsky Boulevard 4/17-kw 12

Moscow 101000
Tel: 294.20.24
Telex: 7825 hewpak su

YUGOSLAVIA
Iskra Commerce, n.solo.
Zastopstvo Hewlett-Packard
Oblicev Venac 26
YU 11000 Beograd
Tel: 636-955
Telex: 11530

Iskra Commerce, n.solo.
Zastopstvo Hewlett-Packard
Miklosiceva 38/VII
YU-61000 Ljubljana
Tel: 321-674, 315-879
Telex: 31583

Subject Index

A

ABORTIO statement,228
Acknowledge protocol,140
Active Controller (CA),86,113,117
Active Listener (LA),86,113,117
Active Talker (TA),86,113,117
Active-in select code,58
Active-out select code,58
Address,6
Addressing:
 BCD,170
 Defined,5
 GPIO,194
 HP-IB,87,100,127
Advanced GPIO interfacing,209
Advanced HP-IB interfacing,108
Advanced serial interfacing,140
AND function,42
AND operator,39
ASCII,7,310
ASSERT statement,229
Async registers,146
Async troubleshooting hints,161
Asynchronous data transmission,133
Auto-answer routine,144
Auto-disconnect (serial),152
Auto-originate routine,144
Auto-response (GPIO),220
Auto-RTS (serial),159

B

Base conversion functions,45
Baud rate (serial),148,153
BCD:
 Coding,163
 Formats,164,184
 I/O statements,188
 Interface,163
 Interface errors,189
BINAND function,230
Binary:
 AND function,42
 Complement function,43
 Conversions,46,47
 ENTER,23
 Exclusive OR function,43
 Functions,41
 Images,18
 Inclusive OR function,42
 OUTPUT,18
 Representation,37
BINCMP function,231
BINEOR function,232
BINIOR function,233
Bit,36
BIT function,44,234
Bit test function,44
Blank lines, output,19
Branch precedence table,71
Branching on bits,44
BREAK signal,150,156,162
BTD function,47,235
Buffer:
 Control,60
 Defined,56,61
 Empty (GPIO),210
 Status,58
Busy to ready (GPIO),198
Byte,36
Byte output,18

C

Cancelling conversions,29
Character images:
 Input,22
 Output,17
Character length (serial),151
Choosing the Source or Destination,5
CLEAR statement,236
Clearing bits,39
Comma separator:
 Input,21,22
 Output,15
Compact field,8
Compatibility, interface,4
Complement:
 2's,36
 Function,43
 Operator,38
Complementing bytes,43
Control lines:
 BCD interface,176
 GPIO interface,212
HP-IB interface,122
CONTROL statement,60,81,82,237
Control, buffer,60
Conventions Used to Represent Syntax,227
Conversion:
 Base 10 to base 16,46
 Base 10 to base 2,46
 Base 10 to base 8,46
 Base 16 to base 10,47
 Base 2 to base 10,47
 Base 8 to base 10,47
 Between alternate bases,48
 Turning off,29
 With TRANSFER,56,61
CONVERT statement,28,238
Converting I/O Data,28
CRT IS statement,6
CTL line:
 BCD interface,181
 GPIO interface,195

D

Data communications equipment,132
 Data formats (BCD),164,184,186
 Data rate:
 BCD interface,165
 GPIO interface,210
 Data terminal equipment,132
 DC1/DC3 handshake,140
 DCE,132
 Decimal point (BCD),180
 Decimal point, output,15
 Default formats (BCD),164,166
 Delay, handshake (GPIO),204
 Device Clear (SDC,DCL),89,107,108,112,117,125
 Device selector,5,87

Diagnosing errors,31
 Digit characters:
 Input,21
 Output,14
 Digit selection (BCD),178
 Drive capability (GPIO),193
 DTB\$ function,46,240
 DTE,132
 DTH\$ function,46,241
 DTO\$ function,46,242
 Dual channel formatting (BCD),186
 Duplex:
 Full,142
 Half,143

