GPIB SYSTEMS CONCEPTS

GPIB

COMPANY CONFIDENTIAL

Written and compiled by
Bill Baunach

Tektronix®
COMMITTED TO EXCELLENCE

TABLE OF CONTENTS

I. CUSTOMERS, SYSTEMS, AND YOU I-1

Why Customers Buy GPIB I-2
Tek's Commitment I-5
Sales Opportunity I-6
Using This Workbook I-8

II. SYSTEM OVERVIEW II-1

Hardware II-1
Software II-10
Benefits II-17
Self-Test II-21

III. THE GENERAL PURPOSE INTERFACE BUS III-1

IEEE 488 Definitions III-2
Typical Data Transfer III-10
Hardware Characteristics III-12
GPIB-Compatible III-14
System Specifications III-19
Self-Test III-23

IV. TEK CODES AND FORMATS IV-1

Codes and Formats Definitions IV-2
Benefits IV-16
Self-Test IV-18

V. 4050 OPERATION V-1

Getting Started V-1
Arithmetic V-5
Editing Keys V-10
Peripheral Control Keys V-12
Simple 4050 Operation V-13
User Definable Keys (UDK's) V-15
Peripherals V-16
Demonstration Software V-17
Self-Test V-28

VI. 4050 BASIC PROGRAMMING VI-1

What is a Program VI-2
Computation in Basic VI-6
Print VI-10
Program Control VI-16
Putting Intelligence In Your Program VI-17
More Control VI-21
Subroutines VI-25
Strings VI-29
Input/Output over the GPIB VI-37
RBYTE/WBYTE VI-40
Serial Poll VI-42
Tape Commands VI-45
User Definable Keys VI-49
Miscellaneous VI-53
Bibliography VI-54
Self Test VI-55

APPENDICES

A. ASCII/GPIB CHART A-1
B. ANSWERS TO SELF-TESTS B-1
C. GLOSSARY C-1

I. CUSTOMERS, SYSTEMS, AND YOU

The use of the IEEE 488 General Purpose Interface Bus (GPIB) on Tektronix instruments offers you, as Tek Sales Engineers, a new set of opportunities (and challenges). The GPIB is making a significant impact on Tektronix and on your job. GPIB systems are important because they provide a major benefit to our customers. This booklet is designed to help you understand how GPIB systems work, why customers need them, Tektronix' commitment to GPIB, and the sales opportunity created by your customers' needs.

The current economy and the competitive business climate are creating serious problems for your customers. As a result, customers are demanding increases in the productivity of their labor and in the productivity of their capital (money).

GPIB instruments provide you with a significant opportunity, because they provide the benefit of increased productivity to your customer. The systems capability that GPIB makes practical can be a big part of the solution to the economic difficulties facing our customers. This systems capability is a logical evolutionary development of the technology which supports our traditional test and measurement marketplace.

GPIB has changed the way customers perceive the solutions to their measurement problems. Now, customers can easily connect many products together. Stand-alone products will continue to be purchased, but many of your customers are now interested in systems and the products which can be put into them. Figure 1-1 diagrams the new situation.

GPIB systems are part of a long-term trend. Customers who require test and measurement gear such as we provide are becoming more systems oriented. Let's look at why this is.

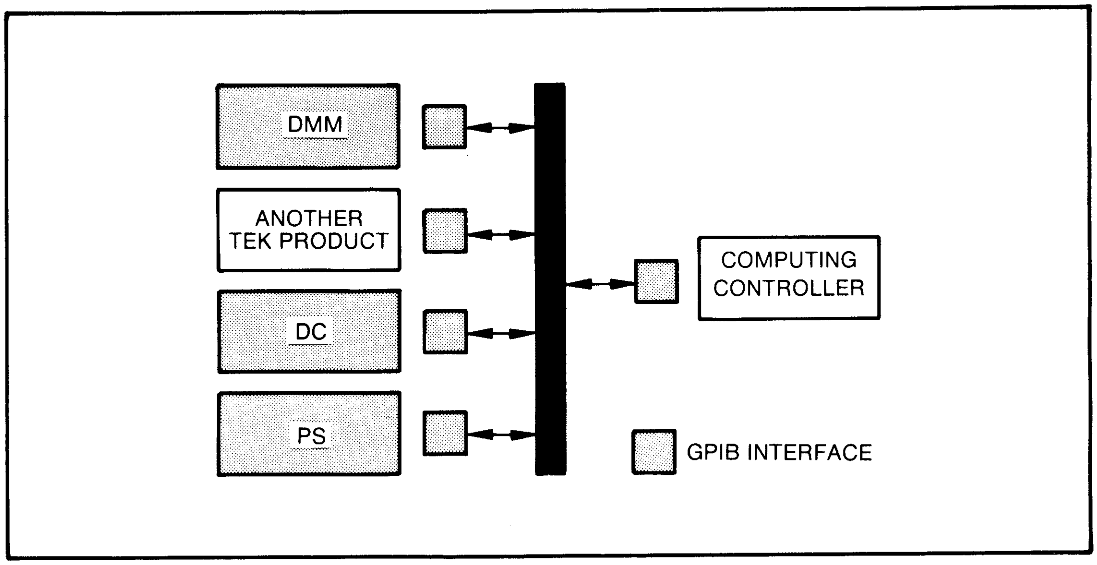


Fig. 1.1. GPIB enables collections of products to work together. It provides for increased capabilities beyond stand-alone products.

WHY CUSTOMERS BUY GPIB

As mentioned earlier, many of the benefits that customers derive from systems boil down to one thing: increased productivity, the ability to produce more or better products with less time, less money, or both. Systems with GPIB programmable instruments provide this increased productivity through:

- Saved time and money, less labor and less skilled labor
- Automated measurements, documented results
- Fewer errors
- More exhaustive testing
- Testing of complex devices which could not practically be tested manually

- Increased speed and accuracy of tests
- Easy-to-reconfigure systems
- Consistent results

Because electronic equipment is becoming more complex, and because the supply of highly trained labor is very limited, manufacturers are finding it necessary to automate their measurement and test procedures. The need to automate is occurring in all phases of a product life-cycle, i.e., research, design, and manufacturing.

Because of the relative shortage of trained electronic engineers and technicians, those available must be used more effectively. Systems allow routine measurements to be automated, freeing engineers for more creative tasks such as design and analysis, and freeing technicians to solve the really tough problems.

Before the GPIB was developed, development of instrument systems was economically feasible only in manufacturing environments where the volume of testing was very high, or where the cost of the system was low compared to the value of the measurement/test results. The high costs of these systems resulted from the fact that a custom interface had to be designed for each instrument in a system, a costly proposition. These custom interfaces also resulted in high programming and maintenance costs. Now the GPIB makes more systems economical and practical.

It is now possible to assemble an automated measurement system with relative ease, so system development costs can be reduced dramatically. Thus, automated measurement systems are now feasible for engineering benches, low-volume production, quality assurance, and many other applications where such systems were previously impractical or too expensive.

MARKET ENVIRONMENT

The GPIB is a product feature purchased by our traditional customers. Virtually all GPIB products are eventually interfaced with controllers into systems, either temporarily or permanently.

Tek did some research in 1978 to find out to what degree GPIB had permeated the marketplace. We polled about 200 Tek customers. At that time we found that:

- 67% have GPIB equipment
- 39% currently use GPIB gear in systems
- 28% currently use GPIB gear stand-alone (28 + 39 = 67)
- Of the 28% using products stand-alone,
    - 12% will put it into a system within 12 months
    - 12% will put it into a system between 12 and 18 months
    - 4% are hedge buyers

Therefore, of the group questioned, fully 63% either are or will be using GPIB products in systems. Your customers are buying or thinking about buying systems.

Today, on these instruments where the customer pays extra for the GPIB (468, 5223, 492), more than one-fourth are being ordered with the GPIB option.

TEK'S COMMITMENT

Tektronix is committed to supplying GPIB-compatible instruments. There is a corporate policy which states that all new Tek products must be GPIB-compatible unless it is unreasonable for the intended marketplace.

There are two reasons for this. The research that we have done including inputs from the sales force, indicates that our customers are demanding GPIB compatibility for their instruments, i.e., our products. Some customers, as many of you already know, will not or cannot buy test and measurement gear unless it is GPIB-compatible. That is strong motiviation.

The other reason is that Tek itself needs to increase productivity to maintain long-term profitability. We have recognized that computer-aided systems can provide a significant means to achieve such an increase. We are establishing computer-aided systems in many diverse facits of our operation -- research, design, manufacture (we even have some robots), and test. Since computer-aided sytems are helping us to increase productivity, then surely they can help our customers as well. For test and measurement kinds of products, GPIB compatibility is a major step towards the development of systems solutions.

Fortunately, Tek is not entirely new to the field of instrument systems. We have been actively engaged in selling waveform-oriented systems (Signal Processing Systems) since 1973. This expertise in waveform-related systems is an important advantage to us, and waveform products (digitizers and spectrum analyzers) will continue to be a major part of our GPIB product offerings. It is also important because we have had the advantage of several years' experience to learn how best to handle software interfaces for measurement instruments and data. The result of this learning is the Tek Codes and Formats Standard; it is a major step toward improving compatibility between devices.

SALES OPPORTUNITY

Tektronix GPIB instruments present you with a new set of challenges and opportunities. With the new challenges that GPIB products pose come the rewards of selling large ticket systems. The way you sell will be affected. For example, with collections of products such as GPIB systems suggest, there will need to be more team selling activities among the different sales and support groups. The GPIB presents a challenge to all of us. Because it is a major customer benefit, it represents a significant opportunity as well.

GPIB systems must often be sold higher in an organization than stand-alone instruments. As package price increases, the purchase approval moves further up in the chain of command. When you sell at higher levels, product features are not of great interest to the manager. More important is the economic benefit; you will need to sell productivity. When you are selling complete systems (those sales that include a controller), you will seldom be making a pitch to the end user. Even as you get closer to the end user though, you will need to sell specific product (system) benefits rather than features because there will be too many features to discuss. However, you must know the features which provide the benefit.

When you start to work with customers who are thinking about buying GPIB products, they will fall into three categories:

- Those buying stand-alone as a hedge or for future system application

- Those buying a complete system

- Those buying a product for use in a system using either a Tek or non-Tek controller

Although what you emphasize with each customer will be different, every one will want and need to discuss system concepts. Newcomers to GPIB systems will need to learn about general concepts and benefits. Those who have some experience with GPIB systems will be more interested in specific product compatibilities in regard to their applications and/or existing systems needs.

The rewards in selling GPIB systems are commensurate with the challenges that they pose. GPIB-compatible products are typically priced at a premium because of the digital hardware and software which make them easy to interface. (Remember that the alternative is expensive custom interfacing.) Full systems, of course, carry a fairly high pricetag. Selling one is gratifying and moves you closer to target more quickly.

There are a couple of other rewards less obvious than the commissions from selling high price products. Selling a system usually means easily selling additional complementary products where the extra dollars sold is far greater than the effort required to earn them. For example, once a computer terminal is sold, it is easier to sell a hard copy unit. Or, more to the point, once a TM5000 mainframe is sold, it will be easier to convince a customer to fill in all the holes. Our experience in SPS has shown that a complete system sale will typically net three times more dollars than an (SPS) stand-alone product sale. The effort expended is not three times as much. Also, when a customer wants to add new products to a system that customer will normally turn to the source of previous products.

There is also an advantage in selling higher in an organization. When you start to sell to high level managers and learn their concerns, you will gain a perspective of business organizations that you cannot acquire when selling to a service department manager. You may even learn enough about a company to find additional departments that need our products which you might not otherwise have found. And the business acumen gained will be invaluable.

USING THIS BOOKLET

The primary purpse of this booklet is to enable you to effectively represent Tektronix' GPIB products, sytems, and services in a competent, positive, and productive manner.

Specifically, to be able to effectively sell Tektronix' GPIB products and systems, you will need to be able to:

- Speak intelligently about GPIB, systems, and software.
- Relate software and systems potentials to customers.
- Describe the features and benefits of Tek's Codes and Formats Standard.
- Be confident and effective in the presence of Tek controllers, especially 4052 and 4041.
- Connect pieces of gear on the GPIB.
- Operate computer demo tapes for instrumentation products.
- Recover from difficulties on demo tapes.
- List programs.
- Read (analyze) simple programs that are well documented.
- Control GPIB devices from the 4052 keyboard.

This is a tall order, but a necessary one. Remember, the long term trend is towards systems. This booklet is a step towards meeting these objectives. We have tried to make this workbook as brief and concise as possible without making any assumptions about prior system/programming experience. It has been our intent that all the material contained in this booklet is directly applicable to your sales environment, acknowledging that each of you has an individual approach to your customers.

A word about programming. Chapter VI is a short primer on programming the 4052 with special emphasis on instrumentation applications. It is not our intent that you become a computer programmer. However, introduction to programming is essential for you to understand the needs and concerns of your customers and to understand the potentials of products, systems, and software. Just as it is impossible to describe the flavor of an apple to a person who has never tasted one, it is equaly impossible for you to discuss GPIB systems without taking a bite into programming.

Computers have added a number of words to our language, and although it may be circumstantial, it seems that ever since the advent of computers, acronyms have invaded the simplest of sentences. To make life a little easier, all acronyms and computer/systems terms are included in a glossary in Appendix C.

Probably the best approach to the remaining chapters of this booklet is to read the self test at the end of a given chapter first. These self tests cover the major objectives of each chapter. They do not, however, cover the explanations of why a particular answer is the way it is. Knowing the questions on the tests will help provide direction for your reading. Answers to the self tests are in Appendix B.

If you get stuck in a problem or have a question about a concept, the IDD Systems Analysts have been asked to give you a hand when you need it. You are also encouraged to call Bob Young, ext. 1504 (Walker Road), of GPI Marketing (formerly TM500) if you have questions -- he is expecting your call.

II. SYSTEM OVERVIEW

The system solves your customers' needs. The GPIB helps make systems possible by providing a common link between system components. GPIB allows the various devices to communicate and therefore to function as a whole. But remember, customers have the total system uppermost in their minds, not the interfacing scheme.

In this chapter, we are going to describe system 1) hardware components, 2) software components, and 3) benefits.

HARDWARE

Virtually all instrumentation systems have at minimum the following generic components.

- Some INSTRUMENTS. There are either stimulus instruments, such as function generators, pulse generators, etc. or measurement instruments, such as counters, waveform digitizers, and multimeters.

- A CONTROLLER. This controls all system communication, makes decisions, establishes the sequence of activity, and processes results. This is generally a small computer.

- A KEYBOARD or KEYPAD, which provides user input and interaction with the system. Lets the user command the system.

- A DISPLAY, which provides the user output to monitor the system. Usually this is a crt or LED display.

- A MASS STORAGE device to hold software and data -- probably a tape unit or disk drive.

Except for a very few cases, all systems must have one of each of the above hardware components.

Here is an example of a typical instrumentation system:

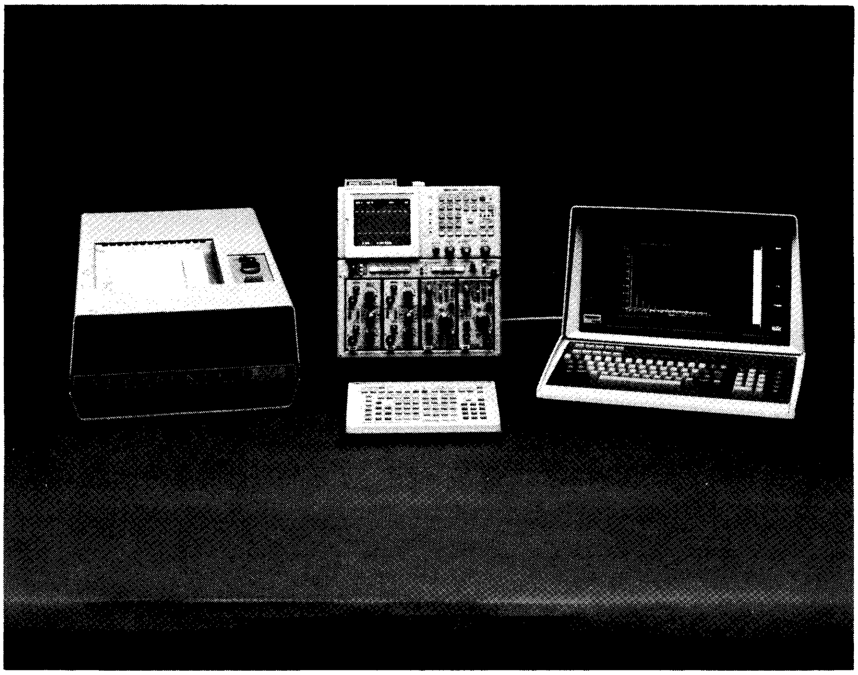


Fig. 2.1. In a WP1310 system, the 4052 contains four of the five generic system components.

Figure 2.1 shows the WP1310 which roughly consists of a 7854 and a 4052. Do these two units include the five generic system components mentioned above? Sure, and this points to a common fact. Many times, more than one system component (function) is built into one unit. In the case of the 4052, the unit includes: 1) the CONTROLLER, a microprocessor, memory, and associated hardware; 2) the KEYBOARD; 3) the DISPLAY, crt; and 4) the MASS STORAGE, the DC300 tape drive. This leaves the 7854 as the INSTRUMENT.

The reasons for breaking out the various system components are: 1) so that you don't forget to discuss any of them when you are consulting with customers; 2) because a customer may already own some of the components; and 3) because customers may want to choose among several alternatives in each category to help meet their specific requirements.

The system component list above stated minimum requirements. Here is the same list in a general form:

- INSTRUMENTS. Measurement (acquisition) and stimulus.

- CONTROLLER. Communicates, sequences, and processes.

- User INPUT devices. Keyboards are a must, but we might also include graphic tablets, joysticks, thumbwheels, and some card readers.

- User OUTPUT devices. Usually crt screens, but also might include line printers, hard copy units, and plotters.

- MASS STORAGE devices. Hard and flexible disks, cartridge tapes, and large mag tapes.

- MULTIFUNCTION INTERFACES. Programmable signal routing switches (multiplexers), D/A and A/D converters, and other DUT interfaces. Used especially in production testing environments.

Figure 2.2 shows how the above components are logically connected. Not surprisingly, the system controller is at the center of the action.

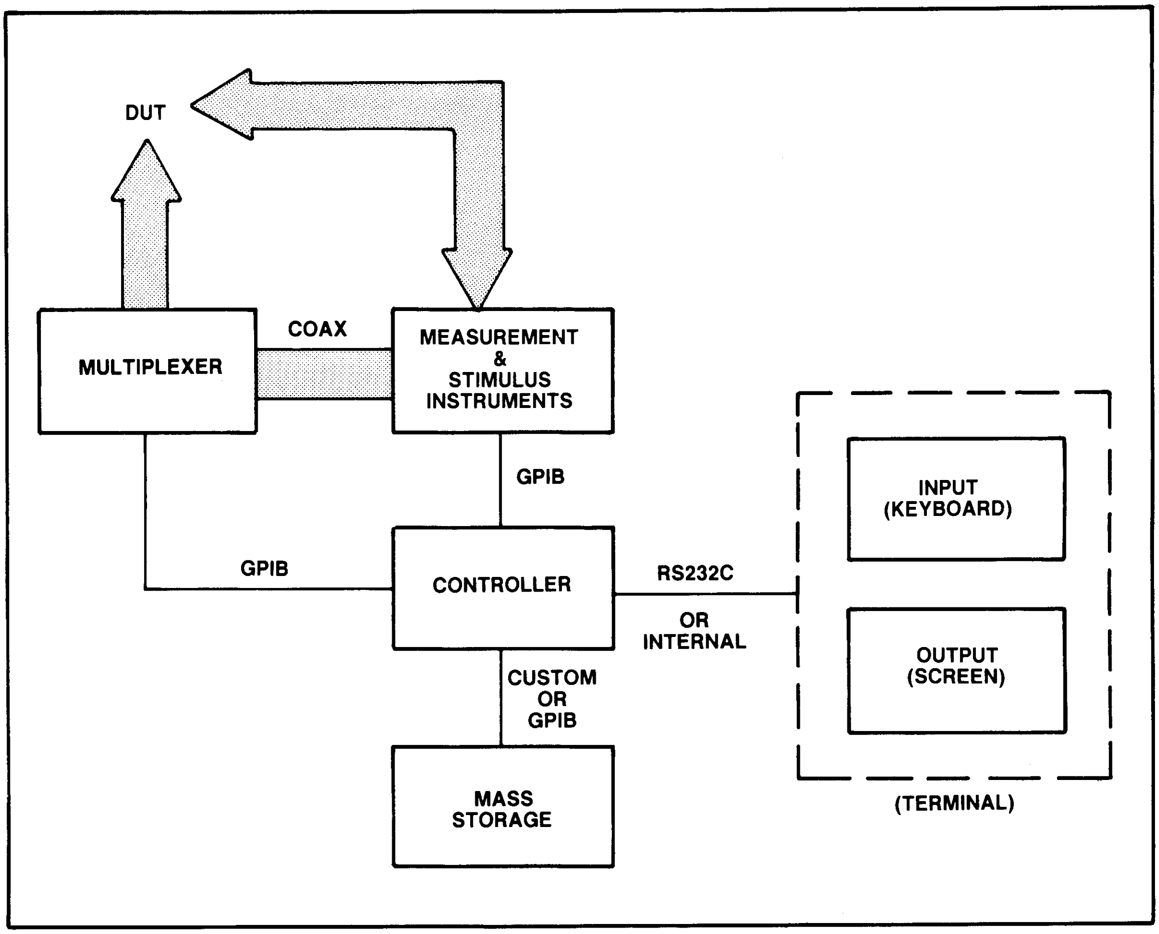


Fig. 2.2. A generalized instrumentation system schematic.

Let's take a moment to look at the considerations a customer may need to take into account regarding each class of device and how it impacts the total system capability. We will begin to see here, and discuss more in the next chapter, that a customer is interested not only in the capability (i.e. specifications) of particular measurement instruments, but of the system as well. And that this system specification is from no means derived by the sum of the specs of the various components.

First, though, let's look at the considerations regarding components.

INSTRUMENTS

With GPIB systems, you have the opportunity to learn many new technical and sales skills; however, all the skills and knowledge that you have used in the past are still required. You must offer test instruments which provide the salient signal characteristics which meet the customer's measurement requirement. Here we are talking about the entire range of products that you have been selling in the past. A system product has a GPIB interface on the back, but there are many other considerations when selecting an appropriate instrument. These include throughput, level of intelligence, compliance with Tek Codes and Formats, etc. These special considerations are discussed in the next chapter in the section "System Considerations." Examples from the current Instruments Division product line include: 7854, 5223, 468, 492P, 7612D, 7912AD, and the CG551AP, and the first wave of TM5000 products.

CONTROLLERS

Considerations include the size of user memory (temporary memory to store programs and data, contents lost when power is turned off), computational speed, interface (communication) speed, ease of programming (language), ease of use and availability of peripherals, and built-ins (such as graphics screen, tape drive, and GPIB port). As an example, let's suppose two customers are going to make different measurements, but they both need a net output (answer) twice a second. The first customer can make a measurement with a DMM and the second with a waveform digitizer. The second customer needs a controller with an interface speed substantially faster than the first since the second customer will be transferring as much as 1,000 times more data over the GPIB in the same amount of time. Therefore, a controller with a much faster interface data rate is required. Tek's GPIB controllers are the 4050 Series and the new 4041.

INPUT DEVICES

For instrumentation systems, the standard user input device is a keyboard; however, these days, User Definable Keys are a great convenience to the operator. UDK's are typically special keys on the keyboard which have the ability to call software routines that are in the controller. Since users can write these routines in any way they choose, the keys are said to be "user definable." The benefit is that a user need only press one key, rather than a whole sequence.

The 4050 Series instruments have their own input and output devices built in, but for the new 4041, any of Tek's terminals is an appropriate input device. These includes the 4006, 4010, 4014, 4024, and 4025 terminals. (Actually the 4041 can be used without a terminal by using the LED display and the optional keyboard.)

OUTPUT DEVICES

The customer may frequently have a need for more than one kind of output device. The standard will normally be a screen for text (such as for listing programs), but may also include graphics capability. In addition, most folks will need some form of hard documentation. On small systems this might be a thermal tape printer. For a normal terminal it will more likely be a hard copy unit or a line printer. For some customers who supply the U.S. Government, the hard copy documentation is an important feature of the system because each piece of gear they ship has to have the calibration results affixed to it. The ability to automate this documentation requirement may provide a huge cost savings. Yet other customers will want a plotter because they need to produce graphs which are print quality for trade journals or which are in color. Examples include the 4631 and 4611 hard copy units, the 4662 plotter, and the terminals mentioned above under input devices.

MASS STORAGE

Customer considerations pertaining to mass storage are: 1) speed--the rate at which data must be stored for each experiment/DUT in semi-permanent form (such as for reliability statistics); and 2) quantity--if the customer needs to store complete waveform records for each test, then a lot of storage is required. There are three typical forms of mass storage these days: a) serial tape cartridges (like the 4050 tape); b) flexible disks; and c) hard disks. The above are in order of speed, quantity, and expense. Hard disks are the fastest, hold the most data, and are the most expensive. Currently Tek's mass storage devices include the 4907 flexible disk file manager and the 4924 cassette tape drive as well as the built-in tape drive in the 4050 Series.

MULTIFUNCTION INTERFACES

Aha! A new category. Multifunction interfaces include signal multiplexers, A/D and D/A converters, and other interfacing devices that help connect a DUT to a measurement or stimulus instrument.

Multiplexers will be important devices in production test and other dedicated system applications where signals will need to be routed to different points on the DUT. The multiplexer can be used also to connect different points to a measurement device or from several DUT's to one instrument (see Figure 2.3).

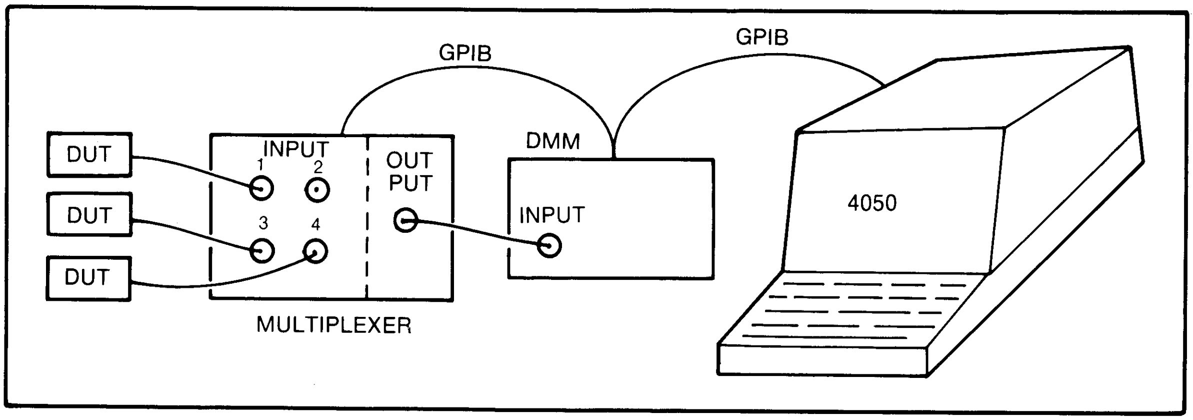


Fig. 2.3. A signal multiplexer is used to switch many incoming signals to one output signal or vice versa.

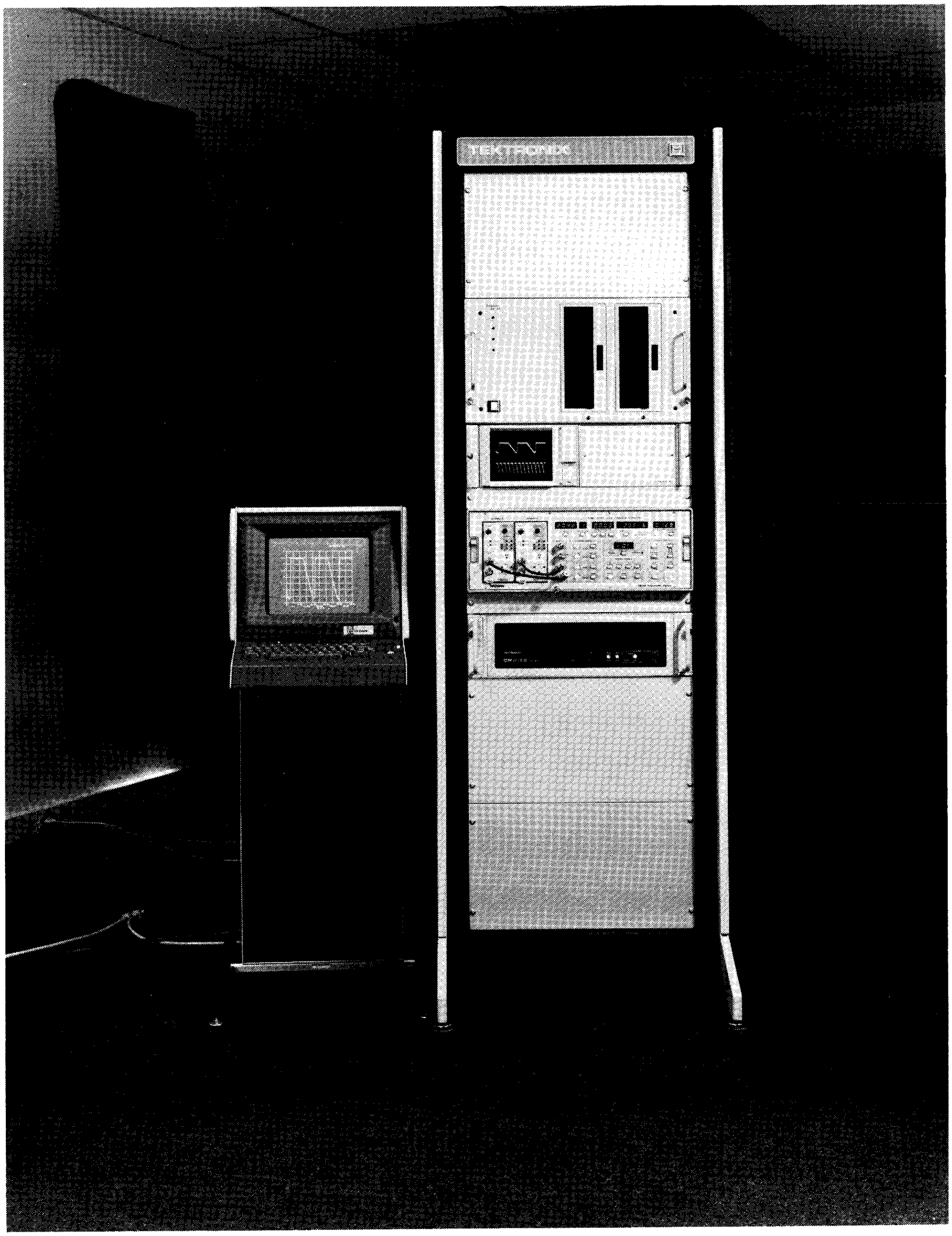


Fig. 2.4. The elements in a WP3100 are different from those in a WP1310, but both systems have the five generic system components.

Several parameters must be considered when selecting a multiplexer including insertion loss, cross talk, maximum voltage, bandwidth, etc. Tek will have signal multiplexers on the market with the introduction of the first wave of TM5000.

The above discussion was somewhat lengthy because I wanted to give you some idea of the myriad ideas that will be in your customers' heads when they start talking about systems. When a customer starts talking about the finer details of mass storage requirements, it is time to bring in the aid of your local IDDSE.

One last example of a system, Figure 2.4, shows a WP3100 with some options. Take a moment to identify the generic components of the system. (Note: The 4010 terminal is typical in that the INPUT and OUTPUT are built into one unit but logically separate.)

SOFTWARE

Stand-alone products by themselves can do what they always have done -- provide specific tools. In manual tests, an operator is always required to set up the instrument(s), apply it (them) to the device under test, make the actual measurement, compute the specific parameter required, and interpret the results. Although we will discuss benefits later, we can see that a system is going to assist the user. Software can take care of many of the above activities.

Just as there are minimum requirements for hardware in a system, there are also minimum requirements for software. They are as follows:

- APPLICATION SOFTWARE. These are the programs and routines written to perform the specific measurement and documentation task that the customer is trying to accomplish. They are normally written by the customer or an outside consultant whom the customer hires.

- IMPLEMENTATION LANGUAGE. This is the high level language (e.g., BASIC) in which the customer programs the application. It is sometimes referred to as the operating system software. It usually includes the first level interfacing software (called drivers) to all outside equipment, plus math packages to ease the programmer's job (remember that even division is a complex operation to a computer). The implementation language is virtually always provided by the vendor of the controller.

- DEVICE DEPENDENT CODE. This is the specific command set of each of the GPIB instruments. Each device has its own set of control words, and these must be learned for the user to have complete command over a unit.

Let's discuss each level of software in just a little more detail.

