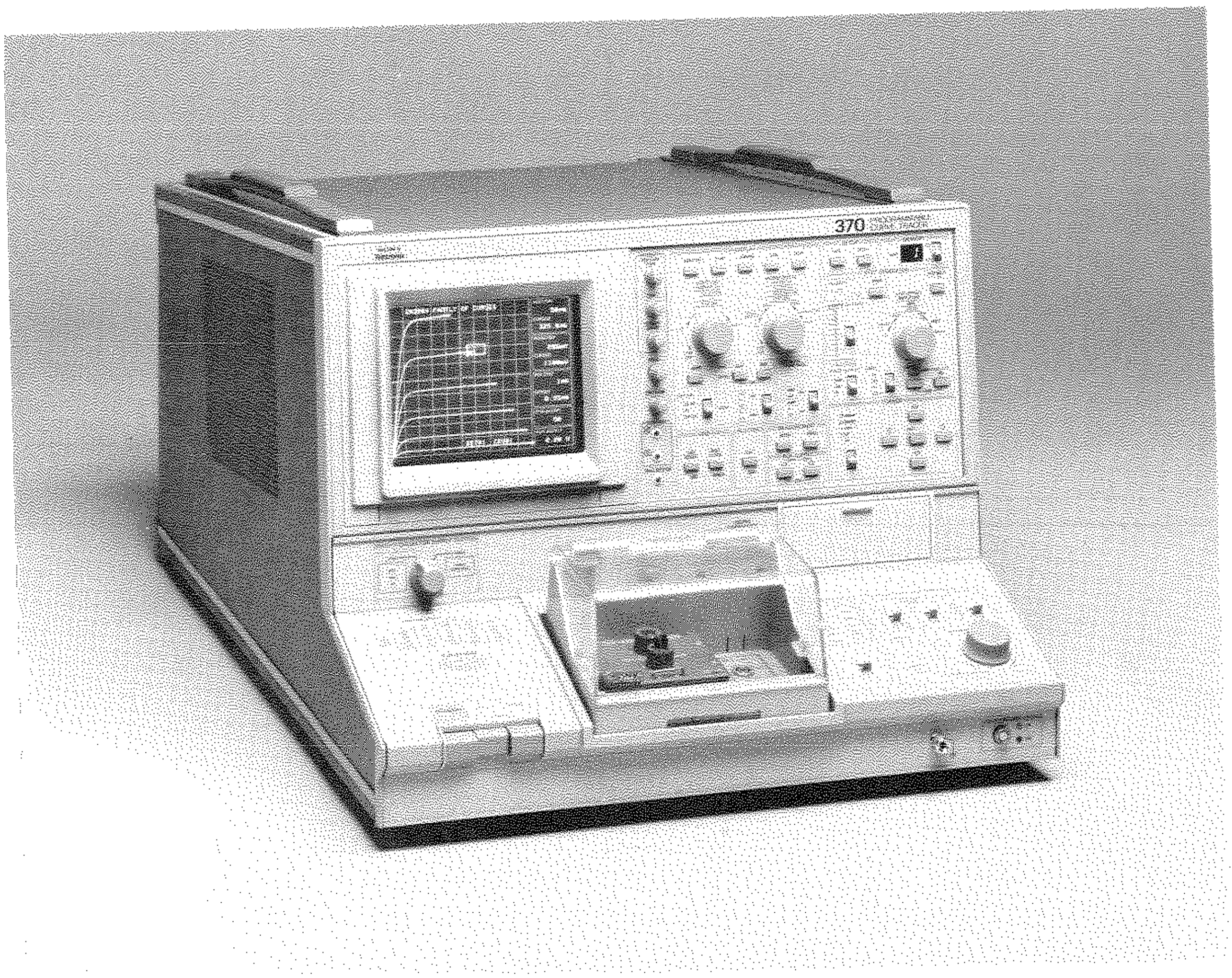
A 340.

TEK INTERFACING INFORMATION

PRODUCT GROUP 48
070-6067-00

370 Instrument Interfacing Guide

REFERENCE COPY



SONY
Tektronix

The information presented in this interfacing guide is provided for instructional purposes only. Tektronix, Inc. does not warrant or represent in any way the accuracy or completeness of any program herein or its fitness for a user's particular purpose.

For additional copies, order:
070-6067-00
First printing OCT 1986

TABLE OF CONTENTS

Section 4 Controlling the 370 over the bus

Section 5 Instrument setup over the bus

Section 6 Data storage and transfer

Section 7 Device-dependent message formats

Section 8 Scope of programmability

Section 9 Commands tables

Section 10 Service requests

Section 11 Miscellaneous information

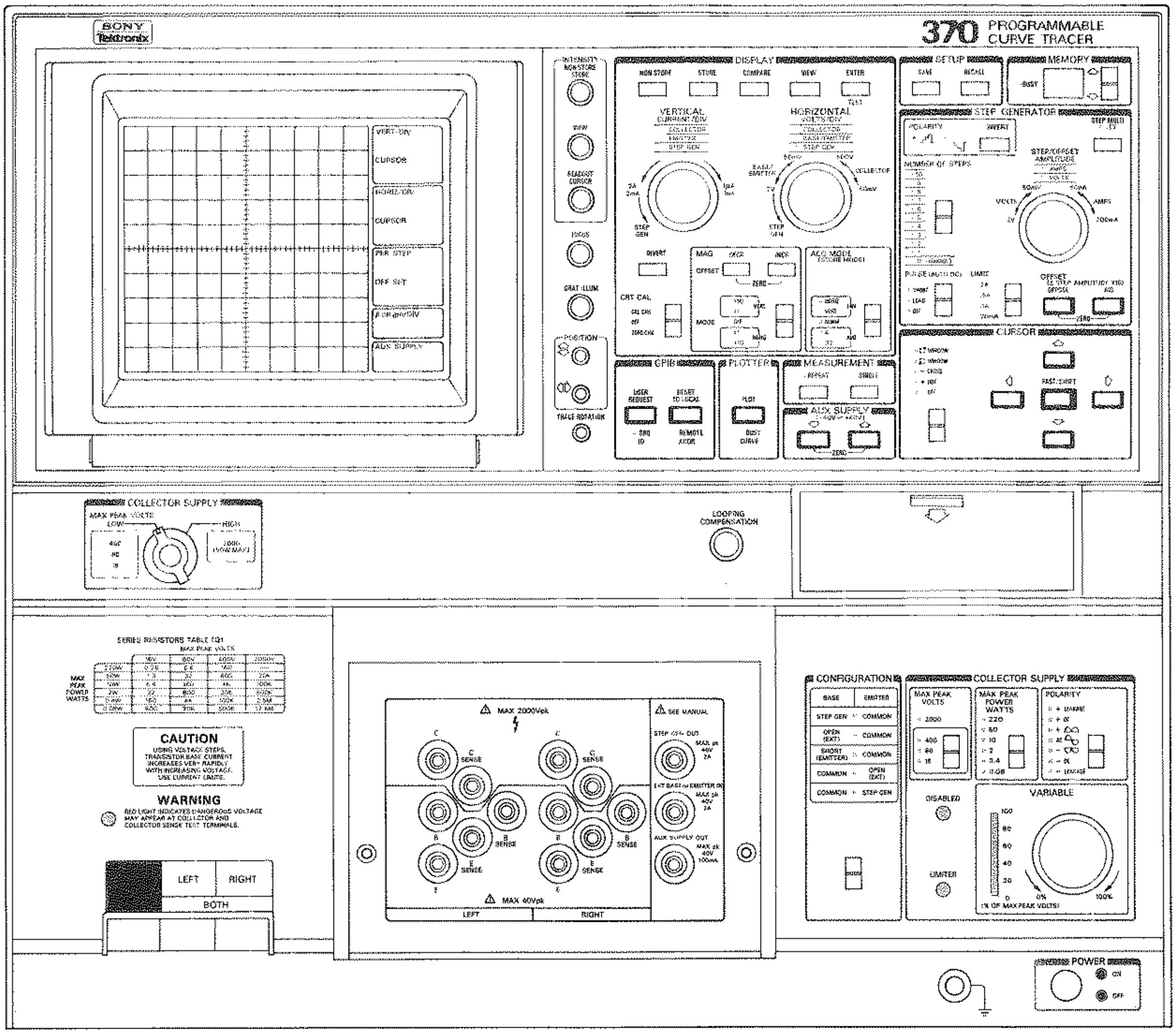


Figure 1. Front-Panel Controls.

INTRODUCTION

This Interfacing Guide is designed to help you get started quickly and easily using the 370 Programmable Curve Tracer with the IEEE General Purpose Interface Bus or GPIB. This guide explains not only how to set up the curve tracer for GPIB operation, but also how to communicate with it using a variety of controllers. Sample programs are also included. This document does not take the place of the 370 Operators Manual or any other publication supplied with your curve tracer or your system controller. Instead, it is intended to help you set up your system and get started making programmable measurements.

GPIB communication with the 370 conforms to Tektronix Codes and Formats, which standardizes and optimizes bus messages for human and machine readability.

We assume you are generally familiar with the GPIB and have an appropriate bus controller and controller language. The GPIB is documented in ANSI/IEEE Std 488-1978, IEEE Standard Interface for Programmable Instruments. This guide is not meant to be a tutorial on GPIB operation so it would be good to have access to the ANSI/IEEE Std 488 along with the manuals for the controller you are using.

This guide is divided into eleven sections. The first six discuss GPIB control of the type 370 curve tracer in general terms. The seventh through the tenth go into more detail on command syntax, etc. and include a complete reference table of commands available to control your 370. The last shows you examples of programs and provides a reference chart of ASCII codes and GPIB codes.

Initially, reading the first six sections will get you started. As you get involved in more detailed work you will want to use the remaining sections.

This guide corresponds to the 370 version denoted "V81.1,F1.01". This can be verified from the front panel by pressing the FAST and the ID buttons together and reading the results on the CRT.

Section 1 -- Instrument programmability

370 PROGRAMMING CAPABILITIES

Overview

The 370 curve tracer adds the major feature of programmability to the many capabilities of the standard curve tracer. With front-panel settings interface that allows communication with any of a variety of possible instrument controllers.

Front-panel setups, data acquisition, and data transfer can now be easily handled under program control. Among the advantages is that setups and tests, even a series of them, are quickly reproducible without error. Also, data and measurements are easily acquired and quickly stored for later reference and analysis.

Scope and limitations

While the 370 is programmable, it does require an operator for most situations. A few significant operations are not subject to program control: inserting and removing the device being tested, closing and opening the protective cover, switching between left and right sockets in the test fixture, and switching the collector supply to and from the 2 kV range.

Also, while setups can be stored and recalled from either the bubble memory or the instrument controller, the curve tracer must leave any sequencing or decision making to the program running in the controller or to the operator.

The following surveys the programmable features the 370 makes available to you.

OTHER FEATURES

Front panel control

Most controls on the front panel can be controlled both manually and over the bus. Besides the exceptions already noted, the CRT display adjustments are also left for the operator.

Setups transfers

Measurement setups can be loaded all-at-once with a single program instruction, or, using a series of instructions, they can be loaded a few controls at a time.

Display control

The CRT display can show data from the device or devices under test, from the bubble memory, or from the controller. Comparisons can be made by showing curves from different sources at the same time. Each display can be labeled with several words of text.

Bubble-memory control

With the bubble-memory cartridge you now can avoid repetitive and error-prone knob twiddling in reestablishing a measurement setup. You can also store measurement data in the cartridge; data such as you might want to use for device comparisons. Use of these storage locations can be controlled from both the front panel and the instrument controller.

Each cartridge has 16 numbered locations for storing setups. Each location can hold the settings for all the programmable controls on the front panel. It can also hold 24 characters of text, for labeling the setup.

Each cartridge also has 16 numbered locations for storing measurement results stored as waveforms. Each waveform consists of a preamble of information needed for scaling and other interpretation, and an array of numeric data representing the characteristic curve for that measurement. These locations also have room for 24 characters of text, for labeling the measurement data.

Data acquisition control

Data can be acquired in a number of ways, all of which can be controlled from the bus as well as from the front panel. Generally, the data is acquired as a waveform of 1024 digitized points.

Cursor control and readout

The dot cursor can be located at any desired point along the 1024 points being digitized. The cross-hair and window cursors can be located at any desired location within the graticule.

The location of any of the three sets of cursors, dot, cross-hair, or window, can be controlled from the bus and the resulting data values, in units of volts (horizontal) and amperes (vertical), can be read out over the bus.

Data transfer

The bulk of the data you will want to copy from the curve tracer will be characteristic curves, along with the parameters, such as scale factors, needed to interpret them. These parameters are collected into an ASCII string called the preamble. The preamble and curve data are available individually or together. Together they are called a waveform.

Under GPIB control, preambles, curves, and waveforms can be copied from the 370 to the controller. They can later be restored to the curve tracer.

Plotter control

If a plotter is connected to the curve tracer a plot of the display can be initiated from the bus.

Diagnostics

Along with internal digital control of the instrument come both the need for, and capability to accomplish, extensive self-diagnostics, which verify the internal functioning of the instrument. Some of these tests can be initiated over the bus.

During operation with an instrument controller there are also a number of significant conditions and events the 370 can detect and then report to the controller.

Section 2 -- GPIB and 370 basics

GPIB REVIEW

Introduction

The general purpose interface bus, or GPIB, is a standardized, digital interface for interconnecting up to 15 self-contained instruments, controllers, and other devices. The instruments may be such units as your 370 curve tracer, signal generators, digital multimeters, or digital oscilloscopes. The ANSI and IEEE standard defines two aspects of the interface: the hardware and a basic communication protocol.

The hardware consists of a set of interface circuits in each device along with standard, 24-wire cabling for interconnecting the devices in a system. The 24 wires include 16 lines used for signaling: eight for addresses or data, three for handshaking during data transfers, and five for interface management. Most data are transferred as a series of eight-bit bytes transferred over the eight data lines.

The basic communication protocol specifies a set of predefined **interface messages** for system organization and housekeeping but indicates only the basic requirements for communicating other information such as setup commands and measurement results. It does not define the meaning or format of the latter, termed **device-dependent messages.**

The meaningful messages for each instrument are specified by the instrument manufacturer and are usually spelled out in the instrument manual. Tek has standardized on a higher level protocol for all its GPIB instruments, calling it Tek Codes and Formats, or TC & F.

A typical GPIB system will include a controller and one or more instruments, such as the 370 curve tracer. Some instruments are talk only devices while others can both talk and listen. Your 370 does both.

Having the controller linked to the 370, and possibly other instruments, enables you to do work in four major task areas:

- Program development
- System control
- Data processing
- Display and storage

These are discussed in the following paragraphs.

Program development

Program development includes the functions of writing, editing, and debugging the programs needed to control the instruments in the system.

Controlling the system

When actually running a program, the controller assigns tasks to the instruments, coordinates communication, handles error conditions, and monitors the system's progress. This instrument control task can be further divided into five functions: Addressing instruments, sending commands, transmitting and receiving data, handling interrupts, and monitoring device status.

Let's look at each of these functions individually.

Addressing instruments -- The controller selects an instrument by addressing it. Each instrument on a bus is assigned a unique primary address in the range 0 through 30. The controller uses this address to tell an instrument to talk or listen.

Sending data and commands -- Device-dependent messages carry commands and data from the controller to the 370 and return instrument status information and measurement data. The ATN line is asserted during the sending of interface messages, thus distinguishing them from device-dependent messages. See figure 2.

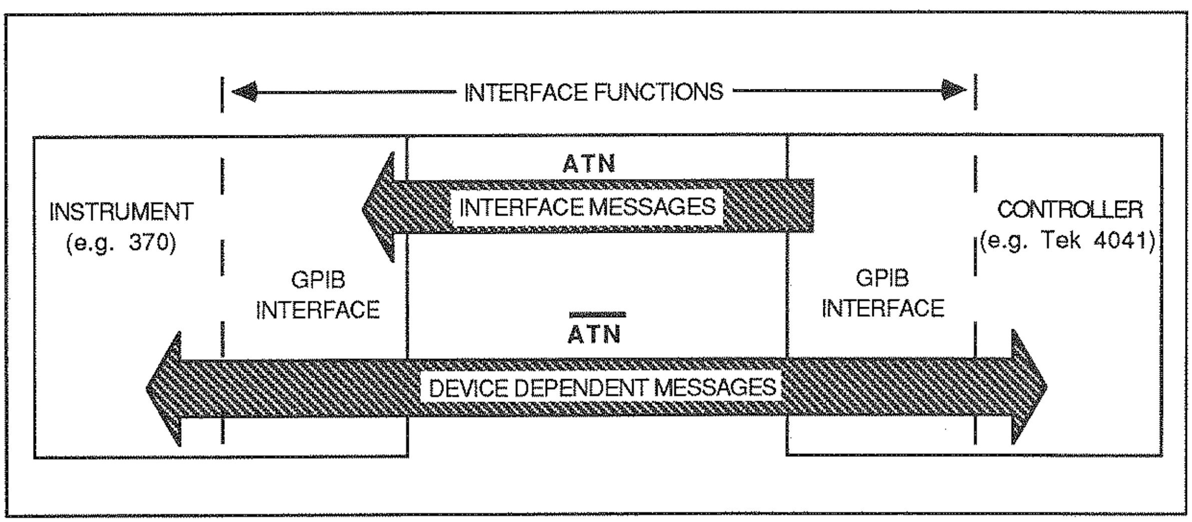


Figure 2. The controller sends interface messages with Attention (ATN) asserted. These messages control interface functions. Device dependent messages, sent with ATN unasserted, control instrument functions and transfer data.