E

EBCDIC output,30
 Eliminating the line-feed requirement,24
 Empty buffer,57,58
 Empty pointer (buffer),57,58,60
 ENABLE INTR statement,66,81,243
 ENABLE KBD statement,75,244
 End-of-line:
 Branch,65,66
 Sequence,82,115,116,159,211
 Sequence images,19
 ENQ/ACK handshake,140
 ENTER USING statement,20,245
 ENTER:
 Conversions,29
 Free field,10
 Statement,9,20,245
 Entering:
 Line-feeds,26
 Numeric data,9
 String data,10
 EOI input termination,27

EOL control:
 GPIO interface,211
 HP-IB interface,115
 Serial interface,160
 ERRN statement,31
 ERROM statement,31,247
 Error handling,31
 Error messages:
 BCD interface,189
 GPIO interface,223
 HP-IB interface,130
 Serial interface,162
 ERRSC statement,31,248
 European format:
 Input,29
 Output,15
 Exclusive OR function,43
 Exclusive OR operator,40
 Exponent digits (BCD),179
 Exponents:
 Input,21
 Output,14

F

FHS transfers:
 BCD interface,188
 GPIO interface,210
 HP-IB interface,110
 Field and statement terminators,25
 Field width,8
 Fill pointer (buffer),57,58,60
 Flag line:
 BCD interface,181
 GPIO interface,195

Format of data (BCD),164,184
 Formatted:
 ENTER,20
 I/O operations,13
 OUTPUT,13
 Framing error,156
 Free field,8,10
 Full buffer,57,58
 Full duplex connection,142
 Function digits (BCD),168,179,182

G

Go-To-Local (GTL),90,125
 GPIO:
 I/O statements,222

Interface,191
 Interface errors,223
 Group Execute Trigger (GET),89,112,117,125

H

Half duplex connection,143
 HALT statement,249
 Handshake:
 Control (GPIO),202
 Defined,4
 ENQ/ACK,140
 GPIO interface,195
 Lines (BCD),164,176,177
 Lines (GPIO),212
 Modem,141,152
 XON/XOFF,140

Hardware interrupt,65,66,81,98,104
 Hex,37
 Hex conversions,46,47
 HP-IB:
 Control lines,115,119,123
 Control registers,112,115
 Control responses,125
 Error messages,130
 I/O statements,108,128
 Message definitions,120
 Registers,112,115,116,283

Select code, 5
 Status registers, 116
 HTD function, 47, 250

Hung ENTER statements, 24
 Hung HP-IB bus, 106

I

I/O statements:
 BCD interface, 188
 GPIO interface, 222
 HP-IB interface, 108
 Serial interface, 162

Ignoring input characters, 23
 Image overflow, 16
 IMAGE statement, 251

Images:
 Advice, 28
 Binary input, 23
 Binary output, 18
 End-of-line sequence, 19
 Numeric input, 21
 Numeric output, 14
 String input, 22
 String output, 17
 Terminator, 26

Inclusive OR function, 42
 Inclusive OR operator, 39
 INDEX conversion, 29
 Input handshakes (GPIO), 198
 Input terminations (serial), 157, 158
 Installing the I/O ROM, 2
 Integer, 41

Interface Clear (IFC), 89, 113, 117, 125
 Interface:
 BCD, 163
 Compatibility, 4
 Functional diagram, 3
 GPIO, 191
 HP-IB, 85
 Serial, 131

Internal printer, 6
 Interrupt:
 Branching, 65
 Enable, 66
 Polling (GPIO), 218
 Pulse width (GPIO), 216
 TRANSFER, 51
 Transfers (GPIO), 209
 Transfers (HP-IB), 110
 Transfers (serial), 143

Interrupts:
 BCD interface, 173, 177
 GPIO interface, 215
 HP-IB interface, 102
 Serial interface, 146