APPLICATIONS

When the system the customer has decided to purchase is delivered, the real investment in the system is just beginning. Now the task begins of codifying the measurement process in terms the system can deal with; i.e., the customer must program the application. Each customer has a very specific measurement situation which is peculiar entirely to that customer, and might include statistical data needs, documentation needs, go/no-go boundary limits, and computational requirements. The people making the measurement are the best qualified to define the process which the system must execute to accomplish their task. Application software defines this process, and the fact that each customer perceives his or her situation as unique is why "general purpose" application software is not practical. As sales engineers you should not need to get involved in your customer's application software; however you will have application software of your own, called demonstration programs. The advantage to you of these demo programs is that they can exercise GPIB instruments without your having to get too familiar with the language (device dependent codes) of each one. There is a special advantage to your customers as well. If you provide them a listing of the demonstration program, they will have an excellent example of how the various components of the system (languages and hardware) all work together. Such an example frequently makes the difference between customers new to controllers and programmable instruments who are completely mind-boggled, and those who have some experience and training and the confidence to undertake the programming of their application. In addition, the demo programs will sometimes have program segments (called subroutines) which are directly applicable to the customers' needs, and they may want to lift this directly out of your demo and into their applications. Great!

Here is a segment of application software which sets a hypothetical programmable power supply (device address 3) to 15.7 volts, written for a Tek 4052:

```
PRINT @3: "15.7"
```

Although most application programs will be longer than this one, the line illustrates that a user needs to know three things to successfully make a system operate:

1. The specific tasks and in what sequence the system is to perform them (the application).

2. The computer's language (implementation language).

3. The language (commands) that the instruments require to make them function properly (device dependent codes).

The user seldom has to know much about the operation of GPIB since the software (drivers) automatically handles GPIB operation.

IMPLEMENTATION LANGUAGES

Implementation languages (generally referred to as high level languages) such as BASIC or FORTRAN provide the application designer a vehicle with which to more easily program a computer to do a specific task. These languages generally use English words and algebraic equations to specify to the computer what to do. Typically the implementation language will include: a) math routines to interpret and compute complicated (as well as simple) arithmetic expressions; b) first level communications routines (called drivers) to handle interface functions at the hardware level; c) control routines which can make decisions based on other calculations and looping control (computers love to do repetitive tasks); and d) formatting routines to handle input and output tasks for the user (to the terminal) and to mass storage devices.

Tektronix is supplying GPIB controllers in which the implementation language is BASIC. BASIC is a good language to work with, but it is not the only possible implementation language. In case your customers should confront you with some other possibilities, let's discuss some of the alternatives and their pros and cons.

BASIC is a common, easy-to-learn, easy-to-program, <u>interpretive</u> language. A word of caution--although there is an ANSI standard minimal BASIC which defines some of the rudimentary commands, each BASIC tends to have its own extensions (new or modified commands) and subtle syntax (spelling and placement) conventions. An interpretive language is one in which the user types in English-like lines of code (the program) and, when executed, the computer interprets (translates) <u>each line</u>, one at a time, into the machine language that the computer is able to execute. The computer executes one line and then goes on to the next. The disadvantage to this is that if the computer is in a loop (executing the same lines repeatedly), it has to translate and execute this same group of lines over and over. The language has no ability to remember the translated machine code and, of course, the translation process takes time. So the advantages of BASIC are: easy-to-learn, industry familiarity and acceptance, highly interactive, and easy-to-modify programs. The disadvantage is that for some applications, BASIC is too slow. Curiously, the fact that BASIC's do not comply with an overall standard is an advantage in that we can define extensions which allow for extremely flexible, yet easy-to-use, GPIB control and input/output statements. This is true of the 4050 Series and especially true of the new 4041.

FORTRAN, ALGOL, and PASCAL are all compiler languages. Compilers take the entire program that the user writes and translates the entire thing to machine code at one time. The computer then throws out the original user's typed program and the translater program. What is left is a small, neat bundle of machine code that directly executes the user's original intent. Most compilers have some sort of standardization of the language specification. The advantages of compiler-based languages are: reduce space (memory) requirements, speed efficiency, and language standardization. Disadvantages are: more difficult languages and set-up parameters, the need to learn editors and resource allocation systems, and more time-consuming program modification.

There is one other option a user has in writing an application program for a system: assembly languge. Assembly language is really the language of a particular computer or microprocessor, except expressed in mnemonics that a human can more readily understand. (An assembler is a small program that converts the human-understandable mnemonics into machine code.) For example:

```
MOV L,H        00111100
```

The command MOV L,H is a mnemonic instruction which the 8080 microprocessor can execute if given the correct binary representation, which is 00111100. There is a one-for-one mapping between an assembly language mnemonic and its equivalent machine-code binary instruction. The advantages for a customer in programming like this are maximum optimization for speed and memory utilization, complete flexibilty in using a given processor's architectural advantages, and flexibility in adding non-standard hardware directly into the processor's bus structure. The reasons very few folks do this (i.e., the disadvantages) are: the need to know a processor's architecture and detailed operation, many times more time-consuming to write, excessively difficult to modify after project completion, and very detailed documentation is required. Typically, the only situation where a customer would program an application using assembly language would be in a dedicated production environment where each second saved in a test cycle representes many dollars saved, and where the system is to remain unaltered for several years.

A word about firmware. Firmware is software which has been placed in read-only memories (ROM's). There is no real difference between firmware and software. They compute the same, they control the same, they both have bugs (errors), and they look identical on paper. However, because firmware is committed to ROM's, it is more difficult to change (which is usually done by replacing the ROM's). Our BASIC is implemented in firmware and, therefore, has the advantage that when you turn the power on to the computer, the software (i.e., BASIC) is ready to operate. In a computer that does not use firmware, a user must first load the language from a mass storage device into the computer's volatile memory (RAM - read/write memory). From this point on it is all the same. (Note: This is also true for GPIB instruments. Most instruments have firmware which controls their functions and operation but it is all just software that has been committed to ROM. Not many instrument designers want to put a mass storage device inside an instrument, although it has been done.)

DEVICE-DEPENDENT CODE

All instruments which communicate across the GPIB will have some sort of specific command set unique to each device. For example:

```
FREQ 10.7M;SPAN 500K
```

will put the 492P into a center frequency position of 10.7 MHz with a resolution of 500K Hz/div. Sending the 7854 this same sequence of characters has an altogether different effect -- namely to give the user an error message and to sit idle. Therefore, the user must learn the commands (device-dependent code) for each instrument to be used. Tektronix has made this chore easier by defining common message structures and error code generation schemes (see Tek Codes and Formats in Chapter IV). Nonetheless, a spectrum analyzer is a spectrum analyzer and a scope is a scope and never the device-dependent codes shall meet. (That is, the actual commands themselves will be different for each device.)

The codes which are device-dependent will always be the same for a specific product, but the computer statement which sends them to the instrument will depend on the implementation language being used. For example, suppose we want to send "FREQ 10.7M" to a 492P with with GPIB address 7, then:

```
(1)  200 PRINT @7:"FREQ 10.7M"
(2)  200 PUT "FREQ 10.7M" INTO @Ø,39
```

where (1) is the application line that would be used for the 4052, and (2) is what it would look like for SPS BASIC (the 39 at the end of the statement is the absolute GPIB listen address, 32+7 = 39). To do it in FORTRAN would look something like:

```
     WRITE (39,50)
50   FORMAT (' FREQ 10.7M'/)
```

and to send the same message in assembly language would look like a small novel filled with gibberish.

People may sometimes ask, "Are your instruments programmed in BASIC?" You can now see that this question does not make sense. BASIC is a high level implementation language used in many controllers. The instruments never see the implementation language of the controller (BASIC, PASCAL, HPL, FORTRAN, etc.); they see only their device-dependent commands which are sent by the controller. (HPL is the quasi-BASIC language used by the HP 9825.)

By the way, the manner (format) in which data comes out of the instrument is also called device-dependent code and is essential for the user to know. Since the GPIB standard doesn't define this, it is possible for data to come out as ASCII, EBCDIC*, binary, BCD, etc. Again, the Tek Codes and Formats Standard has taken steps to ensure that there is compatibility and consistency between Tek products, thereby making the customer's job a lot easier when multiple Tek products are used.

A last note about software as a whole. For the user, the software reference manual is the single most important part of the product he has to work with during the design. Unlike hardware (a scope, for example), a software package is totally useless without a well-written reference manual, because, without a manual, the user has no idea what the commands are. So you have something new to sell - the fact that Tek writes and produces some of the most thorough, easily read reference manuals in the industry. Don't take this lightly; anyone who has tried to work through some of our competitors' reference manuals knows just how frustrating and time-consuming working with a poor set of manuals can be.

BENEFITS

Every customer has a unique set of reasons for buying anything, but with systems it is especially important to home in early on the specific features and benefits in which a particular customer is interested. The reason is simple: there are so many features and so much flexibility in any given system that to describe or demonstrate the reasonable combinations of just the significant features may literally take days or weeks. Those of you who try to sell a 7854 by demonstrating every button on the waveform calculator know how time-consuming this is. With systems, the problems will just get worse and the chances to bungle in front of a customer increase substantially. Therefore, probably the best way to sell systems (or anything else) is to sell benefits that solve specific customer needs, and when doing

*See Glossary for definitions

a demo, generally stick to the demo tape provided. Doing extemporaneous programming to attempt to solve a particular customer situation is very dangerous. It invites trouble, demonstrates lapses in your product knowledge, and leads to further, "Well, then let me see you do this."

So what are frequent system benefits? First, in many cases you may not sell an entire system, but just a GPIB instrument. However, even if you sell the whole thing, some of the most important benefits to the customer will be in your selecting measurement equipment which meets the technical specifications of that customer's measurement requirement. First and foremost, the customer has to make a needed measurement.

Systems proper, however, do have added benefits. These are to:

Reduce labor costs. This most often is a result of the speed of the automated system. For example, if prior to the system five people were required to test all of the parameters of a device, then conceivably with one system and one operator all the same measurements could be made, and in substantially shorter time. Another way to reduce labor cost is for the system to do entirely an operation that had been done by hand. For example, some labs used to take photos of scope traces, hand digitize the waveform, and then load this data into a computer for processing. With a waveform digitizer and a computer, the activity of hand digitizing and typing data into a computer is replaced. Another way to reduce labor cost is by lowering the skill level required to make certain measurements. Be careful with this one, as a system may require relatively expensive programmers and maintenance personnel. However, if the system is to go in a dedicated production environment, frequently actual measurement tasks can be made by folks who have little understanding of the system, of software, of instrumentation, or even of the device being tested.

Release engineers from drudgery and increase the use of their creative skills. The rapid growth of the electronics industry has caused a shortage of qualified electrical and electronic engineers. Measurement systems have the potential to increase the effectiveness of engineers by relieving them from doing routine measurements. For example, an engineer designing a filter will periodically need to test the design over a specific frequency range to see how the design is progressing. Depending on the resolution requirement, the frequency range, and available equipment, the designer may need to make fifty or more measurements, each requiring setting a generator, making a reading, and plotting the data. Not only is this extremely boring, it is time-consuming and, therefore, expensive. An appropriate combination of acquisition and stimulus equipment coupled to a computer and a short program can literally get the job done in seconds. The engineer has more time to design and the customer gets his product out the door that much faster.

Provide insight by coupling analysis with measurement. Many times a system is able to provide information to a user in a different form from that provided by measurement devices alone. And sometimes this new way to look at the data makes the difference between an unsolved problem and a solved one. The system provides two capabilities that a user may never easily have had before: 1) complex mathematical computation capability, and 2) data storage and statistical analysis capability. As an example, suppose you are trying to electrically harden equipment to withstand large electro-magnetic pulses (like signals generated from lightning). To do this the customer must characterize the frequency vs. amplitude waveform of the equipment-under-test when lightning strikes. Because of the single-shot nature, only a waveform digitizer coupled with a controller able to compute the FFT (Fast Fourier Transform) will be capable of providing the necessary data. Or, as another example, folks who use large turbo-machinery need to monitor the turbines for bearing wear and damage. They do this by digitizing the output of transducers mounted on the bearings and then taking the FFT to look at the spectral output. When the amplitudes exceed a certain level, they know it is time to replace the bearings.

One last example: suppose a customer has installed a system to QC a particular GPIB compatible counter. They decide to check the accuracy by generating different frequencies on a frequency synthesizer and log the output of the counters onto the disk. At the end of the week they average the data and realize that for 25% of the counters, they are getting inaccurate reading at one particular frequency. By giving that data to the engineers, they are able to find a component problem that might otherwise have gone undetected.

Reduce human error and increase measurement accuracy and consistency. Particularly in a production test environment, the system benefit of being able to reduce measurement error is especially easy to claim. Measurement errors are made when: 1) a less-skilled operator does not read the instrument properly, 2) the operator is bored and reads the instrument in a sloppy manner, 3) the operator cannot visually resolve the output of the instrument simply due to visual limitation; or 4) multiple operators making the same measurement use slightly different methods to do so. There is little question that a system can make these measurements more accurately, more consistently, and generally faster, without ever getting tired or bored. The net result is usually a consistent, high-quality output from the operation which makes your customer more productive and more competitive.

CHAPTER II SELF TEST

Answer the following questions. Then compare your responses to the answers in Appendix B at the back of this booklet.

1. What are the five minimum generic hardware components that will be found in any instrumentation system?

2. Draw a block diagram of the components and how they are connected to one another.

3. List two specific examples by product number of each generic component from the Tektronix product line. (One product may be found in more than one category.)

4. What is the sixth system hardware component that is likely to be found in most ATE and production test environments?

5. If a customer needs to generate print-quality graphics from the results of an experiment, what kind of specific product would you recommend?

6. List three considerations a customer may take into account when selecting a controller for an instrumentation system? (Do not include price.)

7. List two basic classes of mass storage devices.

8. List the three minimum software components required in an operational instrumentation system.

9. In a couple of sentences, describe the difference between a compiler and an interpreter.

10. Give two reasons why it is unlikely that a customer would implement a system using assembly language.

11. As far as results are concerned, would you say that firmware and software are significantly different? Why?

12. Why are manuals important to a system implementor?

13. List three benefits which are unique to systems and the system feature which allows this benefit to be realized.

III. THE GENERAL PURPOSE INTERFACE BUS

The concept of instrumentation systems began to evolve in people's minds during the 1960's when computers started to invade business and scientific domains. So some folks thought that putting together a computer and a digital instrument would be a practical idea in high volume applications. The reason for the "high volume" qualification is that the cost of interfacing was so high that the payback had to be high also.

The problem with these early systems was that custom interfacing was required. The problems with custom interfacing are:

1. The designer's need to learn the bus architecture of the computer.
2. Need to design specific interface (tailored, naturally to the particular instrument's requirements).
3. Need to document interface.
4. Need to train maintenance personnel.
5. And, the worst, the need to start the whole list again if a new instrument is added.

Figure 3.1 represents the problems of the old way of interfacing.

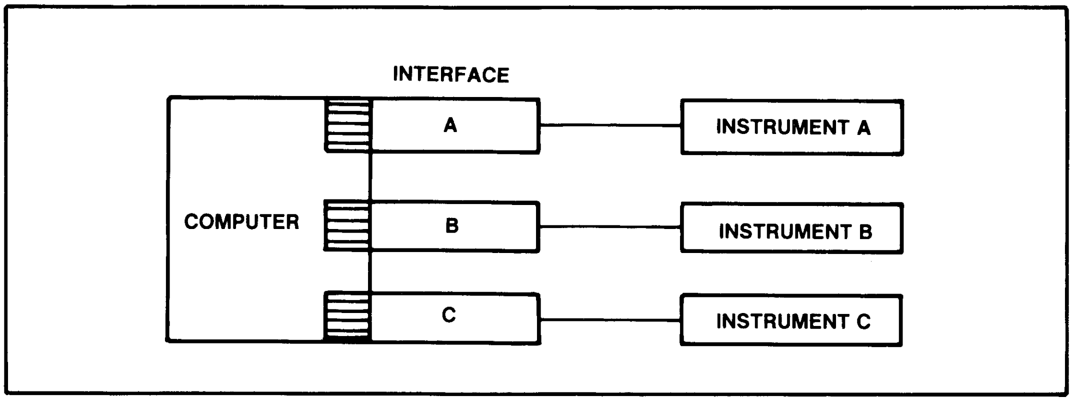


Fig. 3.1. Problem before GPIB: 1) too many lines; 2) each interface different; 3) point-to-point connection only.

The concept of a general-purpose interface bus evolved at Hewlett-Packard to solve some of these problems. The IEEE 488 1978 standard is what resulted.

IEEE 488 DEFINITIONS

The IEEE 488 1978 is a booklet published by the IEEE which defines an interface standard intended for small instrumentation systems. The interface designs based on this standard are called, variously, the GPIB, the HP-IB (HP's implementation of IEEE 488), IEC 625-1 interface (an international standards group), and the IEEE 488 interface.

The IEEE 488 standard defines the mechanical, electrical, and functional characteristics of the interface. This leaves the operational characteristics to be defined by the instrument designer.

(Just because a manufacturer claims that a product is IEEE 488 compatible does not necessarily mean that it actually complies to the standard. Tektronix extends every effort to ensure that all GPIB products do, in fact, comply precisely with the standard.)

MECHANICAL

The standard defines precisely the nature of the 24-pin connector which is mounted on each instrument. The bus is defined as a standard cable with the proper mating connectors and 24 wires (shown in Figure 3.2). (The IEC 625-1 European standard uses a 25-pin connector much like an RS232C connector.) Sixteen of the wires carry signals; the remainder are grounds and the cable shield.

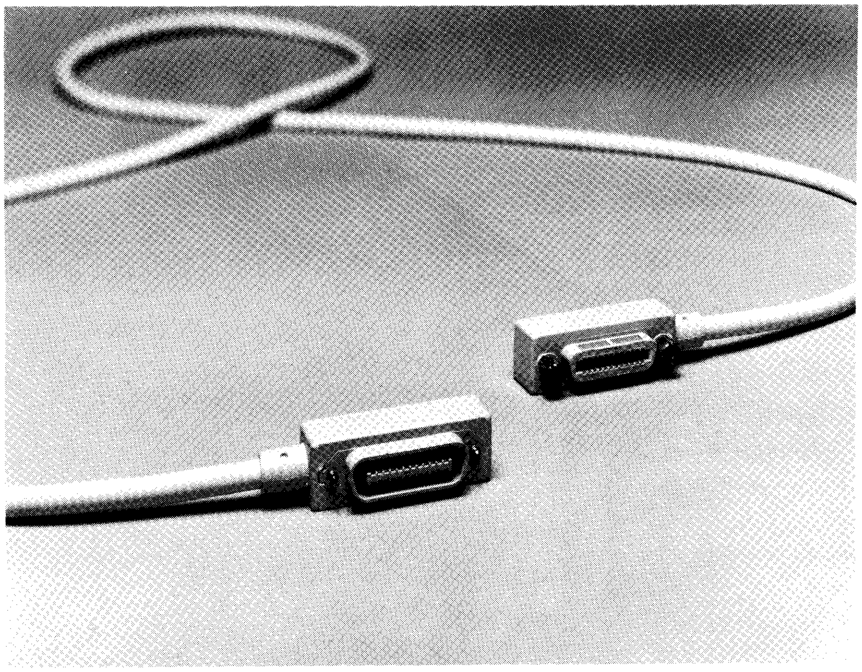


Fig. 3.2. The GPIB connector.

ELECTRICAL

The standard defines the driver and receiver circuits for the interface. The voltage and current values required at connector nodes are based on TTL technology (voltage not to exceed 5.25 V). And logic levels are also defined, whereby a signal line is asserted (logic true) when in a low-voltage state (≤ 0.8 V), and unasserted (logic false) when in high-voltage state (≥ 2.0 V) and is called negative true logic.

FUNCTIONAL

The functional elements of the GPIB are problably the least understood, yet some of the most important, aspects of the standard as far as systems use and development are concerned. However, before we can discuss the functional elements of the GPIB, we need some signal line definitions.

Figure 3.3 shows a "typical" GPIB instrumentation configuration, it also breaks out the GPIB into three sub-groups which are called: the DATA bus, the INTERFACE-MANAGEMENT bus, and the TRANSFER (or handshake) bus.

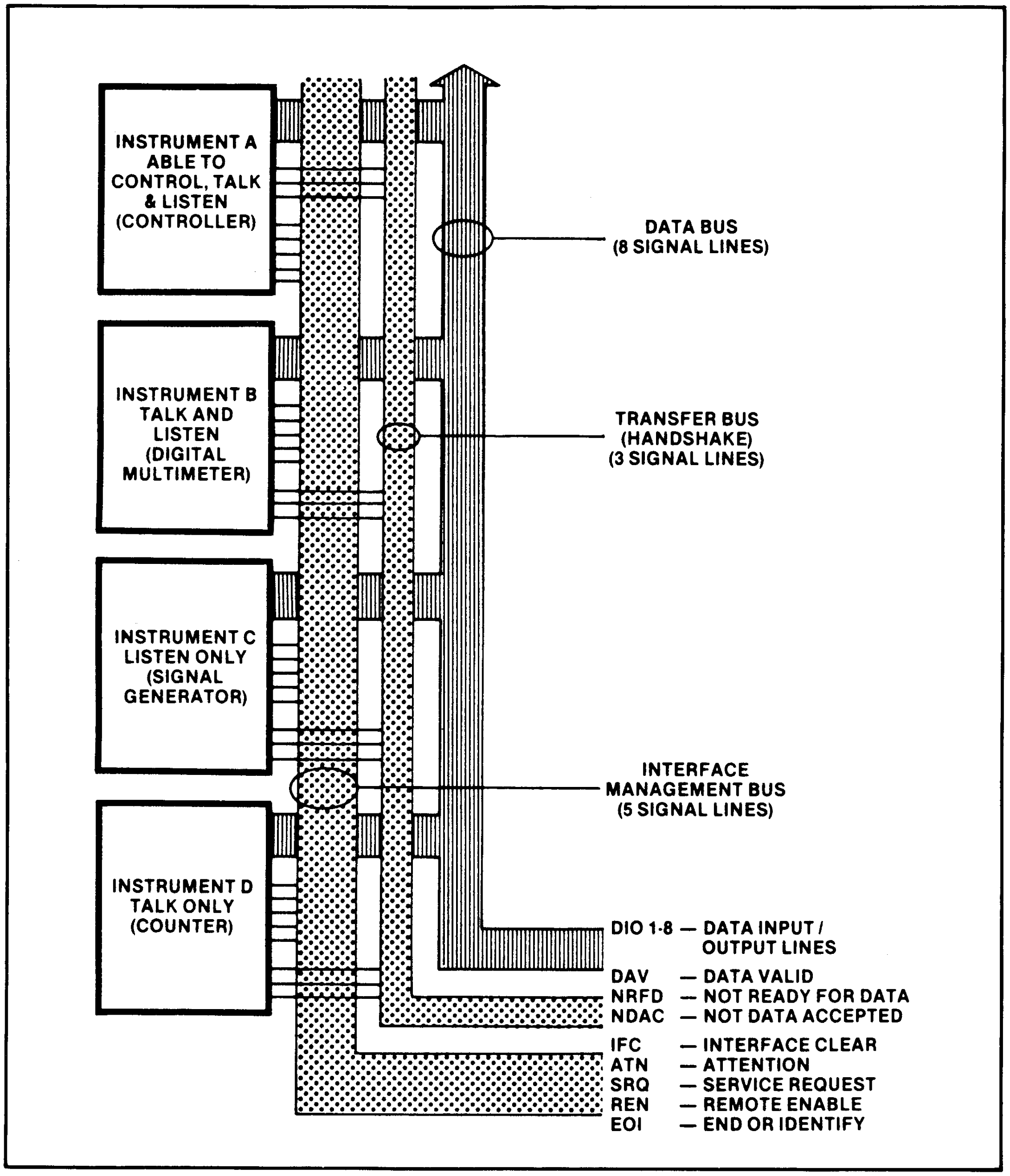

INSTRUMENT A ABLE TO CONTROL, TALK & LISTEN (CONTROLLER)
INSTRUMENT B TALK AND LISTEN (DIGITAL MULTIMETER)
INSTRUMENT C LISTEN ONLY (SIGNAL GENERATOR)
INSTRUMENT D TALK ONLY (COUNTER)
DATA BUS (8 SIGNAL LINES)
TRANSFER BUS (HANDSHAKE) (3 SIGNAL LINES)
INTERFACE MANAGEMENT BUS (5 SIGNAL LINES)
DIO 1-8 — DATA INPUT / OUTPUT LINES
DAV — DATA VALID
NRFD — NOT READY FOR DATA
NDAC — NOT DATA ACCEPTED
IFC — INTERFACE CLEAR
ATN — ATTENTION
SRQ — SERVICE REQUEST
REN — REMOTE ENABLE
EOI — END OR IDENTIFY

Fig. 3.3. The GPIB is made up of a data bus, a transfer bus, and an interface management bus. Devices connected to the bus can be categorized as Talkers, Listeners, Controllers, or combinations thereof.

DATA

The data bus has eight of the 16 signal lines of the GPIB. These lines are bidirectional, carrying a data byte either to or from a given instrument. As with all the signal lines, these lines are shared among all of the devices connected to the system, which is why the GPIB is called a bus. Therefore, information that is being made available to one instrument is, in fact, available to all instruments simultaneously. Because the lines are shared, the controller must designate which instruments are to use the bus at any one time. This is discussed below.

TRANSFER

Three lines (NRFD - Not Ready For Data; DAV - Data Valid; and NDAC - Not Data Accepted)) are used for a handshake to ensure that data is properly sent and received. The handshake sequence is completely defined by two of the interface functions (SH and AH, see Table 1) specified in the IEEE-488. Therefore users never need to concern themselves with the operation of the handshake bus. If it is not working correctly, then one of the devices on the bus needs to be repaired.

There are a couple of characteristics of data transfers, however, that are worth remembering. A complete cycle of all three handshake signals occurs for each byte of information transmitted on the data bus. This insures that the GPIB is fully asynchronous. With an asynchronous bus, both slow and fast devices can communicate on the bus. The rate of data transfer is determined by the speed at which data can be transferred to/from the slowest device involved in the transaction. Notice that the transfer rate is not determined by the speed of the slowest device connected on the bus, but by the speed of the slowest device involved (addressed) in the current transmission.

So, the data rate automatically increases whenever the slower devices are not involved in the transaction. The transfer bus is so fully asynchronous that even if an instrument should radically change transmission rates in the middle of a single message, the data will not be lost.

INTERFACE-MANAGEMENT

The remaining five signal lines are used for managing the bus. The most important of these for understanding how the GPIB works is the attention line, ATN.

Since all the instruments share the GPIB lines, the controller must designate which instruments are to use the bus for talking and listening at any given time. So, one major responsibility of the CONTROLLER is to specify (address) one TALKER to send data, and one or more LISTENERS to receive data. The controller does this by sending specific interface messages over the data bus. Interface messages, which are completely defined by the IEEE 488 standard (summarized in Table 1), are distinguished from device-dependent messages (identical to device dependent codes described in Chapter II) whenever the controller asserts the ATN line. To reiterate, only the controller can assert the ATN signal and, when it does, the information present concurrently on the data bus is to be interpreted by the instruments as an interface message. If ATN is unasserted then the information present on the data bus is data or commands (device-dependent messages) being sent to or from instruments and/or the controller.

A complete summary of allowable interface messages is provided in Table 1. The messages themselves are listed in the third column. The ten interface functions which are able to acknowledge these messages are listed and described in the first two columns. We'll come back to these in just a moment.

The other four interface-management signals are:

- Service Request (SRQ) - Any instrument can assert this line at any time to request service from the controller. It is the controller's responsibility to decide when to acknowledge the SRQ, and then to determine which instrument has requested service and why.

- End or Identify (EOI) - for most manufacturers (HP excluded), this line is asserted by the talker concurrent with the last data byte in a given message. In this manner it indicates to the listeners and the controller that it has completed sending a particular device-dependent message. (HP simply does not use this line and generally sends an ASCII line-feed character as the last data byte. See Chapter IV.)

- Remote Enable (REN) - The controller asserts this line to tell the instruments that they are going to be controlled remotely (i.e., from the computer). When REN is asserted and the instrument is addressed, this signal causes the instruments to ignore inputs from the front panel and to accept information sent to them over the GPIB.

- Interface Clear (IFC) - The controller asserts (actually pulses) this line to place all the device interfaces in a known quiescent state, e.g., initialization or power-up state. The devices must respond to this signal even if they are not addressed to be listeners.

TABLE 1. INTERFACE FUNCTIONS AND CORRESPONDING COMMANDS

FUNCTION	DESCRIPTION	ASSOCIATED MULTILINE (8-bit INTERFACE MESSAGES
Command and address are merged in these 8-bit messages		
Talker (T)	allows instrument to send data	MTA My Talk Address MSA My Secondary Address
Listener (L)	allows instrument to receive data	MLA My Listen Address MSA My Secondary Address
Source Handshake (SH)	synchronizes message transmission	none
Acceptor Handshake (AH)	synchronizes message reception	none
8-bit messages comprising commands		
Remote-Local (RL)	allows instrument to select between GPIB interface and front-panel programming	GTL (Go To Local) LLO (Local Lockout)
Device Clear (DC)	puts instrument in initial state	DCL (Device Clear) SDC (Selected Device
Device Trigger (DT)	starts some basic operation of instrument	GET (Group Execute Trigger)
Service Request (SR)	request service from controller	SPE (Serial Poll Enable) SPD (Serial Poll Disable)
Parallel Poll (PP)	allows up to eight instruments simultaneously to return a status bit to the controller	PPC (Parallel Poll Configure) PPU (Parallel Poll Unconfigure) PPE (Parallel Poll Enable) PPD (Parallel Poll Disable)
Controller (C)	sends device addresses and other interface messages	UNT (Untalk) UNL (Unlisten) TCT (Take Control)

Now, let's get back to the functional elements of the IEEE 488 standard. The standard defines ten functions or tasks that a given instrument's interface may perform (Listed in Table 1). These functions are activated or deactivated by various interface messages sent by the controller to the instrument interfaces. All of these functions are optional, allowing the designer of GPIB-compatible instruments to choose only those suitable for the particular instrument. For example, a simple power supply might only require Listener and Acceptor Handshake functions. With these, the power supply could receive commands from the controller regarding what voltage to put out. Another example is the 468. It requires only the Talker, Source Handshake, and Acceptor Handshake functions. Since the 468 does not acknowledge any device-dependent commands, it does not ever need to be a listener. However, to acknowledge its talk address, the 468 does require the Acceptor Handshake function. With that it knows when to start transmitting its data.

More complex GPIB instruments incorporate a half dozen or so of the functions, but will generally exclude the controller function. However, you and your customers should check to be sure that, for a given application, the desired functions are incorporated. Many instruments are labeled "IEEE 488 Compatible" or "GPIB Compatible," but in the extreme, the label could only mean that the instruments have the standard connector (with termination) and none of the functions implemented!

Further, even though an instrument may be able to perform the general function required, the user needs to know the capabilities of the instrument with regard to that function. Within each function, the IEEE 488 standard defines several levels of capability. (These functional subsets are frequently listed in a user's manual to define instrument capability: e.g., AH1, SH1, L2, T2, RL0, DC1, C0, etc. The numbers following the function abbreviations describe a specific interface capability. A 0 means the function is not implemented.) The prospective buyer/user should check with the instrument's vendor to be sure that the instrument can perform as needed.

A TYPICAL DATA TRANSFER

Now that the various signal lines, message types, and functions have been explained, let's see what happens on the bus when, for example, a simple power supply is programmed to put out 15.7 V. First, the controller asserts the attention line and sends out the listen address of the power supply (the user will have set that address with the switches on the back panel of the power supply). At this point, the controller is sending interface messages, and all the devices are receiving the data by handshaking with the controller. Only the device whose address switches match the listen address sent by the controller will become a listener. Other devices ignore this interface message (see Figure 3.4). Next, the controller unasserts the attention line, and makes itself a talker.

Now, a device-dependent message is sent to the power supply, which at this point is a listener. In this example, the device-dependent message consists of the four characters "15.7" coded in ASCII (Figure 3.4c). Note that the End or Identify (EOI) line is asserted with the last character, telling the power supply that the message is through and can now be executed. The power supply then puts out 15.7 V. Figure 3.4 portrays graphically the entire message sequence just described as generated by a 4052.

The simple program developed by the user (who wanted the power supply to put out 15.7 V) enabled the controller to do all that was required.