Interface messages are commands that control interface functions. The IEEE 488 standard specifies these so that they are the same for all devices. There are two kinds: uniline and multiline, where "line" refers to the 16 signal lines within the bus. Multiline messages can be further subdivided into universal commands, addresses, and addressed commands. Figure 3 shows how the different types of GPIB messages are related and indicates the standard three-letter symbols assigned to them.

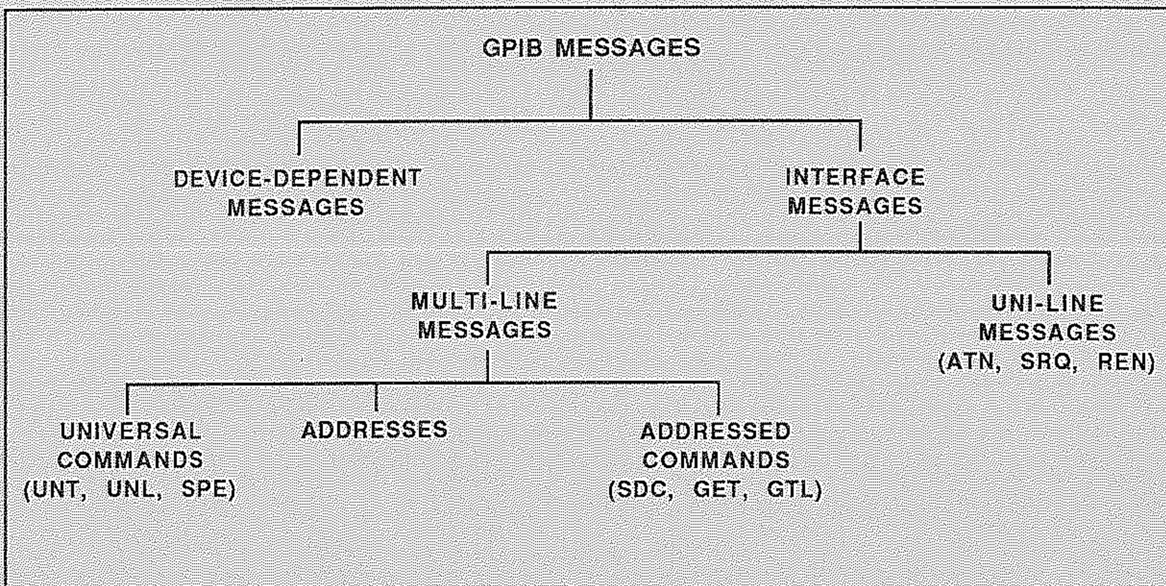


Figure 3. Messages sent over the GPIB can be divided into two general types — interface messages and device-dependent messages. Interface messages are further divided into universal multi-line messages, addressed multi-line messages, and uni-line messages.

Multiline interface messages are sent by placing a byte on the eight data lines of the GPIB and asserting the ATN line. Universal commands affect all devices on the bus while addresses and addressed commands affect only the addressed instruments.

Uniline interface messages are sent by asserting one of the set of five individual interface signal lines of the GPIB: **SRQ** (service request), **ATN** (attention), **IFC** (interface clear), **REN** (remote enable), and **END** (EOI, end or identify).

Device-dependent messages consist of commands or data that control instrument function and communicate instrument status, as well as data from measurements and other information. As noted before, the content and format of these messages has been addresesed with Tektronix Codes and Formats.

All of these message types are significant for the 370.

Transmitting and receiving data -- Most instruments talk (send data) or listen (receive data) to the system controller. The 370 does both. In fact the 370 sends and receives two classes of data: instrument setups and measurement results. There are various possible ways of coding such data. The 370, in keeping with Tek Codes and Formats, uses English key-words for setup and status data and a combination of similar key-words and a form of binary-coded numbers, for measurement data.

Handling interrupts -- The 370 and other devices in the system can generate interrupts to inform the controller of conditions warranting some sort of attention, such as an error condition or the completion of an operation. The controller polls the devices on the bus to find the one generating the interrupt, reads its status, and takes appropriate action.

Processing data

Still another major task of a GPIB system controller is processing the data acquired from instruments. Examples might be extracting key parameters from a family of curves and deciding whether some pass-fail criterion has been exceeded.

Storing and displaying the data

Once data have been sent to the controller they can be stored or displayed, besides being processed. The console screen is one place you may want to display data.

370 INTERFACE CAPABILITIES

Standard interface capabilities

IEEE Std 488 defines a variety of possible interface capabilities for differing needs among various controllers and instruments. The accompanying table summarizes the capabilities realized in the 370. The abbreviations are detailed in the IEEE standard.

Table 1
GPIB Interface Specifications

FUNCTION	SUBSET	NOTE
Source Handshake	SH1	Complete capability
Acceptor Handshake	AH1	Complete capability
Talker	T6	Basic Talker, Serial Poll, Talk Only, Unaddress if MLA
Listener	L4	Basic Listener, Unnaddress if MTA
Service Request	SR1	Complete capability
Remote/Local	RL2	No Local Lockout (LLO)
Parallel Poll	PPØ	No capability
Device Clear	DC1	Complete capability
Device Trigger	DTØ	No capability
Controller	CØ	No capability

Interface messages

The following explains how the curve tracer reacts to standard interface messages. Abbreviations are from IEEE Std 488.

As noted before, a uniline message is sent over a dedicated line and a multiline message is sent using the eight data lines while the ATN line is asserted. In the following descriptions, uniline messages are described as having the appropriate line asserted. Multiline messages are described with their respective ASCII code and decimal value for the eight-bit byte expressed on the eight data lines.

Due to the set of interface functions needed for the 370, not all interface messages would be meaningful. The 370 does not respond to the following.

LLO	**Local lockout**
GET	**Group execute trigger**
PPC	**Parallel poll configure**
PPU	**Parallel poll unconfigure**
TCT	**Take control**

It does respond to or use, the following interface messages, as described.

My Listen and My Talk Address (MLA and MTA) -- The 370's address is established by setting the address select switches on the rear panel. When the 370 receives its own address given with either of these commands it responds by entering the appropriate state, ready to talk or ready to listen.

Attention (ATN) -- With the ATN line asserted, data on the eight data lines are interpreted as an address or interface message. With most controller programming languages, operation of the **ATN** line is transparent to you.

Unlisten (UNL) and **Untalk (UNT)** -- When the **Unlisten (UNL)** message (ASCII **"?", decimal 63**) is received, the curve tracer's listen function is placed in an idle (unaddressed) state. In the idle state, the curve tracer will not accept messages over the GPIB. The **Talk** function is placed in an idle state when the curve tracer receives the **Untalk (UNT)** message (ASCII **"_", decimal 95**). In this state the curve tracer cannot transmit data via the GPIB.

Interface Clear (IFC) -- When the **Interface Clear (IFC)** line is asserted, both the Talk and Listen functions are placed in an idle state. This produces the same effect as receiving both the **Untalk** and **Unlisten** commands. It resets the interface only, clearing the input and output buffer, and does not affect any instrument functions. This can be used to restart communication with the 370.

Device Clear (DCL) -- The **Device Clear (DCL)** message (ASCII **Control T, decimal 20**) reinitializes communication between the 370 and the controller. In response to **Device Clear,** the 370 clears any input and output messages as well as any unexecuted control settings. Any errors and events waiting to be reported, except power-on, are also cleared. If the SRQ line has been asserted for any reason other than power-on, it becomes unasserted when DCL is received.

Selected Device Clear (SDC) -- **Selected Device Clear (SDC)** (ASCII **Control D, decimal 4**) performs the same function as **DCL**; however, only instruments that have been listen-addressed respond to **SDC.**

Go To Local (GTL) -- The **Go To Local (GTL)** message (ASCII **"Control A", decimal 1**) takes the 370 "off bus" and turns off the front panel **REMOTE** indicator, just as the **RESET TO LOCAL** button does,

Remote Enable (REN) -- When the **Remote Enable (REN)** line is asserted and the instrument receives its listen address **(MLA),** the curve tracer is placed in its **Remote State (REMS)** and the front panel **REMOTE** indicator is turned on.

Service Request (SRQ) -- The **Service Request (SRQ)** line is set by the 370 each time it has a change in status to report to the controller or when the operator presses the **User Request** button on the front panel.

Serial Poll Enable and Disable (SPE and SPD) -- The **Serial Poll Enable (SPE)** message (ASCII **Control X, decimal 24**) causes the 370 to transmit its serial-poll status byte when it is talk addressed. The **Serial Poll Disable (SPD)** message (ASCII **Control Y, decimal 25**) message switches the 370 back to its normal operation.

End or Identify (END or EOI) -- The curve tracer or controller sets **End Or Identify (EOI)** simultaneously with the last byte of the command or data, if **LF** (line feed)/**EOI** is currently selected as the message terminator. If **LF** alone, is selected, the message terminator is the character **LF** accompanied by **EOI** and followed by **CR** (carriage return).

Device-dependent messages

Device-dependent messages for the 370 can be thought of in three categories: Commands and queries, setup data, and measurement data.

Commands and queries -- Commands are sent to the 370 to cause it to take some action or change its settings. Some commands may be as short as three characters. Others, more involved, may be much longer.

A notable subcategory of commands is that of query commands or queries. The curve tracer's response to a query is to send another device-dependent message back to the controller. This response communicates the appropriate instrument status, settings, or measurement data. These messages, too, may vary from just a few characters to many characters in length.

Data: Setups -- Since most front-panel settings are programmable, a series of commands can set the 370 up for a particular measurement. In fact, a group of settings or an entire setup can be strung together and sent as a single message.

Queries can be used, in turn, to send individual settings or complete setups back to the controller. This means you can make a setup manually then store it using the bus controller, providing an alternative to storing the setup in the bubble memory.

Data: Measurements -- Usually the most significant data available from your curve tracer are sets of characteristic curves from devices being tested. For any given measurement these data are packaged in two parts: a preamble and a curve. Preambles and curves can be read from the 370 into the controller either separately or combined as a waveform. Later they can be loaded back into the curve tracer for further work.

The preamble contains the information needed for interpreting, scaling, and labeling the numeric information in the curve. This preamble is coded in ASCII characters and is human readable.

The curve is a series of binary-coded numbers expressing the X-Y coordinates of the 1024 digitized points representing the displayed curves. These binary-coded numbers are not easily read by humans so the controller will have to be used to interpret them.

Two other kinds of data that can be read out of the curve tracer are the coordinates of a cursor located on the curves and the text shown in the text-display area of the screen.

Section 3 -- 370 and GPIB setup

The first steps in putting your 370 to work in your system are to set its bus address, choose its message terminator, connect it to the system with a GPIB cable, and turn it on.

SWITCH SETTINGS

Address

Each instrument connected to the bus must have a unique address. This address is used by the controller to direct the flow of data to and from that specific device.

When choosing a bus address for the instrument, keep a few things in mind. First, the address must be unique on the bus. Second, some controllers reserve an address for themselves. For instance the Tektronix 4041 System Controller has a default address of 30, though that can be changed under program control. Also, setting any device to address 31 effectively removes it from the bus -- the device cannot be addressed.

The 370 uses primary addressing only. Sending a secondary address will have no effect.

The address setting can be verified or changed by examining or setting the address switches located on the rear of the 370. See figure 4.

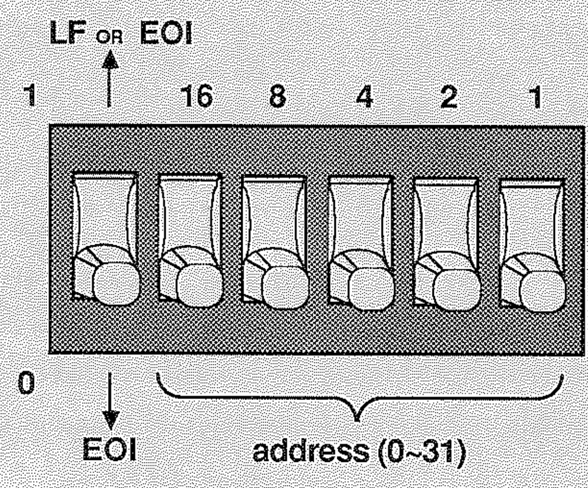


Figure 4. Rear-panel configuration-switch bank.

Message terminator

The terminator is used to indicate the end of a message transfer. The two most common terminators are the **EOI** (end or identify) signal line and the character **LF** (line feed). If **EOI** is selected, the 370 will assert the **EOI** line simultaneaously with the last data byte when sending a message and will recognize the **EOI** line as the terminator when receiving a message. If **LF** is selected, a **CR** (carriage return) and **LF** (line feed) are sent following the last data byte. The **EOI** line is asserted simultaneously with the **LF.** When inputting a message, the 370 will terminate the message upon receiving either the **LF** character or the **EOI** line being asserted.

The best way to determine which terminator to use is to look at the specifications for your controller, and match the terminator recognized by it. For the Tektronix 4041 and the HP 200/300 Series, that is the EOI terminator.

The desired choice of terminator can be set using the remaining switch on the rear-panel configuration-switch bank. See figure 5.

CABLING

Attach the 370 to the GPIB using a standard GPIB cable. A GPIB system may be cabled in two general configurations: star or line. While the star is recommended, these configurations can be mixed as long as the total cable length does not exceed 20 meters and the instruments are distributed on the bus according to a few rules. See figure 5.

First, no more than 15 total devices, including the controller, can be included on a single bus. In addition, to maintain proper electrical characteristics, one device load must be connected for every two meters of cable. Generally, each instrument represents one device load on the bus. The 370 represents one device load.

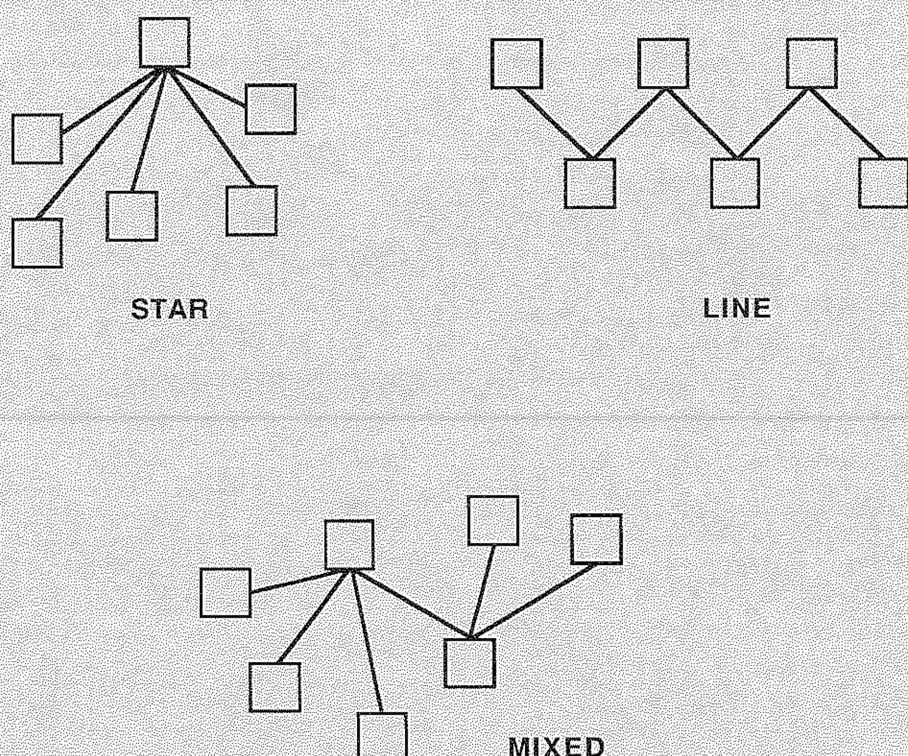


Figure 5. Bus cabling configuration.

POWER UP

Self test

With the 370 cabled and the address and message terminator set, you're ready to power up the system. Keep in mind that when powering up a system with several GPIB instruments on the bus, at least half of them must be powered up before the controller is brought "on-line".

To turn on the 370 press the front-panel power switch. The 370 performs a self-test on power-up, and initializes itself to a predefined state, ready to make measurements. For details of the power-up test see the 370 Operators Manual. The predefined state for the 370 is the same as for the INIt command described later in Commands Table 9.