Inverting bits, 40
 IOBUFFER statement, 63, 253

K

K format for OUTPUT, 14, 17

Keyboard masking, 75

L

Line-feed:
 Entering, 26
 Ignoring input, 24

Listen Address (LAG), 100
 Literal images, 17
 Local Lockout (LLO), 90, 125
 LOCA, LOCKOUT statement, 92, 109, 129, 256

LOCAL statement, 109, 128, 255
 Logic polarity:
 BCD interface, 180
 GPIO interface, 205
 Logical operators, 38
 Look-up table conversions, 29

M

Mantissa digits (BCD), 178
 Mark state (RS-232), 133
 Minus sign, output, 15
 Modem:
 Control, 148
 Signal lines, 141
 Status, 149

Use of, 132, 141
 Multiple devices (BCD), 168
 Multiple interrupts:
 BCD interface, 174
 GPIO interface, 218
 My Listen Address (MLA), 88, 112, 117
 My Talk Address (MTA), 88, 112, 117

N

Negative sign, output, 15
 Negative words, 36
 Non-controller, 97, 99
 Normalization:
 BCD interface, 180
 GPIO interface, 205

NOT function, 43
 Number of ports (GPIO), 193
 Numeric data input, 9, 21
 Numeric data output, 8, 14
 Numeric OUTPUT images, 14

O

Octal, 37
 Octal conversions, 46, 47
 OFF EOT statement, 71, 257
 OFF INTR statement, 67, 258
 OFF TIMEOUT statement, 69, 259
 ON EOT statement, 66, 70, 71, 260
 ON INTR statement, 66, 71, 261
 ON TIMEOUT statement, 68, 71, 106, 262
 Operating modes (BCD), 164

OR function, 42, 43
 OR operator, 39
 OTD function, 47, 263
 OUTPUT USING statement, 13, 264
 OUTPUT:
 Conversions, 29
 Images, 14
 Statement, 8, 13, 264

Output:

BCD interface, 187
 Blank lines, 19
 Drive capability (GPIO), 193

Handshakes (GPIO), 196
 Inhibit (GPIO), 205, 207
 Overflow of images, 16
 Overlap, 52, 53

P

PAIRS conversion, 29
 Parallel Poll, 90, 125
 Parallel Poll Configure (PPC), 99, 125
 Parity error:
 GPIO interface, 214, 216
 Serial interface, 154, 156
 Parity:
 GPIO interface, 214
 Serial interface, 134, 150
 Part I: Beginner's Guide, 1
 Part II: Bits and Bytes and Such, 33
 Part III: Advanced I/O Operations, 49
 Part IV: Interface Programming Techniques, 83
 Part V: Appendix, 225
 Partial fields (BCD), 170
 Partial handshake (GPIO), 195
 PASS CONTROL statement, 109, 129, 266
 Peripheral device, 1
 Place value, bits, 35, 36

Polarity, logical:
 BCD interface, 180
 GPIO interface, 205
 Port 10 (BCD), 171, 177, 183
 Ports:
 16-bit (GPIO), 195
 8-bit (GPIO), 194
 PPOLL function, 109, 129, 267
 Primary address:
 BCD interface, 170
 Defined, 6
 GPIO interface, 194
 HP-IB interface, 87, 127
 Printer interface (serial), 135, 135
 PRINTER IS statement, 6, 85
 Printing to peripheral devices, 6
 Punctuation characters:
 Input, 21
 Output, 15

Q

Quick reference, 227

R

Radix point, output, 15
 Ready to busy (GPIO), 198
 Receiver control (serial), 154
 Reference:
 HP-IB, 108
 Interface registers, 283
 Syntax, 227
 Registers:
 BCD interface, 175, 288
 Buffer, 58, 60
 GPIO interface, 291
 HP-IB interface, 283
 Serial interface, 146, 285
 Remote Enable (REN), 90, 92, 108, 125