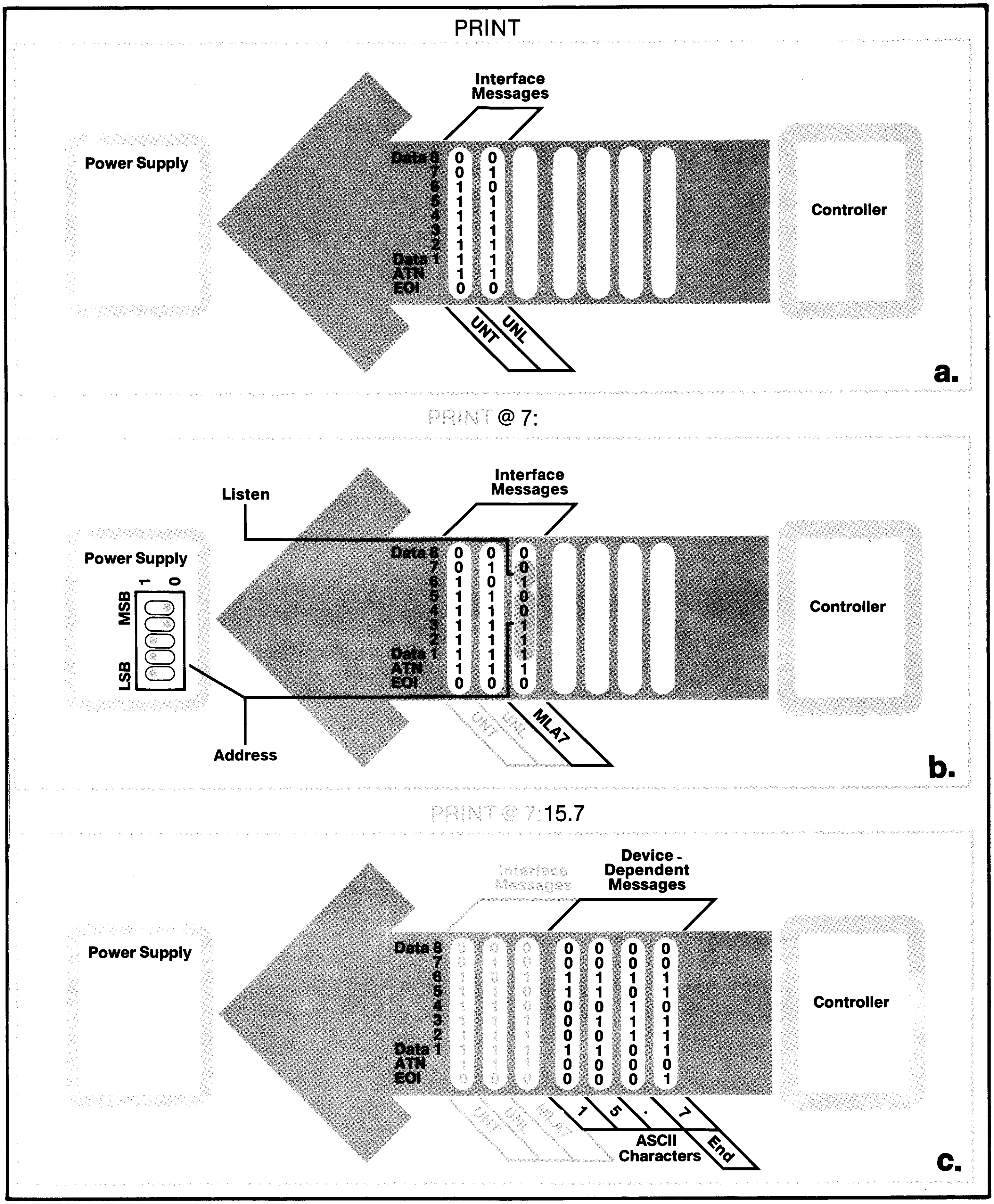


Fig. 3.4. This sequence shows that first the interface message is sent (a,b) and then is followed by the device-dependent data (c). The combined message illustrates the activity of the ATN and EOI lines.

GPIB HARDWARE CHARACTERISTICS

There are some hardware characteristics of the GPIB which fall out from the mechanical and electrical specifics but which were not mentioned earlier. Because of the impact that these characteristics may have on your customers' systems, they are worth remembering.

- Cable length of up to 20 meters with a device load for every two meters of cable. This implies that the GPIB is intended for bench-top or rack-mounted systems, not for intra-building systems.

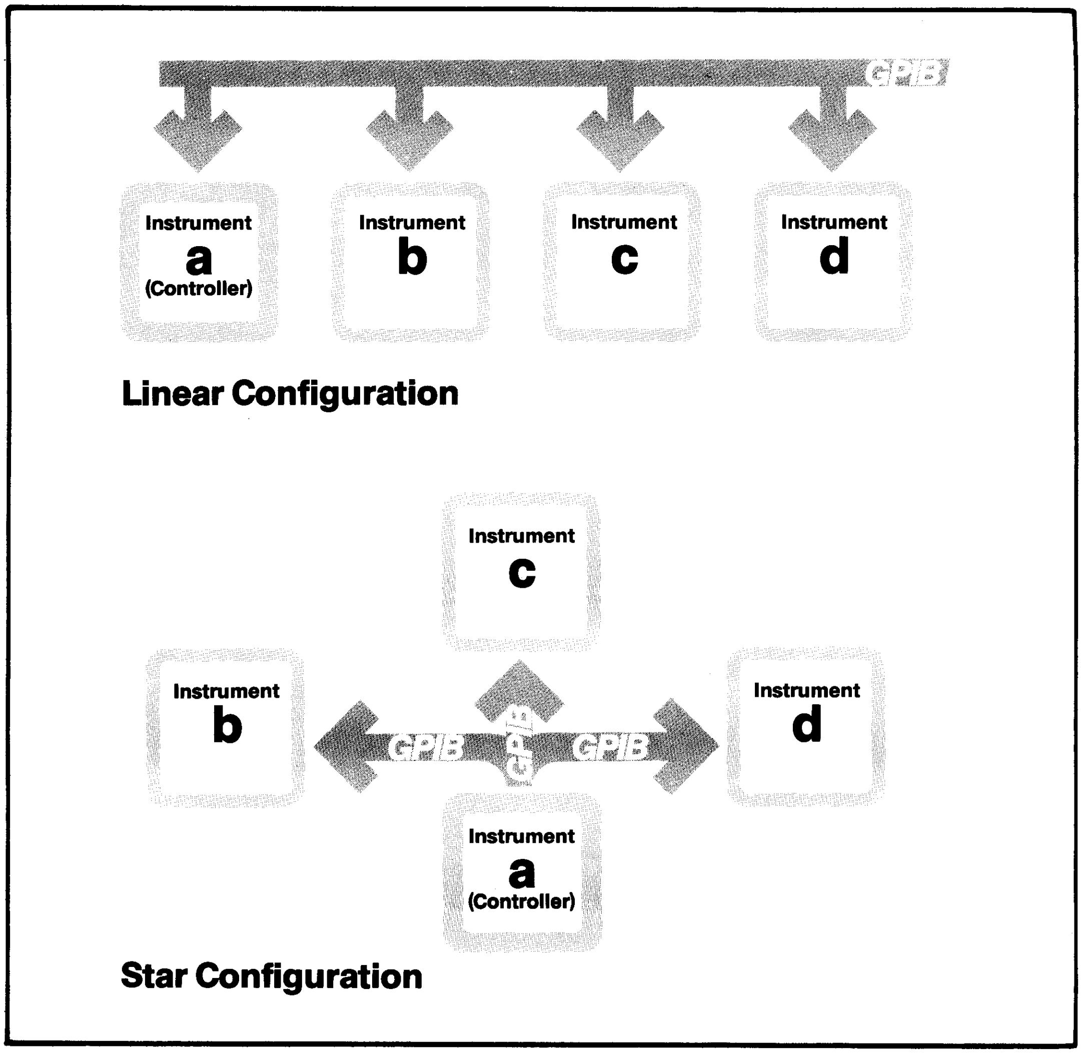


Fig. 3.5. The GPIB, with its standard connector, gives versatility in system set-up.

- Up to 15 devices (1 controller and 14 instruments) may be connected in linear, star, or combination configurations (see Figure 3.5). This allows for greater flexibility than conventional point-to-point interfacing like that which we discussed at the beginning of this chapter.

- Maximum data rate of up to 250 kilobytes/second over a distance of 20 meters (with 2 meters per device), or 1 megabyte/second over a distance of 15 meters, tuned up. See IEEE standard 488 1978 for details. Obviously this specifies maximum data rates and for any given combination of devices the speed of transfer will depend on the instruments involved in the transmission. We'll discuss throughput in the next section.

- More than two-thirds of the connected devices must be powered-up in order for any GPIB communication to occur. This is worth mentioning for two reasons. If you are doing a demonstration of just one programmable product and the controller but have three instruments connected to the system, then it will be necessary to turn on at least two of them. Also, your customers may have fairly sizable collections of equipment all connected together on the GPIB. However, during a particular test or experiment, the user only needs a couple of devices. If he doesn't disconnect the other units, he will need to turn on more than two-thirds of them.

GPIB-COMPATIBLE

The term "GPIB-Compatible" has different meanings to different people. In this section we will discuss what GPIB-compatible means technically and also introduce the concept of "system-compatible." Finally, we will work with the idea of system specifications and why these may be intrinsically different from a summary of the specs for all the devices.

HARDWARE VS. SOFTWARE

As we learned earlier, the term GPIB-compatible, in the extreme technical sense, only means that a device has the GPIB connector attached to the box. It may have no wires connected to the inside of the box. However, this is not very realistic.

A manufacturer who wants his unit to communicate will have implemented some subset of the ten interface functions that we described earlier. At the absolute minimum, a manufacturer will have included the Acceptor Handshake function and either the Listener function or the Talker plus Source Handshake functions. With these minimum functions implemented a device will be able to 1) handshake properly, 2) respond to its ADDRESS, and, 3) either send or receive some kind of data. That is a lot more than "it's only guaranteed not to smoke!"

There is a catch, however. I just said that a device is able to either send or receive "some kind of data." The device-dependent messages, the data, are in no way defined by the IEEE standard. Unfortunately, some manufacturers have been pretty creative in how they define and structure the messages that control their instruments. For example, the message "2OR5S*" may mean, "Send a voltage measurement in range 5 at a rate of 20/second continuously." The good news is that this message is in ASCII and not EBCDIC* as it might have been.

*See Glossary

The point I want to make, however, is this: if we have a product that truly conforms to the IEEE 488 standard, along with adequate documentation (about 70%), a completely flexible GPIB controller and software driver (about 75% of those with GPIB capability), and a creative and well-versed software engineer (hard to say how many of these there are), then no matter what the data format is like, we can guarantee communication will occur. The problems with this are: 1) that it may take an inordinately long time to achieve the communication; 2) internal software documentation will be complex and/or incomprehensible, and 3) software maintenance, when required, will be long, tedious, and expensive. We will talk about Tek's solution to these problems in the next chapter (Tek Codes and Formats).

SYSTEM CONSIDERATIONS

We have already discussed that, when they select equipment for a GPIB system, customers will need to consider the interface functions provided by a given instrument and the ease of use (programming). In this section we will discuss how a given application may require different types of instrument "intelligence," and what these types are.

There are, roughly speaking, four distinct categories of instrument intelligence.

1. non-programmable
2. programmable
3. distributed processing
4. key-stroke programmable

As with any list of this sort, there are gray areas between some categories and different levels of capability within each class. Nonetheless, I think this list will suffice when talking with your customers.

NON-PROGRAMMABLE

An example of a non-programmable instrument is the 468. When told to do so, the 468 sends its waveform. That's it. That is its major capability on the GPIB. Devices like this are good for monitoring activities which only change within a given setting's range, or where a user is constantly present to be interactive with the controls. However, non-programmable products are not good in automated production test environments where an entire multi-faceted test procedure is expected to be handled essentially without operator interaction. As it applies to GPIB, non-programmable refers to the inability to remotely (via GPIB) change any operational settings or controls of an instrument.

PROGRAMMABLE

An instrument is programmable if some or all of its functions or settings can be remotely actuated via GPIB. There are many levels of programmability. In fully programmable products all settings are externally alterable. In partially programmable products, some subset of the controls is remotely programmable. Examples of fully programmable instruments include the 7612D, 7912AD, 492P, CG551AP, and all first-wave TM5000. Examples of partially programmable products include the 5223 and the 7854 (although the mainframe is generally fully programmable, the necessary plug-ins are non-programmable).

The degree to which an instrument is programmable determines the degree to which an application can be more fully automated. Customers who are trying to completely automate a test procedure will need to buy instruments which allow remote control of those functions that they will require to change. Most measurements do not require that all settings be changed and, for this reason, some controls are more likely than others to be programmable or not. For example, the focus control on a scope does not generally need to be changed during the course of a measurement and, therefore, may not necessarily be made programmable by the manufacturer.

DISTRIBUTED PROCESSING

Distributed processing is defined as the capability of an instrument to manipulate (i.e., process) data independent of the act of data acquisition or signal generation. Generally speaking, products which have distributed processing capabilities will be acquisition instruments and not stimulus instruments. What would be the point in doing data analysis in a power supply or a function generator, since the data is normally going in to those devices, not coming back to the controller? An example of distributed processing would be the 7912AD's capability to normalize its data. Remember the 7912AD raw data has two trace edges and may have some horizontal addresses with no data at all. The 7912AD has internal routines which interpolate for non-existant data and which compute one single trace out of the original two. Another example is the 7854's ability to compute rise-time, pulse width, integrated waveform, signal averaging, etc.

For the most part, the distributed processing capabilities of a given instrument will provide the kind of computations that we would expect most customers to make anyway. This is the benefit to the customer. Because the processing is done in the instrument, the user is not required to program that computation in the main controller. In waveform products, the distributed processing capability might include the calculation of a waveform parameter (such as P-P, or risetime), and sending this single value will suffice, rather than sending the entire waveform; therefore, data transmission is much faster. Programming becomes easier, calculation speed is usually faster (because the routines are written in assembly language in the instrument versus BASIC in the controller), and the customer will be able to use the device more effectively (the routines in the instrument may trigger use of them). The most important benefit is usually ease of use; the unit is more friendly.

KEYSTROKE PROGRAMMABLE

Think of your handy programmable calculator and you know what keystroke programmable means. A more rigorous definition is the ability of an instrument to store a sequence of instructions, provide (request) input and output, and choose between alternative actions based on inputs or processed information. The obvious example of keystroke programmable instrument is the 7854. The waveform calculator allows the user to store any sequence of the available keys, to automatically acquire data, and to make decisions (the IF keys) based on that data.

Typically, being keystroke programmable is not an important characteristic of a system device, but important when the device is to be used stand-alone. The ability to store data and make decisions is generally optimized in the system controller. Nevertheless, a particular program sequence may execute substantially faster or easier in the instrument. With a keystroke programmable product plus a controller, you have a distributed network; a network is a system of devices with more than one controller, each processing data independently of the other but occassionally communicating with each other.

A subset of keystroke programmable which may be very important to a system user, is an instrument's ability to be internally sequenced. Internally sequenced is the capability of an instrument to store only a sequence of commands and to execute them either repeatedly or on some sort of trigger (such as Group Execute Trigger). Frequently, the ability to be internally sequenced is restricted to a narrow range of an instrument's command set. For example, there may not be much point in sequencing the horizontal and vertical amplifier sections of an oscilloscope. But it may be very useful to sequence "single sweep arm, acquire data" so that those commands can be repeated very rapidly a dozen times or whatever is required. Doing this decreases transmission time because the commands do not need to be sent repeatedly.

Finally, note that the level of programmability, having distributed processing capability and being keystroke programmable are each product features which are independent of the others. They do not logically form a hierarchy (although some overlapping always occurs). So, when the customers are making their instrument selection, all three categories of instrument intelligence must be considered and judged in light of their measurement and system requirements.

SYSTEM SPECIFICATIONS

When trying to determine the measurement capability of a system, the user has to consider several factors. The first of these you are all familiar with; that is, the measurement capability of the individual instruments which make up a system. There are two other specifications (at least) which are unique to a given system configuration and which, in fact, can never be fully specified until the application is completely developed. These are throughput and, for lack of a better term, software-enhanced calibration.

Throughput is simply the time it takes to complete one cycle of a major activity of the system. This doesn't necessarily mean that it has to be one complete cycle of the full application of the system. An example: let's say a 5223 is coupled with a 4052 and an R08 Signal Processing ROM pack (which includes the FFT) to monitor the vibration of the bearings of a large turbine. The system process is to capture a single record of data, transfer the data to the 4052, compute the FFT, check to see if any amplitude is greater than a predetermined threshold level, and sound a buzzer if it is. This process constitutes one cycle of the system application, and the time it takes to complete one cycle is the throughput. Although this application is not time-critical and, therefore, throughput is not a particular customer concern, let's look at the factors which cause the throughput time to increase.

Each subprocess (i.e., capture record, transfer data, compute FFT, do comparison check) within a given application cycle takes some amount of time to complete; typically, these times add to total throughput time in a serial fashion. Taking our example above, none of the activities we described could have occurred until the previous activity had finished. So, recording the data takes time, added to the time to transfer the data, added to the time to compute the FFT, etc.

It is because these processes add serially that a given throughput specification cannot normally be known prior to full application development. However, throughput rates that we can provide will include 1) the time it takes to transfer data out of an instrument (and to a given controller), and 2) the amount of time it takes to do complex data processing (like the FFT). Production test applications will be the most time critical because they will need to justify the expense of the system in terms of the increased pieces tested/hour. For waveform digitizers or spectrum analyzers, the largest time consumers will be data transfers (because of the large quantity of data) and numerical processing (for the same reason). For counters, multimeters, and most stimulus products, the factors are in test set-up, measurement time, and in data processing. Both of these assume that the applications are in more or less completely automated environments. If this is not true, then the single overriding time consideration will be operator interaction.

Software-enhanced calibration is the capability of an instrumentation system to increase the accuracy or response characteristics of a particular instrument beyond its individual specifications. For example, if a particular digital oscilloscope has a bandwidth of 100 MHz, then we can, through software, characterize the roll-off of that unit, given that we have a leveled sinewave generator good to 500 MHz. In the future, when we acquire data we can map the original data through a correction program, the output of which is a roll-off corrected signal, and thereby effectively increasing the bandwidth of the original scope. These digital filtering techniques are by no means simple procedures, but may be worth discussing with your most technically competent customers.

A simpler example would be when the user has a programmable function generator and needs a leveled sinewave through some frequency range with greater accuracy than is specified. As long as the output characteristics are consistent (i.e., non-random) and, given either a scope or a digital multimeter with the desired level of accuracy, a correction table could be generated that provides a different amplitude command for each frequency to the function generator to force it to generate the proper output. Again, these techniques are not simple, but they are capabilities that stand-alone instruments cannot provide.

REVIEW

Let's take one last brief look at some of the considerations that you and your customers will need to ponder when selecting instruments for a particular application.

Does the instrument:

- Execute some or all front-panel functions (i.e., is it partially or fully programmable)?
- Request service and respond to status inquiries?
- Process measurement results before sending (i.e., data reduction of analysis, distributed processing)?
- Store and execute mini-programs to be triggered through GPIB (i.e, key stroke programmable or internally sequenced)?
- Execute self-diagnostic programs?
- Accept English-like commands?
- Check the syntax of commands before executing? And with what constraints?
- Check the validity of what the instrument is being programmed to do and then do nothing if it can't execute properly?
- Follow Tek's Codes and Formats standard?

And above all else:

- Does it meet the technical specifications required to make the measurement?

CHAPTER III

SELF TEST

1. How many signal lines comprise the data bus on the GPIB?

2. What primary aspect of the GPIB is defined by the mechanical specification?

3. List two problems that a system designer faced when trying to interface products prior to GPIB. Explain how GPIB overcomes these.

4. What interface signal line allows instruments to easily distinguish between interface messages and device-dependent messages coming from the controller? In what case is it asserted?

5. What determines the rate of data transfer over the GPIB?

6. Is the GPIB considered synchronous or asynchronous? What is the advantage of this?

7. When can an instrument assert the SRQ line in relation to the other signals on the GPIB?

8. According to the IEEE-488 standard, what interface functions must be present in every GPIB instrument? List two other functional categories.

9. What is the maximum cable length in a GPIB system?

10. How many devices can be connected in a GPIB system?

11. How many devices must be powered up for a GPIB system to run properly?

12. Some people say that GPIB only guarantees that two pieces of gear connected together "won't smoke." Given a flexible controller and even a minimally functional GPIB product, what are two other capabilities that we can safely guarantee?

13. List three types of instrument intelligence and a brief sentence about what each means.

14. What are two kinds of systems specifications that are over and above instrument specs?

15. List three major considerations a customer may make when selecting GPIB-compatible instruments.

IV. TEK CODES AND FORMATS

As we have seen, the IEEE 488 standard has made instruments more compatible in a system environment; that is, they are easier to interface together to perform some task. Establishing this compatibility is the primary function of the standard. However, it is only the first step toward further standardization that will achieve even greater compatibility.

We mentioned earlier that, given a flexible enough implementation language such as our 4050 BASIC, a user could probably get communication to occur on any IEEE 488 compatible instrument. However, it is also conceivable that the applications engineer/programmer will spend as much or more time trying to program and debug the communications activities as on the rest of the application! This is not a productive use of the programmer's or engineer's time.

Perhaps, then, the most important rationale behind Tektronix Codes and Formats is to increase ease-of-use and customer productivity, not only during system development, but also for future software maintenance and system re-configuration. An important benefit of easy-to-use instruments is that, for most applications, it will be easier for an engineer to develop the application software without an external support programmer.

The intention of the Codes and Formats standard is to:

- Define device-dependent message formats and codings and thus minimize the operational incompatabilities encountered in assembling systems from devices compatible with IEEE Standard 488 1978, and

- Minimize the cost and time required to develop system and applications software by making it easier for people to generate and understand the necessary device dependent coding.

Beyond the Codes and Formats standard, there is also a need for a philosophy of designing instruments which are friendly to the user. That is, they are controlled over the bus in easily understood commands and are resistant to operator errors. Since the application of this philosophy is different for each type of instrument, it cannot be included as a specific standard. However it is the application of this philosophy which assures an increase in productivity. Other instruments, whose designs are based on the simplicity of an instrument's hardware logic, are going to be more difficult to use.

CODES AND FORMATS DEFINITIONS

The Tektronix Codes and Format standard augments the IEEE 488 standard by specifying in rigorous detail the codes and formats of device dependent messages between instruments and/or the controller. Codes are the form in which numbers, data, and messages must appear; and formats are the order, syntax, and control protocol that these messages must use.

The standard:

- establishes a common message structure.
- describes communication elements and how they may be combined.
- defines control protocol.
- defines status bytes for error handling.
- standardizes features that are particularly important to test, measurement, and analysis systems.

These five categories encompass the primary features (definitions) of the Tektronix Codes and Formats standard; we will examine the benefits both as we go along and in a summary at the end of the Chapter.

MESSAGE STRUCTURE

The structure of messages sent to or received from a Tektronix instrument are defined very explicitly. A message is a complete transaction over the bus which starts when a device is initially addressed to talk or listen and terminates when the talking device asserts EOI (End or Identify). The message itself consists of one or more message units. If there is more than one message unit, they will be separated by semicolons.

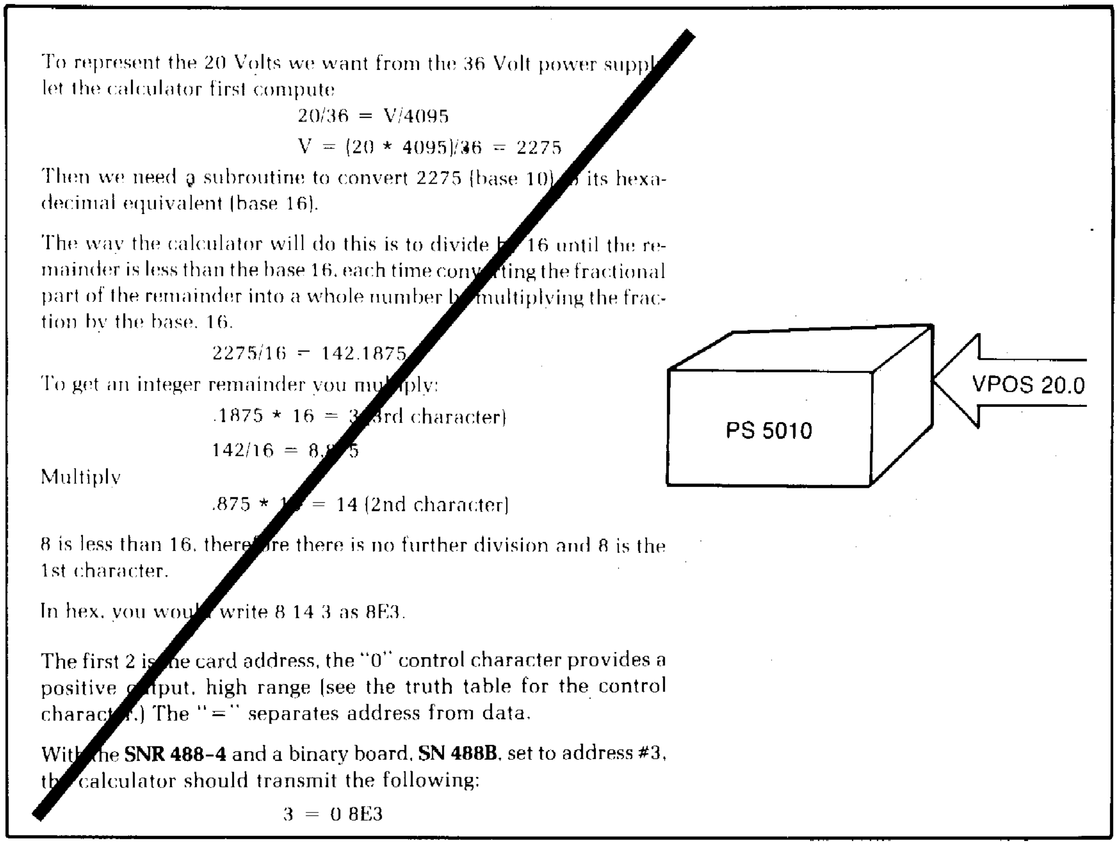
Here are examples of message units:

VPOS 15	(PS5010)
VPOS 15; IPOS .1	(PS5010)
FUNC TRI	(FG5010)

Notice that a message unit consists of a header (VPOS, IPOS, FUNC) followed usually by an argument (15, .1, TRI). (Headers are always characters, whereas arguments may be either characters like TRI or numbers like .1, 15.) A header frequently describes some control on an instrument and the argument describes what that control is to be set to. In the above examples, VPOS 15 means we are going to set the positive voltage supply to 15 volts. IPOS .1 means we want to current limit the positive supply to 100 mA. And finally, FUNC TRI puts a function generator into triangle waveshape function. Again, notice that when we want to send more than one message unit, we separate them with semicolons.

It is also in these message units that we incorporate the "human compatible" design philosophy that is essential for easy-to-use instruments. Ease-of-use or friendliness of instruments is a very real concern to customers in trying to increase productivity.

For example, a power supply can be designed in one of two basic ways. The first is with minimal intelligence so that it can accept some "hieroglyphics" -- which it, in turn, can conveniently interpret and execute. One power supply requires the sequence 0 8 E 3 to put out 20 volts. Here the "0" stands for the 0 - 36 volt range, and the "8E3" is the ASCII representation of the hexadecimal commands required as shown in Figure 4.1. In this case the power supply only needs a four-byte latch and a simple D/A converter to be programmable.



To represent the 20 Volts we want from the 36 Volt power supply let the calculator first compute

$$20/36 = V/4095$$

$$V = (20 * 4095)/36 = 2275$$

Then we need ạ subroutine to convert 2275 (base 10) to its hexadecimal equivalent (base 16).

The way the calculator will do this is to divide by 16 until the remainder is less than the base 16, each time converting the fractional part of the remainder into a whole number by multiplying the fraction by the base, 16.

$$2275/16 = 142.1875$$

To get an integer remainder you multiply:

$$.1875 * 16 = 3 \text{ (3rd character)}$$

$$142/16 = 8.875$$

Multiply

$$.875 * 16 = 14 \text{ (2nd character)}$$

8 is less than 16, therefore there is no further division and 8 is the 1st character.

In hex, you would write 8 14 3 as 8E3.

The first 2 is the card address, the "0" control character provides a positive output, high range (see the truth table for the control character.) The "=" separates address from data.

With the **SNR 488-4** and a binary board, **SN 488B**, set to address #3, the calculator should transmit the following:

$$3 = 0\ 8E3$$

Fig. 4.1. Using human-understandable mneumonics (right) makes programmers and maintenance personnel more productive. Using cheaper hardware logic (left) to keep product price down a few dollars is never really cost-effective for the customer.

On the other hand, the power supply can be designed with a microprocessor and intelligence to accept human readable numbers. In this example, to put out 20 volts, the programmer simply sends the character sequence "VPOS 20.0". This second method of interacting is a great deal more convenient for people, not only when the computer program is first written but also later, when someone other than the original programmer has to find out what the program is supposed to do.

QUERIES

So far we have only really discussed how we control an instrument to make it change some setting. In many cases the instrument is already set up or has been changed by hand and the computer needs to know what these current settings are.

Most instruments cannot satisfy this need. With Tek instruments, there are two ways to find instrument settings. One way is to query (to ask a question of) the instrument for a particular setting. For example, let's say an operator manually set the frequency of the FG5010 to 2.53 KHz. If the computer needed to know this, it would simply send the query "FREQ?" and the FG5010 would respond "FREQ 2530;" for a frequency of 2530 Hz. By the way, if you sent the same command "FREQ?" to a 492P spectrum analyzer, it would respond with the current setting of the center frequency. Remember Tektronix Codes and Formats does not define the specific messages themselves, but the syntax or format in which they will be transmitted. Normally, any header which, with its various arguments, changes an instrument's function, can be used in the form of a query to find out the current setting of that function. For example:

HEADER	ARGUMENT	QUERY
V/D	2.	V/D?
FREQ	10M	FREQ?
VPOS	15	VPOS?

The other way to find the settings of an instrument is to send the query "SET?". Most Tek instruments will respond to this by sending the computer all their current settings and status information. This is a convenient and important feature of our products to the user.

Many customers will want to set up a particular instrument manually to make a certain measurement. Imagine that you are trying to set up an oscilloscope to examine a signal and you have only a vague conception of the signal's parameters. It will likely be easier to adjust the oscilloscope's controls manually (time/div, trigger mode, volts/div) than to try to program these over the GPIB. But once you have found the signal, you want the computer to remember, or LEARN, these settings. People who are familiar with hardware, but new to programming will prefer this manual set-up also. In either case, the application program only needs to send "SET?" and the instrument will send back all its settings. If these are stored on tape or disk, then, when the user next wants to make a similar measurement, he simply sends the original settings that the instrument gave him back into the instrument. The response to "SET?" is data that are exactly in the proper format to set the instrument back to those control positions should they have been changed in the meantime (such as turning the power off). Customers may ask if our products have a LEARN mode; that is exactly what "SET?" provides. This query makes it possible to develop a program using an instrument's front panel as an input to the computer. Using this feature, a programmer may never have to know the instrument's GPIB commands. Very few non-Tek instruments have this capability.

There are other queries that are common to most Tek instruments:

"ERR?" is used for returning detailed error conditions in an instrument. The instrument will respond with a number that is a code for the particular problem.

"ID?" makes an instrument identify itself by sending information as instrument type, model number, firmware version, etc. This feature is useful for identifying a particular device in the field and potentially for self-configuring systems.

Although we will discuss this in detail in Chapter VI, let's put these message units in the context of a 4052 and, say, an FG 5010. To tell the FG (device address 24) to generate square waves, the application program line would look like:

```
310 PRINT @24:"FUNC SQU"
```

To ask it what frequency it is currently generating, we would program:

```
525 PRINT @24:"FREQ?"
530 INPUT @24:A$
```

The string variable A$ would be loaded with the appropriate response from the instrument. If we then typed:

```
PRINT A$
```

the computer would respond with:

```
FREQ 1180;
```

where 1180 means 1.18 kHz. (Program statements such as PRINT A$ typed without a line number will execute immediately. See Chapter V.)

COMMUNICATION ELEMENTS

We have seen how to control an instrument's setting and how to find out where the settings are, but we also need to communicate results. For a DMM the data for a DC volts measurement will be pretty simple, but for an oscilloscope? For that there are lots of data for a single waveform, and there is a need to specify how the data are sent.

It's relatively easy to tell a person that a particular number format is to be used; most people understand what a number is. A microprocessor in an instrument, however, has no such knowledge and must be told -- explicitly and unambiguously -- how to send or how to receive numbers. Otherwise, it can send or receive something that is "obviously" wrong.

Because nearly all of today's GPIB instruments use ASCII-coded characters to send and receive data, Tek has chosen ASCII coding as standard.

In addition, nearly all instruments that send or receive numbers use the ANSI X3.42 standard format. This format states in effect that there are three types of numbers -- integers, reals, and reals with exponents -- and that they should be sent with the most significant character first. Table 1 shows examples of these formats.

If a device makes a group of measurements and is asked to report them, then this requires that a group of numbers be sent. To separate one number from the next, a comma (,) is used. For example, the position coordinates from a digitizer might be sent as .732, 1.52.

TABLE I

NUMBER FORMATS (ANSI X 3.42)

NR1 (Integers)	375 + 8960 -328 +0000	Value of "0" must not contain a minus sign.
NR2 (Reals)	+12.589 1.37592 -00037.5 0.000	Decimal point should be preceded by at least one digit. Value of "0" must not contain a minus sign.
NR3 (Reals with exponent)	-1.51E+03 +51.2E-07 +00.0E+00	Value of "0" must contain an NR2 zero followed by a zero exponent with plus sign.

Note however, that while a number has been defined, its use has not. It does not matter whether the number is sent from a multimeter, a counter, or a spectrum analyzer. In all cases, the syntax or structure of the number is identical. Having this well-defined format for using a number allows instruments to send and receive numbers without confusion.

There are other data types also. In some instances (like plotters or printers) these more complex forms will be sent to a device, but more frequently in Test and Measurement applications, these data types will be coming out of the device:

String Arguments -- for sending text to a display or printer.

Binary Block Arguments -- for sending binary data blocks of known length, such as waveforms.