Power-up SRQ

When the power-up tests have been completed, the 370 asserts the GPIB line called **SRQ.** In the interface, the status byte is set to reflect either a normal power up (65) or the code for the type of failure in the power up test. See the error indications listed in the 370 Operators Manual. If the controller's SRQ interrupt is disabled, the power up SRQ can be ignored. However, the interrupt can only be cleared by performing a serial poll. Handling service requests or interrupts is discussed later. In normal system operation, you will probably want to poll the instruments and check the power up status to see that the instruments powered up normally.

Section 4 -- Controlling the 370 over the bus

SENDING COMMANDS TO THE CURVE TRACER

Most GPIB system controllers and their languages provide high level statements allowing you to send device-dependent messages, such as commands, to any system instrument, in this case the curve tracer. These statements usually consist of three parts:

1. A key word (PRINT, OUTPUT, WRITE, etc.), which causes the action (sending the message over the bus) to occur.
2. An address or logical unit number, which directs the message to a specific instrument.
3. The device-dependent message, which is the actual command, query, or data, to be recognized by the instrument. (Most controllers delimit the device-dependent message with double quotes.)

The following examples show command strings for three controllers and a representative language for each. The first is an IBM PC with a Tektronix GURU card (or National type 2A or 2B card) running BASICA. The second is a Tektronix 4041 Instrument Controller running 4041 BASIC. The third is a Hewlett Packard 200 or 300 Series Scientific Computer running Series 200 or 300 BASIC.

In these examples the 370 command used is the one which sets the cross-hair cursor to mid-screen. In the second and third examples, the GPIB address of the 370 is assumed to be 8.

IBM PC: (The address is assigned elsewhere in the code.)

```
220 WRT$="CRO 500,500"        ' Create command string.
230 CALL IBWRT(DSO%,WRT$)     ' Send command.
```

Tek 4041:

```
220 Print #8:"CRO 500,500"    ! Send command to address 8.
```

HP 200/300 SERIES:

```
220 OUTPUT 708;"CRO 500,500",END
```

A useful addressing variation assigns the 370 address to a variable and uses that variable in place of the specific numeric address. In the following examples, the address is set to 6. This method works for all three of our example controllers when running BASIC and allows you to change the destination of several commands by changing only the value of one variable. This is the scheme used in the IBM PC with GURU card, running BASICA. See the complete IBM example in the back of the guide.

Tek 4041:

```
720 Addr=6
730 Print #addr:"CRO 500,500"
```

HP 200/300 SERIES:

```
720 DEVICE=706
730 OUTPUT DEVICE;"CRO 500,500",END
    or
720 ASSIGN @DEVICE TO 706
730 OUTPUT @DEVICE;"CRO 500,500",END
```

A third way of addressing the instrument, with further advantages, is to use a **logical unit number (LUN).** Following is an example using an OPEN statement in 4041 BASIC to set up a LUN associated with a particular 370 GPIB address instead of using a variable. Logical unit numbers can conserve programming time, since it is only necessary to specify a list of parameters (a stream specification) once. From then on, in the program, you only need to refer to the logical unit number. In the following example, the LUN is 0 and the instrument address is 2. Refer to the 4041 Programmer's Reference Manual for more information on logical units. Here the instrument bus address is 2.

Tek 4041:

```
120 Open #0:"GPIB0(pri=2):"
      .
      .
400 Print #0:"CRO 500,500"
```

In this example, note that any of the 370 commands listed in **Commands Tables A** through **J,** may be substituted for what's inside the quotation marks on line 400. Some command strings will be as short as three characters. More elaborate ones can be 250 characters, or even more.

The following examples show how the syntax of several different controllers can vary. In these examples, the 370 is LUN 10. Once you understand the input and output statements of your controller, just plug in the appropriate 370 commands.

CONTROLLER LANGUAGE	OUTPUT COMMAND
IBM PC with BASICA	WRT$="string" CALL IBWRT(DEV%,WRT$)
Tek 4041 BASIC	Print #10:"string"
HP 200/300 SERIES BASIC	OUTPUT 710;"string",END
HP 9825/200-SERIES HPL	wrt 710,"string"

SENDING QUERIES AND RECEIVING RESPONSES

370 commands with a question mark (**?**) following the header are query commands which solicit information from the curve tracer. After the controller sends a query command, it must acquire the resulting response from the curve tracer. Examples using the **HOR**iz**?** query command follow.

IBM PC: (See the full example at the end of the guide.)

```
150 WRT$="HOR?"
160 CALL IBWRT(DSO%,WRT$)          ' Send query.
170 RD$=SPACE$(100)
180 CALL IBRD(DSO%,RD$)            ' Input response.
```

Tek 4041:

```
150 Dim set$ to 100
160 Input #10 prompt "HOR?":set$  ! Query LUN 10 and input response.
```

HP 200/300 SERIES:

```
150 DIM SET$(100)
160 OUTPUT 710;"HOR?",END
170 ENTER 710;SET$
```

In these examples, a string is dimensioned to 100 characters in order to store the incoming information. The controller sends HOR? over the bus to the 370 curve tracer located at primary address 10. The controller then assigns the instrument at address 10 to be a talker and inputs the characters into the target variable, SET$. The following shows a possible response, a 29 character string. The variable, SET$, now contains this string of characters showing the current status of the horizontal controls.

HORIZ COLLECT:1E-3,OFFSET:0.0

Most commands have a corresponding query command. See **Commands Tables A** through **J** later in this guide.

The following list shows how query responses are input from a variety of controllers.

CONTROLLER LANGUAGE	INPUT COMMAND
IBM PC with BASICA	CALL IBRD$(DSO%,RD$)
Tek 4041 BASIC	Input #10:s$
HP 200/300 SERIES BASIC	INPUT 710;S$,END
HP 9825/200-SERIES HPL	red 710,S$

Section 5 -- Instrument setup over the bus

One popular use of the GPIB with the 370 curve tracer is to store front-panel setups on a storage medium for later recall. These are then used in setting up the curve tracer for repeating specific tests. This is accomplished by using a query command to acquire an ASCII string representing the front-panel setup, from the 370. This string is saved by the controller. Later, this same setup can be restored by sending the stored string back over the bus to the instrument.

STORING SETUPS

To bring the 370 front-panel setup data into the controller use the **SET?** query and input the response into a string variable or variables that can hold up to 400 characters. From there it can be stored on any medium available to the controller, such as magnetic tape or disk.

IBM PC: (BASICA is limited to a 255 character maximum string so the IBRD command will have to be executed twice to take in the full possible extent of the settings string.)

```
400 WRT$="SET?"                   ' Set up query command string.
410 RD$=SPACE$(200)               ' Assign space for response string.
420 CALL IBWRT(DSO%,WRT$)         ' Send query.
430 CALL IBRD(DSO%,RD$)           ' Input response.
```

4041:

```
400 Dim set$ to 400                ! Dimension string variable.
410 Input #8 prompt "SET?":set$    ! Input current setup into set$.
```

HP 200/300 SERIES:

```
400 DIM SET$(400)
410 OUTPUT 710;"SET?",END
420 ENTER 710;SET?
```

A settings string that is being handled by the controller can, of course be modified. Use the appropriate string manipulation commands in the controller language to search for, modify, or replace the needed parts of the string. Just be sure that the resulting string meets the syntax requirements of the 16 individual commands making up the whole and that they are in the proper sequence.

To save the current setup in the bubble memory choose an appropriate setup storage location number then send the command **SAVe <NR1>**, where the number, **<NR1>**, identifies that setup storage location.

LOADING SETUPS

To reverse the above process for setups stored by the controller, simply take the appropriate settings string from wherever it has been stored and send that string itself to the 370. There is no need for any preface or other command since the settings string is made up of the very commands that are needed. When it is necessary to break the settings string, as in BASICA, the break should occur where a semicolon would appear and that semicolon should be dropped. The following examples use the settings strings that were saved in the preceding set of examples.

IBM PC:

```
700 CALL IBWRT(DEV%,SETA$)
710 CALL IBWRT(DEV%,SETB$)
```

Tek 4041:

```
700 Print #10;set$
```

HP 200/300 Series:

```
700 OUTPUT 710;SET$,END
```

If the settings were stored in the bubble memory you need to remember which of the 16 bubble-memory locations was used. Then send the command **RECall <NR1>,** where the number, <NR1>, identifies that setup storage location.

While these procedures can make setups quite quickly and without error, there remain a few manual settings the operator may have to make. One of these is the collector supply High-Low Switch, if it is to be switched into or out of the 2000 volt range. Two others are the Left-Right-Standby Switch and the Cover. One way to remind the operator what needs to be done is to send a message using the text display area of the 370 screen. Send an appropriate message of up to 24 characters using the **TEXt <string>** command.

Section 6 -- Data storage and transfer

OVERVIEW

Once the 370 has been set up to make a desired measurement, the next step in realizing the advantages of digital storage is to **acquire** the data with the 370 and **store** it in the bubble memory. This in turn allows you to **plot** it, if you have an attached plotter, or **copy** it into the controller. From the controller you can **store** it on other media, **compare** it with other data, **calculate** with it, or **display** it. See the diagram later in this section, showing the possible routes for data transfers.

In most cases, acquiring data with the 370 involves two parts, both done automatically by the instrument. The first is to code the necessary scale factors and other parameters, into a series of words and numbers. The words are English but are standardized so they can also be read by a computer. This part of the data is called the **preamble.**

The second part is to digitize and code the displayed curves. This converts the curves to a series of binary numbers representing the horizontal and vertical locations of a sequence of points along the curves. This part of the data is called the **curve.** Together with the preamble it becomes a complete **waveform**.

Acquiring data within the 370

To acquire data, set the display function to store mode. The command to do this over the bus is **DIS**play **STO**re. At this point the current set of parameters are recorded and the curves being shown are digitized. They can now be copied over the bus to the controller or to an attached plotter.

Note, the **ACQ**uire command does not cause an acquisition as such. Think of it as a key word used to set some parameters dealing with the manner in which the acquisition is to be done.

Data structure

The preamble and curve are each a string of eight-bit bytes. The **preamble** is a string of ASCII characters: letters, numerals, and punctuation. Each character is represented by one byte. The major part of a **curve** is a sequence of binary-coded numbers, which is prefixed by a 25 character ASCII string identifying the curve.

In keeping with the Tektronix Codes and Formats standard the word curve refers to the set of 1024 data points representing a measurement. In this section of the guide we distinguish a set of data (string of numbers) from the family of characteristic curves which it represents by qualifying the characteristic curves as **members** or a **family**. Thus, the word **curve**, by itself, refers to a data set representing a number of **members** making up a **family** of characteristic curves.

In a family of curves there can be as many as 11 individual member curves, each representing a separate step or sweep. The set of 1024 data points contains nearly equal numbers of points for each individual member curve in the family. Thus there will be at least 93 points for each individual member curve in the family. The sweeps are done with a half-sine waveform, while the points are digitized at uniform time intervals. This results in the data points being closer together near the extreme end of each individual member curve.

Each of the 1024 data points is represented by two ten-bit numbers, a horizontal coordinate and a vertical coordinate. Each number, in turn, is coded into two bytes in the data string. Thus it takes four bytes to represent each data point and the binary part of a full **curve** will take 4096 consecutive bytes plus two bytes at the start for a byte count and one byte at the end for a checksum value, or 4099 bytes total.

The numbers are coded in two's-complement binary format. The low-order eight bits are stored in the second byte while the two high-order bits are stored in the low-order positions of the first byte and the sign bit fills the remaining bits in the high-order byte.

MOVING DATA TO THE CONTROLLER

To move data from the 370 to the controller requires that a bubble-memory cartridge, unlocked (not write-protected), be in place in the 370. The cartridge is unlocked by moving the plastic tab to uncover the window in the cartridge body.

Preamble

To take in a waveform preamble from the 370, send the command query **WFM**pre? The response will be a string containing 17 parameters, each shown as a label and value pair. See Commands Table G for details. Follow the command query with an appropriate statement to input the response, allowing for a string about 410 characters long.

Tek 4041:

```
300 Dim pream$(410)
310 Input #8 prompt "WFM?":pream$
```

HP 200/300 Series:

```
300 DIM PREAM$(410)
310 OUTPUT 718;"WFM?",END
320 ENTER 718;PREAM$
```

Curve

Curves transferred from the 370 to the controller are copied from the bubble memory. The bubble-memory location used is the same as for the last preamble transfer. Before a given curve can be transferred the corresponding preamble must have been transferred.

To copy the data then, first, send a **DIS**play **VIE**w:**<NR1>** command to set the desired location index. Then, send a **WFM**pre? query to transfer the preamble from that location. Last, send a **CUR**ve? query to transfer the curve from that location. Of course the last two steps can be combined by sending a **WAV**eform? query to transfer both the preamble and waveform from the indicated location.

For the curve data, in general, you will need to provide for a string of about 4125 bytes. The response will be a short (25 bytes or characters) ASCII string identifying the curve, two bytes giving the number (in binary) of data points to follow, the 4096 bytes representing the data proper, and a checksum (one binary byte).

Waveform

One other command, **WAV**frm?, combines the functions of **WFM**pre? and **CUR**ve? and returns the whole waveform: preamble plus curve, with an ASCII semicolon between them.

Another possibility for storing data under control of the bus is to put the waveform into a bubble-memory waveform-storage location. Use the **ENT**er **<NR1>** command, where the number, **<NR1>**, is the storage location index.

LOADING DATA FROM THE CONTROLLER

To move data from the controller to the 370 requires that an unlocked bubble-memory cartridge be in place.

Preamble

The command **WFM**pre **<string>**, where **<string>** is a waveform preamble, will load that preamble into the 370. The preamble will be stored in the bubble-memory location indicated by the WFID INDEX.

Curve

When sending a waveform to the 370, the target is the bubble-memory location set by the preceding preamble transfer.

To send a curve from the controller to the 370 use the command, **CUR**ve **<string>**, where **<string>** consists of a short ASCII string (CURVID . . .) followed by a string of binary bytes. It is made up as shown in **Commands Table G.**

OTHER TYPES OF DATA

Although waveforms are the principal data developed with the 370, there are two other types of data: cursor readout and text.

Cursor readout

With the dot or cross-hair cursor located at a desired point in the display you can determine that point in terms of display units. Request the data with the **REA**dout? command. The response is the word **READOUT** followed by a pair of numbers giving the cursor location in terms of physical units: horizontal location in volts and vertical location in amperes. The window cursor can be used similarly but the readout value depends on the last manual setting as to whether it comes from the upper-right or lower-left corner of the rectangular window.

Text

Although it's not data in the same sense, there may be meaningful information asssociated with a display in the text area. This can be read over the bus with the **TEX**t? query command. Of course similar data can be added to a display with the **TEX**t command.

SUMMARY

There are then, several kinds of data and they differ in content and format. There are a number of ways to move those various kinds of data from place to place. The following diagram shows the possibilities.

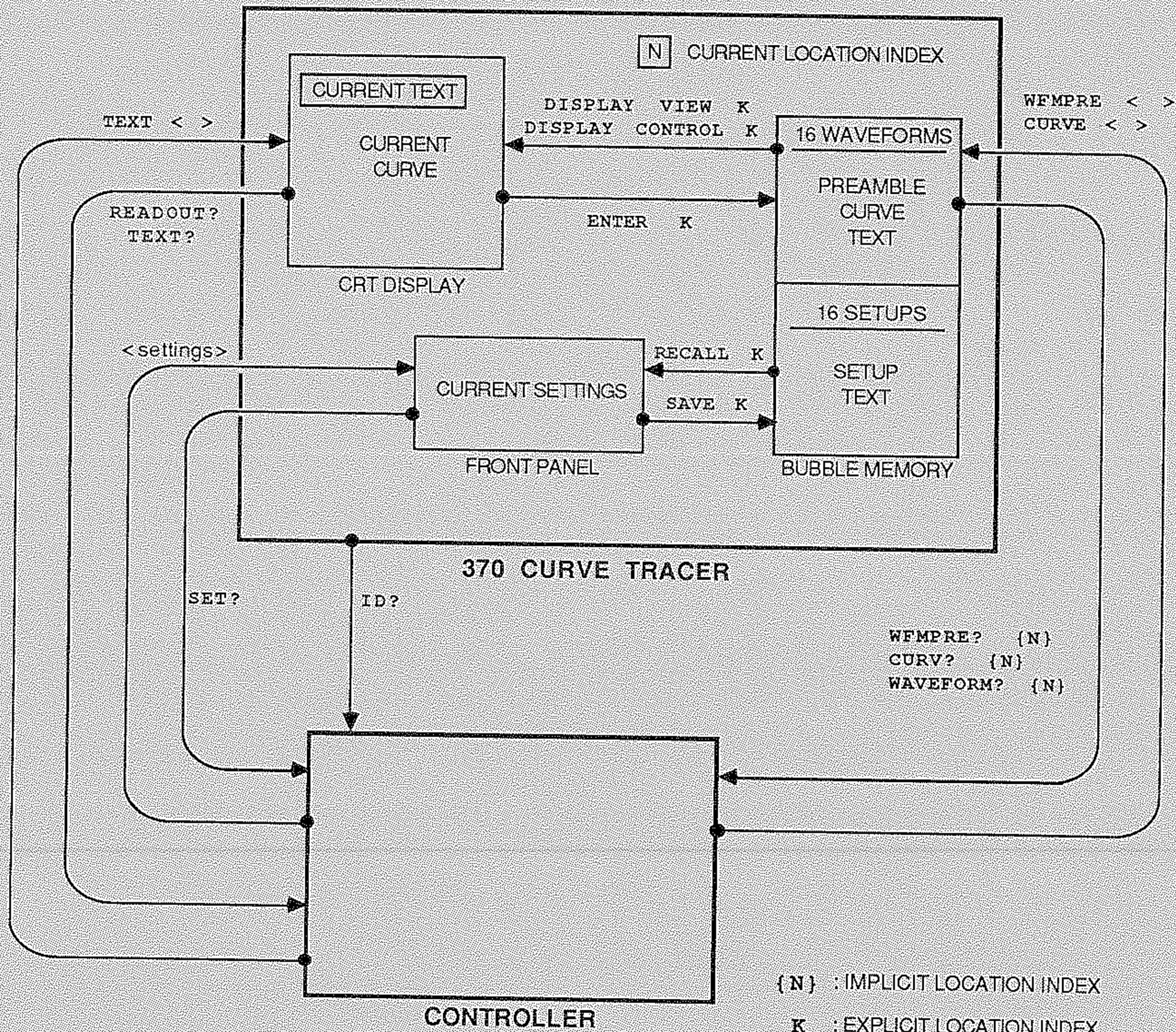


Figure 6. Data Transfers Under GPIB Control.