REMOTE statement, 92, 109, 129, 188, 268
 Removing the I/O ROM, 3
 Replication of images, 16
 REQUEST statement, 105, 109, 129, 269
 RESET statement, 270
 Resetting the peripheral device (GPIO), 213
 RESUME statement, 271
 ROM:
 Drawer, 2
 Installation, 2
 Removal, 3
 Socket, 2
 RS-232 troubleshooting hints, 161
 RS-232-C interface, 131

S

Sales and Service Offices, 285
 Secondary addressing, 99, 120, 273
 Secondary commands, 99, 120, 273
 SEND statement, 99, 100, 110, 129, 272
 Serial Poll, 90, 103, 125
 Serial:
 Data character, 134, 151
 I/O statements, 162
 Interface, 131
 Interface errors, 162
 Interface registers, 146
 Troubleshooting hints, 161
 Service Request (SRQ), 66, 67, 90, 102, 105, 112, 117, 125
 SET TIMEOUT statement, 68, 275
 Setting bits, 39
 Sign bit, 36
 Sign bit polarity (BCD), 183
 Sign character, output, 15
 Skip to end of record, 23
 Skipping unwanted characters, 23

Space state (RS-232), 133
 SPOLL function, 103, 110, 129, 276
 Start bit, 134
 States of an RS-232 line, 133
 STATUS statement, 58, 79, 80, 277
 Status:
 Buffer, 58
 Lines (BCD), 176
 Lines (GPIO), 212
 Stop bits, 134, 150
 String data:
 Input, 10
 Output, 8
 String images, 17
 String images, input, 22
 Strobe handshake (GPIO), 195, 204
 Suppressing carriage-return/line-feed, 19
 Syntax Reference, 227
 System Controller (SC), 86

T

Take Control (TCT), 89, 104, 125
 Talk Address (TAD), 100
 Teletype interface (serial), 139
 Terminal interface (serial), 138
 Terminating with EOI, 27
 Terminator:
 Field, 25
 Images, 26
 Statement, 25
 Testing bits, 44
 Timeout, 68
 Timing compatibility, 4
 Trailing blanks, 8
 Transfer rate:
 BCD interface, 165
 GPIO interface, 210
TRANSFER:
 BCD interface, 188

Fast-handshake, 56, 62
 GPIO interface, 209
 HP-IB interface, 101
 Interrupt, 51, 62
 Statement, (in), 63, 280
 Statement, (out), 61, 278
 Terminations, 63, 64, 68, 70
 Transmitter control (serial), 154
 TRIGGER statement, 94, 110, 129, 282
Trigger:
 Byte (GPIO), 219
 HP-IB interface, 89
 Signal (GPIO), 220
 Triggering transfers (GPIO), 219
 Troubleshooting hints (serial), 161
 Two's complement, 36

U

Unaddress, 86, 89
 Unformatted ENTER, 9

Unformatted OUTPUT, 8
 Unlisten (UNL), 86

W

Width of field, 8
 Word, 36

Word output, 18

X

XON/XOFF handshake, 140

Error Messages

Error No. Message	Meaning	Possible Corrective Action
101 XFR	This is only a warning. It is issued when a program is paused with an I/O TRANSFER still active. Do not attempt to modify a program when a TRANSFER is active.	Before you modify or rerun the program, stop all active transfers with a RESET, HALT, or ABORTIO instruction; or press the RESET key.
110 I/O CARD	An interface has failed self-test. This indicates a probable hardware problem.	ERRSC can be used to determine which interface has failed. Try recycling the power (turn computer off, then back on again). If the interface still fails, contact the authorized HP-85 dealer or the HP sales and service office from which you purchased your HP-85.
111 I/O OPER	The I/O operation attempted is not valid with the type of interface being used. Some examples are: specifying a status or control register that does not exist, using a primary address with an RS-232 interface, or using an I/O statement that is not defined for the interface being used.	ERRL can be used to identify the improper statement. Check this statement in the Syntax Reference section to determine if it is defined for the interface being used. If the statement is valid, check the appropriate Interface Programming section to get details on the proper mode or configuration required for the statement used.
112 I/O ROM	The I/O ROM has failed the checksum self-test. This indicates a probable hardware problem.	Try recycling the power (turn the computer off, then back on again). If the error keeps recurring, contact the authorized HP-85 dealer or the HP sales and service office from which you purchased your HP-85.