End Block -- for sending binary data of unknown length.

These same general formats can be used for all types of communications over the bus -- commands to instruments, data from instruments, text to be displayed, and others.

CONTROL PROTOCOL

While standardizing the Codes and Formats syntax does foster greater compatibility between devices using the GPIB, it alone does not solve all compatibility problems. Well-defined operational conventions are also needed. Conventions for using the GPIB are analogous to good manners when using the telephone -- one party should not hang up before the other has finished speaking. The following is an example of the lack of good manners in two devices using the GPIB.

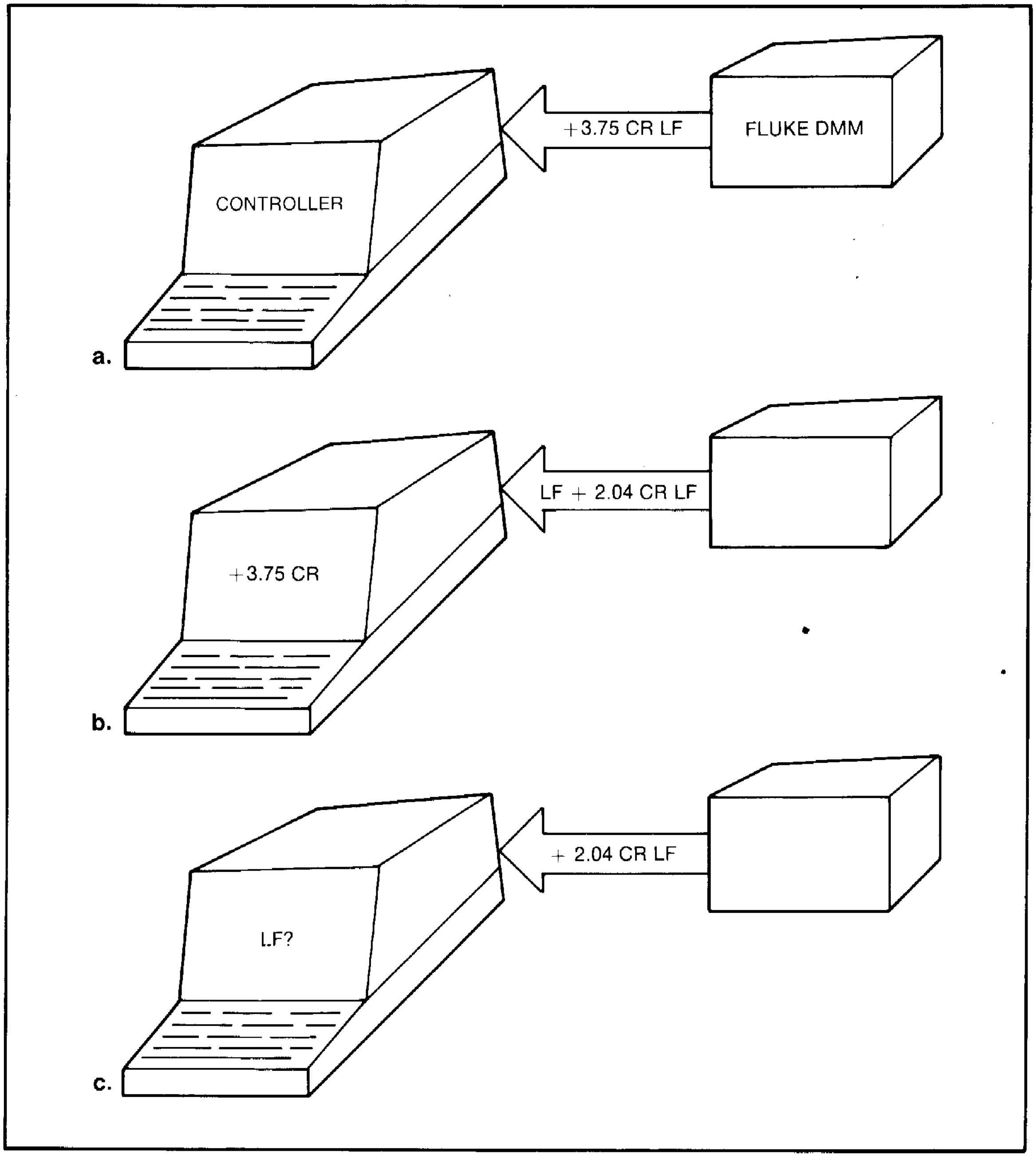


Fig. 4.2. Without a standard message termination, the user must take care that each listener understands the same message termination convention. In this example, the talking instrument uses CR LF as the message termination (a), while the listening controller understands CR as the message terminator (b). When the controller accepts the next message, the first character encountered is the LF left from the last message (c), which the controller may interpret as an illegal character since it is expecting a numeric value.

Suppose a computer has requested a voltage reading from a multimeter over the GPIB. The multimeter sends the number and terminates the transmission with the characters CR (carriage return) followed by LF (line feed) (Figure 4.2A). The computer, however, understands the CR by itself terminates a message. The computer "hangs up" after receiving the CR and leaves the LF character in the multimeter unsent (Figure 4.2B). The next time the computer asks for a multimeter reading, the multimeter sends LF, the character left over from the previous measurement, followed by the measured values (Figure 4.2B). The computer does not know what to make of a number preceded by an LF character. It stops and indicates an error. Although both devices are GPIB compatible, they do not work together because the IEEE 488 standard has not defined the conventions, "manners," for how the bus is to be used. To avoid such incompatibilities, a standard way to terminate messages is needed. Two methods are commonly used. The first is to send some printer format characters such as CR or CR LF. The other is to assert the EOI line when the last data byte of a message is sent. The first method was adequate for simple instruments that sent or received only ASCII coded numbers. However, today's more intelligent devices have to send messages representing digitized waveforms or programs for a microprocessor in an instrument. Some of these messages may contain binary data to reduce transfer time. Certain sequences of binary coded bytes (00001101, 00001010), if interpreted as ASCII, will appear to be a CR LF and thus be misinterpreted. The second termination method has no such problems. Asserting the EOI line unambiguously terminates the message.

The Codes and Formats standard states that instruments sending messages must terminate them by asserting the EOI line concurrently with the last byte of the message (Figure 4.3).

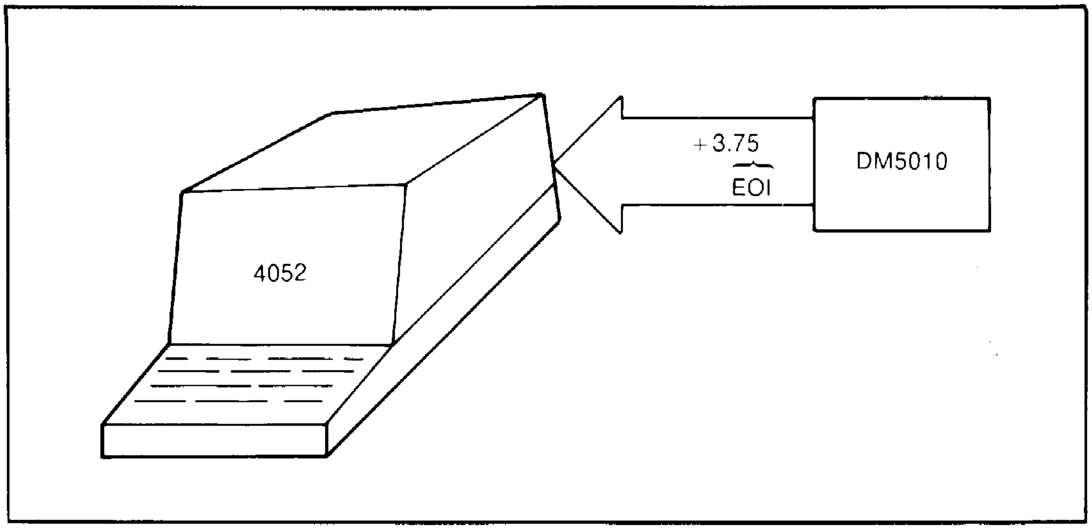


Fig. 4.3. Setting the EOI line concurrent with the last byte of a message provides an unambiguous message terminating convention.

Other problems can be created by instruments which execute each individual command as received. For example, suppose a programmable high-voltage power supply that can be set as high as 1000 V has been set to put out 10 V and limit current at 2 A. Then it is sent the message "VOLTS 1000, CURR 10E-03", that is, 1000 volts output limited at 10 mA. Lacking good message handling conventions, the supply goes to 1000 V output immediately upon receiving the first part of the message. But because the current limit is still 2 A, the value from the previous setting, the supply either crowbars or damages the equipment connected to it. For proper operation, the programmer should have changed the current setting first. Only then should the voltage be changed. It is much easier and safer using a power supply which does not execute any command before the entire message is received and terminated by asserting the EOI line (Figure 4.4).

We can define a message as a complete block of information that begins when a device starts sending data and ends when EOI is sent or received concurrently with the last data byte.

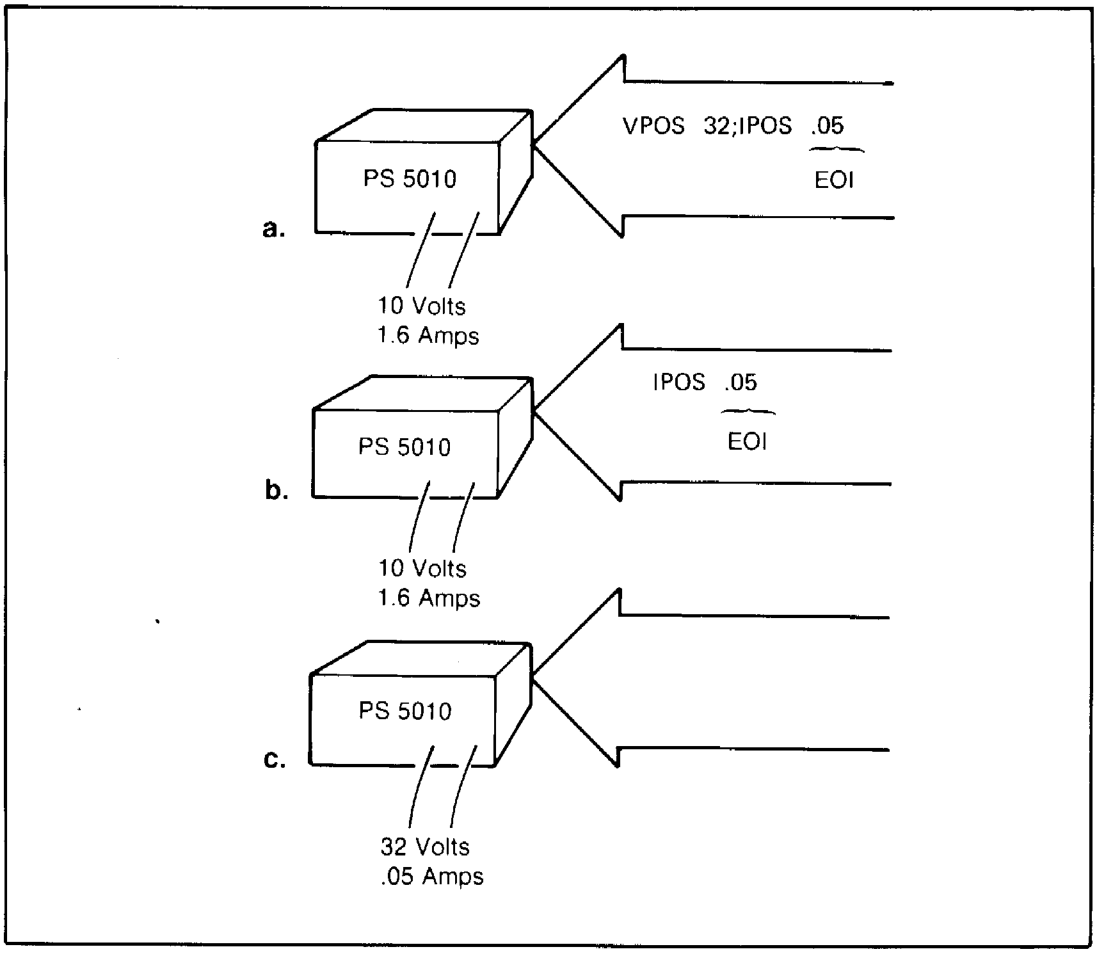


Fig. 4.4. Tektronix instruments wait for the message terminator (EOI) before execution of any command in the sequence.

Talkers Talk. There is a further refinement to the message convention. When a device is made a talker, it should always say something. If it has nothing to say and will have nothing to say for an indefinite period of time, it should send a byte of all ones concurrent with EOI. This lets the listening device know that no meaningful data is forthcoming. Therefore, the "talked with nothing to say" byte is a null message. This convention prevents hanging up of the GPIB while the computer waits for a device to talk that will never send a message.

Listeners Listen. A listening device should always handshake. It should not stop handshaking just because it does not understand or cannot execute a particular message.

When In Doubt, Shout. If the listening device is confused after EOI is received, it should send out a service request and, on a serial poll, notify the controller that nonsense has been received. Under no circumstances should a device execute a message it does not understand. Some non-Tektronix devices do not follow this convention -- with disastrous results. A particular power supply can be sent four letter O's instead of four zeros, a common human mistake, and this supply will put out its maximum voltage instead of the intended zero volts.

ERROR HANDING AND STATUS BYTES

The IEEE 488 standard defines a facility for an instrument to send a byte of status data to the computer, but, except for one bit, the standard does not define the meaning of the bits. The IEEE 488 standard assigns bit 7 to mean that a device is or is not requesting service. Thus, bit 7 cannot be used for other purposes.

However, there is a common need for instruments to report certain kinds of status or errors to the computer. So a status byte convention is established for this by the Tek standard. One common need is for instruments to report if they are busy or ready. Bit 5 is used for this purpose. Another common need is for instruments to report if they are encountering abnormal conditions. Bit 6 is selected.

There are more complex conditions besides busy/ready or normal/abnormal. These are listed in Table II. While these status bytes are generally useful for most purposes, certain instruments may have conditions that are peculiar to them. To report these status states, bit 8 is used to indicate that the status byte is not the common type, but particular to an instrument.

Providing a standard coding for the status byte is convenient for the person programming the computer that runs the instrument system. If all the instruments have common status byte codings, then a common status byte handling routine is written for all instruments, not a separate one for each. But even with all the possibilities allowed by status bytes, it is often necessary to send more detailed information to a computer. In these cases the query "ERR?" is used. The instrument will then send back an explicit error code to the computer.

TABLE II

TEK CODES AND FORMATS

STATUS-BYTE DEFINITIONS

		Decimal	
		X=0	X=1
Abnormal Conditions	Binary		
ERR query requested	011X 0000	96	112
Command error	011X 0001	97	113
Execution error	011X 0010	98	114
Internal error	011X 0011	99	115
Power fail	011X 0100	100	116
Execution error warning	011X 0101	101	117
Internal error warning	011X 0110	102	118

X=1 if instrument busy

Normal Conditions			
No status to report	000X 0000	0	16
SRQ query request	010X 0000	64	80
Power on	010X 0001	65	81
Operation complete	010X 0010	66	82

BENEFITS

Since we have been discussing specific benefits that match with particular features, the following list summarizes the major overall benefits of Tek Codes and Formats.

System users who incorporate instruments that comply with the Codes and Formats Standard as well as use easily understood commands, will find that:

- Writing the system software will be easier because different instruments communicate with the same data format and with the same protocol. Therefore, one applications-level driver will be sufficient for all.

- Programs will be self-documenting. As long as commands are human intelligible, programs will be easier to read, write, and especially maintain.

- System change, or expansion, will require less software modification. Many support subroutines, such as error handling and data drivers, can continue unchanged.

- Programmer only needs to learn one set of conventions for all instruments, not one for each. Training time is shorter.

- Development time is faster.

- Debug time is shorter and easier because there are only a few central applications level software drivers.

Basically, then, the features of our intelligent instruments make them both compatible with computers and friendly to people.

Given a powerful enough computer, and a clever enough programmer, most of today's devices that use the IEEE 488 bus can be made to do whatever they were designed to do: compatibility can be forced. Without well-defined codes and formats and without operational conventions and easily understood commands, instruments appear incompatible or unfriendly. With Tek Codes and Formats and with known operational conventions, devices using the GPIB become friendly as well as compatible, thereby allowing the user to spend more time on the task at hand, rather than figuring out how to make the system work -- increased productivity with Tek GPIB instruments.

IV. SELF-TEST

1. What are three general areas that Codes and Formats define?

2. Codes and Formats is a specification for device dependent messages or interface messages. Which one?

3. One type of message unit consists of a header and an argument. For example: FUNC SIN. This command would cause an FG5010 to generate a sinewave. If someone had manually set up the instrument, what query message could the controller send to find out what type of waveshape the FG was generating?

4. What ASCII character does Codes and Formats define to separate message units?

5. What command do most Tek instruments respond to which cause them to send the position (value) of all their controls and status?

6. What query will elicit detailed error information?

7. List two types of communication elements defined by Codes and Formats.

8. How do all Tek instruments designate the termination of a particular message?

9. List three user benefits of Codes and Formats.

V. 4050 OPERATION

As we have discussed earlier, the 4052 (or 4051 or 4054) has four of the five generic system functions built into one single unit. These four generic functions are the user input (keyboard), the user output (crt screen), the controller (the microprocessor, memory, BASIC, interfaces, etc.), and the mass storage device (the DC300 tape unit). This chapter will discuss the operation of each of these parts and some examples and exercises of their use.

This chapter and the next will require that you have a 4050 Series controller available. In the following sections, the text will describe some functions of the 4050, then a couple of examples will be shown, followed by a few exercises. Unless you are very familiar with the 4050, do all the exercises so that you can become comfortable with it. In the near future you will be doing a fair percentage of your demonstrations with a 4050 Series controller. And even when you are not doing a demo, you will find yourself discussing controllers much more than in the past. Knowing and seeing how one works will amply pay off.

GETTING STARTED

Turn ON the 4050 (hereafter we will use 4050 to mean any one of the three 4050 Series controllers) by flipping the power switch located under the right front corner of the unit (Figure 5.1). The four green indicator lights on the panel will turn on briefly, but only the POWER light should remain on. Disconnect any GPIB gear that may be attached to the controller.

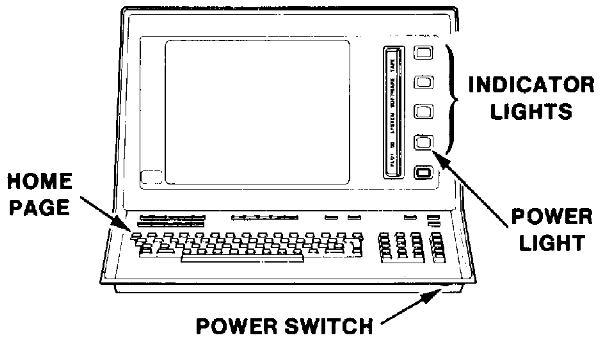


Fig. 5.1. Power switch, indicator lights, and PAGE key.

When the 4050 powers up, it is immediately ready to go to work. Remember, BASIC is implemented in ROM (Read Only Memory). That the operating system is immediately available at power-up is a feature of firmware-implemented language. Other systems will require that the operating system software be loaded from a mass storage device.

At power-up, you will notice the screen "writing up," becomming bright. This is normal. Press the HOME/PAGE key (Figure 5.1) and the screen will be blank except for a small faint blinking rectangle, called the cursor, in the upper-left corner of the screen.

The cursor is analogous to the position of the carriage on a typewriter. Whenever you strike a character, that character will appear at the current position of the cursor. As we will see later, the cursor sometimes becomes a blinking question mark when the 4050 is under program control and is waiting for input from the keyboard.

Before we can describe the 4050, there are a couple of definitions worth knowing. The following is a very short 4050 program:

```
1ØØ PAGE
11Ø PRINT "THIS IS A SHORT PROGRAM"
12Ø END
```

The 4050 has two general modes of command entry. When a user types *statements* (i.e., complete 4050 executable lines) with *line numbers* preceding them as in the above program, then the command entry is said to be in *deferred* or *program mode*. LIne numbers (e.g., 100, 110, 120 above) are used by the computer so that it knows in what order the programmer wants the statements executed. In deferred mode, the statements will not execute until the user types RUN; program execution is deferred until that time.

If we type a statement on the 4050 keyboard without a line number, then when we press the RETURN key, the statement will execute immediately; this kind of command entry is called *immediate mode*. Now, knowing what line numbers are and the difference between immediate and deferred modes, let's use the 4050.

Type about a half-dozen arbitrary characters (trying *not* to spell a word) on the 4050, and follow these by pressing the RETURN key. Notice that as soon as you hit the RETURN key, the characters are retyped, but a message which says SYNTAX ERROR and an arrow is positioned over the third character. It should look something like this:

```
  ↓  SYNTAX ERROR
SLDFLKZ
```

When this happens, it means that you typed something that the 4050 did not know how to interpret. You can bet that this is not the last time you will see this message. It will happen whenever you misspell a word or type a statement in the wrong order. Order, spelling, the placement of commas and spaces are, in computer jargon, called syntax. Although syntax isn't much fun, it is essential for the computer to be able to execute your intent as directed by your program lines.

Error messages work in the users behalf to indicate problems that otherwise wouldn't show up until the program was executed.

Anyway, to clear this error condition when it occurs, press the CLEAR key (Figure 5.2). Then retype the line correctly. You will notice that, if you try using the RETURN key to clear a syntax error, it will simply retype your original line and the error message.

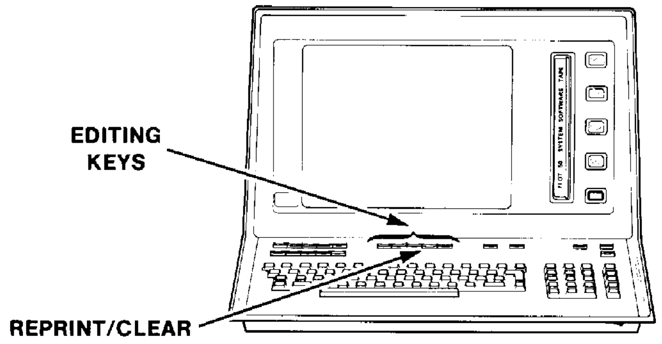


Fig. 5.2. The REPRINT/CLEAR key.

When the screen is full, it is indicated by a faint blinking F (for full) in the upper-left corner. Pressing the PAGE key erases the screen and returns the cursor to the HOME (upper-left corner) position.

Also, if the screen information should fade out on you, it is because there has been no screen activity for 100 seconds. To return the screen to full brightness, just press the SHIFT key. (Pressing any other key will also bring the screen back, but the character is also typed, which might not be desired.)

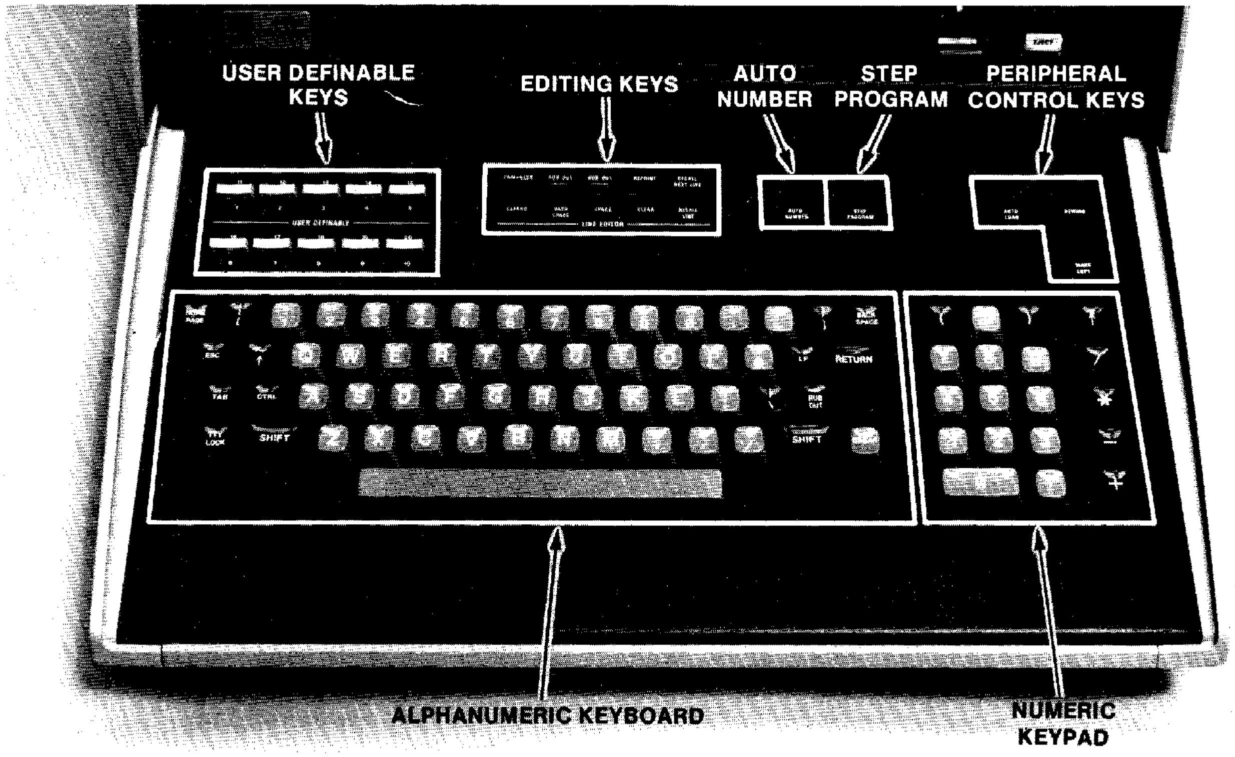


Fig. 5.3. Keyboard controls.

ARITHMETIC

The 4050 can be used as a simple calculator without the need to write a program at all. To do this, just type the calculation on the NUMERIC KEYPAD (Figure 5.3) in the same manner you would write the computation on paper. To make the 4050 actually compute the expression, the RETURN key must be pressed.

Here are some examples:

```
7+8 (RETURN)      Addition
 15
12-3 (RETURN)     Subtraction
  9
120*3 (RETURN)    Multiplication
  360
```

```
144/12 (RETURN)  Division
 12
12↑2 (RETURN)    Powers
 144
```

These are all pretty simple because they involve only one operator (i.e., +, -, /, *,). However, it is important to note that between any two operands (numbers or intermediate results) one operator is required. For example, 3(2) might be read as 3 times 2, however the 4050 will give you a syntax error because it needs an operator.

When we start mixing operators, we have to be aware of the computer's math hierarchy. This simply means that multiplication and division operations are calculated prior to addition and subtraction; powers are calculated before multiplication and division. Parentheses allow for computations done in a sequence other than the defined heirarchy. For example:

```
3+5*4 (RETURN)
 23                   Multiplication done first
(3+5)*4 (RETURN)
 32                   Parentheses change thc order
4*3↑2 (RETURN)
 36                   Power done first
(4*3)↑2 (RETURN)
 144                  Parentheses change the order
```

Multiplication and division are on the same level of hierarchy, and addition and subtraction are both one level lower. In these cases, the computation is performed left-to-right. For example:

```
4/2*3 (RETURN)
  6                   Computation done left-to-right
4/(2*3) (RETURN)
0.666666666667        Parentheses change the order
```

Care must be exercised in translating algebraic expressions into BASIC. Note that arithmetic expressions are represented in BASIC by a single line of numbers and symbols. This can present some difficulties in translating an expression like the following:

$$\frac{4X + 7}{2X - 3}$$

A first attempt to convert this expression into an arithmetic expression in BASIC might yield this:

```
4*X + 7/2*X-3
```

This comes out to be interpreted as:

$$4X + \frac{7X}{2} - 3$$

The problem can be resolved through the use of parentheses:

```
(4*X + 7)/(2*X-3)
```

Parentheses are used in BASIC as they are used in conventional algebraic notation.

With the following arithmetic expressions, first calculate them yourself the way you think the 4050 would, then check your answers by typing them into the machine.

2+8/4 _______
4*2-6 _______
6↑2/9 _______
4-(2+3) _______
12/2*3 _______
2↑(2+1) _______

So far we have been using mostly whole numbers, but the numbers that the 4050 uses are the same as those defined by the Tek Codes and Formats. There are integer numbers: 45, -27, 1, 0; there are decimal numbers: 2.7, -8.9, -6.0; and there are numbers expressed in scientific notation: 2.0E-3, -7.9E8, 0.E0. Scientific notation is composed of a mantissa and an exponent. For example:

$$\underbrace{\underline{2.83}}_{\text{mantissa}}\ \underbrace{\underline{\text{E6}}}_{\text{exponent}}$$

This is equivalent to the value 2.83 X 10^6.

The 4050 controllers have 12 digits of precision (this is a computer term analogous to resolution) and a numeric range of 8E307 to -8E307. Anywhere you would want or expect to specify a number (except line numbers), you can use any of the above notations.

Suppose we want to compute the system risetime of a scope with risetime of 3.5E-9 seconds (3.5nS) and a probe with risetime of 7.0E-8 seconds (70nS), then:

$$T_{R\ \text{system}} = \sqrt{(T_{R\ \text{probe}})^2 + (T_{R\ \text{scope}})^2}$$

or

```
(3.5E-9↑2 + 7.E-8↑2)↑.5
  7.008744538E-8
```

Remember that a power of 1/2 is equivalent to a square root.

The 4050 Series controllers also provide users with commonly used functions. These are:

ABS	absolute value
ACS	arc cosine
ASN	arc sine
ATN	arc tangent
COS	cosine

EXP	e^x
INT	integer part
LGT	log base 10
LOG	log base e
SIN	sine
SQR	square root
TAN	tangent

There are also some special functions for matrices and arrays (see 4050 Operators manual for details). You must type functions followed by parentheses as in these examples:

```
7*LGT(100)
SQR(3.5↑2+7.0↑2)
```

where the part inside the parentheses is called the argument of the function. Functions are used in expressions just like any other value. Type the above examples and see what you get.

A word about the trigonometric functions. When the 4050 is powered up, all trig functions are computed in radians. If you want to use degrees, you must type SET DEG either as part of your program or in immediate mode (i.e., without a line number so that it executes immediately). To return to radians, type SET RAD.

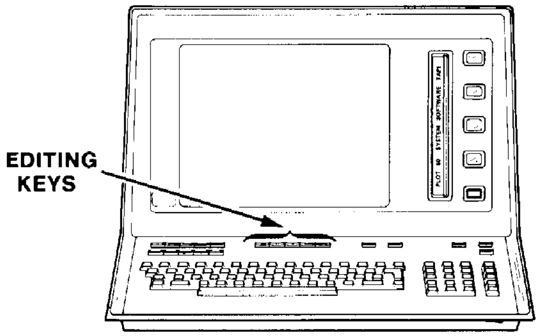


Fig. 5.4. The Editing keys.

EDITING KEYS

For the purposes of this text, we will only discuss some of the editing keys. For a complete desciption read pages 9-21 through 9-23 of the 4050 Series Operator's Manual.

Following are explanations of the functions of the editing keys. Notice that the rubout, backspace, and space functions can also be performed with keys on the alphanumeric keyboard.

← RUBOUT

Pressing this key while holding down the SHIFT key is equivalent to pressing the RUB OUT key on the alphanumeric keyboard. If a character is displayed beneath the cursor, it is replaced with the "space" character. If a character is not displayed beneath the cursor, the cursor backspaces one character position and then performs its function.

BACKSPACE

Pressing this key by itself duplicates the function of the BACKSPACE key on the alphanumeric keyboard; the cursor moves one character position to the left.

→ RUBOUT

Pressing this key while holding down the SHIFT key is equivalent to pressing the RUBOUT editing key, except that the cursor moves to the right instead of the left. If a character is displayed beneath the cursor, it is replaced with a "space" character. If a character is not displayed beneath the cursor, the cursor moves one character position to the right and then performs its function.

SPACE

Pressing this key by itself is equivalent to pressing the SPACE bar on the alphanumeric keyboard. The cursor moves one character position to the right.

REPRINT — Pressing this key while holding down the SHIFT key displays the current contents of the line buffer on the next display line. The cursor also moves down one line while retaining its position in the line.

This provides a good tool to use when a line has type-overs, rubouts, etc. Pressing SHIFT-REPRINT provides a "clean" line to check before pressing the RETURN key.

CLEAR — Pressing this key by itself clears (erases) the contents of the line buffer. This is a good tool to use when it's easier to rewrite a statement than to edit it.

RECALL LINE — Entering a line number and then pressing this key alone recalls that program line from RAM and places a copy of it into the line buffer. The cursor is positioned at the end of the recalled line. You can perform your editing functions and then press RETURN to replace the initial line with the edited line.

Do the following short exercise. Type the following program line into the 4050:

```
1ØØ PRINT "THIS IS AN EXAMPLE OF A PRINT STATEMENT"
```

Conclude this line by pressing RETURN. Now experiment with the editing keys; use especially RECALL LINE (that's how you will get it back into the line buffer), RUBOUT, BACKSPACE, and REPRINT. Reprint the line after using Rubout and Backspace to get a feel for how they differ.

Perhaps the simplest way to edit a program line, however, is to simply type the same line number over followed by the correct statement. This will always update the statement at that line number to the most recently typed line.