Section 7 -- Device-dependent-message formats

Device-dependent messages travel both ways between the instrument and controller. They set instrument controls; request and return the instrument status; request, return, and send waveforms; and request and return results of other measurements. Most messages are sent as strings of ASCII characters. However, data representing sets of curves are sent as series of binary-coded numbers.

Commands are sent from the controller to the instrument. Each starts with a key word called a **header** which is usually followed by various possible **arguments** to further detail the instruction. The key words used for command headers and arguments are mnemonics related to specific instrument controls and functions. Multiple commands can be sent in one message. Many command headers can be issued with an attached question mark (?), turning them into **queries** which prompt the instrument for certain information.

Responses to queries either contain information about instrument status or they contain measurement data. Generally status messages are made up of the same key words used for commands. The only device-dependent messages not readily read by persons are those made up of curve data.

Each message ends with the message terminator, which was discussed earlier. In most cases the controller or its language takes care of the message terminator and you need not be concerned with it once it has been chosen and set.

COMMAND MESSAGES

Commands for the 370 Curve Tracer, like those for other Tektronix GPIB instruments, follow the conventions in the Tektronix **Standard Codes and Formats**. Each command starts with a key-word header which is often followed by one or more key-word arguments to further specify the action to be taken. The key words for the commands were chosen to be as understandable as possible, while still allowing a familiar user to shorten most of them to only a few characters. Syntax is also standardized to make the commands easier to learn.

In most of this guide key words for headers and arguments are listed in a combination of uppercase and lowercase letters. The instrument accepts any abbreviated header or argument containing at least all the characters shown in uppercase. We show them as uppercase for emphasis only. The 370 ignores case. Any characters added to the abbreviated (uppercase) version must be those shown in lowercase and in the order shown. In this section only, the commands are expressed in a variety of valid ways, to illustrate the flexibility possible.

The following are all valid versions of the **INIt** command, which resets the 370 to the initial state following power-up.

```
INI
INIT
Ini
Init
ini
init
inIT
```

Commands Tables A through **J** describe all the 370 commands and queries. The first column lists the header key-word. The second and third columns list arguments that may be associated with the command. Brief descriptions are shown in the fourth column along with examples.

Headers

Each command consists of at least a header.

```
<header>
```

Each command header is unique and in some cases is all that is needed to invoke the command.

```
INIt
```

Note: **CURS**or can be shortened only to four characters. The three characters **CUR** will be interpreted as **CURVE**.

Arguments

Many commands require the addition of arguments to the header to describe exactly what is to be done. If there is more to the command than just the header, the header must be followed by at least one space. Otherwise, the 370 treats all spaces, linefeeds, and tabs as "white space" and ignores them in analyzing messages from the controller.

```
<header> <argument>
```

In some cases, the argument is simply a single word or a number.

```
CURSOR OFF
rqs on
Pkvolt 16
```

Linked arguments

In other cases the argument itself requires another argument. When an argument to an argument is required, a colon links the two, hence the second is called a **linked argument.**

```
<header> <argument>:<linked argument>
```

```
DISPLAY INVERT:ON
Horiz step:0.5
ACQ AVG:32
acquire avg:4
```

Queries

For most commands there is a corresponding query formed by adding a question mark to the header key word. Do not put a space between the last character of the key word and the question mark.

Queries for the 370 need only the header and question mark, though the response will usually be more involved.

```
<header>?

ID?
horiz?
CONFIG?
Stpgen?
```

Query: `STP?`

Typical Response: `STPGEN NUMBER: 5,PULSE:OFF,OFFSET: 0.00, INVERT OFF,MULT:OFF,CLIMIT:0.02,CURRENT:50.0E-9`

Multiple arguments

Where a header has multiple arguments, the successive arguments (or argument pairs if the arguments have linked arguments) must be separated by commas.

```
<header> <first arg>:<link arg>,<second arg>:<link arg>


Vert  collect:0.05,offset:-0.1
STP  CUR:2E-6,MUL:ON,NUM:5
dis  vie:16,inv:on
```

Numeric-argument formats

Many commands have numeric arguments. The numbers are decimal (base 10) values. They are expressed in three different formats, denoted <NR1>, **<NR2>** and **<NR3>**.

Symbol	Number Format	Examples
<NR1>	Integer	+1, 2, -1, -10, 0
<NR2>	Explicit Decimal Point	-3.2, +5.1, 1.2, .0, 0.
<NR3>	Floating Point, Exponential, or Scientific Notation	-12.3E-2, .005e-6, 0.000E-3 6.7E+4, 2.35E-3, 0.e0, 125E-6

Generally, an **<NR1>** argument must be sent to the 370 in that format (ie., without decimal point). An **<NR2>** argument may be sent to the 370 in either **<NR2>** or **<NR1>** format. Similarly, an **<NR3>** argument may be sent in **<NR3>**, **<NR2>** or **<NR1>** format.

Command	Valid Forms
RECall <NR1>	REC 12
DISplay VIEw:<NR1>	DIS VIE:7
VERt OFFset:<NR2>	VERT OFFSET:2 ver off:-1.5
VERt COLlect:<NR3>	ver col:2 VER COL:0.5 vert collect:1.5E-2

Multiple-command messages

You may put multiple commands into one message by separating individual commands with semicolons.

```
<first command>;<second command>;<third command>

PKPOWER 10;CSPOL PNORMAL;MEASURE SINGLE
pkpow 10;cspol pnorm;meas single
PKP 10;CSP PNO;MEA SIN
CURSOR OFF;MAG OFF;HORIZ OFFSET:2;ACQUIRE NORMAL;
STPGEN NUMBER;3
```

With multiple commands in the message the message terminator is needed only once, at the end of the message. Again, most controllers and their languages take care of this for you.

OTHER MESSAGES

Besides receiving commands and queries the 370 can receive data and send responses to the queries. The latter can be quite short (a word and a number) or fairly long (a full set of panel settings). Measurement information can likewise be short (a word and a number) or lengthy (a full waveform).

ASCII strings

As mentioned before, the only device-dependent messages not sent as ASCII strings are the binary data used for curves. All other messages, both to and from the 370, are ASCII strings made up of numbers or key words pertaining to the parameters of interest.

Key-word messages -- An example, key-word exchange resulting in a simple response is the following.

Query:

```
PST?
```

Response:

```
PSTATUS  BUSY
```

Another example follows, this one resulting in a more lengthy response.

Query:

```
STP?
```

Response:

```
STPGEN  NUMBER:5,PULSE:OFF,OFFSET:-1.5,INVERT:OFF,
MULT:OFF,CLIMIT:.1,VOLTAGE:2.0E-3
```

Number messages -- Numbers other than those representing waveforms are sent as strings of ASCII characters. With some controllers you may have to explicitly convert these to numeric values in order to use them in calculations. Other controllers or languages may provide a more direct conversion, as the Tektronix 4041 does.

Suppose we are using a 4041 for our controller and we have the following case.

Query:
```
AUX?
```
Response:
```
AUX -7.38
```

The 4041 can handle this either of two ways.

The first method inputs the whole, nine character response into a string variable, **aux$.** To extract the numeric part would require use of the **SEG$** function and to convert that to a numeric value would require use of the **VAL** function.

```
210 Input #8:prompt "AUX?":aux$
```

The second method inputs the numeric value directly into an implicit, short floating point variable, **auxlev.**

```
210 Input #8:prompt "AUX?":auxlev
```

This requires no additional manipulation.

Preambles

Preambles are necessary to interpret the numeric information in the following curve data. Within a preamble, 25 parameters are specified. The first 9 are unique to the 370 curve tracer and are included as a substring linked to the **WFID**: label. The other 16 include ten that have fixed values and six that vary with the particular data sent.

Within the **WFID**: substring the parameters are separated by slashes, while the entire substring is delimited by a pair of double quote marks. Most of the **WFID**: string is rather strictly defined, with each parameter value being right justified in a fixed length field. An exception is the BGM value which may vary in field length.

The remainder of the preamble uses standard punctuation. A colon links each parameter label with its corresponding value and the individual label and value pairs are separated with commas.

A complete preamble might look like this.

```
WFMPRE WFID:"INDEX  16/VERT  .001/HORIZ  1.0/STEP  .002/OFFSET  -.5/B
GM  500/AUX  0.0/ACQAVG/TEXT"Sample 14A, Oct 17,1986",ENCDG:BIN,
NR.PT:1024,PT.FMT:XY,XMULT:1.0,XZERO:0,XOFF:    0.0,XUNIT:V,YMULT:1.0,
YZERO:0,YOFF:0.0,YUNIT:A,BYT/NR:2,BN.FMT  :RP,BIT/NT:10,CRVCHK:
CHKSM0,LN.FMT:VECTOR
```

Curves

Curve data sets are usually much longer than any other kind. Most of the time a set of curve data will be about 4122 bytes long, with most of the bytes being binary-coded numbers. Thus, most of the string of data is not directly readable, but must be interpreted by the controller.

An example might look like this.

```
CURVE CURVID:"INDEX    9",%NNXXYYXXYY  .  .  .  XXYYC
```

This breaks down as follows. It starts with an ASCII string of 25 characters.

```
CURVE CURVID:"INDEX    9",%
```

This is followed by a series of binary bytes. The first of these is two bytes giving the number of data bytes to follow, plus one (typically 4097).

```
NN
```

Then come the 4096 data bytes. Each of the 1024 data points on the curve is represented by four bytes, 2 for the 10 bits of the X coordinate and 2 for the 10 bits of the Y coordinate.

```
XXYYXXYY  .  .  .  XXYY
```

And finally there is one byte which is the checksum for the preceding 4098 data bytes.

Section 8 -- Scope of programmability

Having an instrument programmable over the bus doesn't make it possible to do everything from the controller. It is still necessary to insert and remove the devices being tested. Safety too, is important. The high-voltage supply and protective cover are also manually controlled.

Beyond that most of the functions not requiring observation of the display or access to the instrument can be controlled remotely.

FRONT-PANEL FUNCTIONS THAT CAN BE BOTH SET AND QUERIED

The following table lists functions that can be set and also can be queried over the bus.

Table 2.

Category	Function	Command & Table
Measurement setup		
Test circuit	Configuration	CON, E
Collector supply	Maximum peak voltage	PKV, C
	Maximum peak power	PKP, C
	Variable voltage	VCS, C
	Polarity	CSP, C
Step generator	Step type (V or I)	STP, D
	Step amplitude	
	Offset	
	Limit	
	0.1x multiplier	
	Invert	
	Number of steps	
	Pulse	
	Pulse width	
Auxiliary supply	Level	AUX, F
Measurement	Repeat, single	MEA, F

Table 2 (Cont'd).

Category	Function	Command & Table
Data acquisition and display		
Display	Non-store, store, view, or compare	DIS, A
	Acquisition mode	ACQ, A
	Horizontal source	HOR, A
	Horizontal sensitivity	
	Vertical source	VER, A
	Vertical sensitivity	
	Invert	DIS, A
	Magnifier mode	MAG, A
	Magnifier offset	
	CRT calibration	DIS, A
Data output		
Bubble memory	Enter	ENT, A
	Save	SAV, F
	Recall	REC, F
Plotter	Start	PLO, F
	Status	PST, F
Cursors	Off	CURS, B
	Dot	DOT, B
	Cross-hair	CRO, B
	Window	WIN, B

LIMITATIONS AND PARTIAL PROGRAMMABILITY

The following table summarizes those functions for which there is only limited access from the bus.

Table 3.
Functions not completely programmable

Control	Set from bub mem	Set from bus	Query from bus	Manual action
Left-right-standby switch	no	no	LRS?	control
Cover	no	no	COV?	control
Collector supply				
Peak voltage: 16 to 400	yes *	PKV *	PKV?	control
Peak voltage: 2000	no *	no *	PKV?	control
High-low switch	no	no	HIL?	control
Display mode	no	DIS	DIS?	control
Readout: cursor loc (V & A)	no	no	REA?	observe
Memory				
Enter, wfm	no	ENT	no	control
Recall, setup	no	REC	no	control
Save, setup	no	SAV	no	control
Index, wfm	no	DIS	no	observe
Index, setup	no	SAV	no	observe
GPIB				
Initiate SRQ	no	NA	poll	control
Reset to local	no	GTL	NA	control
Plotter				
Initiate plot	no	PLO	NA	control
Report status	no	NA	PST?	observe
Initiate self test	no	TES	NA	control
Report identification	no	no	ID?	control
Show help list	no	no	HEL?	none
Report event status	no	no	EVE?	observe

*** Switching the collector supply to and from the 2 kV range is manual only.**

Section 9 -- Commands Tables

Commands Table A
DISPLAY COMMANDS

Command	Argument	Link Arg	Definition
ACQuire	ENVelope:	VERt	Set the acquisition mode.
		HORiz	ACQuire ENVelope: VERt
	NORmal		ACQuire NORmal
	AVG:	4	ACQuire AVG: 32
		32	
ACQuire?			Respond with the current acquisition mode. ACQUIRE ENVELOPE: <axis>, ACQUIRE NORMAL, or ACQUIRE AVG: <val> where <axis> = VERT or HORIZ and <val> = 4 or 32

Commands Table A (Cont'd)
DISPLAY COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
DISplay	NSTore		Select the NON-STORE mode.
	STOre		Select the STORE mode.
	VIEw:	<NR1>	Select the VIEW mode. DISplay VIEw:<index> where: <index> = 1, 2, . . . , 16.
	COMpare:	<NR1>	Select the COMPARE mode. DISplay COMpare:<index> where: <index> = 1, 2, . . . , 16.
	INVert:	ON OFF	Set the display invert mode. DISplay INVert: OFF
	CRTcal:	ZERochk OFF CALchk	Set the CRT check mode. DISplay CRTcal: ZERochk
DISplay?			Respond with current display information. DISPLAY <mode1>, <mode2>, <mode3> where: <mode1> = NSTORE, STORE, VIEW: <NR1>, or COMPARE: <NR1>, <mode2> = INVERT: OFF or INVERT: ON, and <mode3> = CRTCAL: ZEROCHK, CRTCAL: OFF, or CRTCAL: CALCHK

Commands Table A (Cont'd)
DISPLAY COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
ENTer	<NR1>		Store the displayed curve data in the bubble-memory curve-location specified. This is meaningful only when already in store mode. **ENT**er **<index>** where: **<index>** = **1, 2, 3, . . . ,** or **16**.
HORiz	**STE**p		Select the horizontal-display source and sensitivity (volt/div). **HOR**iz **<source>:<volt>** When **<source>** is **COL**lect, **<volt>** may be **5.0E-2** to **5.0E+2**. When **<source>** is **BAS**e, **<volt>** may be **5.0E-2** to **2.0E+0**.
	COLlect:	<NR3>	
	BASe:	<NR3>	
	OFFset:	<NR2>	Set the horizontal-display offset (div). **HORI**z **OFF**set:**<val>** where: **<val>** = **-10.0** to **+10.0** in steps of **0.5**
HORiz?			Respond with the horizontal-display source, sensitivity (volt/div), and offset. **HORIZ <source>: <volt>, OFFSET:<val>** where: **<source>** = **STEP, COLLECT,** or **BASE,** **<volt>** = sensitivity (volt/div), and **<val>** = offset(div).
MAG	**VER**t:	<NR1>	Set the vertical or horizontal display magnifier to **x1** or **x10**. Only one of the two can be set at x10 at any given time. **MAG VER**t: **<val>** where **<val>** = **1** or **10**
	OFF		
	HORiz:	<NR1>	
MAG?			Respond with the display magnifier setting. **MAG <mode>:<val>** where: **<mode>** = **VERT, OFF,** or **HORIZ,** and **<val>** = **1** or **10**.