Error No.	Message	Possible Corrective Action
113	<p>An interface-dependent error.</p> <p>HP-IB: The statement used requires the interface to be system controller.</p> <p>Serial: UART receiver overrun; data has been lost.</p> <p>BCD: Attempting to put the interface into an illegal mode.</p> <p>GPIO: An odd number of bytes was transferred when the interface was configured for 16-bit words.</p>	<p>ERRSC can be used to determine the source of the error. Refer to the appropriate Interface Programming section to get details on the error and possible corrective actions.</p>
114	<p>An interface-dependent error.</p> <p>HP-IB: The statement used requires the interface to be active controller.</p> <p>Serial: Receiver buffer overrun; data has been lost.</p> <p>BCD: Port 10 not currently available.</p> <p>GPIO: FHS TRANSFER aborted by ST0.</p>	<p>ERRSC can be used to determine the source of the error. Refer to the appropriate Interface Programming section to get details on the error and possible corrective actions.</p>
115	<p>An interface-dependent error.</p> <p>HP-IB: The statement used requires the interface to be addressed to talk.</p> <p>Serial: Automatic disconnect forced.</p> <p>BCD: FHS TRANSFER aborted by FLGB.</p> <p>GPIO: Interface configuration does not allow an output enable or output operation on Port A or Port B.</p>	<p>ERRSC can be used to determine the source of the error. Refer to the appropriate Interface Programming section to get details on the error and possible corrective actions.</p>
116	<p>An interface-dependent error.</p> <p>HP-IB: The statement used requires the interface to be addressed to listen.</p> <p>Serial: This error number not currently used.</p> <p>BCD: Data direction mismatch on current operation.</p> <p>GPIO: Cannot start operation because handshake CTL line is not in proper state.</p>	<p>ERRSC can be used to determine the source of the error. Refer to the appropriate Interface Programming section to get details on the error and possible corrective actions.</p>

Error No.	Message	Possible Corrective Action
117	<p>An interface-dependent error.</p> <p>HP-IB: The statement used requires the interface to be non-controller.</p> <p>Serial: This error number not currently used.</p> <p>BCD: Interface command has been directed to a non-existent field.</p> <p>GPIO: This error number not currently used.</p>	ERRSC can be used to determine the source of the error. Refer to the appropriate Interface Programming section to get details on the error and possible corrective actions.
118	<p>An interface-dependent error.</p> <p>HP-IB: This error number not currently used.</p> <p>Serial: This error number not currently used.</p> <p>BCD: Cannot start operation because CTL line is not in the proper state.</p> <p>GPIO: This error number not currently used.</p>	ERRSC can be used to determine the source of the error. Refer to the appropriate Interface Programming section to get details on the error and possible corrective actions.
119	<p>An interface-dependent error.</p> <p>HP-IB: This error number not currently used.</p> <p>Serial: This error number not currently used.</p> <p>BCD: Data format does not match the mode of the interface.</p> <p>GPIO: This error number not currently used.</p>	ERRSC can be used to determine the source of the error. Refer to the appropriate Interface Programming section to get details on the error and possible corrective actions.
120	An interface-dependent error. This error number not currently used.	
121	An interface-dependent error. This error number not currently used.	
122	An interface-dependent error. This error number not currently used.	
123 NO ":";	Syntax error. A semicolon delimiter was expected in the statement.	Put the semicolon where it belongs!