PERIPHERAL CONTROL KEYS

Figure 5.5 shows the peripheral control keys.

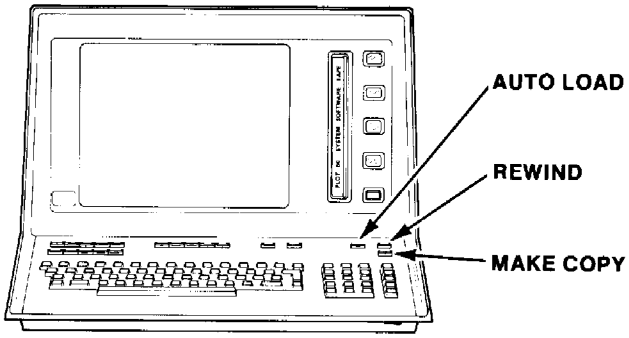


Fig. 5.5. The Peripheral Control keys.

AUTO LOAD Pressing the AUTO LOAD key rewinds the internal magnetic tape, locates the first ASCII program on the tape, loads the program into the 4050 memory (RAM), and begins executing the program. (The program doesn't have to be located in the first tape file.)

REWIND Pressing the REWIND key causes the 4050 to rewind the tape cartridge in the magnetic tape unit. Pressing this key is the same as executing the BASIC statement FIND Ø.

MAKE COPY Pressing the MAKE COPY key causes an attached hard copy unit (an optional peripheral) such as the 4611 or 4631 to make a paper copy of the information on the display.

SIMPLE 4050 OPERATION

To see the 4050 actually operate, we are going to need to enter a couple of short programs (We will explain in detail what the program statements do in Chapter VI. This section is to familiarize you wilth the DEL ALL, LIST, and RUN commands.) Before typing in any new programs, however, we need to clear the computer's memory of any previously entered programs or variables. On the 4050, type:

DEL ALL (RETURN)

This command deletes the contents of the entire user's memory.

Now type in the following program (note that on a computer a zero (Ø) and the letter O have completely different effects and cannot be used interchangeably):

```
100 PAGE
110 PRINT "THIS IS A SHORT PROGRAM"
120 END
```

To make sure that you have typed this correctly, use the LIST command to print out the program. Do this now by typing in: LIST (RETURN).

If it is OK, we can RUN it; if not, use the EDIT keys to correct the program. (Remember, the easiest way to edit a line is simply to retype the line number followed by the correct statement.) Type: RUN (RETURN). The RUN command is used to begin program execution from the lowest line number.

Now, without deleting the current program in memory, type the following lines:

```
500 PAGE
510 PRINT "THIS PROGRAM IS IN AN INFINITE LOOP"
520 HOME
530 GO TO 510
```

To list the program or to verify correct entry, you can type just LIST or you can type LIST 500, 530. (By now you will probably have noticed that the 4050 does not do anything until the RETURN key is pressed. The EDIT keys function without first pressing RETURN because they work on the current line. From now on, we will assume that all lines must be terminated with a RETURN.) LIST, by itself, prints out all program lines currently in memory (RAM -- everything the user enters is in RAM; BASIC is in ROM). LIST with two line numbers following it (e.g., LIST 500, 530) will print out only the lines of the program between and including those two line numbers.

To execute this program, type: RUN 500. By putting a line number following RUN, we can begin the execution of a program at any point. As you probably have gathered by now,this second program is in an infinite loop. Quoting from the operator's manual:

BREAK

> The BREAK key is used to interrupt a program. Two levels of interrupt are provided: a program interrupt and a program abort. For a program interrupt, press the BREAK key once. This causes the BREAK indicator on the front panel to light up. Program execution stops after the current BASIC line, an interrupt message appears on the screen, and the BREAK light goes out. To restart the program at the point of interruption, enter RUN followed by the line number that is printed in the message and press RETURN.
>
> For a program abort, press the BREAK key twice in quick succession or press it once while the BREAK light is on. Program execution is aborted immediately.
>
> To restart the program at the beginning, enter RUN and press RETURN. Attempting to restart the program at the point of interruption may give you questionable results, depending on the operation in progress when the interrupt occurred.

USER DEFINABLE KEYS (UDK's)

We will discuss how to program the UDK's in the next chapter. They are mentioned here so that you are familiarized with their benefits and their location on the keyboard. The UDK's will be used extensively in various demo packages for some instruments.

The UDK's are the ten keys in the upper left corner of the 4050 keyboard. With these ten keys, you can make up to 20 separate subroutine calls. Ten are actuated by simply pressing the appropriate key; the other ten are actuated by first holding down the shift key and then pressing the UDK.

Basically speaking, the User Definable Keys provide a convenient means either to interrupt a program so that it can go off and run some previously defined subroutine, or to cause a branch at a decision point to one of a selection of alternative software paths. The user has complete control over what software activity the UDK will initiate.

User Definable Keys are frequently a benefit to your customers because, through applications programming, the customer can develop software which is to be used by operators totally unfamiliar with computers or programming. The less skilled operator may need to respond to the 4050 occassionally, such as for verifying that a set-up operation has been completed. By using the UDK's, the operator never needs to interact with the BASIC language itself. In this manner, the operator needs to press only one key, rather than many. Easier-to-use, faster, easier-to-learn--all these benefits make for increased productivity.

PERIPHERALS

When a customer is judging a computer either for an instrumentation control application or for practically any other use, one of the main considerations will be what kind of peripherals are already designed to hook up to the product. The 4050 has a GPIB interface port, so the literal answer is that the 4050 is designed to work with any IEEE 488-compatible product. However, the 4050 Series controllers do have some specific peripherals for which all of the communication drivers are built into the BASIC language. This allows the programmer/engineer to control these devices from a high-level language and not get bogged down in specific device codes.

The following Tek products will probably be the most common non-instrument peripherals your customers will be most likely to ask about.

- 4631 HARD COPY UNIT. Provides high quality, high resolution paper copies of any image written on the 4050 crt.

- 4611 HARD COPY UNIT. Provides the same function as the 4631, but is a lower resolution, lower cost alternative. Per-copy cost is substantially less expensive as well.

- 4907 FILE MANAGER. Provides floppy disk mass storage where files can be accessed directly (as opposed to sequentially as on the internal tape) and by name (rather than by number). Provides twice the storage of a tape on a single disk. Comes in one, two, and three drive configurations.

- 4924 CARTRIDGE TAPE DRIVE. This is an auxiliary tape drive which uses tapes indentical to the 4050 Series controller. Data format and control commands are the same as those for the 4050 internal tape cartridge.

- 4662 DIGITAL PLOTTER. Provides exceptionally high resolution plots with various pens and on many different media (paper, mylar, etc.). The 4663 is a two-pen plotter capable of handling much larger paper.

- ROM PACKS. These provide software (high level) extensions to the already powerful 4050 BASIC. Note especially the two Signal Processing ROM packs. These provide essential array processing functions that would be useful on waveforms of any digitizer or spectrum analyzer. These should be an easy additional item for any waveform instrument/4050 combination sale. They are well worth the price to customers. Functions included on the R07: maximum, minimum, integration, differentiation, and quick graphics. On the R08, they include: FFT, convolution, correlation, and related utility functions.

DEMONSTRATION SOFTWARE

As Tek continues to bring out more GPIB-compatible products, you will probably see your product demonstrations begin to change. Although most of the new products can be demonstrated in a stand alone fashion, a few cannot (e.g. the 7912AD), and most will have capabilities available via the GPIB which are simply not accessible from the front panel (e.g., the 492P). Still other products just do not make any sense without a controller (e.g., the CG551AP cannot increase scope calibration productivity without one, and the 7612D cannot make any measurements without one).

Some increasing percentage of your product demos, therefore, will be made with a controller attached. In many cases, the controller will be a 4052. The graphics capability of the 4052 lends itself particularly well to these kinds of presentations.

Each GPIB product announced will generally be introduced to the field accompanied by a demo tape. These tapes make it easier to demo the major features and benefits of a product and will allow you not to have to remember the specific command set for each instrument. Also, as mentioned earlier, these programs (as a listing) can be given to your customers and will almost always speed up the learning process and reduce frustration.

Remember, these tapes are your applications software to help you sell. Although the demo tapes are written to be easy to use, it is important that you see the tape run once so that you know what it does, and so that you can most effectively incorporate its action into your sales call.

Because the business units recognize the importance these tapes will have in supporting your sales demos, standards have been created for these programs so that there is assurance that they are easy-to-use and create minimal operator traps of their own. These standards also provide commonality among the demo tapes so that once you have learned the skills to operate one of them, the rest will be easier to learn.

Demo tapes generally fall into one of three categories:

- Page-to-page tutorial
- Menu selectable features/applications
- Specific application demonstration

Because each of these categories is distinctly different, we have included an example of the output of each. You may discover that not every type or demo program is completely suitable for every product/controller demo you want to give. Normally, however, they are intended to be general enough to cover a broad range of different customers' interests.

The most important single instruction for running any demo tape is to follow closely the directions that will be printed on the screen as the demo tape progresses. Once in a while the original programmer forgets to check for a valid entry. For example, if the program asks for you to select a menu item between 1 and 5 and you type 7, occasionally you can get some weird results--typically, the program it will catch the error and ask you to re-enter the number. To avoid difficulty, follow the printed instructions.

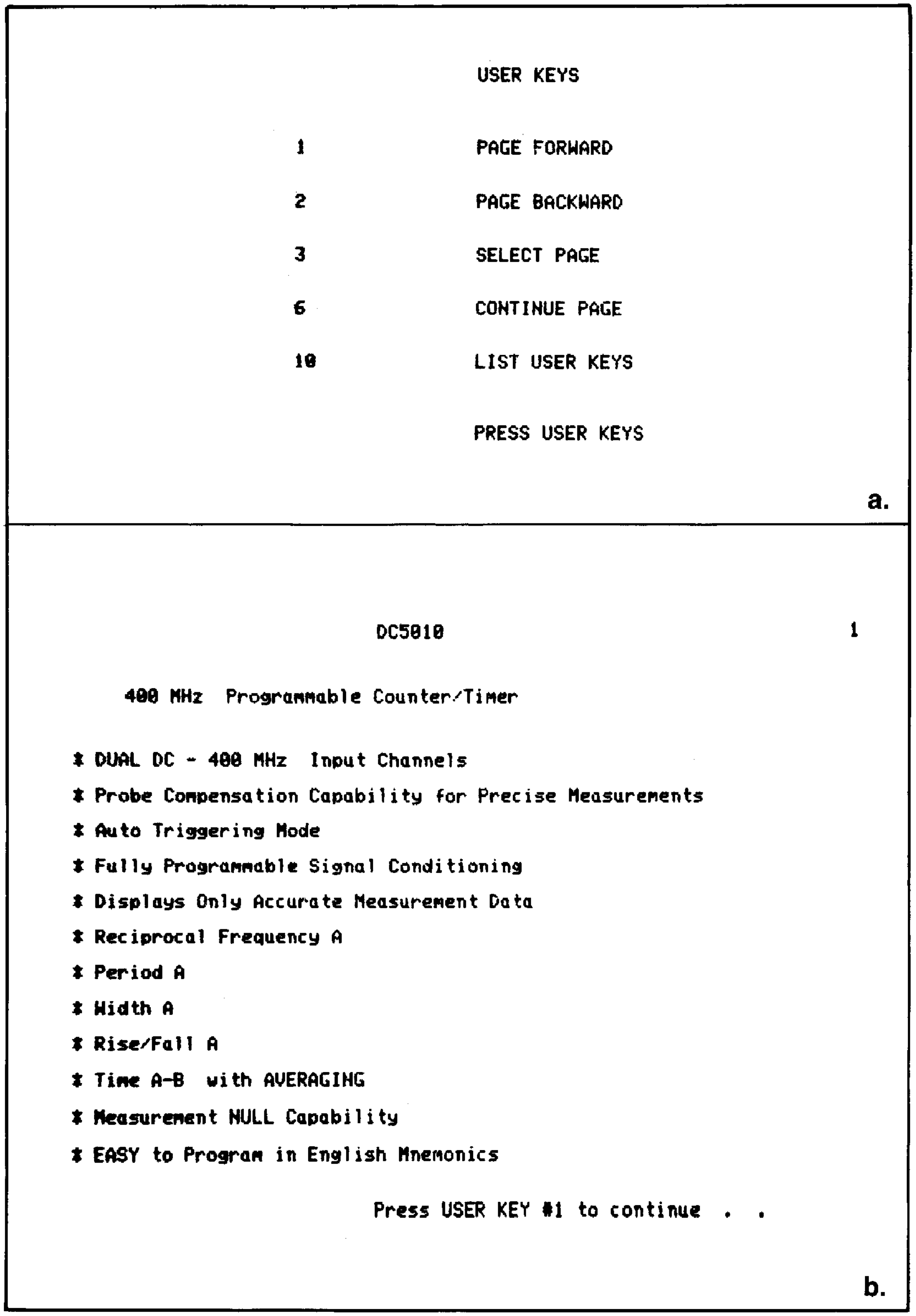
PAGE-TO-PAGE TUTORIAL

Take a look at all of Figure 5.6. This is an excerpt of the output of the DC5010 demo tape.

After auto-loading the tapes, the controller will ask you for the address of the DC5010 (not shown). Then the software lists the various features inherent in the product (Fig. 5.6b). Following that, by pressing UDK 1, the program follows a sequence through illustrations and examples of those features, one at a time. Figures 5.6c through 5.6f are examples of the formats used to show some of those features. (Notice the page numbers in the upper right corner of each page.)

The User Definable Keys for page-to-page tutorials are to move the user from one page to the next (UDK 1) or to move to an arbitrary page in the demo (UDK 3). The UDK's used in this fashion have limited flexibility.

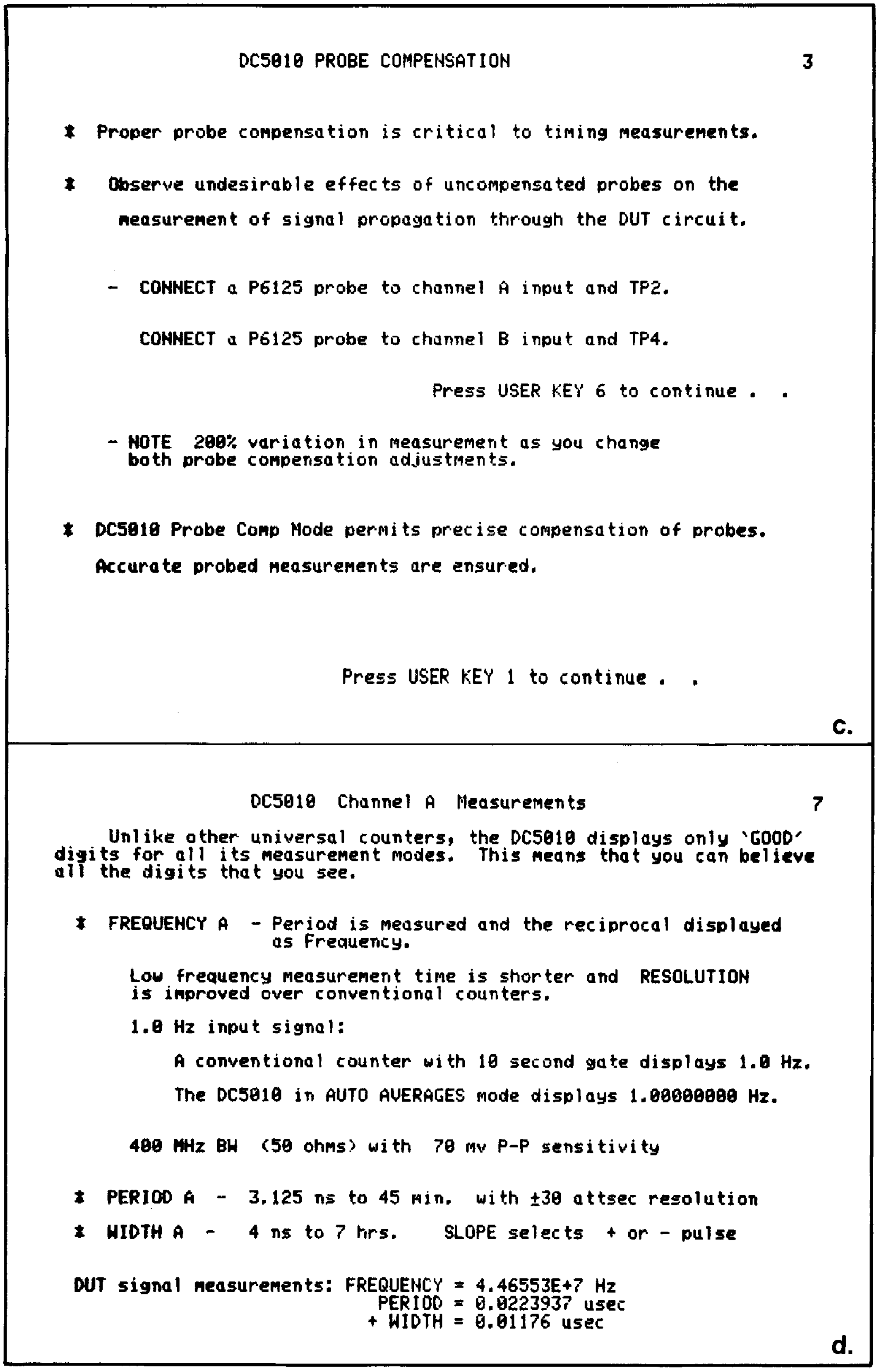
UDK 10, like all demos, will list the available UDK menu which, in this case, is Figure 5.6a.



```
                         USER KEYS

        1                PAGE FORWARD

        2                PAGE BACKWARD

        3                SELECT PAGE

        6                CONTINUE PAGE

        10               LIST USER KEYS

                         PRESS USER KEYS
```

```
                    DC5010                              1

   400 MHz  Programmable Counter/Timer

 * DUAL DC - 400 MHz  Input Channels
 * Probe Compensation Capability for Precise Measurements
 * Auto Triggering Mode
 * Fully Programmable Signal Conditioning
 * Displays Only Accurate Measurement Data
 * Reciprocal Frequency A
 * Period A
 * Width A
 * Rise/Fall A
 * Time A-B  with AVERAGING
 * Measurement NULL Capability
 * EASY to Program in English Mnemonics
                       Press USER KEY #1 to continue . .
```

Fig. 5.6.



DC5010 PROBE COMPENSATION 3

* Proper probe compensation is critical to timing measurements.

* Observe undesirable effects of uncompensated probes on the measurement of signal propagation through the DUT circuit.

- CONNECT a P6125 probe to channel A input and TP2.

  CONNECT a P6125 probe to channel B input and TP4.

Press USER KEY 6 to continue . .

- NOTE 200% variation in measurement as you change both probe compensation adjustments.

* DC5010 Probe Comp Mode permits precise compensation of probes.
  Accurate probed measurements are ensured.

Press USER KEY 1 to continue . .

c.

DC5010 Channel A Measurements 7

Unlike other universal counters, the DC5010 displays only `GOOD' digits for all its measurement modes. This means that you can believe all the digits that you see.

* FREQUENCY A - Period is measured and the reciprocal displayed as Frequency.

  Low frequency measurement time is shorter and RESOLUTION is improved over conventional counters.

  1.0 Hz input signal:

  A conventional counter with 10 second gate displays 1.0 Hz.

  The DC5010 in AUTO AVERAGES mode displays 1.00000000 Hz.

  400 MHz BW (50 ohms) with 70 mv P-P sensitivity

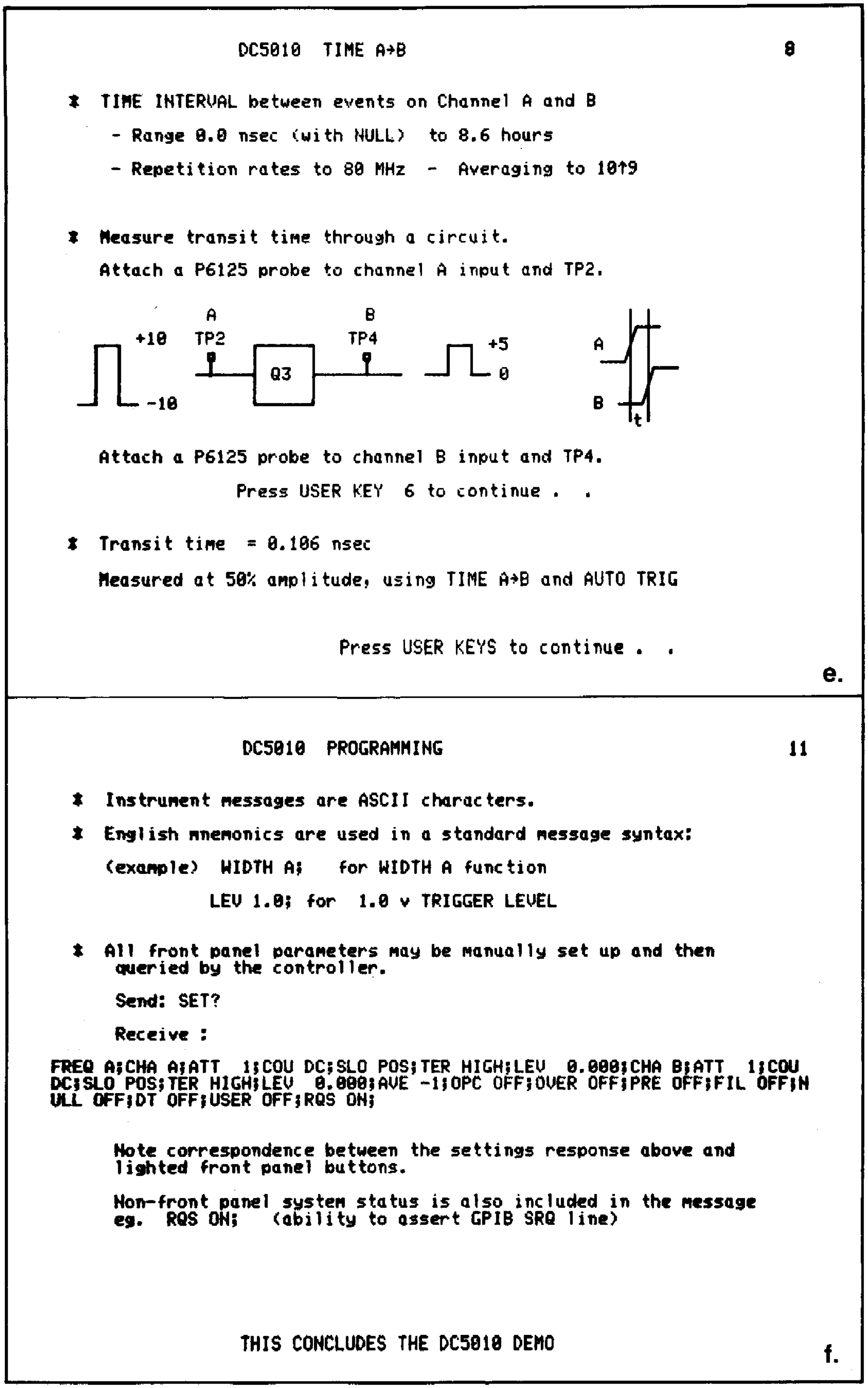
* PERIOD A - 3.125 ns to 45 min. with ±30 attsec resolution

* WIDTH A - 4 ns to 7 hrs. SLOPE selects + or - pulse

```
DUT signal measurements: FREQUENCY = 4.46553E+7 Hz
                            PERIOD = 0.0223937 usec
                           + WIDTH = 0.01176 usec
```

d.

Fig. 5.6. (continued)



DC5010 TIME A→B 8

- TIME INTERVAL between events on Channel A and B
  - Range 0.0 nsec (with NULL) to 8.6 hours
  - Repetition rates to 80 MHz - Averaging to 10↑9

- Measure transit time through a circuit.

  Attach a P6125 probe to channel A input and TP2.

Attach a P6125 probe to channel B input and TP4.

Press USER KEY 6 to continue . .

- Transit time = 0.106 nsec

  Measured at 50% amplitude, using TIME A→B and AUTO TRIG

Press USER KEYS to continue . .

e.

DC5010 PROGRAMMING 11

- Instrument messages are ASCII characters.
- English mnemonics are used in a standard message syntax:

  (example) WIDTH A; for WIDTH A function

  LEV 1.0; for 1.0 v TRIGGER LEVEL

- All front panel parameters may be manually set up and then queried by the controller.

  Send: SET?

  Receive :

```
FREQ A;CHA A;ATT  1;COU DC;SLO POS;TER HIGH;LEV  0.000;CHA B;ATT  1;COU
DC;SLO POS;TER HIGH;LEV  0.000;AVE -1;OPC OFF;OVER OFF;PRE OFF;FIL OFF;N
ULL OFF;DT OFF;USER OFF;RQS ON;
```

Note correspondence between the settings response above and lighted front panel buttons.

Non-front panel system status is also included in the message eg. RQS ON; (ability to assert GPIB SRQ line)

THIS CONCLUDES THE DC5010 DEMO

f.

Fig. 5.6. (continued)

This example is characteristic of page-to-page tutorial demos. They tend to lead you through some or all of the features of the particular product. As with this example, the features will be shown primarily via the 4050 keyboard, with only occassional references to the front panel of the product. Therefore, the demo tape shows the product from the systems vantage point and would augment the typical stand-alone product demo that would generally be given first.

Because these demos are designed to provide a complete overview of a product, they are usually lengthy. For this reason, a page-to-page demo will have a page-forward, page-backward, and go-to-page-n feature to allow you to tailor your demo to specific customer needs. You only need to show those pages of the demo-tape where your customer's interests lie. Trying to show an entire page-to-page tutorial depending at the stage of the sale, is too time consuming and ineffective. Use only what you need.

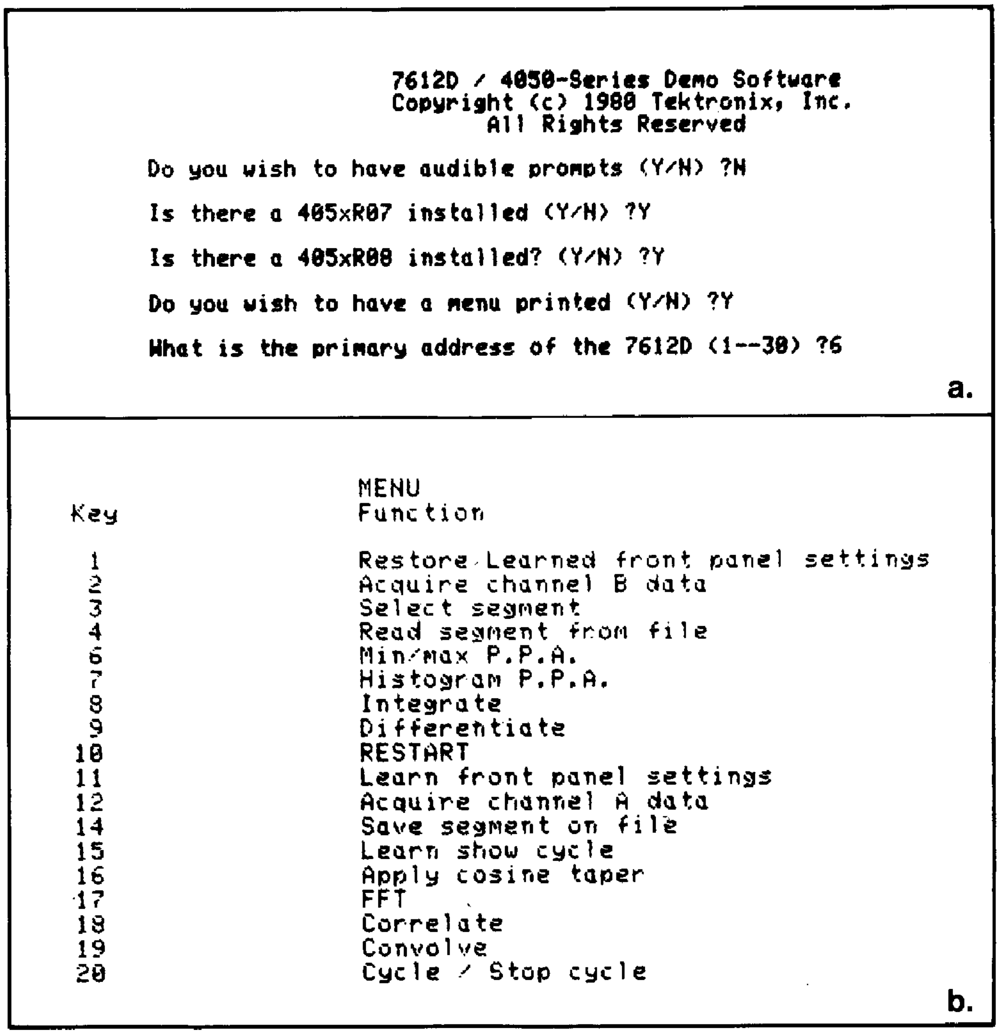
MENU SELECTABLE FEATURES/APPLICATIONS

These kinds of demos make extensive use of the UDK's or a master menu. This kind of software allows you to select a set of specific features or applications to demo, based on a list of choices. To make a choice, you simply press the appropriate UDK indicated on the master list. This allows you to specify the order and extent of the features that you feel are important to the customer. Menu selection is in contrast to the page-to-page type of program where you are more inclined to follow the order provided by the tape. Each has advantages. Frequently a menu-selectable format will not provide the customer with a tutorial service to the same degree that the page-to-page format will.

Figure 5.7 is an excerpt of the output of the 7612D demo tape. The first frame determines what hardware you have available (Fig. 5.7a) and what the address of the 7612D is. After the initial questions and answers, the "master menu" is listed (Fig. 5.7b). The software will wait here until a UDK is pressed.

This is the main "decision branch" of the program and allows you to control what feature you want the customers to see next. In this example, acquiring data is typically the first task desired, so you would press UDK 2 (or 12). From that point, any of the computations may be appropriate. Examples shown are determination of pulse parameters from histogram data (Fig. 5.7d), and the FFT (Fig. 5.7e).

Again, you can see that this type of format gives you the flexibility to establish the order of features and benefits you feel is important to show to a particular customer.



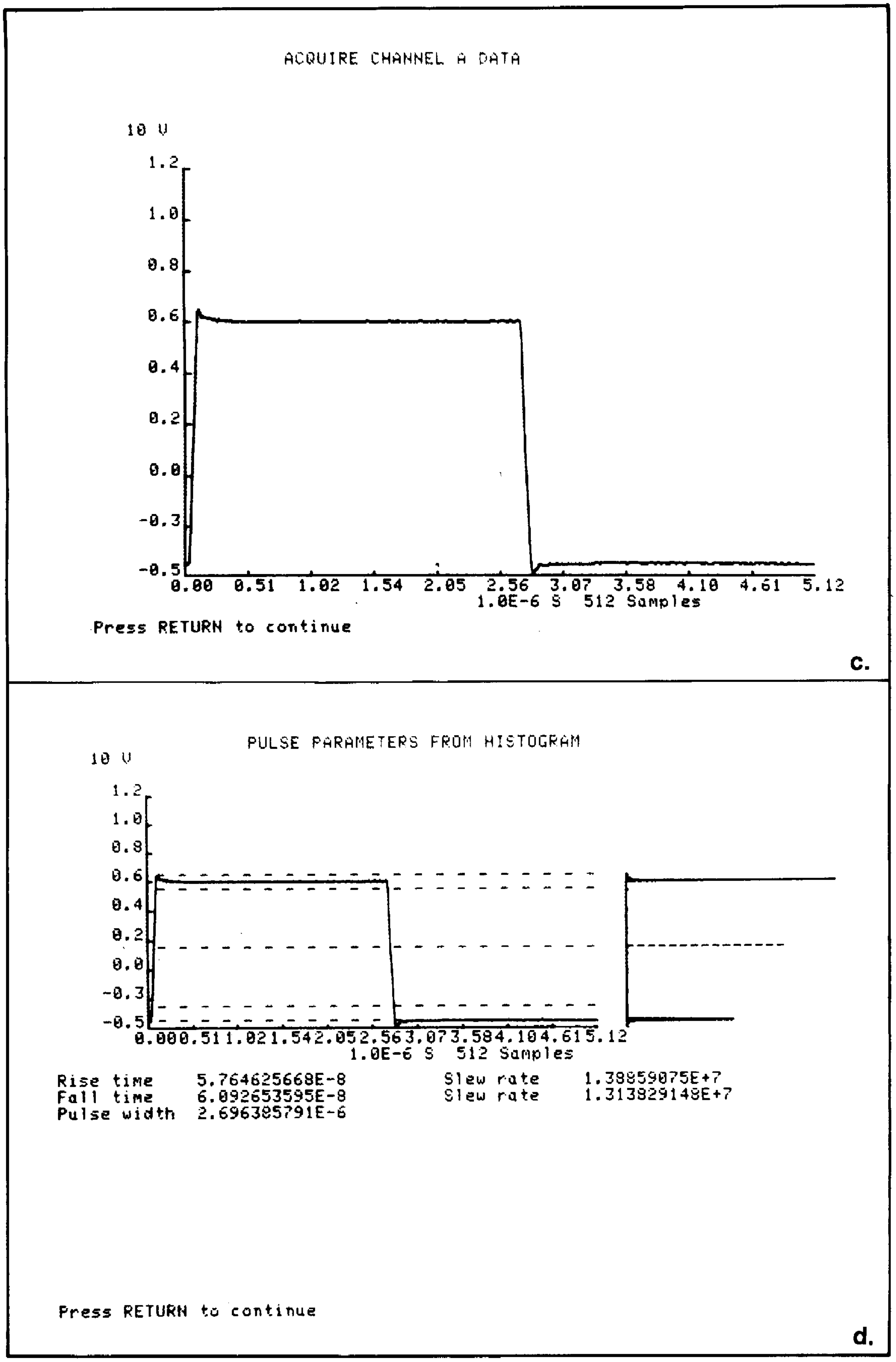
```
                    7612D / 4050-Series Demo Software
                    Copyright (c) 1980 Tektronix, Inc.
                           All Rights Reserved

Do you wish to have audible prompts (Y/N) ?N

Is there a 405xR07 installed (Y/N) ?Y

Is there a 405xR08 installed? (Y/N) ?Y

Do you wish to have a menu printed (Y/N) ?Y

What is the primary address of the 7612D (1--30) ?6
```

a.

```
                MENU
Key             Function

 1              Restore/Learned front panel settings
 2              Acquire channel B data
 3              Select segment
 4              Read segment from file
 6              Min/max P.P.A.
 7              Histogram P.P.A.
 8              Integrate
 9              Differentiate
10              RESTART
11              Learn front panel settings
12              Acquire channel A data
14              Save segment on file
15              Learn show cycle
16              Apply cosine taper
17              FFT
18              Correlate
19              Convolve
20              Cycle / Stop cycle
```

b.