Commands Table A (Cont'd)
DISPLAY COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
VERt	STEp		
	COLlect:	<NR3>	Set the vertical display source and sensitivity (A/div). **VERt COLlect:<amp>** where: **<amp>** = **1.0E-6** to **2.0E+0** when COLLECTOR POLARITY is **not** in leakage mode, and **<amp>** = **1.0E-9** to **2.0E-3** when COLLECTOR POLARITY **is** in leakage mode.
	OFFset:	<NR2>	Set the vertical display offset (div). **VERt OFF:<val>** where: **<val>** = **-10** to **+10** in increments of **0.5**
VERt?			Respond with the vertical display source, sensitivity, and offset. **VERT STEP, OFFSET: <val2>** **VERT COLLECT: <val1>, OFFSET: <val2>** where **<val1>** = sensitivity (A/div), and **<val2>** = offset (div).

Commands Table B
CURSOR COMMANDS

Command	Argument	Link Arg	Definition

Also note the **REA**dout? query in Commands Table H, for extracting data values from the cursor.

Command	Argument	Link Arg	Definition
CURSor	**OFF**		Turn off cursor. **CURS**or **OFF**
DOT	**<NR1>**		Set the dot cursor on the specified data point in the current curve. **DOT <data>** where **<data>** = **1, 2, 3, ... 1024**
DOT?			Respond with the dot cursor position. **DOT <NR1>**
CROss	**<NR1>,<NR1>** **<NR1>**		Set the cross-hair cursor to the specified position on the CRT. **CRO**ss **<data1>, <data2>** where **<data1>** = **0, 1, 2, 3,** . . . , or **1000,** **horizontal position** **<data2>** = **0, 1, 2, 3,** . . . , or **1000,** **vertical position**
CROss**?**			Respond with the cross-hair cursor position. **CROSS <NR1>, <NR1>**

Commands Table B (Cont'd)
CURSOR COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
WINdow	<NR1>,<NR1>, <NR1>,<NR1>		Set the window cursor to the specified position on the CRT. **WINDOW <data1>, <data2>, <data3>, <data4>** where: <data1> = 0, 1, 2, 3, . . . , or 1000 Lower left horizontal position <data2> = 0, 1, 2, 3, . . . , or 1000 Lower left vertical position <data3> = 0, 1, 2, 3, . . . , or 1000 Upper right horizontal position <data4> = 0, 1, 2, 3, . . . , or 1000 Upper right vertical position
WINdow?			Respond with the window-cursor position. **WINDOW <NR1>, <NR1>, <NR1>, <NR1>**

Commands Table C
COLLECTOR-SUPPLY COMMANDS

Command	Argument	Link Arg	Definition
CSPol	PLEakage PDC PNOrmal AC NNOrmal NDC NLEakage		Select the collector-supply polarity and mode. CSPol <mode> where <mode> = PLE, PDC, PNO, AC, NNO, NDC, or NLE and PLE = +LEAKAGE PDC = +DC PNOr = +NORMAL AC = AC NNOr = -NORMAL NDC = -DC NLE = -LEAKAGE
CSPol?			Respond with the collector-supply polarity and mode. CSPOL <mode> where <mode> = PLEAKAGE, PDC, PNORMAL, AC, NNORMAL, NDC, or NLEAKAGE.
HILowsw?			Respond with the collector HIGH-LOW switch status. HILOWSW <flag> where <flag> = LOW or HIGH

Commands Table C (Cont'd)
COLLECTOR-SUPPLY COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
PKPower	220.0 50.0 10.0 2.0 0.4 0.08		Set the maximum peak power, in watts. **PKPower <set>** where **<set>** = 220.0, 50.0, 10.0, 2.0, 0.4, or 0.08.
PKPower?			Respond with the maximum peak power setting, in watts. **PKPOWER <set>** where **<set>** = 220.0, 50.0, 10.0, 2.0, 0.4, or 0.08.

Commands Table C (Cont'd)
COLLECTOR-SUPPLY COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
PKVolt	16 80 400		Set the maximum peak voltage. The front panel HIGH-LOW switch can override this command. Setting it to 2000 must be done manually. Trying to do so under program control results in event code 204. **PKV**olt **<set>** where **<set>** = **16**, **80**, or **400**.
PKVolt**?**			Respond with the current, maximum peak voltage setting. **PKVOLT <set>** where **<set>** = **16**, **80**, **400**, or **2000**.
VCSpply	**<NR2>**		Set the variable collector-supply. The argument is stated as a percentage. Allowed increments are 0.1%. **VCS**pply **<data>** where **<data>** = **0.0**, **0.1**, . . . , **99.9**, or **100.0**.
VCSpply**?**			Respond with the variable collector-supply setting. **VCSPPLY <data>** **<data>** = **0.0**, **0.1**, . . . , **99.9**, or **100.0**.

Commands Table D
STEP-GENERATOR COMMANDS

Command	Argument	Link Arg	Definition
STPgen	CURrent: VOLtage:	<NR3> <NR3>	Set the step generator to provide current or voltage steps, and set the step size in amperes or volts. STPgen <source>: <val> where <source> = CUR or VOL <val> = 5.0E-8 through 2.0E-1, but not 0, for current step size (ampere/step) <val> = 5.0E-2 through 2.0E+0, but not 0, for voltage step size (volt/step) STPgen CURrent: 1.0E-3
	NUMber:	<NR1>	Number of steps to be generated. STPgen NUMber :<val> where <val> = 0, 1, 2, . . . , 10.
	INVert:	ON OFF	Set the step generator to invert mode. STPgen INVert :<mode> where <mode> = ON or OFF.
	MULt:	ON OFF	Set the step generator to 0.1X mode. STPgen MULt: <mode> where <mode> = ON or OFF.
	PULse:	OFF SHOrt LONg	Disable or enable pulse mode and set the pulse duration. STPgen PULse: <mode> where <mode> = OFF, SHORT, or LONG, and SHORT = 80 microseconds LONG = 300 microseconds.
	CLImit:	<NR2>	Set the step generator current limitation STPgen CLImit: <val> where <val> = 0.02, 0.1, 0.5, or 2.0.
	OFFset:	<NR3>	Set the offset of the step generator. STPgen OFFset: <val> <val> = -10.0, -9.9, . . , -0.0, 0.0, +0.1, . . . , +9.9, or +10.0.

Commands Table E
CONFIGURATION COMMANDS

Command	Argument	Link Arg	Definition
STPgen?			Respond with the step generator source, amps/step or volt/step, number of steps, pulse mode, offset, invert mode, 0.1X mode, and current limitation. **STPGEN <num>, <pulse>, <offset>, <invert>, <mult>, <clim>, <typ:size>** where **<num>** = number of steps **<pulse>** = pulse mode **<offset>** = offset value **<invert>** = invert mode status **<mult>** = 0.1X mode status **<clim>** = current limitation value **<typ:size>** = **CURRENT: size (A/step)** or **VOLTAGE: size (V/step)**.
CONfig	**BSG**en **BOP**en **BSH**ort **ESG**en **EOP**en		Set the test circuit configuration. **CON**fig **<mode>** where **<mode>** = **BSG**, **BOP**, **BSH**, **ESG**, or **EOP** and **BSG** = BASE STEP-GEN & EMITTER COMMON **BOP** = BASE OPEN & EMITTER COMMON **BSH** = BASE SHORT & EMITTER COMMON **ESG** = BASE COMMON & EMITTER STEP-GEN **EOP** = BASE COMMON & EMITTER OPEN
CONfig?			Respond with the configuration **CONFIG <mode>** where **<mode>** = **BSGEN**, **BOPEN**, **BSHORT**, **ESGEN**, or **EOPEN**

Commands Table F
MISCELLANEOUS COMMANDS

Command	Argument	Link Arg	Definition
AUX	<NR2>		Set the AUX output to the voltage specified. **AUX <voltage>** where **<voltage>** = -40.00, -39.98, . . . , 0.00, . . . , +39.98, or +40.00 (0.02 volt steps)
AUX?			Respond with the current AUX output voltage. **AUX <data>** **<data>** = -40.00, . . . , or +40.00.
COVer?			Respond with the protective cover status. **COVER <status>** where **<status> = ON or OFF** and **ON = for cover closed** **OFF = for cover open**
LRSsw?			Respond with the LEFT-RIGHT-STANDBY switch status **LRSSW <status>** where **<status>** = **LEFT,** **RIGHT,** **STANDBY,** or **BOTH.**

Commands Table F (Cont'd)
MISCELLANEOUS COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
MEAsure	REPeat SINgle		Set the measurement mode. MEAsure <mode> where <mode> = REPeat or SINGLE.
MEAsure?			Respond with the current measurement mode. MEASURE <mode> where <mode> = REPEAT or SINGLE.
PLOt	ALL CURve		Select and start the plotter output. PLOt <mode> where <mode> = ALL or CURve and ALL = FULL CUR = CURVE
PSTatus?			Respond with the current plotter status. PSTATUS <status> where <status> = READY or BUSY and READY = idle mode BUSY = busy mode
RECall	<NR1>		Recall the front-panel setup data from the bubble-memory location specified. RECall <index> where <index> = 1, 2, 3, . . . , or 16.
SAVe	<NR1>		Save the current setup in the bubble-memory setup location specified. SAVe <index> where <index> = 1, 2, 3, . . . , or 16

Commands Table G
WAVEFORM TRANSFER COMMANDS

Command	Argument	Link Arg	Definition
CURve	<string>		Load this curve into the display and the specified memory location. CURve <string> where <string> = CURVID:<crvid> % <binary data> where <crvid> = "INDEX <NR1)" <binary data> = <count><first point> . . . . <last point><checksum> where <count> = two bytes indicating the number of data points plus one <point> = two bytes indicating the X coordinate and two bytes indicating the Y coordinate for a point (00 through FF) <checksum> = one byte, the 2's complement of the modulo 256 sum of the preceding binary data
CURve?			Respond with the curve data for the most recent preamble query. CURVE CURVID <curvid>, % <binary data> where <crvid> = "INDEX <NR1)" <binary data> = <count><first point> . . . . <last point><checksum> where <count> = two bytes indicating the number of data points plus one <point> = two bytes indicating the X coordinate and two bytes indicating the Y coordinate for a point (00 through FF) <checksum> = one byte, the 2's complement of the modulo 256 sum of the preceding binary data

Commands Table G (Cont'd)
WAVEFORM TRANSFER COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
WAVfrm?			Respond with both the preamble and curve data for the current waveform. See the discussions of **WFM**pre? and **CUR**ve? for details. The preamble and curve are separated by a semicolon.
WFMpre	<string>		Load this waveform preamble into the memory location indicated by the current contents of the memory index display. WFMPRE <STRING> where <string> = WFID:<wfid>, ENCDG:BIN, NR.PT:<point>, PT.FMT:XY, XMULT:<x multi>, XZERO:0, XOFF:<xoff>, XUNIT:V, YMULT:<y multi>, YZERO:0, YOFF:<yoff>, YUNIT:A, BYT/NR:2, BN.FMT:RP, BIT/NR:10, CRVCHK:CHKSM0, LN.FMT:<format> where <wfid> = "INDEX <num>/ VERT <amp>/HORIZ <volt>/ STEP <step>/ OFFSET <offset>/ BGM <para>/ AUX <aux>/ ACQ <acq>/ TEXT <txt>" where <num> = memory location, <NR1> <amp> = sensitivity, A/div <volt> = sensitivity, V/div <step> = step amplitude, V or A <offset> = step offset, div <para> = beta or GM <aux> = aux supply setting <acq> = acquisition mode (AVG, NOR, or ENV) <txt> = readout of text area <point> = number of points in curve (1 through 1024) <x multi> = horizontal scale factor <x off> = horizontal offset <y multi> = vertical scale factor <y off> = vertical offset <format> = VECTOR or DOT

Commands Table G (Cont'd)
WAVEFORM TRANSFER COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
WFMpre?			Respond with the waveform preamble from the memory location specified. WFMPRE WFID:<wfid>, ENCDG:BIN, NR.PT:<point>, PT.FMT:XY, XMULT:<x multi>, XZERO:0, XOFF:<x off>, XUNIT:V, YMULT:<y multi>, YZERO:0, YOFF:<y off>, YUNIT:A, BYT/NR:2, BN.FMT:RP, BIT/NR:10, CRVCHK:CHKSM0, LN.FMT:<format> where <wfid> = "INDEX <num>/VERT <amp> /HORIZ <volt>/STEP <step> /OFFSET <offset>/BGM para> /AUX <aux>/ACQ <acq> /TEXT <txt>" where <num> = memory location number <amp> = sensitivity, A/div <volt> = sensitivity, V/div <step> = readout of step amplitude <offset> = readout of step offset <para> = readout of beta or GM <aux> = readout of aux supply <acq> = acquisition mode of curve (AVG, NOR, or ENV) <txt> = readout of text area <point> = number of points in curve <x multi> = horizontal scale factor <x off> = horizontal offset <y multi> = vertical scale factor <y off> = vertical offset <format> = VECTOR or DOT

Commands Table H
CRT-READOUT TRANSFER COMMANDS

Command	Argument	Link Arg	Definition
REAdout?			Respond with the parameter readout from the current cursor location. Window-cursor values are for the upper-right or lower-left corner, depending on the last, manual, front-panel cursor-control setting. **READOUT <xread>, <yread>** where **<xread> = horizontal reading in volts** **<yread> = vertical reading in amperes** If the cursor is off screen the values will have question marks before them and are not usable.
TEXt	<string>		Display this text in the CRT text area. **TEX**t "**<text>**" where **<text> = A message no longer than 24 characters**
TEXt?			Respond with the text currently displayed in the CRT text area. **TEXT "<text>"** where **<text> = The message from the CRT text area, no longer than 24 characters.** Note: While this text may be stored in bubble memory along with the settings, it can be sent over the bus only with this query.

Commands Table I
INSTRUMENT PARAMETER COMMANDS

Command	Argument	Link Arg	Definition
HELp?			Respond with a list of all valid command and query headers. CONFIG, READOUT, TEXT, CROSS, DOT, WINDOW, CURSOR, DISPLAY, ACQUIRE, MAG, HORIZ, VERT, STEPGEN, MEASURE, ENTER, RECALL, SAVE, PLOT, PSTATUS, HILOWSW, LRSSW, COVER, AUX, PKVOLT, PKPOWER, CSPOL, VCSPPLY, WFMPRE, CURVE, WAVFRM, RQS, OPC, EVENT, TEST, INIT, ID, SET
ID?			Respond with the 370's ID: ID SONY_TEK/370, V<nbr1>, F<nbr2> where <nbr1> = current model number <nbr2> = current firmware version

Commands Table I (Cont'd)
INSTRUMENT PARAMETER COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
INIt			Reset the instrument. Settings are to be the same as at power-up and as shown below.

Function	INIt Value
DISplay	STORE
CURsor	OFF
DISplay CRT:	OFF
DISplay INV:	OFF
HORiz OFFset:	0.0
STP CUR:	50.0E-9
STP OFF:	0.0
STP PULse:	OFF
STP INV:	OFF
PKPower	0.08
CSPol	PNORMAL
HORiz COL:	2.0E+2
OPC	OFF
ACQ	NORMAL
MEAsure	REPEAT
MAG	OFF
VERt OFFset:	0.0
AUX	0.00
STP NUM:	5
STP CLI:	0.02
STP MUL:	OFF
VCS	0.0
PKVolt	16
CONfig	BSG
VERt COL:	2.0E+0
RQS	ON

Commands Table I (Cont'd)
INSTRUMENT PARAMETER COMMANDS (Cont'd)

Command	Argument	Link Arg	Definition
SET?			Respond with the front-panel settings. Any text message is not included. CURSOR OFF; MEASURE REPEAT; ACQUIRE NORMAL; DISPLAY STORE, INVERT: OFF, CRTCAL: OFF; HORIZ COLLECT: 2.0E+0, OFFSET: 0.0; VERT COLLECT: 2.0E+0, OFFSET: 0.0; MAG OFF; PKVOLT 16; PKPOWER 0.08; CSPOL PNORMAL; CONFIG BSGEN; STPGEN NUMBER: 5, PULSE: OFF, OFFSET 0.00, INVERT: OFF, MULT: OFF, CLIMIT: 0.02, CURRENT: 50.0E-9; AUX 0.00; VCSPPLY 0.0; RQS ON; OPC OFF; HILOWSW LOW
TESt?			Perform the ROM and RAM checks and respond with the result. TEST ROM:0000,RAM:0000 (Note: See the Operators Manual or Service Manual for codes differing from 0000.)