Error No. Message	Meaning	Possible Corrective Action
124 ISC	Either the interface select code specified is out of range, or there is no interface present set to the specified select code. Interface select codes must be in the range of 3 thru 10. Select codes 1 (CRT) and 2 (internal printer) are allowed for OUTPUT statements only.	Be sure that the interface select code is within the proper range. Pay special attention to variables that are used to hold interface select codes. If the interface select code is OK, be sure that the interface is plugged in properly. Finally, check the switch settings on the interface. (Someone might have changed them last weekend.)
125 ADDR	The primary address specified is improper. Only addresses 00 thru 31 are allowed, but not all interfaces use this entire range.	Be sure that the primary address is within the proper range. Pay special attention to variables that are used to hold addresses or device selectors.
126 BUFFER	Four possible buffer problems: (1) The string variable specified has not been declared as an IOBUFFER. (2) Attempting to ENTER from a buffer which is out of data. (3) Attempting to OUTPUT to a buffer which is already full. (4) Attempting an output TRANSFER with an empty buffer.	Be sure you have included the necessary IOBUFFER statement. Check the logical flow of your program (in what order are the statements executed). Buffer contents can be examined at any time by simply printing or displaying the string variable being used as the buffer. If this doesn't provide enough information, the buffer pointers can be examined with the STATUS statement.
127 NUMBER	An incoming character sequence does not constitute a valid number, or a number being output requires three exponent digits and an "e" format was specified.	If the error is from an output operation, check the magnitude of the number and the format used. If the error is from an input operation, there are many possible causes. Here are some things to look for: more than 255 leading non-numeric characters, unexpected spaces in a character stream when a character-count format is used, punctuation sequences that include potentially numeric characters used in an order that is numerically meaningless.

Error No. Message	Meaning	Possible Corrective Action
128 EARLY TERM	A buffer was emptied before all the ENTER fields were satisfied, or a field terminator was encountered before the specified character count was reached.	Check your incoming character stream, ENTER list, and image specifiers.
129 VAR TYPE	The type (string or numeric) of a variable in an ENTER list does not match with the image specified for that variable.	Check your ENTER list and image specifiers.
130 NO TERM	A required terminator was not received from an interface or buffer during an ENTER statement. Remember that there is a default requirement for a line-feed statement terminator.	Check your incoming character stream, ENTER list, and image specifiers.

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Binary	Oct	Hex	Dec	
NULL	00000000	000	00	0	
SOH	00000001	001	01	1	GTL
STX	00000010	002	02	2	
ETX	00000011	003	03	3	
EOT	00000100	004	04	4	SDC
ENQ	00000101	005	05	5	PPC
ACK	00000110	006	06	6	
BELL	00000111	007	07	7	
BS	00001000	010	08	8	GET
HT	00001001	011	09	9	TCT
LF	00001010	012	0A	10	
VT	00001011	013	0B	11	
FF	00001100	014	0C	12	
CR	00001101	015	0D	13	
SO	00001110	016	0E	14	
SI	00001111	017	0F	15	
DLE	00010000	020	10	16	
DC1	00010001	021	11	17	LLO
DC2	00010010	022	12	18	
DC3	00010011	023	13	19	
DC4	00010100	024	14	20	DCL
NAK	00010101	025	15	21	PPU
SYNC	00010110	026	16	22	
ETB	00010111	027	17	23	
CAN	00011000	030	18	24	SPE
EM	00011001	031	19	25	SPD
SUB	00011010	032	1A	26	
ESC	00011011	033	1B	27	
FS	00011100	034	1C	28	
GS	00011101	035	1D	29	
RS	00011110	036	1E	30	
US	00011111	037	1F	31	