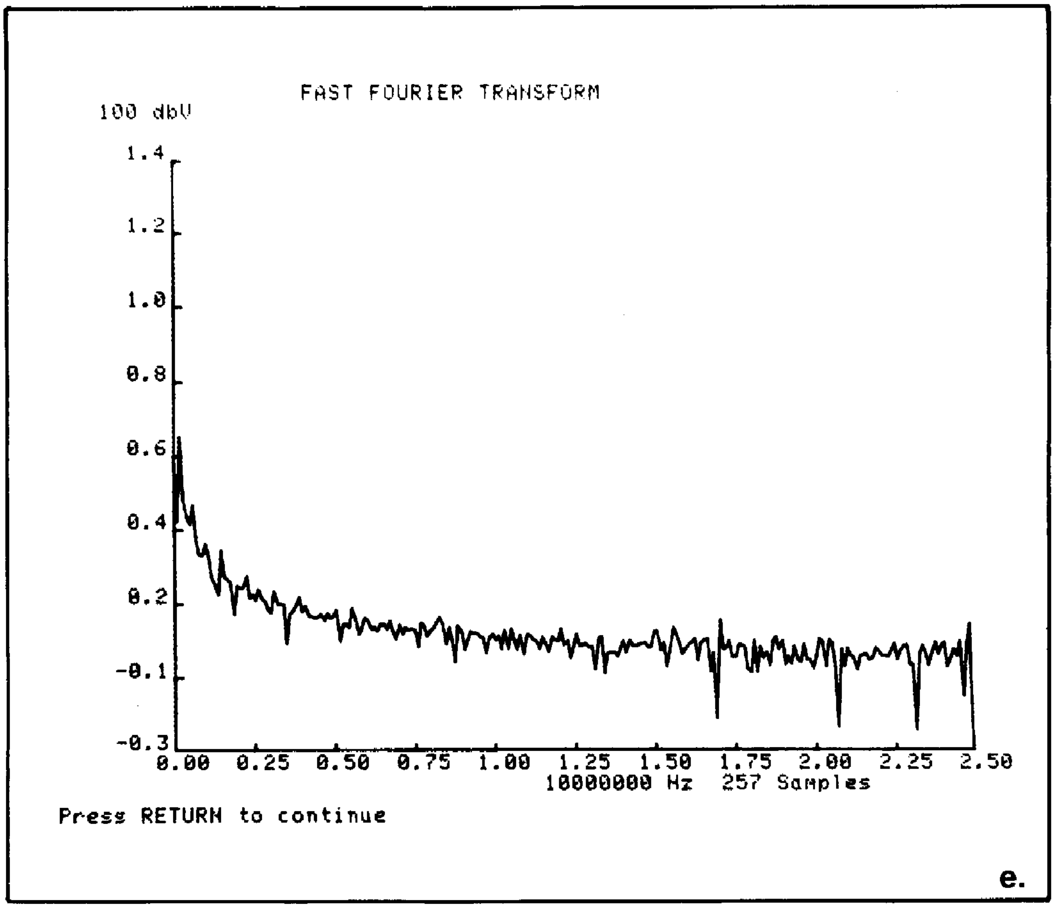
Fig. 5.7.



```
ACQUIRE CHANNEL A DATA

Press RETURN to continue
```

```
PULSE PARAMETERS FROM HISTOGRAM

Rise time    5.764625668E-8          Slew rate    1.38859075E+7
Fall time    6.092653595E-8          Slew rate    1.313829148E+7
Pulse width  2.696385791E-6

Press RETURN to continue
```

Fig. 5.7. (continued)



Press RETURN to continue

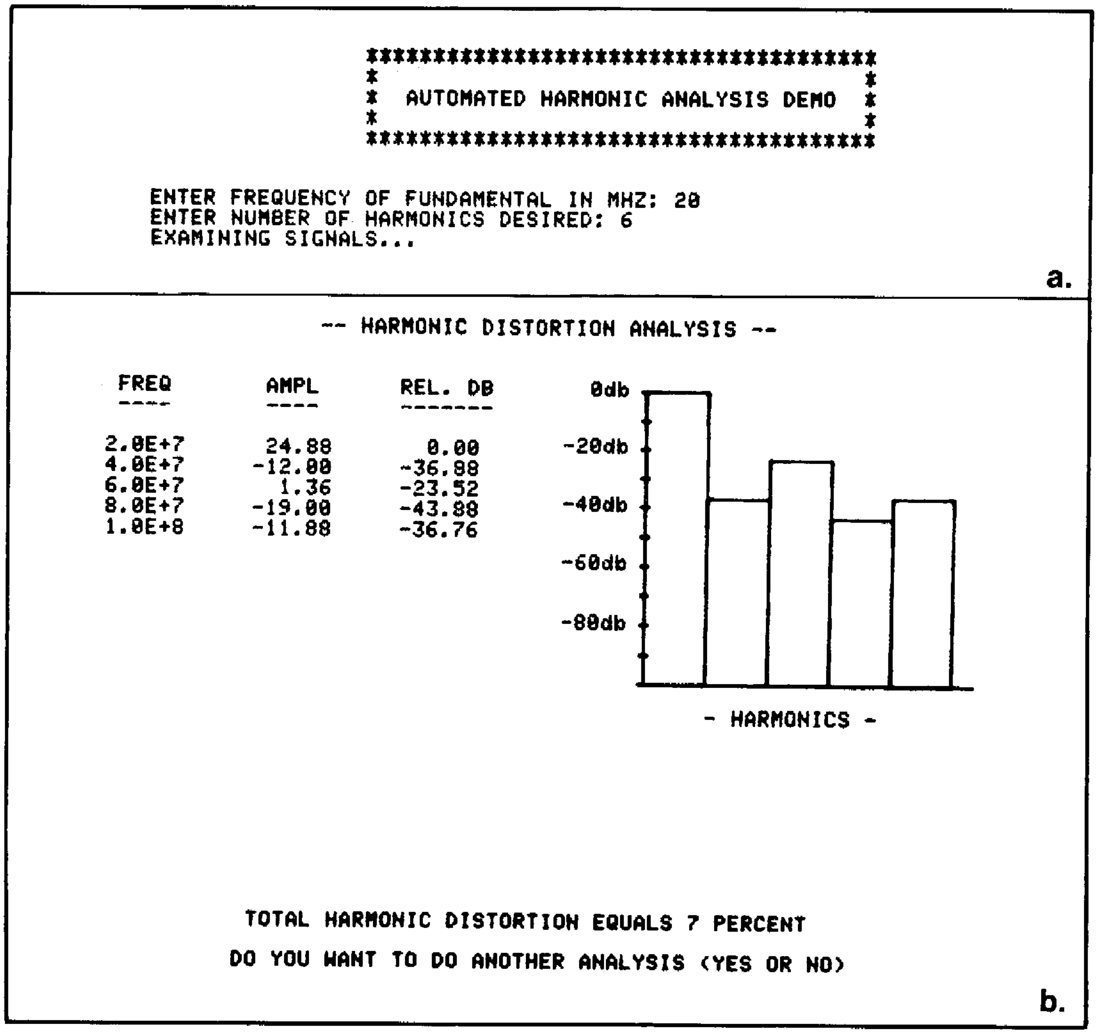
Fig. 5.7. (continued)

SPECIFIC APPLICATION DEMONSTRATION

Because applications for different products will be very diverse, there is a wide range of styles, techniques, and formats that these kinds of demo tapes may use. In fact, there are as many possibilities in this category as there are potential applications for your customers.

The common characteristic with this class of demo tape is that it demonstrates a product's features by inference. That is, while the program's apparent task is to describe and execute some particular application, it is, at the same time, using various features of a product to demonstrate that application. If it is at all similar to the customer's application, the demo tape becomes an extremely effective sales tool. Not only will customers see their applications being performed, they will know that software for those general tasks have already been writtten. A listing of the demo tape provides customers with an enormous headstart in getting their application software written.

Figure 5.8 shows the two frames which make up a very effective applications demo for the 492P. The application here is to compute the Total Harmonic Distortion for a sine wave which is attached to the 492P. What this illustration cannot show is how the software sequences the 492P through many frequency ranges, finds the individual harmonics, determines the amplitudes, and calculates the THD. The whole process takes about a minute and a half. If that is what the customer wants to do, this may be the extent of the product demo.



```
**************************************
*                                    *
*  AUTOMATED HARMONIC ANALYSIS DEMO  *
*                                    *
**************************************

ENTER FREQUENCY OF FUNDAMENTAL IN MHZ: 20
ENTER NUMBER OF HARMONICS DESIRED: 6
EXAMINING SIGNALS...
```

a.

```
          -- HARMONIC DISTORTION ANALYSIS --

 FREQ        AMPL        REL. DB
 ----        ----        -------

2.0E+7       24.88         0.00
4.0E+7      -12.00       -36.88
6.0E+7        1.36       -23.52
8.0E+7      -19.00       -43.88
1.0E+8      -11.88       -36.76



        TOTAL HARMONIC DISTORTION EQUALS 7 PERCENT
       DO YOU WANT TO DO ANOTHER ANALYSIS (YES OR NO)
```

b.

Fig. 5.8.

V. SELF TEST

Using a 4051 or 4052, demonstrate to your own satisfaction that you know how to do the following:

1. Power-up the 4050 and erase the screen.

2. Type the following statement into the 4050:

```
1ØØ PRINT "THE BUTLER DID IT"
```

using the EDIT keys change this to read:

```
1ØØ PRINT "THE BUTLER DID NOT DO IT"
```

3. Calculate the following arithmetic statements:

a. $\frac{4.25-1.78}{6.23x9E2}$ = __________

b. $\frac{7.5}{3.2-8.8^{2}}$ = __________

4. Print lines 200 through 400 on the 4050 screen of a program in user memory.

Answer the questions below. Then check your responses against the Answer Key provided in Appendix B.

5. What does the BREAK key do?

6. To enter a line into memory, or to get any immediate mode line to execute, what key must be pressed at the end of the line?

7. What is the difference between immediate mode and deferred mode?

8. Write the 4050 math expression which would calculate the following:

$$\sqrt{\frac{73.6 + 13.2}{(14.6-8)^2}} = \underline{\hspace{3em}}$$

9. What is a benefit of UDK's (User Definable Keys)?

10. List two general types of demonstration software.

11. Which UDK in standard Tek demonstration programs is always used to return you back to the master menu or master selection list?

12. What additional product(s) should you almost always be able to sell with any waveform instrument/4050 product combination?

13. List two other important non-instrument peripherals that a customer might be interested in when buying a 4050-series controller.

VI. 4050 BASIC

In working with a computer, a programmer must communicate with the computer. In order to do that, the programmer must use a language that the computer understands. There are many different computer languages. The one described in this chapter is BASIC. It is composed of English words and common mathematical symbols. When arranged in a precise and logical manner, these words and symbols become instructions to the computer and, eventually, programs.

Computers actually have a pretty limited range of capabilities. The tasks they can perform, however, they do many times faster and more accurately than people. These capabilities can be grouped into four areas:

- Calculation - number crunching
- Control - specifying a sequence of activities based on inputs and making decisions which allow for alternative sequences
- Remembering - storing and analyzing large quantities of data
- Communication - accepting inputs and generating outputs in a variety of forms

As we will see, almost every computer program has elements of all of these.

The following sections are a brief introduction to BASIC. The last page of this chapter lists several books or manuals that are a more complete introduction to BASIC. The best book as far as the 4050 is concerned is PLOT 50 Introduction To Programming In Basic, since it completely describes the 4050 language with its particular syntax. Not all BASIC's are alike so, if you use an alternate source book, be aware of differences in syntax and some command names.

There is information in this chapter that is repeated from the previous chapter, particularly regarding formation of mathematical expressions. This was done to make both chapters more complete. Since it is written in a slightly different form, it probably won't hurt to read through it quickly. If you get hung up on a concept, remember to get assistance from the Systems Analysts or GPI Marketing.

WHAT IS A PROGRAM?

```
100 INPUT A,B,C
110 LET Z=A+B+C
120 LET Z=Z/3
130 PRINT Z
140 END
```

This program consists of five statements. Every statement is preceded by a line number. Line numbers must be positive integers within the range of 1 to 65000. You can enter statements into the computer with any line number, but the computer will execute your program in numerical sequence, starting at the lowest line number. (Some instructions alter this flow. These are discussed later.)

Just as line numbers are required for program statements, so are commands required; these are special words that describe the action the computer is to take when the statement is executed. In line 100, the action is to INPUT information from the keyboard. In this case, three variables are input, having assigned variable names of A, B, and C. When the computer executes this statement, the variables A, B, and C will contain the values you entered from the keyboard.

Line 110 contains a LET statement. It tells the computer to LET the previously undefined variable Z contain the sum of the input values of A, B, and C. Had you entered 1, 2, and 3 when the INPUT instruction was performed, Z would now contain the value of 6. Notice that the equal sign does not mean the same as it does in a mathematical equation. Here, it means "let the variable on the left of the equal sign become equal to the value of the expression on the right."

An expression is a combination of defined variables, operators, and constants that result in a numeric value. Variables and constants are also called operands, or what the expression is to evaluate. Operators define the arithmetic operation to be performed in the expression (i.e., multiply, add, etc.).

Line 120 is an example of an expression containing a variable, operator, and constant. The "3" in line 120 is defined as a constant because it can never be redefined, or placed to the left of the equal sign. The value of 3 is 3 no matter what programming language you use.

The value of variable Z will be replaced by the value of the expression on the right side of the equal sign. The old value of Z will be lost forever. The operator is the "/" sign, which tells the computer to divide the value of the variable Z by the constant 3.

In line 130, the instruction is to PRINT the computed value of Z on the terminal. Line 140 simply tells the computer to halt execution of the program.

Let's take a moment to execute this program. Before typing this program into the 4050, first type the command DELETE ALL. This will clear the memory (RAM) of any previous programs and variables. After executing this command we are assured of a clear scratch pad to start working on our new problem.

```
100 INPUT A,B,C
110 LET Z=A+B+C
120 LET Z=Z/3
130 PRINT Z
140 END
```

Take time now and type in the above program.

When it is all typed in, check it for correctness by using the LIST command. (If you need to edit it, use the Edit Keys. See Chapter V.) If it is all right, begin execution by typing RUN. When the program first starts executing, notice that the cursor has become a blinking question mark "?" and that the I/O indicator light is turned on over on the right side of the 4050. The blinking ? means that the computer is waiting for the user to type data in on the keyboard. You can do this either by entering three numbers separated by commas, or three numbers separated by the RETURN key.

Review

The example is an entire program, consisting of input, processing, and output. Before going further, let's review the fundamentals just covered.

So far, we have discussed:

PROGRAMS, which contain a series of logically ordered

STATEMENTS, which are instructions to the computer to perform certain operations. Each statement must be preceded by a

LINE NUMBER, a positive integer (whole number) between 1 and 65000. Statements will be executed by the computer in sequence, beginning with the lowest numbered statement. Line numbers are generally "gapped" (usally by 10) to make room for additional statements.

COMMANDS, or statement types, tell the computer what kind of operation to perform in processing the statement.

VARIABLES are used to hold information (data values). A variable is named with one or two characters, the first being a letter, the second (optional) must be a digit (0-9). Examples of variable names are T, T9, Z. (BASIC on the 4041 will allow variables to have eight character names.)

CONSTANTS are actual numbers, and must be placed to the right of the equal sign in a LET statement.

OPERATORS are symbols used to define the mathematical operation to be performed in the statement. Operators are:

↑ meaning raise to the power of
* meaning multiply
/ meaning divide
+ meaning add
- meaning subtract

(Note: The = sign really means "replaced by" not "equal to.")

OPERANDS are defined variables or constants that, in conjunction with the operator, determine the value of the expression.

EXPRESSIONS are a series of operands and operators that are evaluated by the statement.

BASIC is a "free form language." All this means is that spaces don't matter in your statements. The only rule about spaces is that one must follow the last character of the command whether it is PRINT, LET, or whatever. Both the following examples are correct.

```
LET X5 = Z      +      3/      9
LET X5=Z+3/9
```

In this chapter, each section will conclude with one or more additional examples similar to the ones described in the text. These additional examples will not be as thoroughly described and are to be used to test your understanding of the concepts introduced.

Look over the following examples. If you do not completely understand how they would operate, type them into the 4050 and execute them.

```
90 INIT
100 INPUT A3,B0,C9
110 LET Z=A3+B0+C9
120 PRINT Z
130 END
```

INIT intitializes the 4052. All variables are set to a "neutral" state (neither zero or space), other 4052 characteristics are set to their default conditions.

```
90 INIT
100 INPUT A3
110 INPUT B0
120 LET R=Z3/B0
130 PRINT R
140 PRINT Z
```

Why doesn't Z print?

COMPUTATIONS IN BASIC

Most mathematical problems can be solved with your computer. But before your computer can know how to do the job, you must tell it exactly what steps to take and in what order to take them. You have to use the computer's language.

BASIC recognizes the following symbols for arithmetic operations. The priorities are discussed shortly.

SYMBOL	EXAMPLE	MEANING
↑	X↑Y	Exponentiation, 1st priority
*	X*Y	Multiplication, 2nd priority
/	X/Y	Division, 2nd priority
+	X+Y	Addition, 3rd priority
-	X-Y	Subtraction, 3rd priority

Writing expressions, or formulas, in BASIC is similar to writing an algebraic expression. However, all arithmetic operators (symbols) must be present. With algebra, it is possible to say XY and mean "X times Y," but not in the computer's language. The multiplication symbol must be present. "X*Y" is correct. If the symbol were left out, the 4050 would see "XY" and give you a syntax error.

Besides the five mathematical operators, other special "functions" are included in 4050 BASIC to further extend its power. In the following table, X may be a constant, variable, or expression.

FUNCTION	MEANING
SQR(X)	square root of X
EXP(X)	e raised to the X power
LOG(X)	natural logarithm of X
LGT(X)	logarithm to the base 10
ABS(X)	absolute value of X
INT(X)	next lowest integer from X
SIN(X)	sine of X
COS(X)	cosine of X
TAN(X)	tangent of X
ACS(X)	arc cosine of X
ASN(X)	arc sine of X
ATN(X)	arc tangent of X
RND(X)	generate random number between 0 and 1
SGN(X)	returns a +1 if X is positive, 0 if X is zero, -1 if X is negative

These functions are used within the expression to be evaluated. For example, to solve the problem:

$$X = \frac{\sqrt{Y^2 + Z^2}}{3Z}$$

The following statement would work.

```
LET X = SQR(Y↑2+Z↑2)/(3*Z)
```

It is assumed, of course, that the variables Y and Z have been previously defined, in either a LET, INPUT, or READ statement.

In the expression above, the computer finds the value of Y squared, and then adds the value of Z squared, creating the "argument" for the square root function. In "computerese," an argument is some value that a routine or function needs to do its job. The square root function needs something to take the square root of.

When an expression contains operators of the same priority, the calculations are performed in a left-to-right direction. For example, in the expression:

```
3+2-3+5
```

the computer would add 3 and 2, getting 5. It then would subtract 3 from 5, resulting in 2, and finally add 5 to get a final value of 7.

Watch what happens when you mix priorities.

```
X+.707*Y-1
```

In this expression, the computer will first find the product of .707 times Y, then add X, and finally subtract 1. If you had wanted to muliply the sum of X and .707 by Y, and then subtract 1, the expression should be written as:

```
(X+.707)*Y-1
```

Parentheses are also used to get around some restrictions in BASIC. For example, no two operators may appear side by side. To solve the addition of 3 plus minus 4, the following statement could be used:

```
LET X = 3+(-4)
```

Parentheses can also be used to nest functions within function arguments. For example:

```
LET X=SIN(SQR(4*Z))
LET X=ABS(COS(SQR(3.14159)))
LET X=LOG(EXP(4*SQR(5)))
```

These are legal BASIC statements.

On the 4050, all trigonometric functions are calculated based on angles expressed in radians. This can be changed, however. If you want the angles expressed in degrees, just enter a program line of SET DEG and all further calculations will be computed in that manner. To return to radians, enter SET RAD.

The command LET, by the way, is optional. That is, if you leave it out of a program statement, it will still perform the same activity. Henceforth, we will leave out the word LET in all assignment commands.

Here are some program examples:

```
100 INIT
110 INPUT A5
120 X9=SIN(A5)
130 PRINT X9
```

```
100 INIT
105 LET Z=1
110 LET X=SIN(SQR(4*Z))
120 PRINT X
```

```
100 INIT
105 LET Z=1
110 X=ABS(COS(SQR(3.1415)))
120 PRINT X
```

```
100 INIT
105 SET DEGREES
110 X=ABS(COS(SQR(3.1415)))
120 PRINT X
```

PRINT

The PRINT command is one of the more versatile commands in BASIC.

Calculations may be performed in PRINT statements. A simple program that computes the length of the hypotenuse of a right triangle, when the length of the other two sides are known, could be written in two statements.

```
100 INPUT X,Y
110 PRINT "THE LENGTH OF THE THIRD SIDE IS, ";SQR(X↑2+Y↑2)
```

Line 100 gets the values of the two known sides from you, and line 110 prints the message, computes the answer, and then prints it. Quote marks (") are used whenever the programmer wants to print a specific message. The quotes surround (programmers say "delimit") the message to be printed. The semicolon in line 110 delimits the end of the message and the value to be printed. If a comma had been used, the value of the third side would have been printed in a tab zone, right justified about 18 spaces from the end of the message. The semicolon packs the two fields (the message and the value of the expression) closer together, leaving only two spaces between the lst value (the message) and the next.

To execute this program, just type RUN after entering the two statements. When the question mark appears, enter two numbers (X and Y), separated by a comma. The answer appears as soon as you enter the carriage return. Remember, though, if you want to enter a new program after an old one has been run, type DELETE ALL first to clear memory.

Output from several PRINT statements can be printed on the same line by using the semicolon. Try the following:

```
100 J=1
110 PRINT J;
120 J=J+1
130 GO TO 110
```

Note: If your mathematical sense is bothered by J = J + 1, you should know that, in LET statements, the = sign does not mean "equals." Rather, the = sign means "replaced by." In this case, the old value of J, plus one, replaces J in storage.

This program prints each value of J in a tab zone, across the screen of the terminal. When the line is full, a carriage return and line feed is automatically inserted, and printing continues on the next line.

We will see also that the PRINT command is very useful in controlling devices over the GPIB. We'll come back to that later.

DRESSING UP YOUR PROGRAM

Most programs can be improved; the problem is knowing when to stop. Improvements in our averaging program are obvious. How many numbers should be entered? One way to find out is to LIST the program and look for INPUT statements. Another is to enter one value at a time, each followed by a carriage return. If the proper number of values has not been entered yet, the computer continues with the cursor question mark until enough values have been entered to satisfy the INPUT statement.

The easiest way is to have the program tell you just what is expected. Try this example (be sure to DELETE ALL before entering the example).

```
100 PRINT "ENTER THE THREE VALUES, SEPARATED BY COMMAS"
110 INPUT A,B,C
120 Z=(A+B+C)/3
130 PRINT Z,"IS THE AVERAGE"
140 STOP
```

Note the changes. A new PRINT instruction has been added to tell the user what input is expected.

Type RUN and carriage return and check your output. It should look like this:

```
RUN
ENTER THE THREE VALUES, SEPARATED BY COMMAS
1,2,3 <------------------------------------ The user enters these numbers
 2                 IS THE AVERAGE

STOP IN LINE 140
```

It's time now for you to write some programs. Before you do that, however, let's look at one more example of a slightly more complex program.

```
100 PRINT "This program computes the roots of a two degree polynmial"
110 PRINT "Enter A, B, and C of the equation AX↑2+BX+C=0"
120 INPUT A,B,C
130 REM
140 REM    FIRST WE WILL SEE IF THERE ARE ANY IMAGINARY ROOTS
150 REM
160 R=B↑2-4*A*C
170 REM
180 REM    IF THERE ARE IMAGINARY ROOTS, WE WILL GOTO 350
190 REM
200 IF R<0 THEN 350
210 REM
220 REM    IF THE ROOTS ARE REAL, GO AHEAD AND CALCULATE THEM
230 REM
240 R1=(-B+SQR(R))/(2*A)
250 R2=(-B-SQR(R))/(2*A)
260 REM
270 REM    PRINT THE RESULTS
280 REM
290 PRINT
300 PRINT "The roots of the equation  ";A;"X↑2+";B;"X+";C;"=0"
310 PRINT "are  ";R1;"   and ";R2
320 PRINT
330 PRINT
340 GO TO 110
350 REM
360 REM    HERE IS OUR "ERROR" HANDLING ROUTINE
370 REM
380 PRINT "THERE ARE NO REAL ROOTS"
390 PRINT
400 PRINT
410 GO TO 110
```

This program is pretty straightforward, although there are a couple of commands we have not discussed yet. All of the REM commands (for REMARK) are simply comments to anyone who might be reading the program. These remarks are extremely valuable to programmers who might have to come back to a program after a long time has elapsed. They are also useful to a new programmer who may be required to update or maintain a program that someone else has written.

This program is actually very similar to the examples we have just seen. Lines 100 through 120 request data; lines 160, 240, and 250 compute some arithmetic; and line 310 prints out the result.

We'll discuss IF...THEN and GO TO statements in the next section.

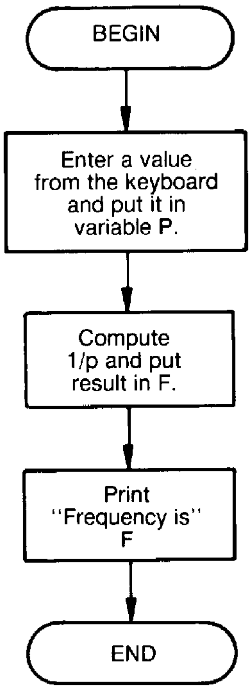
Here are a couple of problems for you to program and then to enter into the 4050. When writing programs, do it on paper, not the 4050. You will find this less frustrating and far less prone to error.

For these first couple of exercises, we will provide a flowchart. Flowcharts are extremely valuable tools in diagramming the flow of activity and information through a program. Without a flowchart to guide you, programming a task becomes a hit or miss proposition. Even experienced programmers will generally write a flowchart for any application that looks like it will require more than 20 lines of code. You might be amazed at how complex a program can be written with just 20 lines. The example we just looked at only had 12 executable lines.

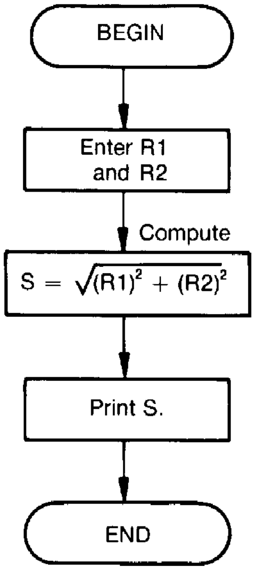
Flowcharts are quite simple, really. Beginning and endings of programs are shown by an oval. Computations and input/output activities are diagrammed with a rectangle. That's all we need for now. By the way, possible answers to the exercises are at the end of the chapter. There are, of course, many possible ways to program and solve any given problem. Very rarely do two experienced programmers write exactly the same instructions to solve a particular problem.

EXERCISES

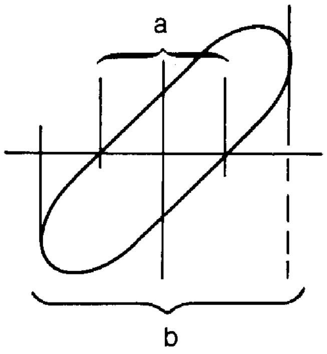
1. Ask for the period of a waveform then compute and print its frequency.


BEGIN
Enter a value from the keyboard and put it in variable P.
Compute 1/p and put result in F.
Print "Frequency is" F
END

2. Write a program that requests the risetime of a probe and the risetime of an oscilloscope and computes the system risetime.

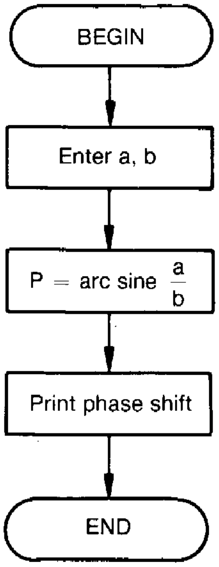


3. To determine the phase-shift of an oscilloscope, we can put it into XY mode and put the same sinewave signal into both sides. The result we get will look something like this:



To calculate the phase shift, we measure the length of a and b and compute arc sine (a/b).

Write a program to request a and b and compute the phase shift.



PROGRAM CONTROL

Program control commands are those BASIC commands which alter the order of execution from the normal flow. Remember that the normal flow is from the lowest line number up through ever-increasing line numbers.

Let's take care of two control commands right off the top. The STOP and END commands can be used anywhere in a program when you want it to terminate execution. Normally you would expect to put them at the physical end of a program but, frequently, you will see them in various spots in the middle. This is either because the logical end of the program is not at the physical end (a very common occurrence because of subroutines and error handling routines), or because there are multiple paths the program may take and each path requires a termination point. STOP and END are, for the most part, identical, except that the STOP command will print at the line where it stops. END doesn't print anything. Try the following brief exercise to see the difference between STOP and END.

```
100 INIT
110 INPUT V3
120 R4=V3/PI
130 PRINT R4
140 STOP
```

Line 120 — PI is a function for the trig. value 3.1415---

Type RUN What happens?

Change line 140 to: 140 END. Now what happens?

Another control command is GOTO. GOTO is used to transfer program control to a specified line number unconditionally, i.e., every time a GOTO command is encountered, the program is transferred to the line specified in the statement. GOTO's are often found at the end of programs or at the end of other logical routines, typically in spots you might otherwise use STOP, except that you want to run the program repeatedly. GOTO's can be used to create "loops." A "loop" is a section of programming which circles back on itself some number of times. The following example is an infinite loop:

```
100 I=0
110 I=I+1
120 PRINT "THIS IS LINE ";I
130 GO TO 110
```

Note that I=I+1 is "mathematically" invalid. What I=I+1 really means to the 4052 (or any computer) is "I is-replaced-by I+1. That is, the equal sign means "replaced by" to the 4052.

In line 120 the semi-colon (;) suppresses the spacing to just one space (rather than 18 in BASIC).

Try this by changing the semi-colon to a comma.

Virtually all "endless loop" programs are created with at least one GOTO. The BREAK key is used to terminate these programs. If you are not sure what the above program does, type it in and RUN it.

PUTTING INTELLIGENCE IN YOUR PROGRAM

So far, the way we have been using the 4050 is more or less as an overgrown calculator. Computers have the ability to "make decisions" based on inputs and this is one of their major features.

The following program sums a list of numbers. It will accept any quantity of numbers but will print the sum only when you enter a zero.

```
100 PRINT "THIS PROGRAM SUMS VALUES"
110 PRINT "ENTERING A VALUE OF ZERO"
120 PRINT "PRINTS THE SUM AND STOPS"
130 S=0
140 PRINT "ENTER A VALUE"
150 INPUT V
160 IF V=0 THEN 190
170 S=S+V
180 GO TO 140
190 PRINT "THE SUM IS ";S
200 STOP
```

Line 130 "forces" S=0.
Doing this assures us that the value named S will be zero to begin with.

Note that lines 140 prints a "prompt message" to help the person using the program.

DELETE 130, type RUN and see what message you get.

Line 130 initializes variable S to a value of Ø. Since this is the variable in which we are going to hold the sum, we have to give it a beginning value. Most large programs will have several lines of initialization. Lines 140 and 150 request and accept a value from the keyboard.

Line 160 is our decision-making line. If the user has entered a zero, the program transfers its control to line 190. If the value is anything other than zero, the program just "falls through" to the next line at 170. Line 170 adds the new value V to the summation variable S. S is keeping the running total as the user enters each new value; then we go back for another number.