Commands Table J
STATUS AND EVENT REPORTING COMMANDS

Command	Argument	Link Arg	Definition
EVEnt?			Return the event code for the most recent event. **EVENT <code>** where **<code> = refer to following table.**
OPC	ON OFF		Enable or disable assertion of service request upon completion of an operation.
OPC?			Respond with the current status of the operation complete service request feature. **OPC ON** = enabled, or **OPC OFF** = disabled.
RQS	ON OFF		Enable or disable assertion of service requests (SRQs).
RQS?			Respond with the current status of the service request feature. **RQS ON** = enabled, or **RQS OFF** = disabled.

COMMANDS KEY-WORD CROSS-REFERENCE

* indicates header key-word for command or query

	Key Word	Usage	Command	Table
	AC	arg	CSPol	C
*	**ACQ**uire	cmd		A
*	**ACQ**uire**?**	query		A
	ALL	arg	PLOt	F
*	**AUX**	cmd		F
*	**AUX?**	query		F
	AVG	arg	ACQuire	A
	BASe	arg	HORiz	A
	BOPen	arg	CONfig	E
	BSGen	arg	CONfig	E
	BSHort	arg	CONfig	E
	CALchk	link arg	DISplay	A
	CLImit	arg	STPgen	D
	COLlect	arg	HORiz	A
	COLlect	arg	VERt	A
	COMpare	arg	DISplay	A
*	**CON**fig	cmd		E
*	**CON**fig**?**	query		E
*	**COV**er**?**	query		F
*	**CRO**ss	cmd		B
*	**CRO**ss**?**	query		B
	CRTcal	arg	DISplay	A
*	**CSP**ol	cmd		C
*	**CSP**ol**?**	query		C
	CURrent	arg	STPgen	D
*	**CURS**or	cmd		B
*	**CUR**ve	cmd		G
*	**CUR**ve**?**	query		G
	CURve	arg	PLOt	F

COMMANDS KEY-WORD CROSS-REFERENCE (Cont'd)

	Key Word	Usage	Command	Table
*	DISplay	cmd		A
*	DISplay?	query		A
*	DOT	cmd		B
*	DOT?	query		B
*	ENTer	cmd		A
	ENVelope	arg	ACQuire	A
	EOPen	arg	CONfig	E
	ESGen	arg	CONfig	E
*	EVEnt?	query		J
*	HELp?	query		I
*	HILowssw	cmd		C
*	HORiz	cmd		A
*	HORiz?	query		A
	HORiz	arg	MAG	A
	HORiz	link arg	ACQuire	A
*	ID?	query		I
*	INIt	cmd		I
	INVert	arg	DISplay	A
	INVert	arg	STPgen	D
	LONg	link arg	STPgen	D
*	LRSsw?	query		F
*	MAG	cmd		A
*	MAG?	query		A
*	MEAsure	cmd		F
*	MEAsure?	query		F
	MULt	arg	STPgen	D
	NDC	arg	CSPol	C
	NLEakage	arg	CSPol	C
	NNOrmal	arg	CSPol	C
	NORmal	arg	ACQuire	A
	NSTore	arg	DISplay	A
	NUMber	arg	STPgen	D

COMMANDS KEY-WORD CROSS-REFERENCE (Cont'd)

	Key Word	Usage	Command	Table
	OFF	arg	CURsor	B
	OFF	arg	MAG	A
	OFF	arg	OPC	J
	OFF	arg	RQS	J
	OFF	link arg	DISplay	A
	OFF	link arg	STPgen	D
	OFFset	arg	HORiz	D
	OFFset	arg	STPgen	D
	OFFset	arg	VERt	A
	ON	arg	OPC	J
	ON	arg	RQS	J
	ON	link arg	DISplay	A
	ON	link arg	STPgen	D
*	**OPC**	cmd		J
*	**OPC?**	query		J
	PDC	arg	CSPol	C
*	**PKP**ower	cmd		C
*	**PKP**ower**?**	query		C
*	**PKV**olt	cmd		C
*	**PKV**olt**?**	query		C
*	**PLO**t	cmd		F
	PLEakage	arg	CSPol	C
	PNOrmal	arg	CSPol	C
*	**PST**atus**?**	query		F
	PULse	arg	STPgen	D
*	**REA**dout**?**	query		H
*	**REC**all	cmd		F
	REPeat	arg	MEAsure	F
*	**RQS**	cmd		J
*	**RQS?**	query		J

COMMANDS KEY-WORD CROSS-REFERENCE (Cont'd)

	Key Word	Usage	Command	Table
*	**SAV**e	cmd		F
*	**SET?**	query		I
	SHOrt	link arg	STPgen	D
	SINgle	arg	MEAsure	F
	STEp	arg	HORiz	A
	STEp	arg	VERt	A
	STOre	arg	DISplay	A
*	**STP**gen	cmd		D
*	**STP**gen**?**	query		D
*	**TES**t**?**	query		I
*	**TEX**t	cmd		H
*	**TEX**t**?**	query		H
*	**VCS**pply	cmd		C
*	**VCS**pply**?**	query		C
*	**VER**t	cmd		A
	VERt	arg	MAG	A
	VERt	link arg	ACQuire	A
*	**VER**t**?**	query		A
	VIEw	arg	DISplay	A
	VOLtage	arg	STPgen	D
*	**WAV**frm**?**	query		G
*	**WFM**pre	cmd		G
*	**WFM**pre**?**	query		G
*	**WIN**dow	cmd		B
*	**WIN**dow**?**	query		B
	ZERochk	link arg	DISplay	A

Section 10 -- Service requests

HANDLING SERVICE REQUESTS

The standard GPIB status and error reporting system used by the 370 sends interrupt messages to the bus controller by asserting the **service request (SRQ)** line on the bus. This **SRQ** message indicates that either an error or a change in status has occurred.

To service an interrupt, the controller "polls" the instruments on the bus. The instrument asserting SRQ, the 370 in this case, returns a **status byte,** indicating the category of the event that caused the SRQ. Each SRQ is automatically cleared after it is reported in response to the poll. If there is more than one event to report, the instrument reasserts SRQ until all pending events have been reported. A complete list of status bytes that can occur is found in **Table 4.**

After polling the 370 to determine the status byte, you can obtain more detailed information about the event that caused the SRQ by sending the **EVE**nt**?** query. The response to an **EVE**nt**?** is an **event code**, an <NR1> number which is a code for certain specific conditions that may have occurred. **Table 5** lists the event codes returned by the 370.

If the status byte and event code are not read and cleared immediately they can be accessed later. In case there are multiple events, only the latest status byte and the one pending are saved. Event codes, however, are kept in a 10-deep, last-in-first-out (LIFO) buffer, for later recall.

The following program segments demonstrate the basics of handling an SRQ along with its corresponding status byte and event code. The status byte and event code are printed on the controller console screen to show the instrument status.

IBM PC: (This is for use with the auto-serial-poll flag disabled in the National card configuration-file. Also see the expanded example later in the guide.)

```
800 REM ***   SIMPLE SRQ HANDLER FOR 370    ***
810 CALL IBRSP(BD%,SPR%)
820 WRT$="EVENT?"
830 CALL IBWRT(BD%,WRT$)
840 RD$=SPACE$(100)
850 CALL IBRD(BD%,RD$)
860 PRINT "STATUS=";SPR%,"EVENT=";RD$
```

Tek 4041:

```
800 ! ***   Simple SRQ Handler for 370    ***
810 Poll stb,dev                 ! Poll bus.  Store status byte in "stb."
820 Input #dev prompt "EVENT?":event ! Send "EVENT?"  Input response.
830 Print "STATUS= ";stb;"  EVENT= ";event    ! Show status and event.
```

HP 200/300 SERIES:

```
800 REM ***   SIMPLE SRQ HANDLER FOR 370    ***
810 STB=SPOLL(DEV)             ! Poll device previously defined.
820 OUTPUT DEV;"EVENT?",END    ! Send "EVENT?" query.
830 ENTER DEV;EVENT$           ! Input response.
840 PRINT "STATUS= ";STB;"  EVENT= ";EVENT$   ! Show status and event.
```

MASKING SERVICE REQUESTS

A subset of SRQs for communicating that certain instrument processes have been finished, are the **operation complete SRQ**s **(OPC SRQ)**. These allow you to determine when the 370 has finished one operation so that you can proceed to the next task only when ready for it.

You may not always want your program to be interrupted by SRQs or OPC SRQs. Either set can be masked so that the 370 does not assert them until the mask is removed. This is done with the **RQS** and **OPC** commands.

RQS ON enables the 370 to assert an SRQ when it has an event to report. If this feature is turned off **(RQS OFF),** up to 10 events are still accumulated and can be retrieved with successive **EVE**ent**?** queries.

Similarly, **OPC ON** enables the 370 to assert an OPC SRQ upon completing an operation.

Note, the **RQS?** and **OPC?** queries respond only with whether the function is enabled **(ON)** or disabled **(OFF)**. They, by themselves, do not give you any other status or event information.

STATUS-BYTE TABLE

Table 4
Status Byte Responses

8	7	6	5	4	3	2	1	Decimal	Condition
0	1	0	0	0	0	0	1	65	Power On
0	1	0	0	0	0	1	0	66	Operation Complete
0	1	0	0	0	0	1	1	67	User Request
0	1	0	0	0	1	0	0	68	Plotter Output Complete
0	1	0	0	0	1	0	1	69	Collector Supply Recover
0	1	1	0	0	0	0	1	97	Command Error
0	1	1	0	0	0	1	0	98	Execution Error
0	1	1	0	0	0	1	1	99	Internal Error

- Bits 4–1 → Four-bit status code
- Bit 6 → Abnormal (1)/normal (0) condition
- Bit 7 → SRQ asserted (depends on RQS command)

Power On — This occurs when the power is turned on, after having been off.

Operation Complete — This status byte is set when the 370 completes an acquisition while in store mode.

User Request — This occurs when the front-panel RQS key is pressed.

Plotter Output Complete — This status byte is set when the 370 completes the plotter output operations.

Collector Supply Recover — This status byte is set when a PLL error or series resistor overheat error is recovered.

Command Error — This status byte is set when a message cannot be parsed or recognized.

Execution Error — This status byte is set when a message is parsed and is recognized, but cannot be executed, such as AUX 50.

Internal Error — This status byte indicates that the 370 microcomputer has discovered a malfunction that could cause the instrument to operate incorrectly or that there has been an operator error while accessing bubble memory.

EVENT-CODE TABLE

Table 5
Status Bytes and Event Codes

Status Byte	Event Code	Meaning
System Events		
0	0	No Error.
65	401	Power On.
66	402	Operation Complete (MASK OPC).
67	403	User Request (RQS key).
68	404	Plotter Output Complete.
69	405	Collector Supply Recovered.
Command Errors		
97	101	Command Header Error.
" "	103	Command Argument Error.
" "	106	Command Syntax Error.
" "	108	Checksum Error.
" "	109	Byte Count Error.
Execution Errors		
98	201	Command not executable in local mode.
" "	203	Output buffer overflow; remaining output lost.
" "	204	Setting conflicts.
" "	205	Argument out of range.
Internal Errors		
99	303	Phase lock system failed.
" "	305	Series Resistor is overheated.
" "	306	Plotter fail.
" "	307	Bubble I/O error.

Section 11 -- Miscellaneous information

PROGRAM EXAMPLES

IBM PC with GURU (4 examples)

Send simple command to 370	CROSS
Query status of horizontal	HOR
Query settings, display on screen	SET_DISPLAY
Query settings, store and display	SET_COPY
Handle service requests	SRQ

HP 200/300 Series (6 examples)

Talk and listen	TEST_370
Send simple command to 370	T370_CRO
Query status of horizontal	T370_HOR
Query all settings	T370_SET
Handle service requests	T370_SRQ
Create a preamble	T370_PREAM

ASCII AND GPIB CODE CHART

CROSS

```
10       CLEAR    ,59745!          ' IBM BASICA Decl   Rev C.3
20       IBINIT1 = 59745!          ' date:11-30-84
30       IBINIT2 = IBINIT1 + 3     ' Lines 10 thru 60 MUST be in your program.
40       BLOAD "bib.m",IBINIT1
50       CALL IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,IBONL,
IBRSC,IBSRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,IBEOS,IBTMO,IBEOT,IBRDF,IBWRTF)
60       CALL IBINIT2(IBGTS,IBCAC,IBWAIT,IBPOKE,IBWRT,IBWRTA,IBCMD,IBCMDA,
IBRD,IBRDA,IBSTOP,IBRPP,IBRSP,IBDIAG,IBXTRC,IBRDI,IBWRTI,IBRDIA,IBWRTIA,
IBSTA%,IBERR%,IBCNT%)
70 '******     END DECLARATIONS FOR TEK GURU GPIB INTERFACE      ************
80 '
90 '
100 '   PROGRAM:     370IBM1.BAS
110 '   PURPOSE:     SEND A SIMPLE COMMAND TO THE 370 CURVE TRACER
120 '                USING AN IBM PC CONTROLLER AND
130 '                BASICA PROGRAMMING LANGUAGE
140 '
150 '
160 '
170     BD$ = "GPIB0"
180     CALL IBFIND ( BD$, BD% )                'setup GPIB interface board 0
190 '
200     DEV$ = "DEV5"                           'known device name on board 0
210     CALL IBFIND ( DEV$, DEV% )              'setup device 5 on GPIB board 0
220 '
230     INPUT "What is primary address for 370 on GPIB? ";PRIMARY%
240 '
250 '           address 0 is reserved for the GPIB, address 31 places
260 '           the 370 off line.  valid addresses are 1 through 30
270     IF PRIMARY% >= 1 AND PRIMARY% <= 30  THEN GOTO 310
280     PRINT "primary address of 370 must be 1 through 30"
290     GOTO 230
300 '
310     CALL IBPAD ( DEV%, PRIMARY% )           'reset dev% for 370
320 '
330     WRT$ = "cross 450,650"                  'message to send
340     CALL IBWRT ( DEV%, WRT$ )               'send message to 370 via GPIB
350     END
```

HOR

```
10       CLEAR   ,59745!           ' IBM BASICA Decl  Rev C.3
20       IBINIT1 = 59745!          ' date:11-30-84
30       IBINIT2 = IBINIT1 + 3     ' Lines 10 thru 60 MUST be in your program.
40       BLOAD "bib.m",IBINIT1
50       CALL IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,IBONL,
IBRSC,IBSRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,IBEOS,IBTMO,IBEOT,IBRDF,IBWRTF)
60       CALL IBINIT2(IBGTS,IBCAC,IBWAIT,IBPOKE,IBWRT,IBWRTA,IBCMD,IBCMDA,
IBRD,IBRDA,IBSTOP,IBRPP,IBRSP,IBDIAG,IBXTRC,IBRDI,IBWRTI,IBRDIA,IBWRTIA,
IBSTA%,IBERR%,IBCNT%)
70 '******    END DECLARATIONS FOR TEK GURU GPIB INTERFACE       ************
80 '
90 '
100 '    PROGRAM:    370IBM2.BAS
110 '    PURPOSE:    QUERY STATUS OF HORIZONTAL
120 '                USING AN IBM PC CONTROLLER AND
130 '                BASICA PROGRAMMING LANGUAGE
140 '
150 '
160 '
170      BD$ = "GPIB0"
180      CALL IBFIND ( BD$, BD% )               'setup GPIB interface board 0
190 '
200      DEV$ = "DEV5"                          'known device name on board 0
210      CALL IBFIND ( DEV$, DEV% )             'setup device 5 on GPIB board 0
220 '
230      INPUT "What is primary address for 370 on GPIB? ";PRIMARY%
240 '
250 '            address 0 is reserved for the GPIB, address 31 places
260 '            the 370 off line.  valid addresses are 1 through 30
270      IF PRIMARY% >= 1 AND PRIMARY% <= 30  THEN GOTO 310
280      PRINT "primary address of 370 must be 1 through 30"
290      GOTO 230
300 '
310      CALL IBPAD ( DEV%, PRIMARY% )          'reset dev% for 370
320 '
330      WRT$ = "hor?"                          'message to send
340      CALL IBWRT ( DEV%, WRT$ )              'send message to 370 via GPIB
350 '
360      RD$ = SPACE$( 80)                      'dimension a blank string
370      CALL IBRD( DEV%, RD$)                  'read 370 over GPIB
380      RD$ = LEFT$( RD$, IBCNT%)              'remove right blanks
390      PRINT RD$                              'print 370 response
400 '
410      END
```