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Binary	Oct	Hex	Dec	
space	00100000	040	20	32	LA0
!	00100001	041	21	33	LA1
”	00100010	042	22	34	LA2
#	00100011	043	23	35	LA3
\$	00100100	044	24	36	LA4
%	00100101	045	25	37	LA5
&	00100110	046	26	38	LA6
*	00100111	047	27	39	LA7
(00101000	050	28	40	LA8
)	00101001	051	29	41	LA9
*	00101010	052	2A	42	LA10
+	00101011	053	2B	43	LA11
,	00101100	054	2C	44	LA12
-	00101101	055	2D	45	LA13
.	00101110	056	2E	46	LA14
/	00101111	057	2F	47	LA15
0	00110000	060	30	48	LA16
1	00110001	061	31	49	LA17
2	00110010	062	32	50	LA18
3	00110011	063	33	51	LA19
4	00110100	064	34	52	LA20
5	00110101	065	35	53	LA21
6	00110110	066	36	54	LA22
7	00110111	067	37	55	LA23
8	00111000	070	38	56	LA24
9	00111001	071	39	57	LA25
:	00111010	072	3A	58	LA26
,,	00111011	073	3B	59	LA27
<	00111100	074	3C	60	LA28
=	00111101	075	3D	61	LA29
>	00111110	076	3E	62	LA30
?	00111111	077	3F	63	UNL

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Binary	Oct	Hex	Dec	
@	01000000	100	40	64	TA0
A	01000001	101	41	65	TA1
B	01000010	102	42	66	TA2
C	01000011	103	43	67	TA3
D	01000100	104	44	68	TA4
E	01000101	105	45	69	TA5
F	01000110	106	46	70	TA6
G	01000111	107	47	71	TA7
H	01001000	110	48	72	TA8
I	01001001	111	49	73	TA9
J	01001010	112	4A	74	TA10
K	01001011	113	4B	75	TA11
L	01001100	114	4C	76	TA12
M	01001101	115	4D	77	TA13
N	01001110	116	4E	78	TA14
O	01001111	117	4F	79	TA15
P	01010000	120	50	80	TA16
Q	01010001	121	51	81	TA17
R	01010010	122	52	82	TA18
S	01010011	123	53	83	TA19
T	01010100	124	54	84	TA20
U	01010101	125	55	85	TA21
V	01010110	126	56	86	TA22
W	01010111	127	57	87	TA23
X	01011000	130	58	88	TA24
Y	01011001	131	59	89	TA25
Z	01011010	132	5A	90	TA26
[01011011	133	5B	91	TA27
\	01011100	134	5C	92	TA28
]	01011101	135	5D	93	TA29
^	01011110	136	5E	94	TA30
_	01011111	137	5F	95	UNT

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Binary	Oct	Hex	Dec	
,	01100000	140	60	96	SC0
a	01100001	141	61	97	SC1
b	01100010	142	62	98	SC2
c	01100011	143	63	99	SC3
d	01100100	144	64	100	SC4
e	01100101	145	65	101	SC5
f	01100110	146	66	102	SC6
g	01100111	147	67	103	SC7
h	01101000	150	68	104	SC8
i	01101001	151	69	105	SC9
j	01101010	152	6A	106	SC10
k	01101011	153	6B	107	SC11
l	01101100	154	6C	108	SC12
m	01101101	155	6D	109	SC13
n	01101110	156	6E	110	SC14
o	01101111	157	6F	111	SC15
p	01110000	160	70	112	SC16
q	01110001	161	71	113	SC17
r	01110010	162	72	114	SC18
s	01110011	163	73	115	SC19
t	01110100	164	74	116	SC20
u	01110101	165	75	117	SC21
v	01110110	166	76	118	SC22
w	01110111	167	77	119	SC23
x	01111000	170	78	120	SC24
y	01111001	171	79	121	SC25
z	01111010	172	7A	122	SC26
{	01111011	173	7B	123	SC27
	01111100	174	7C	124	SC28
}	01111101	175	7D	125	SC29
~	01111110	176	7E	126	SC30
DEL	01111111	177	7F	127	SC31