Eventually, when the user types zero, control jumps to line 190 where we print the sum.

The format of an IF statement is:

```
IF condition THEN line number
```

The condition is the logical expression that is tested. If the logical expression is true, the computer branches to the specified line number. If the statement is false, the computer goes on to the next line of the program.

The condition is similar to an expression, but no action is taken other than evaluation of the condition to determine if it is true or false.

For example, the statement:

```
IF X=Z THEN 2ØØ
```

would send the computer off to statement 200 if X was equal to Z. If X were not equal to Z (a false statement), then program control goes to the line immediately following the IF statement.

Relational operators are used to define a condition. On either side of the condition may appear variables, constants, or expressions.

RELATIONAL OPERATORS	EXAMPLE			MEANING
=	IF	X=Z	THEN	Equal to
<	IF	4+Q<T*Z	THEN	Less than
<=	IF	Z<=Y	THEN	Less than or equal to
>	IF	5*Z>Z+Y	THEN	Greater than
>=	IF	Y>=Z	THEN	Greater than or equal to
<>	IF	Y<>Z	THEN	Not equal to

Note: The = sign in an IF statement means "equal to," unlike the same symbol in LET statements.

A very common application in a production test environment will be to measure a value and to see if it is within specified limits. For example, a DMM will be connected to a DUT which is stimulated by a power supply. We will want to see if the output of the DUT responds within a particular range.

The following program simulates the above problem where the "pass" range is from 5 to 6 volts:

```
90 INIT
100 PRINT "ENTER VOLTAGE"
110 INPUT V
120 IF V=5 THEN 150
130 PRINT "DEFECT-OUTPUT LOW"
140 GO TO 100
150 IF V=6 THEN 180
160 PRINT "DEFECT-OUTPUT HIGH"
170 GO TO 100
180 PRINT "PART PASSES"
190 GO TO 100
```

How would you stop this program? (one way would be to add: 115 IF V=-999 THEN 200)

Try it and see.

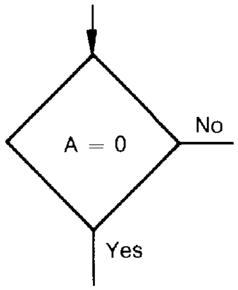
By providing a logical ending to this program, we have used -999 (a "control" value). -999 was used because in this case it could never be a real value found with this data.

Type this program in and RUN it if you wish.

In this program, lines 120 and 150 are testing the boundary conditions; if they are both true, the part passes. If either one of them is false, then the part is out of range.

EXERCISES

The IF...THEN statement adds one more flowchart diagram to our collection. It looks like this:

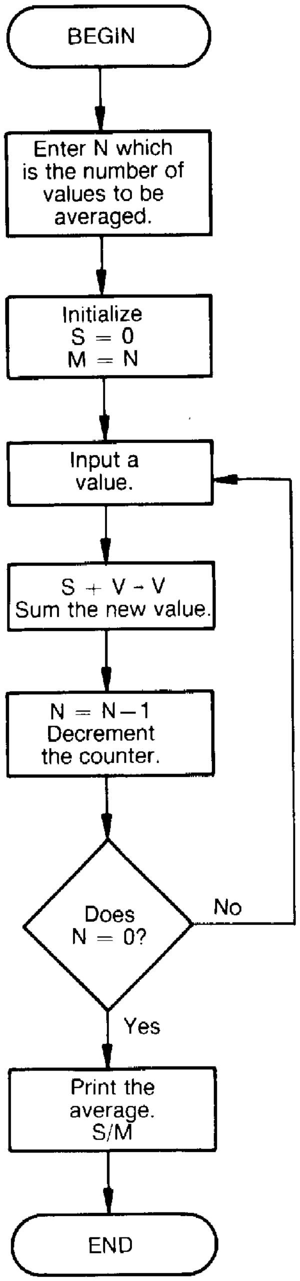


GOTO commands will simply appear as lines going from one box to another point in a program. There will be arrows on the line to describe the direction of program flow.

4. Write a program which allows the user to enter upper and lower boundary limits. Then, based on keyboard entries, check to see if the keyboard entry is above, below, or within the appropriate range.

   Write a flowchart first. Use a separate sheet of paper.

5. Write a program which averages n numbers. (Again, there are many wys to write this program. The flowchart diagrams one method.)


BEGIN
Enter N which is the number of values to be averaged.
Initialize S = 0 M = N
Input a value.
S + V → V Sum the new value.
N = N−1 Decrement the counter.
Does N = 0?
No
Yes
Print the average. S/M
END

MORE CONTROL

Let's look at another way to write the program you just wrote for exercise 4. In exercise 4 we used a counting variable N to keep track of how many values had been entered. We can also do this with a FOR/NEXT loop. It might look like this:

```
90 INIT
100 PRINT "INPUT NUMBER OF VALUES TO BE AVERAGED"
110 S=0
120 INPUT N
130 PRINT "ENTER EACH VALUE, ONE-AT-A-TIME"
140 FOR I=1 TO N
150 INPUT X
160 S=S+X
170 NEXT I
180 PRINT "THE AVERAGE IS ";S/N
190 STOP
```

Note: Actually, line 11Ø S=Ø can be placed anywhere between line 90 and line 140.

Lines 150 and 160 are enclosed in a loop. They will be repeated as many times as necessary to accept the number of data values you have.

FOR/NEXT loops allow you to program boring and repetitive tasks without having to program boring and repetitive statements. For example, two ways to find the sum of the first 10 integers are:

```
100 LET S=1+2+3+4+5+6+7+8+9+10
110 PRINT S

100 FOR X=1 TO 10
110 S=S+X
120 NEXT X
130 PRINT S
```

Both programs used almost the same amount of characters. But you can see that to find the sum of the first thousand integers requires making only one change in the second example, but 990 additions to the first.

The format for the FOR statement is

Index — Final Value

FOR I = J TO K STEP Z

Initial Value — Stepping Value

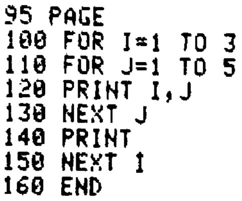
The index must be a simple variable (not dimensioned) and is the variable that will be incremented each time the loop is performed. The intial value may be a constant, simple variable or expression. The index will be set equal to the initial value on the first pass through the loop. Termination of the loop is decided by the final value. Again, a constant, simple variable or expression may be used to tell the computer when the loop is completed.

The stepping value is optional. If it is left out, a value of +1 is assumed. If included in the statement, whatever value (a constant, simple variable, or expession) used will be added to the index on each pass. If the stepping value is negative, the initial value must be greater than the final value.

The terminator of the loop is the NEXT statement. The argument for this statement must match the index variable of the last FOR statement executed.

Any number of statements may appear between the FOR and NEXT statement, even other FOR/NEXT loops. There are some restrictions on this "nesting" of loops, however,

Nesting means that the body of a FOR/NEXT loop can contain an additional FOR/NEXT loop. Consider the following program segment:



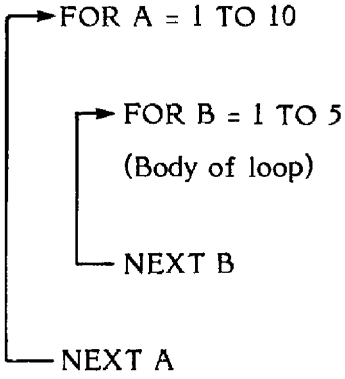
```
95 PAGE
100 FOR I=1 TO 3
110 FOR J=1 TO 5
120 PRINT I,J
130 NEXT J
140 PRINT
150 NEXT I
160 END
```

The first time this program segment is executed, I (in the "outer" loop) retains the value of 1 while J (in the "inner" loop) assumes the values of 1, 2, 3, 4, and 5. Then, I is incremented to the value of 2, and J again becomes 1, 2, 3, 4, and 5.

Finally, I becomes 3, J goes through its 5 values, and the outer loop terminates. This operation can be seen in the output produced by the above program segment.

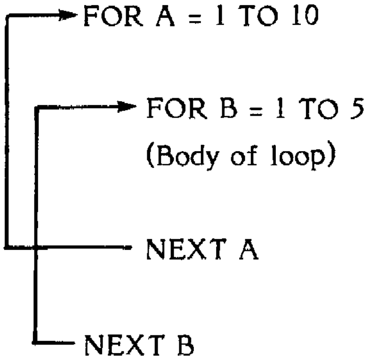
```
 1          1
 1          2
 1          3
 1          4
 1          5

 2          1
 2          2
 2          3
 2          4
 2          5

 3          1
 3          2
 3          3
 3          4
 3          5
```

Loops can be nested in this fashion indefinitely. One thing to watch out for when nesting loops is to make sure that the loops do not "cross." That is, the control paths for nested loops should look like this:



```
FOR A = 1 TO 10

    FOR B = 1 TO 5
    (Body of loop)

    NEXT B

NEXT A
```

The control paths should not look like the following:



```
FOR A = 1 TO 10

    FOR B = 1 TO 5
    (Body of loop)

    NEXT A

NEXT B
```

SUBROUTINES

Suppose you had a calculation to perform at several different points in your program. You could code in the instructions every time it was necessary, but that wastes computer memory. Subroutines avoid this problem. A subroutine is one or more statements that can be called into use from any part of the program.

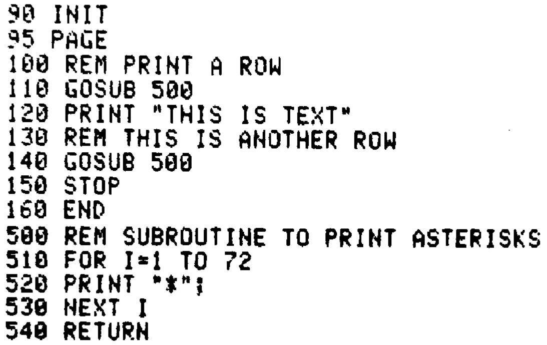
Another way to look at subroutines is to say that they are programs within larger programs. The main program will generally execute the subroutine several times. Sometimes they are even called sub-programs. Anyway, subroutines will require that we know two new commands.

GOSUB line number

and RETURN

The GOSUB command is very much like a GOTO command except that, when it is executed, the program remembers where it came from. At the end of a subroutine, a RETURN statement will send program control back to the line following the one having GOSUB on it.

Let's look at two examples. The first example prints one row of asterisks across the 4050 screen. The screen is 72 characters wide:



```
90 INIT
95 PAGE
100 REM PRINT A ROW
110 GOSUB 500
120 PRINT "THIS IS TEXT"
130 REM THIS IS ANOTHER ROW
140 GOSUB 500
150 STOP
160 END
500 REM SUBROUTINE TO PRINT ASTERISKS
510 FOR I=1 TO 72
520 PRINT "*";
530 NEXT I
540 RETURN
```

Admittedly, this is not too exciting an application, but the important thing is to observe the use of GOSUB and RETURN statements.

In line 110, we execute a GOSUB 500. This is identical to a GOTO 500 (except that the computer remembers where it came from) and program control is transferred to line 500. Lines 510 through 530 print out 72 asterisks all on one line (notice the use of the semicolon at the end of 520). Line 540 returns program control to the line immediately following the GOSUB which called it (i.e., transferred control to it). In line 12 we print some text; then, in line 140, we go back to the subroutine again.

The subroutine executes as before, but when we hit the RETURN statement, we go back this time to line 150. We will see later that the RETURN statement is used in other contexts such as for interrupt handling, and for UDK's, but in each case it acts just the same as here. Whenever RETURN is executed, control is passed back to the line immediately following the point at which control was transferred to the subroutine (in the above example, the GOSUB statements).

Another example. Suppose you are writing a demo program, and you would like to border the entire 4050 screen with asterisks. We know that the screen is 72 characters wide and 33 tall. Of course, it could be done with 32 print statements, but that's pretty inefficient. Here's how it could be done:

```
100 INIT
105 REM LINE 730 HAS 58 SPACES BETWEEN THE QUOTE MARKS
106 REM LINE 740 HAS 12 SPACES FOLLOWING THE QUOTE MARK
110 PAGE
200 GOSUB 700
210 HOME
220 PRINT
230 PRINT " THIS IS FRAME ONE"
234 CALL "WAIT",2
235 PAGE
240 GOSUB 700
250 HOME
260 PRINT
270 PRINT " THIS IS FRAME TWO"
280 END
700 REM     SUBROUTINE TO PRINT FRAME
710 GOSUB 800
720 FOR K=1 TO 33
730 PRINT "*
740 PRINT "            *"
750 NEXT K
760 GOSUB 800
770 RETURN
800 REM     SUBROUTINE TO PRINT ROW OF ASTERISKS
810 FOR J=1 TO 72
820 PRINT "*";
830 NEXT J
840 RETURN
```

This command does not work on the 4051.

This program is somewhat similar to the last example, but there are a few noteworthy differences. Notice first that the same subroutine that we used to print a row of asterisks in the last example is also in this example at lines 800 through 830.

In this program, however, we GOSUB at line 200, and this subroutine is supposed to generate a frame of asterisks. But look, the first thing it does is to go to another subroutine. This is called nesting subroutines. Now the computer not only has to remember that it made one subroutine entry from line 200 but another one from line 710. The computer keeps track of these so that the first RETURN it encounters will send program control back to the statement just following the last GOSUB it executed. If you trace through the program flow of the above program, I think you find that this is what you would have expected.

Type the following program in and RUN it.

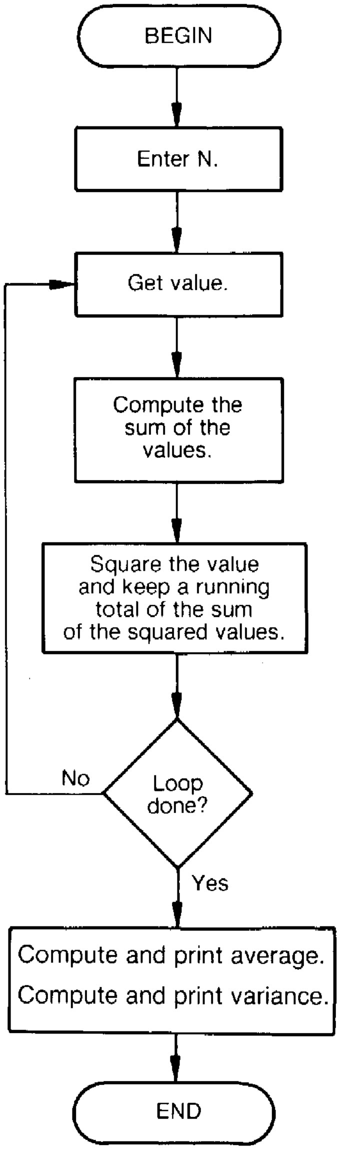
```
90 INIT
100 PAGE
110 REM ANOTHER SUBROUTINE EXAMPLE
120 FOR J=1 TO 10
130 GOSUB 160
140 NEXT J
150 END
160 REM SUROUTINE TO PRINT A PERIOD REPEATEDLY
170 PRINT "NO.";J;"."
180 RETURN
500 REM SUBROUTINE TO PRINT ASTERISKS
510 FOR I=1 TO 72
520 PRINT "*";
530 NEXT I
540 RETURN
```

EXERCISES

6. Using a FOR/NEXT loop, print out the numbers 1 through 15 in a vertical column on the screen.

7. Compute the mean and the variance of a set of N numbers where the computer asks what N is. Use a FOR/NEXT loop. (There is no special notation for FOR/NEXT loops in flowcharts; they look like IF diagrams. This is because any FOR/NEXT loop could be replaced by a counting variable and an IF statement, though less efficiently.)

$$\text{mean} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \text{variance} = \sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}$$


BEGIN
Enter N.
Get value.
Compute the sum of the values.
Square the value and keep a running total of the sum of the squared values.
Loop done?
No
Yes
Compute and print average.
Compute and print variance.
END

STRINGS

All of the programming techniques discussed so far have been concerned with arithmetic data--numbers have been read into the memory, something is done to the numbers, and numbers appear as output. In some of the examples, alphabetic data has been carried along in the form of literal strings used to enhance output or prompt for input. Up to now, however, no operations have been performed on alphabetic data, or "character strings."

BASIC provides a number of built-in facilities for processing character strings or non-numeric data. The reason you are being introduced to strings is that most GPIB products communicate in ASCII strings (see Chapter IV).

String data can be assigned to string variables with the familiar assignment operator. String variables consist of one letter (A through Z) followed by a dollar sign ($), as in A$, B$, X$, etc. String assignment examples:

```
100 A$="ABCDE"
110 B$="PROGRAM"
120 C$=A$
```

Generally, string assignment statements look like:

target string=source string

which means that characters are transferred from the source string to the target string. The source string can be a previously defined string variable (as in line 120 above) or a character string enclosed within quotes (lines 100 and 110 above).

Try this program:

```
90 INIT
95 PAGE
100 LET A$="ABCDE"
110 B$="PROGRAM"
120 C$=A$
```

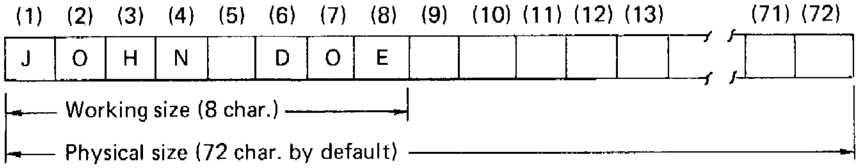
Now add lines 130,140,150 as shown.

```
125 REM  ADD THE FOLLOWING INSTRUCTIONS
130 PRINT C$;"=C$"
140 PRINT B$;"=B$"
150 PRINT A$;"=A$"
```

String variables in the 4050 have a default length of 72 characters. This corresponds to the maximum number of characters that can be printed on one line of the display. The total length of a string variable can be thought to consist of a working size and a physical size. To illustrate, consider a statement like:

```
5ØØ      A$="JOHN DOE"
```

where A$ has not been previously dimensioned. This commits space in the memory as shown below:



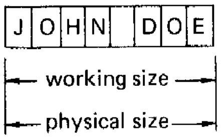
Assuming for the moment that no further characters are to be added to A$, the resulting situation is that a significant amount of space in memory is left unused. The way around this is to allocate space using a DIM statement. DIM statements are also used for numeric arrays, but they are beyond the scope of this text. The form of the DIM statement for strings is:

DIM string variable (numeric expression)

where the numeric expresson is rounded to an integer and becomes the physical size of the string. Strings may be dimensioned longer than 72 characters; the maximum length is bounded only by the amount of memory available. Continuing with the example of "JOHN DOE", a more appropriate utilization of memory results through the following program segment:

```
5ØØ DIM A$(8)
51Ø LET A$="JOHN DOE"
```

This allocates memory for A$ as follows:



Here, the working size and the physical size are the same, resulting in a more efficient use of memory space. In using GPIB products, however, we will see that we frequently need fairly long strings (such as when an instrument returns the settings and status of all its controls).

Of course, you will get an error message if you try to stick more characters into a string variable than it has been dimensioned for.

The INPUT and PRINT statements can be used with string data in essentially the same manner as with numeric data. A statement like:

```
15Ø INPUT A$
```

behaves as before, placing a question mark on the screen indicating that the BASIC interpreter is awaiting data. You then enter the appropriate character string and press RETURN. The string need not be enclosed in quotes. If the string is enclosed in quotes, then the quote marks just become part of the string.

String variables can also be mixed with numeric variables in a PRINT statement, as in:

```
2ØØ PRINT X$;Z;C$
```

or, look at this GPIB example:

```
2ØØ C$="FREQ "
21Ø PRINT @24:C$;H
```

Here, a FG5010 at device address 24 would have changed its frequency to whatever value was contained in the variable H (assuming it was within range).

STRING COMPARISON

Control of program execution can be accomplished through string comparisons resulting in conditional transfers. The method is quite similar to the way in which numeric comparisons are performed, using an IF...THEN...statement as before. The syntax form is essentially the same:

IF condition THEN line number

The only difference is the way the condition is constructed, taking the form:

character-string relational-operator character-string

where the relational-operator, as before, can be any of the six relational operators and character-string can be either a string constant or a string variable.

Some examples of string IF...THEN... statements:

```
100 IF A$=B$ THEN 500
110 IF A$>F$ THEN 600
120 IF B$="END" THEN 300
```

Characters are represented in the 4050 using the ASCII code. In this scheme, each character in the set of 128 characters has a unique decimal equivalent within the range of 0 through 127. Thus, "A" is 65, "6" is 54, "!" is 33, etc. (Appendix A has an ASCII chart showing these decimal values.) The fact that each character is represented numerically forms the basis for making string comparisons. You can say that "A" is less than "B" because "A", in ASCII decimal equivalence, is 65, while "B" is 66.

Character strings are evaluated on a character-by-character basis from left to right. The first character inequality discovered in the two strings determines the relationship. Consider the following two character strings:

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
①	P	A	R	E	N	T	H	E	S	I	S
②	P	A	R	E	N	T	H	E	S	E	S

In comparing these two strings, the BASIC interpreter determines that string 1 is greater than string 2. This relationship is determined when the 10th pair of characters (the first inequality) is examined. The first 9 pairs are equivalent, but in the 10th pair the "I" has a numeric representation greater than "E". No further comparisons are performed once the first inequality is located.

Occasionally in GPIB applications a user will want to link several strings together to form a command to send an instrument.

Concatenation is the process of linking (or joining) character strings together. If you concatenate a "T" and an "H" and an "E", you get "THE". Concatenation is accomplished with the concatenation operator, which is the ampersand symbol (&). You concatenate string constants as shown below:

```
1ØØ A$="CHARACTER"&"STRING"
```

which yields "CHARACTERSTRING". Or, the operation can involve string constants and string variables, as in

```
1ØØ A$=B$&"ES"
11Ø A$="ANTI"&B$
```

Also, concatenation can involve string variables such as:

```
1ØØ A$=B$&C$
```

Remember, the target variable (the string variable to the left of the equal sign) must be dimensioned long enough to accept the new (and presumably longer) string.

The following section on string functions may be skipped. It is included as reference because string functions will invariably show up in any GPIB applications program.

STRING FUNCTIONS

In working with GPIB products, we will be dealing with ASCII strings quite frequently--that is how we communicate with GPIB products much of the time (not all -- remember binary blocks from Chapter IV, etc.)

Several string functions allow us to manipulate these strings so the computer can more effectively deal with certain parts of the data that an instrument sends back. These are usually parts of the "applications level" communications drivers we discussed in Chapters II and IV.

When a string contains an unkown number of characters, as might be the case when bringing in data from a GPIB instrument, you can determine the current length (or number of characters) by employing the built-in length function. This is written and used in a manner which is essentially the same as the mathematical functions.

The LEN function returns an integer indicating the number of characters, or working size, of the specified string variable. An example:

```
1ØØ LET A$="ALPHABET"
11Ø LET X=LEN(A$)
```

In this case, X is assigned the number 8. LEN works just like a math function, except that it has a string argument.

You can extract a substring from an existing string with the three-parameter SEG (for segment) function. This has the form:

string var.=SEG(string var., numeric exp., numeric exp.)

where the first numeric expression specifies the starting position of the substring within the string, and the second numeric expression specifies the number of characters contained in the substring. A short example:

```
100 READ A$
110 B$=SEG(A$,7,5)
120 DATA "HENRY JONES"
130 PRINT B$
```

The READ command and DATA work with each other. DATA permits the programmer to include in the program data to be read by the program (usually constant values)

If you use READ you must also use DATA.

In this case, B$ is assigned the substring "JONES", which is a 5 character substring starting at the 7th position of A$ (counting the blank character). A further example:

```
95 PAGE
100 A$="SUBSTRING"
110 FOR I=1 TO LEN(A$)
120 B$=SEG(A$,I,1)
130 C$=SEG(A$,1,I)
140 PRINT B$,C$
150 NEXT I
```

The output from the above example:

```
S               S
U               SU
B               SUB
S               SUBS
T               SUBST
R               SUBSTR
I               SUBSTRI
N               SUBSTRIN
G               SUBSTRING
```

Sometimes when getting data out of a GPIB product, you get ASCII strings of which only a small portion represents a number. The problem is that, if the computer needs to use the ASCII representation of a number as a computer representation (called floating point) of that number, then we need a method of conversion.

For instance, you might be confronted with ACSII numbers, which are really characters, when you need their decimal counterparts instead. Obviously, you can't expect to get the square root of the letter "7" because it is a character like "X" and "?". What is needed is the means to convert between ASCII numbers (character strings) and their numeric equivalents. This capability is provided by a function called VAL (for value).

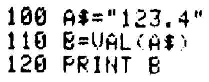
The VAL function takes the form:

```
target string var.=VAL(string var.)
```

where argument string variable refers to an existing character string which contains elements of the following character set:

The letters 0 through 9
The monadic operators "+" and "-"
The decimal point
The exponent symbol "E"

For example:



```
100 A$="123.4"
110 B=VAL(A$)
120 PRINT B
```

Statements 100 and 110 cause the numeric value B to receive the value 123.4. This example is, perhaps, an over-simplification, because it is equivalent to writing:

100 LET B=123.4

However, you can get the idea behind the VAL function.

EXERCISES

Frequently, a program will execute some computation once, then ask the user if it is to be run another time. To do this requires adding a PRINT statement to ask the user the question, an INPUT statement for the reply, and an IF statement so that, if the user says "YES", it goes back to the beginning; otherwise, it stops.

8. Take the program you wrote in exercise 1 and modify it so that it asks if the user wants to execute the program again, and then do it if the response is "YES".

INPUT/OUTPUT OVER THE GPIB

Input and output of data over the GPIB is, in many cases, not much more complex than input and output from the keyboard and screen of the 4050. This is particularly true for products like the TM5000 instrument; data input from waveform products and spectrum analyzers are a little more complex because of the quantity of data and the fact that the data it comes over the bus in binary rather than ASCII numbers.

To send data to an instrument, we use the PRINT statement. Here are some examples:

```
1)   1ØØ      PRINT @18:"FREQ A"      (DC5010)
2)   1ØØ      PRINT @14:"SPAN 1ØK"    (492P)
3)   1ØØ      PRINT @9,1:"V/D .2"     (7A16P)
```

In each of these examples, we are simply changing the setting of some control. Notice the syntax. The PRINT statement is followed by an "@" character which is then followed by a GPIB address (valid addresses are Ø-3Ø) and a colon (:). The first two examples show instruments which are addressed with a primary LISTEN address (because the instrument is going to LISTEN to the controller) only. Example 3 requires both a primary address (9) and a secondary address (1). This is because the 7A16P is a plug-in to a larger product, such as a 7612D, and its connection to the GPIB is not from the back of the plug-in directly, but through the mainframe.

In the above examples, the commands the device-dependent messages are made up of string constants (e.g., "FREQ A", "V/D .2"). It doesn't always have to be like that. Suppose we want to generate a signal which steps through 100 Hz increments using the FG 5010. That would look something like this:

```
1ØØ      FOR I=1ØØØ TO 5ØØØ STEP 1ØØ
11Ø      PRINT @24:"FREQ ";I
         (program to measure response)
38Ø      NEXT I
```

The first time through, the message FREQ 1000 would be sent to the FG; the next time, it would be FREQ 1100 and so on.

To send data to an instrument (i.e., address it to be a LISTENER) we use the PRINT command. To make the instrument send data back to the computer, we use the INPUT command (i.e., make the instrument a TALKER). Normally, though, before we can ask the instrument to tell us something, we must ask it what to tell us. So most INPUT statements are preceded immediately by a PRINT statement. For example:

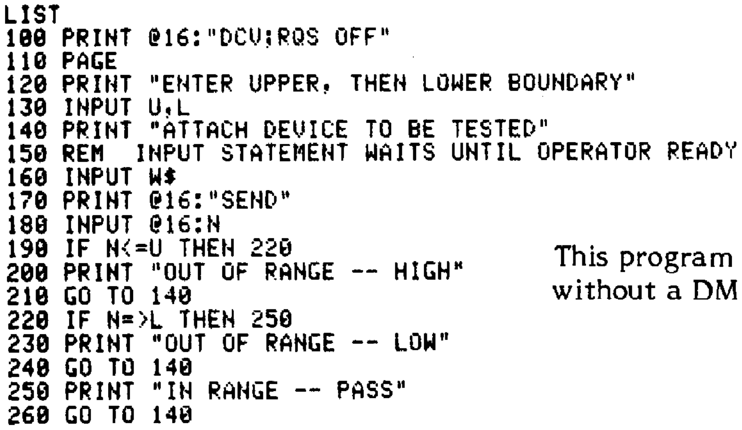
```
1)  1ØØ  PRINT @24:"FUNC?"  (FG5010)
    11Ø  INPUT @24:A$

2)  1ØØ  PRINT @18:"SEND"   (DC5010)
    11Ø  INPUT @18:N

3)  1ØØ  DIM A$(2ØØ)
    11Ø  PRINT @24:"SET?"   (FG5010)
    120  INPUT @24:A$
```

In the first example, we query the FG5010 what function it is currently in. Assuming it is generating sinewaves, A$ will be loaded with the string "FUNC SINE;".

The second example is indicative of several of the TM5000 products. When we send the "SEND" command, the instrument will respond by sending back its current reading, which will depend on what function it happens to be in. So if the DC5010 happens to be in TIME A-to-B mode, it will send back the time interval value. Note that it sends back a value only, and no header information; this is easier for the applications programmer to deal with since it can be read directly into a numeric variable and not a string variable.

The last example retrieves all the setting information from the FG 5010. Except for the length of the string variable, this sequence would look identical for any product conforming to the Tek Codes and Formats.

Let's look at one more example, which is a modification to the program you wrote for exercise 5. This program asks for upper and lower boundary limits, then takes a voltage reading from the DM 5010 to see if it is within that region.



```
LIST
100 PRINT @16:"DCV;RQS OFF"
110 PAGE
120 PRINT "ENTER UPPER, THEN LOWER BOUNDARY"
130 INPUT U,L
140 PRINT "ATTACH DEVICE TO BE TESTED"
150 REM  INPUT STATEMENT WAITS UNTIL OPERATOR READY
160 INPUT W$
170 PRINT @16:"SEND"
180 INPUT @16:N
190 IF N<=U THEN 220
200 PRINT "OUT OF RANGE -- HIGH"
210 GO TO 140
220 IF N=>L THEN 250
230 PRINT "OUT OF RANGE -- LOW"
240 GO TO 140
250 PRINT "IN RANGE -- PASS"
260 GO TO 140
```

This program will not run without a DM5010 attached.

In this program line 100 sets up the DM to monitor DC voltage; it also turns off all SRQ interrupts. (We did this because we haven't discussed polling yet.) We get the boundaries and allow the operator to attach the DUT. Notice that in line 160 we have an INPUT statement; this is not to enter information, but just to wait for the operator to complete attaching the DUT. When he is, he presses RETURN and the program goes on; we never use W$ again in the program.

Lines 170 and 180 get the measured voltage and the rest of the program determines whether it is within range as before.

RBYTE/WBYTE

Occasionally we need to communicate over the GPIB using binary data rather than ASCII. The major advantage to this is that fewer bytes have to be transmitted to convey an equal amount of information. (For example, to send the value 250 over the bus in ASCII takes four bytes: 2-5-0 and a delimiter, usually a comma or space. In binary, it takes one byte which in binary is 11111010.) This is particularly valuable for waveform products where large quantities of data need to be transmitted.

The following program reads a 256 element record from the 7612D. (To communicate with the 7612D, we always need to send a secondary address, even to the mainframe.)

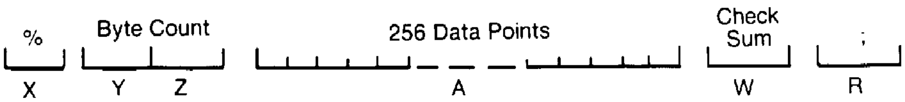
```
100 DIM A(256)
110 PRINT @6,0:"READ A,1"
120 WBYTE @70,96:
130 RBYTE X,Y,Z,A,W,R
```

(scaling routine)
(processing routine)

Although we have not discussed numeric arrays, line 100 creates enough memory space to contain an array (collection of ordered numbers) of 256 elements (each single value in an array is called an element). Line 110 instructs the 7612D to send the waveform record in channel A, record number 1, as soon as it is made a TALKER.

The WBYTE (Write Byte) command in line 120 addresses the 7612D to be a TALKER. The number 70 is created, based on the fact that the 7612D has a device address of 6 and, to create an "absolute TALK address," we add an offset of 64 for TALKERs (for LISTENERs, we add an offset of 32 to the device address to create the absolute LISTEN address). The PRINT and INPUT statements automatically add an offset of 32 and 64, respectively. WBYTE allows for much greater flexibility including setting up instrument to instrument communication without the controller having to act as intermediary. The number 96 is the absolute secondary address of the mainframe. The secondary address, in this case Ø, is added to an offset of 96 to create the absolute SECONDARY address.

Anyway, line 110 addresses the 7612D to be a TALKER. Line 120 causes the 4050 to begin handshaking the data in. The group of variables in this line is due to the Codes and Formats definition of a binary block. The data are loaded into the corresponding variables like this:



Needless to say, when we start using RBYTE and WBYTE, we are beginning to get into more elaborate programming. With our 7612D program we have only scratched the surface, because we also need to scale the data to adjust for voltage sensitivity, ground zero reference, and time-per-sample information.