SET_DISPLAY

```
10       CLEAR   ,59745!          ' IBM BASICA Decl  Rev C.3
20       IBINIT1 = 59745!         ' date:11-30-84
30       IBINIT2 = IBINIT1 + 3    ' Lines 10 thru 60 MUST be in your program.
40       BLOAD "bib.m",IBINIT1
50       CALL IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,IBONL,
IBRSC,IBSRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,IBEOS,IBTMO,IBEOT,IBRDF,IBWRTF)
60       CALL IBINIT2(IBGTS,IBCAC,IBWAIT,IBPOKE,IBWRT,IBWRTA,IBCMD,IBCMDA,
IBRD,IBRDA,IBSTOP,IBRPP,IBRSP,IBDIAG,IBXTRC,IBRDI,IBWRTI,IBRDIA,IBWRTIA,
IBSTA%,IBERR%,IBCNT%)
70 '******    END DECLARATIONS FOR TEK GURU GPIB INTERFACE      ************
80 '
90 '
100 '   PROGRAM:     370IBM3.BAS
110 '   PURPOSE:     QUERY ALL SETTINGS AND PRINT TO SCREEN
120 '                USING AN IBM PC CONTROLLER AND
130 '                BASICA PROGRAMMING LANGUAGE
140 '
150 '
160 '
170     BD$ = "GPIB0"
180     CALL IBFIND ( BD$, BD% )             'setup GPIB interface board 0
190 '
200     DEV$ = "DEV5"                         'known device name on board 0
210     CALL IBFIND ( DEV$, DEV% )            'setup device 5 on GPIB board 0
220 '
230     INPUT "What is primary address for 370 on GPIB? ";PRIMARY%
240 '
250 '            address 0 is reserved for the GPIB, address 31 places
260 '            the 370 off line.  valid addresses are 1 through 30
270     IF PRIMARY% >= 1 AND PRIMARY% <= 30  THEN GOTO 310
280     PRINT "primary address of 370 must be 1 through 30"
290     GOTO 230
300 '
310     CALL IBPAD ( DEV%, PRIMARY% )         'reset dev% for 370
320 '
330     WRT$ = "SET?"                         'message to send
340     CALL IBWRT ( DEV%, WRT$ )             'send message to 370 via GPIB
350 '
360     MAXCHAR = 75                          'user wants a maximum of 75
370 '                                         'characters on each output line
380 '
390     RD$ = SPACE$( MAXCHAR)                'dimension a blank string
400     CALL IBRD( DEV%, RD$)                 'read 370 over GPIB
410     RD$ = LEFT$( RD$, IBCNT%)             'remove right blanks
420     PRINT RD$                             'print 370 response
430 '
440     IF IBCNT% = MAXCHAR THEN GOTO 390     'if IBRD filled array, then
450 '                                         'more GPIB reads are needed
460 '
470     END
```

SET_COPY

```
10       CLEAR    ,59745!           ' IBM BASICA Decl  Rev C.3
20       IBINIT1 = 59745!           ' date:11-30-84
30       IBINIT2 = IBINIT1 + 3      ' Lines 10 thru 60 MUST be in your program.
40       BLOAD "bib.m",IBINIT1
50       CALL IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,IBONL,
IBRSC,IBSRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,IBEOS,IBTMO,IBEOT,IBRDF,IBWRTF)
60       CALL IBINIT2(IBGTS,IBCAC,IBWAIT,IBPOKE,IBWRT,IBWRTA,IBCMD,IBCMDA,
IBRD,IBRDA,IBSTOP,IBRPP,IBRSP,IBDIAG,IBXTRC,IBRDI,IBWRTI,IBRDIA,IBWRTIA,
IBSTA%,IBERR%,IBCNT%)
70 '******    END DECLARATIONS FOR TEK GURU GPIB INTERFACE      ************
80 '
90 '
100 '    PROGRAM:     370IBM4.BAS
110 '    PURPOSE:     LEARN ALL FRONT PANEL SETTINGS AND PLACE IN CONTROLLER
120 '                 IN A FORM SUITABLE FOR DISK STORAGE OR LATER
130 '                 RETRANSMISSION TO 370.
140 '                 IBM PC CONTROLLER WITH BASICA PROGRAMMING LANGUAGE
150 '
160 '
170 '
180      BD$ = "GPIB0"
190      CALL IBFIND ( BD$, BD% )                'setup GPIB interface board 0
200 '
210      DEV$ = "DEV5"                           'known device name on board 0
220      CALL IBFIND ( DEV$, DEV% )              'setup device 5 on GPIB board 0
230 '
240      INPUT "What is primary address for 370 on GPIB? ";PRIMARY%
250 '
260 '            address 0 is reserved for the GPIB, address 31 places
270 '            the 370 off line.  valid addresses are 1 through 30
280      IF PRIMARY% >= 1 AND PRIMARY% <= 30  THEN GOTO 320
290      PRINT "primary address of 370 must be 1 through 30"
300      GOTO 240
310 '
320      CALL IBPAD ( DEV%, PRIMARY% )           'reset dev% for 370
330 '
340      WRT$ = "SET?"                           'message to send
350      CALL IBWRT ( DEV%, WRT$ )               'send message to 370 via GPIB
360 '
370      POINTER = 1                             'set.string pointer
380      DIM  SET.STRING$( 2)                    'make string array to hold
390 '                                            'settings received from 370
400 '
410      SET.STRING$( 1) = ""                    'blank set.string
420      SET.STRING$( 2) = ""                    'blank set.string
430 '
440      MAXCHAR = 255                           'maximum string length is 255
450 '                                            'characters in BASICA
460 '
470      RD$ = SPACE$( MAXCHAR)                  'dimension a blank string
480      CALL IBRD( DEV%, RD$)                   'read 370 over GPIB
490      RD$ = LEFT$( RD$, IBCNT%)               'remove right blanks
500 '
```

SET_COPY (Cont'd)

```
510      PRINT "GPIB RESPONSE: # characters = ";IBCNT%; 'display gpib string
520      PRINT "string received = " RD$
530      PRINT
540 '
550      IF POINTER = 2 THEN GOTO 660
560 '
570 '             PROCESS FIRST GPIB READ
580      I = MAXCHAR                          'search from end and find first
590      IF MID$( RD$, I, 1) <> ";" THEN I = I-1 : GOTO 590      'delimiter
600      SET.STRING$( 1) = LEFT$( RD$, I -1)
610      SET.STRING$( 2) = RIGHT$( RD$, LEN( RD$) -I)
620      POINTER = 2
630      GOTO 470
640 '
650 '             PROCESS SECOND GPIB READ
660      SET.STRING$( 2) = SET.STRING$( 2) + RD$
670 '
680      PRINT : PRINT "STRING 1 STORED IN MEMORY" 'display strings stored
690      PRINT SET.STRING$( 1)'                     in memory
700      PRINT : PRINT "STRING 2 STORED IN MEMORY"
710      PRINT SET.STRING$( 2)
720 '
730      END
```

SRQ

```
10        CLEAR    ,59745!            ' IBM BASICA Decl  Rev C.3
20        IBINIT1 = 59745!            ' date:11-30-84
30        IBINIT2 = IBINIT1 + 3       ' Lines 10 thru 60 MUST be in your program.
40        BLOAD "bib.m",IBINIT1
50        CALL IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,IBONL,
IBRSC,IBSRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,IBEOS,IBTMO,IBEOT,IBRDF,IBWRTF)
60        CALL IBINIT2(IBGTS,IBCAC,IBWAIT,IBPOKE,IBWRT,IBWRTA,IBCMD,IBCMDA,
IBRD,IBRDA,IBSTOP,IBRPP,IBRSP,IBDIAG,IBXTRC,IBRDI,IBWRTI,IBRDIA,IBWRTIA,
IBSTA%,IBERR%,IBCNT%)
70 '******    END DECLARATIONS FOR TEK GURU GPIB INTERFACE       ************
80 '
90 '
100 '    PROGRAM:     370IBM5.BAS
110 '    PURPOSE:     ILLUSTRATE HANDLING OF SERVICE REQUESTS GENERATED
120 '                 FROM THE FRONT PANEL
130 '                 IBM PC CONTROLLER WITH BASICA PROGRAMMING LANGUAGE
140 '
150 '
160 '
170      BD$ = "GPIB0"
180      CALL IBFIND ( BD$, BD% )              'setup GPIB interface board 0
190 '
200      DEV$ = "DEV5"                         'known device name on board 0
210      CALL IBFIND ( DEV$, DEV% )            'setup device 5 on GPIB board 0
220 '
230      INPUT "What is primary address for 370 on GPIB? ";PRIMARY%
240 '
250 '            address 0 is reserved for the GPIB, address 31 places
260 '            the 370 off line.  valid addresses are 1 through 30
270      IF PRIMARY% >= 1 AND PRIMARY% <= 30  THEN GOTO 310
280      PRINT "primary address of 370 must be 1 through 30"
290      GOTO 230
300 '
310      CALL IBPAD ( DEV%, PRIMARY% )         'reset dev% for 370
320 '
330      WRT$ = "RQS ON"                       'enable 370 to assert an SRQ
340      CALL IBWRT( DEV%, WRT$ )              'when it has an event to report
350 '
360 '
370 '            clear pre-existing conditions
380      GOSUB 580                             'get status byte and event code
390      IF S.BYTE% = 0 AND EVENT = 0 THEN GOTO 460
400      PRINT : PRINT "370 reports pre-existing status byte and event:"
410      GOSUB 750                             'print status byte and event
420      GOTO 380                              'empty the buffer
430 '
440 '
450 '            ask user to push SRQ button
460      PRINT : PRINT "press USER REQUEST/SRQ on 370 front panel"
470 '
480      GOSUB 580
490      IF S.BYTE% = 0 THEN GOTO 480         'wait for user input
500 '
```

SRQ (Cont'd)

```
510      GOSUB 750                              'print results
520      GOSUB 580                              'check 370 for empty event stack
530      IF S.BYTE% <> 0 OR EVENT <> 0 THEN GOTO 510
540 '
550      END
560 '    .........................................................
570 '
580 '>>          status byte and event subroutine
590 '
600      CALL IBRSP( DEV%, S.BYTE%)             'return serial pole byte
610      WRT$ = "event?"
620      CALL IBWRT( DEV%, WRT$)                'ask 370 to send event code
630 '
640      RD$ = SPACE$( 80)                      'make 80 character string space
650      CALL IBRD( DEV%, RD$)                  'read 370 response to "event?"
660 '
670      I = IBCNT%                             'find 1st blank from right end
680      IF MID$( RD$, I, 1) <> " " THEN I= I-1 :GOTO 680    'of string
690 '
700      EVENT = VAL( MID$(RD$, I+1, IBCNT% -I))
710 '
720      RETURN
730 '    .........................................................
740 '
750 '>>          print status byte and event
760 '
770      PRINT "status byte = ";S.BYTE%;" event = ";EVENT;" ...";
780      IF EVENT =   0 THEN PRINT " no error"
790      IF EVENT = 401 THEN PRINT " power on"
800      IF EVENT = 402 THEN PRINT " operation complete"
810      IF EVENT = 403 THEN PRINT " user request"
820      IF EVENT = 404 THEN PRINT " plotter output complete"
830      IF EVENT = 405 THEN PRINT " collector supply recovered"
840 '
850      RETURN
```

TEST_370

```
10     !  PROGRAM NAME:   TEST_370
20     !  PURPOSE:        TO TALK AND LISTEN TO A 370 CURVE TRACER
30     !                   FROM AN HP200/300 CONTROLLER
40     Event$="EVE?"
50     DIM Rd$[400]
60     DIM A$[400]
70     PRINT TAB(5);"GPIB TALKER/LISTENER" ! program title
80     PRINT "This program talks and listens to one instrument at a time "
90     PRINT "using its GPIB address.  The terminator is EOI by default."
100    PRINT TAB(25);"INSTRUCTIONS"
110    PRINT "1.  finish each entry by pressing RETURN or ENTER."
120    PRINT "2.  at T/L/C/E ? enter T to Talk to a device"
130    PRINT "                       L to Listen to a device"
140    PRINT "                       C to Change device address"
150    PRINT "                       E to End program"
160    ! ______________________________________________________________
170    PRINT "BEGIN:"
180 Addr_set:   !                SET DEVICE ADDRESS
190    PRINT "enter instrument address";
200    PRINT "(0-31), any other # to end):";
210    INPUT Addr
220    IF Addr<0 OR Addr>31 THEN GOTO End_it
230    Hpib=7
240    Devaddr=Hpib*100+Addr
250    PRINT Addr;", DEV=";Devaddr
260 Do_it:  !                    MAIN LINE
270    INPUT "T/L/C/E ?:",T1$
280    IF T1$="t" OR T1$="T" THEN                 !   TALK
290       GOSUB Talk_to
300       GOTO Do_it
310    END IF
320    IF T1$="l" OR T1$="L" THEN                 !   LISTEN
330       GOSUB Listen_to
340       GOTO Do_it
350    END IF
360    IF T1$="c" OR T1$="C" THEN GOTO Addr_set !   SET ADDRESS
370    IF T1$="e" OR T1$="E" THEN GOTO End_it   !   END PROGRAM
380    GOTO Do_it
390 ! ______________________________________________________________
400 Talk_to: !
410    PRINT
420    INPUT "STRING TO SEND:",A$
430    PRINT "sending: ";A$
440    PRINT
450    OUTPUT Devaddr;A$,END
460    Stat=SPOLL(Devaddr)
```

TEST_370 (Cont'd)

```
470   IF Stat=0 OR Stat=65 THEN
480      GOSUB No_err
490   ELSE
500      GOSUB Err_found
510   END IF
520   RETURN
530 ! ____________________________________________________
540 No_err: !
550   IF A$[LEN(A$)]="?" THEN GOSUB Listen_to
560   RETURN
570 ! ____________________________________________________
580 Err_found: !
590   OUTPUT Devaddr;Event$,END
600   GOSUB Listen_to
610   PRINT "ERROR STATUS BYTE= ";Stat;"    "; !
620   RETURN
630 ! ____________________________________________________
640 Listen_to: !
650   ENTER Devaddr;Rd$
660   IF NUM(Rd$)=255 THEN
670      PRINT Rd$
680      PRINT
690   END IF
700   PRINT "char rec'd ";LEN(Rd$);" ascii = ";NUM(Rd$)
710   PRINT "*";TRIM$(Rd$);"*"
720   RETURN
730 ! ____________________________________________________
740 End_it: !
750   PRINT CHR$(10);"Go to Local";
760   PRINT " (last addressed device responds to manual control again)"
770   PRINT "PROGRAM ENDED"
780   END
790 !______________END OF PROGRAM LIST______________
```

TEST_370 (Cont'd)

```
    GPIB TALKER/LISTENER
This program talks and listens to one instrument at a time
using its GPIB address.  The terminator is EOI by default.
                        INSTRUCTIONS
1.   finish each entry by pressing RETURN or ENTER.
2.   at T/L/C/E ? enter T to Talk to a device
                        L to Listen to a device
                        C to Change device address
                        E to End program
BEGIN:
enter instrument address(0-31), any other # to end): 18 , DEV= 718

sending: SET?

char rec'd  340  ascii =  68
*DOT    1;MEASURE REPEAT;ACQUIRE AVG:32;DISPLAY VIEW: 1,INVERT:OFF,CRTCAL:OFF;HO
RIZ COLLECT:2.0E+0,OFFSET:  0.0;VERT COLLECT:20.0E-3,OFFSET:  5.0;MAG OFF;PKVOLT
 16;PKPOWER 0.4;CSPOL PNORMAL;CONFIG BSGEN;STPGEN NUMBER: 4,PULSE:OFF,OFFSET:  3
.00,INVERT:OFF,MULT:OFF,CLIMIT:0.02,CURRENT:1.0E-3;AUX  -0.02;VCSPPLY  76.8;RQS
ON;OPC ON;HILOWSW LOW*

Go to Local (last addressed device responds to manual control again)
PROGRAM ENDED
```

T370_CRO

```
10    ! PROGRAM:  T370_CRO
20    ! PURPOSE:       SIMPLE COMMAND "CROSS"  ON A 370 CURVE TRACER
30    !                 FROM AN HP200/300 CONTROLLER
40    !
50    Event$="EVENT?"
60    Cmd$="CRO 600,600"
70    DIM Rd$[400]
80    PRINT TAB(5);"GPIB TALKER/LISTENER" ! program title
90    PRINT "Insure that the 370 terminator is EOI"
100   PRINT "  (TERM dip switch set to 0)"
110   ! ____________________________________________________________
120 Addr_set:   !                  SET DEVICE ADDRESS
130   PRINT "enter instrument address";
140   PRINT "(0-31), any other # to end):";
150   INPUT Addr
160   IF Addr<0 OR Addr>31 THEN GOTO End_it
170   Hpib=7
180   Devaddr=Hpib*100+Addr
190   PRINT Addr;", DEV=";Devaddr
200 Do_it:    !                    MAIN LINE
210   PRINT "        "
220   PRINT "Sending: ";Cmd$
230   OUTPUT Devaddr;Cmd$,END
240   Stat=SPOLL(Devaddr)
250   OUTPUT Devaddr;Event$,END
260   ENTER Devaddr;Rd$
270   PRINT "*";TRIM$(Rd$);"*";
280   PRINT "ERROR STATUS BYTE= ";Stat;"    "
290 !______________________END OF PROGRAM LIST______________________
300 End_it: !
310   PRINT "*----- END OF PROGRAM -----*"
320   END
```

```
    GPIB TALKER/LISTENER
Insure that the 370 terminator is EOI
  (TERM dip switch set to 0)
enter instrument address(0-31), any other # to end): 18 , DEV= 718

Sending: CRO 600,600
*EVENT 0*ERROR STATUS BYTE=  0
*----- END OF PROGRAM -----*
```

T370_HOR

```
10    ! PROGRAM:  T370_HOR
20    ! PURPOSE:        QUERY HORIZONTAL STATUS ON A 370 CURVE TRACER
30    !                  FROM AN HP200/300 CONTROLLER
40    !
50    Event$="EVENT?"
60    Query$="HOR?"
70    DIM Rd$[400]
80    DIM A$[400]
90    PRINT TAB(5);"GPIB TALKER/LISTENER" ! program title
100   PRINT "Insure that the 370 terminator is EOI"
110   PRINT "  (TERM dip switch set to 0)"
120   ! ____________________________________________________________
130 Addr_set:   !                   SET DEVICE ADDRESS
140   PRINT "enter instrument address";
150   PRINT "(0-31), any other # to end):";
160   INPUT Addr
170   IF Addr<0 OR Addr>31 THEN GOTO End_it
180   Hpib=7
190   Devaddr=Hpib*100+Addr
200   PRINT Addr;", DEV=";Devaddr
210 Do_it:   !                      MAIN LINE
220   PRINT "      "
230   OUTPUT Devaddr;Query$,END
240   Stat=SPOLL(Devaddr)
250   IF Stat=0 OR Stat=65 THEN
260      GOSUB Listen_to
270   ELSE
280      GOSUB Err_found
290   END IF
300   PRINT "*---------------PROGRAM ENDED---------------*"
310   STOP
320 ! ______________________________________________________________
330 Err_found: !
340   OUTPUT Devaddr;Event$,END
350   GOSUB Listen_to
360   PRINT "ERROR STATUS BYTE= ";Stat;"    "; !
370   RETURN
380 ! ______________________________________________________________
390 Listen_to: !
400   ENTER Devaddr;Rd$
410   IF NUM(Rd$)=255 THEN
420      PRINT Rd$
430      PRINT
440   END IF
450   PRINT "*";TRIM$(Rd$);"*"
460   RETURN
470 !______________________END OF PROGRAM LIST_____________________
480 End_it: !
490   END
```

T370_HOR (Cont'd)

```
    GPIB TALKER/LISTENER
Insure that the 370 terminator is EOI
  (TERM dip switch set to 0)
enter instrument address(0-31), any other # to end): 18 , DEV= 718

*HORIZ COLLECT:2.0E+0,OFFSET:  0.0*
*----------------PROGRAM ENDED---------------*
```

T370_SET

```
10     !  PROGRAM:  T370_SET
20     !  PURPOSE:       QUERY ALL SETTINGS ON A 370 CURVE TRACER
30     !                  FROM AN HP200/300 CONTROLLER
40     !
50     Event$="EVENT?"
60     Query$="SET?"
70     DIM Rd$[400]
80     DIM A$[400]
90     PRINT TAB(5);"GPIB TALKER/LISTENER" ! program title
100    PRINT "Insure that the 370 terminator is EOI"
110    PRINT "  (TERM dip switch set to 0)"
120    ! ____________________________________________________________
130 Addr_set:   !                SET DEVICE ADDRESS
140    PRINT "enter instrument address";
150    PRINT "(0-31), any other # to end):";
160    INPUT Addr
170    IF Addr<0 OR Addr>31 THEN GOTO End_it
180    Hpib=7
190    Devaddr=Hpib*100+Addr
200    PRINT Addr;", DEV=";Devaddr
210 Do_it:   !                   MAIN LINE
220    PRINT "      "
230    OUTPUT Devaddr;Query$,END
240    Stat=SPOLL(Devaddr)
250    IF Stat=0 OR Stat=65 THEN
260       GOSUB Listen_to
270    ELSE
280       GOSUB Err_found
290    END IF
300    PRINT "*---------------PROGRAM ENDED--------------*"
310    STOP
320 ! ______________________________________________________________
330 Err_found: !
340    OUTPUT Devaddr;Event$,END
350    GOSUB Listen_to
360    PRINT "ERROR STATUS BYTE= ";Stat;"   "; !
370    RETURN
380 ! ______________________________________________________________
390 Listen_to: !
400    ENTER Devaddr;Rd$
410    IF NUM(Rd$)=255 THEN
420       PRINT Rd$
430       PRINT
440    END IF
450    PRINT "*";TRIM$(Rd$);"*"
460    RETURN
470 !____________________END OF PROGRAM LIST__________________
480 End_it: !
490    END
```

T370_SET (Cont'd)

```
    GPIB TALKER/LISTENER
Insure that the 370 terminator is EOI
  (TERM dip switch set to 0)
enter instrument address(0-31), any other # to end): 18  , DEV= 718

*CROSS  600, 600;MEASURE REPEAT;ACQUIRE NORMAL;DISPLAY STORE,INVERT:OFF,CRTCAL:O
FF;HORIZ COLLECT:2.0E+0,OFFSET:  0.0;VERT COLLECT:2.0E+0,OFFSET:  0.0;MAG OFF;PK
VOLT 16;PKPOWER 0.08;CSPOL PNORMAL;CONFIG BSGEN;STPGEN NUMBER: 5,PULSE:OFF,OFFSE
T:  0.00,INVERT:OFF,MULT:OFF,CLIMIT:0.02,CURRENT:50.0E-9;AUX   0.00;VCSPPLY  0.
0;RQS ON;OPC OFF;HILOWSW LOW*
*---------------PROGRAM ENDED--------------*
```

T370_SRQ

```
10    ! PROGRAM:  T370_SRQ
20    ! PURPOSE:       HANDLE SERVICE REQUESTS ON A 370 CURVE TRACER
30    !                 FROM AN HP200/300 CONTROLLER
40    !
50    Event$="EVENT?"
60    Query$="RQS ON"
70    DIM Rd$[400]
80    DIM A$[400]
90    PRINT TAB(5);"GPIB TALKER/LISTENER" ! program title
100   PRINT "Insure that the 370 terminator is set to EOI"
110   PRINT "  (TERM dip switch set to 0)"
120   ! ________________________________________________________
130 Addr_set:   !               SET DEVICE ADDRESS
140   PRINT "enter instrument address";
150   PRINT "(0-31), any other # to end):";
160   INPUT Addr
170   IF Addr<0 OR Addr>31 THEN GOTO End_it
180   Hpib=7
190   Devaddr=Hpib*100+Addr
200   PRINT Addr;", DEV=";Devaddr
210 Do_it:    !                 MAIN LINE
220   PRINT "      "
230   OUTPUT Devaddr;Query$,END
240   Stat=SPOLL(Devaddr)
250   IF Stat=0 OR Stat=65 THEN
260      PRINT "press USER REQUEST/SRQ on 370 front panel"
270      GOSUB Trap_it
280   ELSE
290      GOSUB Err_found
300   END IF
310   PRINT "*---------------PROGRAM ENDED--------------*"
320   STOP
330 ! __________________________________________________________
340 Trap_it: !
350   Stat=SPOLL(Devaddr)
360   IF Stat=0 OR Stat=65 THEN GOTO 350
370   GOSUB Err_found
380   RETURN
390 ! __________________________________________________________
400 Err_found: !
410   OUTPUT Devaddr;Event$,END
420   ENTER Devaddr;Rd$
430   PRINT "STATUS=";Stat;"    ";"EVENT=";TRIM$(Rd$)
440   RETURN
450 !_________________END OF PROGRAM LIST_________________
460 End_it: !
470   END
```

T370_SRQ (Cont'd)

```
    GPIB TALKER/LISTENER
Insure that the 370 terminator is set to EOI
  (TERM dip switch set to 0)
enter instrument address(0-31), any other # to end): 18 , DEV= 718

press USER REQUEST/SRQ on 370 front panel
STATUS= 67    EVENT=EVENT 403
*----------------PROGRAM ENDED---------------*
```

T370_PREAM

```
10    ! PROGRAM:  T370_PREAM
20    ! PURPOSE:  CREATE A WAVEFORM PREAMBLE ON A 370 CURVE TRACER
30    !           FROM AN HP200/300 CONTROLLER
40    Event$="EVE?"
50    PRINT TAB(5);"GPIB TALKER/LISTENER"
60    PRINT "Insure that the 370 terminator is EOI"
70    PRINT "  (TERM dip switch set to 0)"
80    ! ____________________________________________________________
90 Addr_set: !                 SET DEVICE ADDRESS
100   PRINT "enter instrument address";
110   PRINT "(0-31), any other # to end):";
120   INPUT Addr
130   IF Addr<0 OR Addr>31 THEN GOTO End_it
140   Hpib=7
150   Devaddr=Hpib*100+Addr
160   PRINT Addr;" , DEV=";Devaddr
170 Do_it:!
180   PRINT
190   !      Note that temporary storage fields of 2, 5, 7, and 24
200   !      are set up to accomodate the necessary left and right
210   !      justification routines because of length dependency
220   DIM Rd$[400]
230   DIM Wfid1$[34],Wfid2$[52],Wfid3$[51],Wfid$[137]
240   DIM Pre1$[200],Pre2$[60],Pre3$[60],Pre4$[13],Pream$[333]
250   DIM Temp2$[2],Temp5$[5],Temp7$[7]
260   DIM Index$[2],Vert$[7],Horiz$[7],Step$[7],Offset$[7]
270   DIM Bgm$[5],Aux$[7],Acq$[3],Text$[24],Xoff$[5],Yoff$[5]
280   DIM In$[50],Out$[50]
290   PRINT "ENTER WFMPRE INFO:"
300   PRINT "WFID INDEX          =";
310   INPUT Index$
320   PRINT Index$
330   Temp2$="  "                        ! Initialize temporary field
340   Temp2$[3-LEN(Index$)]=Index$       ! Right-justify to temp field
350   Index$=Temp2$                      ! Move temp field to original
360   PRINT "     VERT (amps)      =";
370   INPUT Vert$
380   PRINT Vert$
390   Temp7$="       "                   ! Initialize temporary field
400   Temp7$[8-LEN(Vert$)]=Vert$         ! Right-justify to temp field
410   Vert$=Temp7$                       ! Move temp field to original
420   PRINT "     HORIZ (volts)    =";
430   INPUT Horiz$
440   PRINT Horiz$
450   Temp7$="       "                   ! Initialize temporary field
460   Temp7$[8-LEN(Horiz$)]=Horiz$       ! Right-justify to temp field
```

T370_PREAM (Cont'd)

```
470   Horiz$=Temp7$                        ! Move temp field to original
480   PRINT "      STEP (amps)       =";
490   INPUT Step$
500   PRINT Step$
510   Temp7$="       "                     ! Initialize temporary field
520   Temp7$[8-LEN(Step$)]=Step$           ! Right-justify to temp field
530   Step$=Temp7$                         ! Move temp field to original
540   PRINT "      OFFSET (amps)     =";
550   INPUT Offset$
560   PRINT Offset$
570   Temp7$="       "                     ! Initialize temporary field
580   Temp7$[8-LEN(Offset$)]=Offset$       ! Right-justify to temp field
590   Offset$=Temp7$                       ! Move temp field to original
600   PRINT "      BGM               =";
610   INPUT Bgm$
620   PRINT Bgm$
630   FOR I=(LEN(Bgm$)+1) TO 5             ! Left-justify, blank fill
640       Bgm$[I]=" "
650       NEXT I
660   PRINT "      AUX (volts)       =";
670   INPUT Aux$
680   PRINT Aux$
690   Temp7$="       "                     ! Initialize temporary field
700   Temp7$[8-LEN(Aux$)]=Aux$             ! Right-justify to temp field
710   Aux$=Temp7$                          ! Move temp field to temporary
720   PRINT "      ACQ (AVG,NOR,ENV)=";
730   INPUT Acq$
740   PRINT Acq$
750   PRINT "      TEXT              =";
760   INPUT Text$
770   PRINT Text$
780   FOR I=(LEN(Text$)+1) TO 24           ! Left-justify, blank fill
790       Text$[I]=" "
800       NEXT I
810   PRINT "NR.PT (# points)        =";
820   INPUT Nrpt$
830   PRINT Nrpt$
840   PRINT "XMULT (scientific)      =";
850   INPUT Xmult$
860   PRINT Xmult$
870   PRINT "XOFF (integer)          =";
880   INPUT Xoff$
890   PRINT Xoff$
900   Temp5$="     "                       ! Initialize temporary field
910   Temp5$[6-LEN(Xoff$)]=Xoff$           ! Right-justify to temp field
920   Xoff$=Temp5$                         ! Move temp field to original
930   PRINT "YMULT (scientific)      =";
940   INPUT Ymult$
950   PRINT Ymult$
```

T370_PREAM (Cont'd)

```
960   PRINT "YOFF (integer)        =";
970   INPUT Yoff$
980   PRINT Yoff$
990   Temp5$="      "                       ! Initialize temporary field
1000  Temp5$[6-LEN(Yoff$)]=Yoff$            ! Right-justify to temp field
1010  Yoff$=Temp5$                          ! Move temp field to original
1020  PRINT "LN.FMT (VECTOR,DOT)    =";
1030  INPUT Lnfmt$
1040  PRINT Lnfmt$
1050  INPUT "ANY CHANGES?",A$
1060  IF A$="Y" THEN GOTO 290
1070  Wfid1$="WFMPRE WFID:"&CHR$(34)&"INDEX "&Index$&"/VERT "&Vert$
1080  Wfid2$="/HORIZ "&Horiz$&"/STEP "&Step$&"/OFFSET "&Offset$&"/BGM "&Bgm$
1090  Wfid3$="/AUX "&Aux$&"/ACQ "&Acq$&"/TEXT "&Text$&CHR$(34)
1100  Wfid$=Wfid1$&Wfid2$&Wfid3$
1110  Pre1$=Wfid$&",ENCDG:BIN,NR.PT:"&Nrpt$&",PT.FMT:XY,XMULT:"&Xmult$
1120  Pre2$=",XZERO:0,XOFF:"&Xoff$&",XUNIT:V,YMULT:"&Ymult$&",YZERO:0,YOFF:"
1130  Pre3$=Yoff$&",YUNIT:A,BYT/NR:2,BN.FMT:RP,BIT/NR:10,CRVCHK:CHKSM0,"
1140  Pre4$="LN.FMT:"&Lnfmt$
1150  Pream$=Pre1$&Pre2$&Pre3$&Pre4$        ! Concatenate to one string
1160  PRINT "Sending:"
1170  PRINT Pream$
1180  PRINT
1190  Stat=SPOLL(Devaddr)
1200  OUTPUT Devaddr;Pream$,END
1210  Stat=SPOLL(Devaddr)
1220  OUTPUT Devaddr;Event$,END
1230  ENTER Devaddr;Rd$
1240  PRINT "*",TRIM$(Rd$),"*"
1250 End_it:  !
1260  PRINT
1270  PRINT "*---- PROGRAM ENDED ----*"
1280  END
```

T370_PREAM (Cont'd)

```
    GPIB TALKER/LISTENER
Insure that the 370 terminator is EOI
  (TERM dip switch set to 0)
enter instrument address(0-31), any other # to end): 18   , DEV= 718

ENTER WFMPRE INFO:
WFID INDEX              =1
     VERT (amps)        =20mA
     HORIZ (volts)      =2 V
     STEP (amps)        =1mA
     OFFSET (amps)      =3.00mA
     BGM                =20
     AUX (volts)        =-0.02 V
     ACQ (AVG,NOR,ENV)=AVG
     TEXT               =2N3904 ENVELOPE MODE
NR.PT (# points)        =1024
XMULT (scientific)      =+2.0E-2
XOFF (integer)          =12
YMULT (scientific)      =+2.0E-4
YOFF (integer)          =12
LN.FMT (VECTOR,DOT)     =VECTOR
Sending:
WFMPRE WFID:"INDEX  1/VERT    20mA/HORIZ     2 V/STEP     1mA/OFFSET  3.00mA/BGM
 20 /AUX -0.02 V/ACQ AVG/TEXT 2N3904 ENVELOPE MODE    ",ENCDG:BIN,NR.PT:1024,PT.
FMT:XY,XMULT:+2.0E-2,XZERO:0,XOFF:   12,XUNIT:V,YMULT:+2.0E-4,YZERO:0,YOFF:   12
,YUNIT:A,BYT/NR:2,BN.FMT:RP,BIT/NR:10,CRVCHK:CHKSM0,LN.FMT:VECTOR

*         EVENT 0    *

*---- PROGRAM ENDED ----*
```