Write byte (WBYTE) can also be used to send interface messages other than addressing. For example, to send the Device Clear message over the bus (see Table 1, Chapter III), we would type:

```
WBYTE @2Ø:
```

where the "@" symbol means to send the information on the GPIB, and 20 is the decimal equivalent code for DCL (Device Clear) as defined by the IEEE 488 standard. Appendix A provides an ASCII/GPIB chart that shows which numeric codes send particular GPIB interface messages. Take a moment now to look at this chart.

To review then, RBYTE and WBYTE are used to send and receive binary data across the GPIB and WBYTE is also used to send interface messages as defined by the IEEE 488 standard.

EXERCISE

9. Write a program which prompts the user to set up the front panel of an FG5010 (device address 24). Then have the program wait for the user by using an INPUT A$ statement. A$ will not be used later. The program should then ask for and retrieve the settings of the instrument. Finally, print them on the screen of the 4050. (Remember that the settings will require a string variable with enough space for 150 characters.)

10. What command would you use to send an LLO (Local Lockout) interface message to all instruments on the GPIB? (Use Appendix A.)

SERIAL POLL

Until now we have seen that the responsibility of the GPIB applications programmer has been to specify a sequence of activity and to provide for appropriate communication between various devices. There are, of course, many other details of programming a given task, but the last major function we will discuss is that of handling SRQ (Service Request) interrupts. The special characteristic of interrupts which make them somewhat different from our previous discussions is that they may occur asynchronously from any other system activity. For this reason, they need to be handled by special routines which are outside the primary flow of the main program.

There are two steps in dealing with interrupts. The first is acknowledging that an interrupt has occurred. The second is determining which device initiated the interrupt and why.

To enable a program to acknowledge an SRQ interrupt, ON SRQ is used. It looks like:

```
ON SRQ THEN line number
```

where the specified line number is the location where the service routine begins when an SRQ is asserted. This statement is usually one of the first lines of a program because, if an SRQ is being asserted and there is no ON SRQ statement, the 4050 will print an error message (NO SRQ IN IMMEDIATE LINE) and stop.

Remember that for any group of up to 14 products there is only one SRQ signal line coming into the back of the 4050. Therefore, when the 4050 receives an interrupt, the controller has the responsibility to identify which instrument caused the interrupt. This is done using the POLL command. Examples are:

```
POLL    Z,Y;14
POLL    M,N;6;3;9,Ø;L
```

The first example is the simplest possible POLL statement since the two target variables (Z,Y) are always required and are followed by the list of valid GPIB addresses. In the first example the list of addresses includes only one. This means that there is only one GPIB instrument connected to the 4050 capable of generating an SRQ. In the second example, devices with addresses of 6, 3, primary 9, secondary Ø, and the address defined by the value of L are connected. It is important that all valid GPIB addresses are listed in the POLL statement because, if they are not, the 4050 will stop upon receiving SRQ from a non-addressed device.

Here is a typical example of a program which handles SRQ interrupts:

```
100 ON SRQ THEN 500
110 REM      MAIN BODY OF PROGRAM
120 WBYTE @70,96:
130 RBYTE X,Y,Z,A,W,R
210 REM
220 REM
230 REM
350 END
500 REM      INTERRUPT SERVICE ROUTINE
510 POLL M,N;6;3;5
520 IF N=65 THEN 570
530 IF N=81 THEN 570
540 PRINT "ERROR OCCURRED AT ";M
550 PRINT "WHERE 1=ADDRESS 6; 2=ADDR. 3; 3=ADDR. 5"
560 PRINT "ERROR STATUS = ";N
570 RETURN
```

The POLL statement is executed in response to a service request from a peripheral device on the GPIB. Two numeric variables are specified as parameters in the POLL statement followed by a series of I/O addresses. The BASIC interpreter polls the first I/O address in the list, then the second I/O address, then the third, and so on, until the device requesting service is found. It is imperative that the I/O address of the device requesting service is in the list, or program execution will be halted.

After the peripheral device requesting service is found, the device's position in the list is assigned to the first variable specified in the POLL statement. The status word of the device is then sent over the GPIB and assigned to the second variable specified in the POLL statement.

In the program above, line 100 enables the program to acknowledge interrupts. Nothing actually happens at line 100, but it prepares the controller to respond to SRQ interrupts when they occur. Let's assume that an SRQ occurs when the program is executing line 210. When the computer completes executing 210 it jumps to line 500 (based on the ON SRQ statement) as though a GOSUB 500 had occurred (i.e., the controller remembers that it should return to line 220). The POLL statement executes a serial poll which returns a status byte from the instrument into variable N.

In this program, we only check for power-up conditions (status 65 and 81 -- see Table II, Chapter IV), in which case program control simply returns to the program following the point of the interrupt, line 220. If any other status byte is sent, this routine prints it and returns. Although this isn't very elaborate, it is frequently sufficient when debugging programs.

TAPE COMMANDS

There are a few commands to use the internal magnetic tape that you are likely to encounter or use.

FIND

Assuming that you have a program to store or retrieve on tape, the first thing you have to do is position the beginning of the desired file at the recording head. This is done with the positioning command.

```
FIND numeric expression
```

For example:

```
FIND    5
FIND    N+2
```

The numeric expression refers to the file number you are requesting. To position the read/write head at the beginning of the tape (rewind), type FIND 0. This instructs the tape drive to position the tape at the load point, which is the beginning of the tape. (You can also do this by pressing the REWIND key. The only difference between the two methods is that one is programmable, the other is not.)

File 0 does not actually exist. You cannot store anything in file 0; it is used only to indicate the beginning of the tape. However, once you have "found" file 0 (the beginning of a tape), you are then able to create a file that can store a program.

MARK

Files are created with the MARK statement:

```
MARK  numeric expression, numeric expression
```

The first numeric expression (parameter) refers to the quantity of files you want to set up, and the second indicates the length in bytes of the file or files being created.

If you want to create one file with a length of 1000 bytes, enter:

```
MARK 1,1000
```

Similarly, if you want to create 5 files, each with a length of 1200 bytes, enter:

```
MARK 5,1200
```

Typically, however, you only create files one at a time, as they are needed.

It is a good idea to set up files that are larger than necessary; this way room remains in the file to accommodate any later additions to the program being stored. To determine the approximate amount of storage that a given program requires, a good "rule of thumb" is to figure 40 bytes per line of code. (This is a rough approximation--some lines take more, and some lines take less.)

Now, suppose you have written a program and it is currently residing in the random access memory. You want to store it on a brand new tape. You have determined that the program requires, say, 1500 bytes of storage. Enter:

```
FIND 0
```

and the beginning of the tape is positioned at load point. Now, to create the file, type:

```
MARK 1,2000
```

and the Graphic System sets up one file with sufficient length to contain the program plus some additional space. (The 2000 was chosen arbitrarily.) This file now exists with a physical length of 2000. It currently has a logical length of 0 because, as yet, nothing has been stored there. Before the program can be stored in this newly created file, the tape must be positioned at the beginning of the file. To do this, just type:

```
FIND 1
```

and the tape drive positions the tape at the beginning of the file.

SAVE

The actual storing, or "saving," operation is initiatd by another directive.

SAVE (line number(,line number))

SAVE used by itself causes the entire program currently in memory to be stored on tape in ASCII code. (SAVE with one line number stores only the specified line number. SAVE with two line numbers stores only the part of the program bounded by and including the two specified line numbers.)

As soon as you type SAVE and press RETURN, the system records a copy of the program in memory on the tape. (It will record those statements which are preceded by a line number.) The file now has, in addition to its physical length, a logical length. In this case, the physical length is 3000 bytes as established by the MARK statement, and the logical length is 1500 bytes, because the program saved was 1500 bytes.

Before going any forther, let's review what has happened thus far:

- FIND 0 positioned the tape at the beginning load point.
- MARK 1,3000 created a file with a physical length of 3000 bytes.
- FIND 1 positioned the tape at the beginning of the file.
- SAVE caused the machine to make a copy of the program in memory on the tape in file 1.

OLD

Now let's suppose that, after performing some unrelated operations, you want to run an ASCII program previously saved on file 1. Retrieving the program is facilitated with the following statement:

```
OLD
```

To get the program back into memory, enter:

```
FIND 1
OLD
```

FIND 1, again, positions the tape at the beginning of file 1. OLD causes the Graphic System to first delete everything in memory (as though a DELETE ALL statement is executed), and then copy the logical contents of the file into memory.

When you want to save another program on tape, the process is essentially the same. Assuming that you have already utilized file 1 as in the previous discussion, the first thing to do is to type:

```
FIND 2
```

The machine is able to locate file 2 because the process of "marking" file 1 also marks the beginning of file 2. Now, suppose you want to establish one file with a length of 2000 bytes. Enter:

```
MARK 1,2000
```

and the file is created. (Do not confuse the file number with the number of files specified by the MARK statement.) You can now store a program in the second file by entering:

```
FIND 2
SAVE
```

and the system saves the program in memory on the tape, in file 2.

The TLIST Statement

At this point, it should be apparent that you need the ability to determine how many files exist on the tape, and the status of each file. This capability is provided by the additional directive:

TLIST

The TLIST statement lists out (on the screen) information about the tape that is in the tape drive assembly. This listing is typified by the following:

```
TLIST
1       ASCII PROGRAM    3840
2       ASCII PROGRAM    1792
3       ASCII PROGRAM    4096
4       ASCII PROGRAM    1280
5       ASCII PROGRAM    3840
6       ASCII PROGRAM    768
7       NEW              768
8       NEW              1024
9       LAST             768
```

The numbers in the left column indicate the number of the file. The center column identifies the type of file. In this case, the first six files contain programs. Files 7 and 8 are "NEW", indicating they are "empty" and available for use. The "LAST" file (file 9) is self-explanatory; it is the next one to be MARKed and used. The column of numbers on the right indicates the file size, in bytes, of each file, and is always a multiple of 256.

> Note: If you "re-mark" a file, then the file following the one that is "re-marked" always becomes the "LAST" file. Any files that may have been beyond the "re-marked" file are lost.

USER DEFINABLE KEYS

The 10 user definable keys on the upper left of the keyboard allow the user to branch to any of 20 specified BASIC program locations. Ten locations are available by pressing each of the 10 user definable keys; ten more are available by holding down the SHIFT key while pressing each user definable key.

The user definable keys provide a convenient means of interrupting the main program to perform a subroutine already placed in memory by the user. (The 4050 finishes executing the current BASIC line before performing the subroutine.)

To write a program that uses the user definable keys, begin with line number 100. One of the early program statements must be the SET KEY statement which allows the 4050 to respond to the user definable keys while the program is executing. Pressing one of the user definable keys is the same as executing a GOSUB statement. The main program is interrupted and program control is transferred to the line number that is four times the number of the user definable key pressed. These numbers are fixed and cannot be changed. Figure 6.1 shows the line number to which program control is transferred when each user definable key is pressed (in a program with SET KEY established).

The subroutine begins with the statement to which control is transferred and continues to execute statements in sequential order until an END, STOP, or RETURN statement is found or until the BREAK key is pressed. If none of these occur before line number 100 is reached, the system continues into the main program (which begins at line number 100). When a user defined function ends with a RETURN statement, program control is transferred back to the interrupt point in the main program.

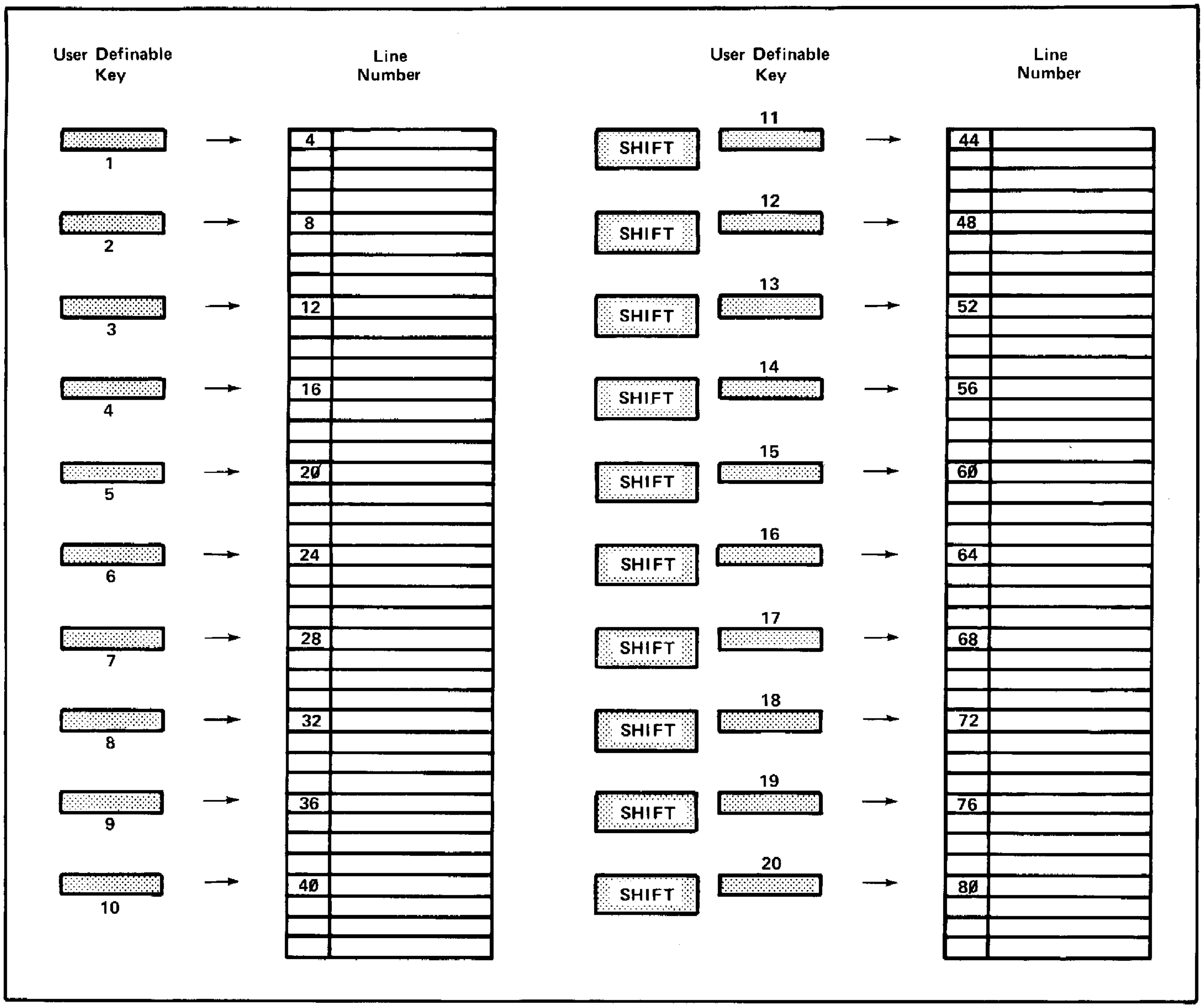


Fig. 6.1. The User Definable key program control transfer table.

Control can be passed from one subroutine line to another subroutine (for example, to a larger one) with a GOSUB statement. Figure 6.2 shows how this looks:

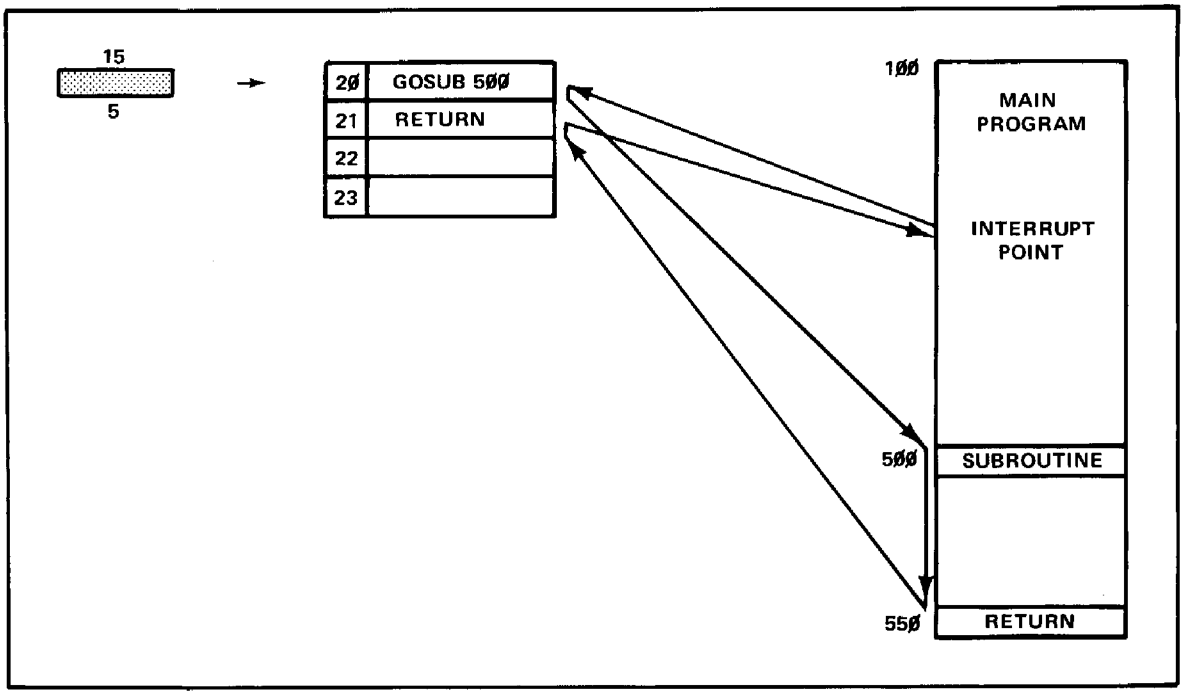


Fig. 6.2. Transferring control between subroutines.

User Definable Key 5 transfers program control to line number 20, a GOSUB 500 instruction. Line 20 then transfers control to line 500, the beginning of a larger subroutine. When the larger subroutine is finished executing, a RETURN statement transfers program control back to line 21, which is also a RETURN statement. Line 21 then transfers program control back to the interruption point in the main program. Here's a program example, type in and run it:

```
1 GO TO 100
4 GO TO 300
8 GO TO 435
100 INIT
110 PAGE
120 REM PROGRAM USING'USER DEFINABLE KEYS'(UDK's)
130 PRINT "UDK'S HELP PEOPLE 'OPERATE' THE 4052"
140 PRINT "-ALL AN OPERATOR NEEDS TO KNOW IS "
150 PRINT "-WHAT UDK TO DEPRESS AND THAT 'KEYS'"
160 PRINT "-THE PROGRAM TO DO A SPECIAL THING"
170 PRINT "  IF-THE-PROGRAMMER-HAS-PROVIDED"
180 PRINT "  FOR-THIS-TO-HAPPEN !!!!!"
190 PRINT
200 PRINT "LET'S LOOK AT AN EXAMPLE"
210 PRINT
220 PRINT "UDK#1 HAS BEEN PROGRAMMED TO JUMP TO"
230 PRINT " A SPECIAL MESSAGE.  TRY IT.PUSH #1"
240 END
300 PRINT "GGGGG"
310 PRINT
320 PRINT
330 PRINT "**** I JUST DEPRESSED UDK#1 !!! ****"
335 PRINT
340 PRINT "(IF YOU'RE WONDERING WHAT CAUSED THE"
350 PRINT " 'BELL' TO BEEP LIST THE PROGRAM AND"
360 PRINT " CHECK THE PRINT INSTRUCTION WHICH"
370 PRINT " HAS 'PRINT 'GGGGG' '"
380 PRINT "  ('BELL' IS MADE BY HOLDING DOWN THE "
390 PRINT "   'CTRL' KEY WHILE DEPRESSING 'G'"
400 PRINT "   --IT'S CALLED 'CONTROL G')"
410 PRINT "   ------ TRY IT IN IMMEDIATE MODE"
420 PRINT "---BUT FIRST REMEMBER THIS---"
425 PRINT "AFTER USING IMMEDIATE MODE PUSH UDK#2"
426 PRINT "--REMEMBER.  UDK#2"
427 PRINT
428 PRINT " (  PRINT'GGGGG'  )  NOTE: 'MEANS QUOTES"
429 END
430 PAGE
435 PRINT "BUT NOW BACK TO UDK'S"
440 PRINT
450 PRINT "THERE ARE 10 UDK 'KEYS' AND EACH HAS"
460 PRINT " A DUAL FUNCTION-NORMAL AND SHIFTED"
470 PRINT "-THIS MEANS 10 KEYS PERMIT 20 SELECTIONS."
480 PRINT "LIST PROGRAM LINES 1 THRU 100.('LIST 1,99')"
490 PRINT "-NOW NOTICE THAT LINE 1 GOTO 100."
500 PRINT "-LINE 4 GOTO 300"
510 PRINT "-LINE 8 GOTO 430"
520 PRINT
530 PRINT "EACH UDK IS PREDEFINED WITH A LINE NO."
540 PRINT " UDK#1 WITH LINE NO. 4,UDK#2 WITH LINE 8"
550 PRINT "  AND SO ON. WHAT LINE NO. WOULD UDK#5"
560 PRINT "  BE ASSOCIATED WITH?"
```

MISCELLANEOUS

There are just a couple more commands that would be useful to know.

The COPY command is used to generate an automatic hard copy under program control. Of course, you need to have either a 4631 or 4611 Hard Copy Unit attached for this to do anything.

And finally, the HOME command is used if you want to reposition the cursor to the upper-left screen position without erasing the page. PAGE is used to erase the screen and to return the cursor to the "home" position.

BIBLIOGRAPHY

Other books on BASIC:

Alcock, Donald. Illustrating BASIC
Cambridge University Press, Cambridge, 1977. An excellent short primer.

Coan, James. Basic BASIC
Hayden Book Comapny, Rochelle Park, 1978. Thorough but a little dry.

Albrecht, Robert, et.al. BASIC: A Self-Teaching Guide
John Wiley, New York, 1978. An excellent programmed instruction book.

Schoman, Kenneth. The BASIC Workbook
Hayden Book Company, Rochelle Park, 1977. This is a workbook with many good exercises; it has almost no text about BASIC.

PLOT 50 Programming in BASIC
Tektronix; Beaverton, 1978. A good comprehensive programming text for the 4050-Series BASIC.

VI. SELF-TEST (continued)

11. Using the following program, fill in the blanks:

```
100 PRINT "ENTER THREE NUMBERS"
110 INPUT A,B,C
120 Z=(A+B+C)/3
130 PRINT "RESULT IS ";Z
140 STOP
```

a. The numbers 100, 110, 120 are called _________ and they establish the _____ of execution.

b. The letters A, B, C, and Z are called ______.

c. In the third line from the top, (A+B+C)/3 is called an ______.

d. In one brief sentence, describe what the above program accomplishes.

12. List the five numeric operators and the appropriate keyboard symbol for each.

13. Write the 4050 statement(s) which would send a FREQ A command to a programmable counter with GPIB address 18.

14. Describe briefly what the following program does. Describe specifically the function of lines 100 and 500.

```
100 ON SRQ THEN 500
110 FOR I=1 TO 1000
120 NEXT I
130 GO TO 110
500 POLL M,N;1;2;3
510 PRINT "SRQ OCCURRED ON DEVICE";M;" STATUS ";N
520 RETURN
```

APPENDIX A

ASCII & IEEE 488 (GPIB) CODE CHART

B7 B6 B5 BITS				Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
B4	B3	B2	B1	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER	
Ø	Ø	Ø	Ø	0 NUL 0 (0)	20 DLE 10 (16)	40 SP 20 (32)	60 0 30 (48)	100 @ 40 (64)	120 P 50 (80)	140 \ 60 (96)	160 p 70 (112)
Ø	Ø	Ø	1	1 GTL SOH 1 (1)	21 LLO DC1 11 (17)	41 ! 21 (33)	61 1 31 (49)	101 A 41 (65)	121 Q 51 (81)	141 a 61 (97)	161 q 71 (113)
Ø	Ø	1	Ø	2 STX 2 (2)	22 DC2 12 (18)	42 " 22 (34)	62 2 32 (50)	102 B 42 (66)	122 R 52 (82)	142 b 62 (98)	162 r 72 (114)
Ø	Ø	1	1	3 ETX 3 (3)	23 DC3 13 (19)	43 # 23 (35)	63 3 33 (51)	103 C 43 (67)	123 S 53 (83)	143 c 63 (99)	163 s 73 (115)
Ø	1	Ø	Ø	4 SDC EOT 4 (4)	24 DCL DC4 14 (20)	44 $ 24 (36)	64 4 34 (52)	104 D 44 (68)	124 T 54 (84)	144 d 64 (100)	164 t 74 (116)
Ø	1	Ø	1	5 PPC ENQ 5 (5)	25 PPU NAK 15 (21)	45 % 25 (37)	65 5 35 (53)	105 E 45 (69)	125 U 55 (85)	145 e 65 (101)	165 u 75 (117)
Ø	1	1	Ø	6 ACK 6 (6)	26 SYN 16 (22)	46 & 26 (38)	66 6 36 (54)	106 F 46 (70)	126 V 56 (86)	146 f 66 (102)	166 v 76 (118)
Ø	1	1	1	7 BEL 7 (7)	27 ETB 17 (23)	47 / 27 (39)	67 7 37 (55)	107 G 47 (71)	127 W 57 (87)	147 g 67 (103)	167 w 77 (119)
1	Ø	Ø	Ø	10 GET BS 8 (8)	30 SPE CAN 18 (24)	50 (28 (40)	70 8 38 (56)	110 H 48 (72)	130 X 58 (88)	150 h 68 (104)	170 x 78 (120)
1	Ø	Ø	1	11 TCT HT 9 (9)	31 SPD EM 19 (25)	51) 29 (41)	71 9 39 (57)	111 I 49 (73)	131 Y 59 (89)	151 i 69 (105)	171 y 79 (121)
1	Ø	1	Ø	12 LF A (10)	32 SUB 1A (26)	52 * 2A (42)	72 : 3A (58)	112 J 4A (74)	132 Z 5A (90)	152 j 6A (106)	172 z 7A (122)
1	Ø	1	1	13 VT B (11)	33 ESC 1B (27)	53 + 2B (43)	73 ; 3B (59)	113 K 4B (75)	133 [5B (91)	153 k 6B (107)	173 { 7B (123)
1	1	Ø	Ø	14 FF C (12)	34 FS 1C (28)	54 , 2C (44)	74 < 3C (60)	114 L 4C (76)	134 \ 5C (92)	154 l 6C (108)	174 ¦ 7C (124)
1	1	Ø	1	15 CR D (13)	35 GS 1D (29)	55 − 2D (45)	75 = 3D (61)	115 M 4D (77)	135] 5D (93)	155 m 6D (109)	175 } 7D (125)
1	1	1	Ø	16 SO E (14)	36 RS 1E (30)	56 . 2E (46)	76 > 3E (62)	116 N 4E (78)	136 ^ 5E (94)	156 n 6E (110)	176 ~ 7E (126)
1	1	1	1	17 SI F (15)	37 US 1F (31)	57 / 2F (47)	77 UNL ? 3F (63)	117 O 4F (79)	137 UNT _ 5F (95)	157 o 6F (111)	177 RUBOUT (DEL) 7F (127)

ADDRESSED COMMANDS

UNIVERSAL COMMANDS

LISTEN ADDRESSES

TALK ADDRESSES

SECONDARY ADDRESSES OR COMMANDS

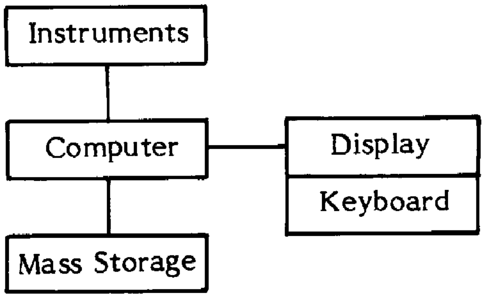
KEY TO CHART

octal	25	PPU	GPIB code
	NAK		ASCII character
hex	15	(21)	decimal

APPENDIX B

ANSWER KEY
CHAPTER II SELF TEST

1. instruments
controller
mass storage
keyboard
display

2. 

3. Instruments:
492P, 7612D, 7912AD, 468, 5223, 7854.
Mass Storage:
4924, 4907, tape cartridge in 4050 series
Keyboard and Display
4006, 4010, 4014, 4024, 4025, and keyboard/display of 4050.
Controllers
4051, 4052, 4054, and 4041.

4. multi-function interfaces or multiplexers

5. plotters, e.g., 4662 or 4663

6. memory size, speed of computation, speed of GPIB I/O, available peripherals, ease-of-programming, built-ins

7. hard disks, floppy disks, cassette tapes, and large mag tapes

8. Applications software
Implementation language - high level language
Device-Dependent codes

9. A compiler converts an entire program to machine code before execution. An interpreter translates and executes only one line of the application program at a time.

10. Need to know processor's detailed architecture, very time consuming to write, difficult to modify, and requires extensive documentation.

11. No. They are both codes which execute identically.

12. Because manuals provide the user the detailed command set and syntax with which to write his application.

13. a. Reduce labor costs by speeding up test operation therefore requiring less labor/piece tested. Also by reducing skill level because of computer control.

    b. Release engineers from drudgery because the computer is able to do repetitive tasks without an operator.

    c. Provide insight by coupling analysis with measurement by having the computer do mathematics computations before documenting results.

    d. Reduce human error and increase measurement accuracy and consistency because digital instruments are typically more accurate and more consistent than several operators making the same measurement.

ANSWER KEY

CHAPTER III SELF TEST

1. 8

2. the connector and the cable

3. (see page III-1)

4. The ATN line is asserted during interface messages; it is unasserted during device-dependent messages.

5. The rate of the slowest device involved in a transmission.

6. Asynchronous. The rate can change to accommodate both fast and slow devices.

7. At any time without regard to any of the other signals.

8. None. For others see page III-8.

9. 20 meters, active device at every 2 meters

10. A total of 15 including the controller.

11. Two-thirds of connected devices.

12. Proper handshaking and some data communication.

13. a. non-programmable
    b. programmable
    c. distributed-processing
    d. keystroke-programmed

    See III-15 thru III-18 for explanation.

14. Throughput and software enhanced calibration.

15. (See page III-22)

ANSWER KEY

CHAPTER IV SELF TEST

1. Establishes a common message structure, describes communications elements, defines control protocol, defines status bytes for error handling, standardizes features important to T & M systems.

2. device-dependent messages

3. FUNC? or
   PRINT @n:"FUNC?" or
   SET?

4. the semicolon (;)

5. SET?

6. ERR?

7. Numbers, Strings, Binary blocks, and End blocks.

8. with the EOI line

9. Writing systems software easier, self-documenting programs, system change and expansion easier, training time shorter, development time is faster, debug time is shorter.

ANSWER KEY
CHAPTER V SELF TEST

3. a. (4.25-1.78)/(6.23*9E2) = 4.405207776E-4
   b. 7.5/(3.2-8.8 ↑ 2) = -0.101023706897

5. Interrupts or aborts a running program.

6. RETURN

7. Deferred mode (program mode) has line numbers in front of each statement and must be executed using the RUN command. Immediate mode uses no line numbers and executes as soon as the RETURN key is pressed.

8. SQR ((73.6+13.2)/(14.6-8) ↑ 2))

9. Allows a lower skilled operator to interact with the 4050 without having to understand BASIC or programming. Allows a user to activate a function using one key rather than many.

10. Page-to-page tutorial
    Menu selectable features
    Specific Application Demo

11. UDK 10

12. The R07 and R08 Signal Processing ROM packs, or possibly a hard copy unit.

13. Hard copy units (4611,4631); plotters (4662, 4663); auxiliary mass storage devices (4907, 4924)

ANSWER KEY

CHAPTER VI SELF-TEST

1.

```
100 INIT
110 PRINT "INPUT PERIOD"
120 INPUT P
130 F=1/P
140 PRINT "FREQUENCY = ";F
```

2.

```
100 INIT
110 PRINT "ENTER RISETIME OF PROBE AND SCOPE"
120 INPUT R1,R2
130 R=SQR(R1↑2+R2↑2)
140 PRINT "SYSTEM RISETIME = ";R
```

3.

```
100 INIT
110 PRINT "ENTER LENGTHS A AND B"
120 SET DEGREES
130 P=ASN(A/B)
140 PRINT "PHASE SHIFT = ";P
```

4.

```
100 PRINT "ENTER UPPER, THEN LOWER BOUNDARIES"
110 INPUT U,L
120 PRINT "ENTER VALUE"
130 INPUT V
140 IF V<=U THEN 170
150 PRINT "VALUE TOO HIGH"
160 GO TO 120
170 IF V=>L THEN 200
180 PRINT "VALUE TOO LOW"
190 GO TO 120
200 PRINT "VALUE IN RANGE"
210 GO TO 120
```

5.

```
100 INIT
110 PRINT "HOW MANY NUMBERS DO YOU WANT TO AVERAGE"
120 INPUT N
130 Z=N
140 S=0
150 PRINT "ENTER EACH VALUE, ONE AT A TIME"
160 INPUT V
170 S=S+V
180 N=N-1
190 IF N>0 THEN 160
200 PRINT "AVERAGE IS ";S/Z
```

6.

```
100 FOR I=1 TO 15
110 PRINT I
120 NEXT I
```

7.

```
100 PRINT "ENTER NUMBER OF VALUES TO BE ENTERED"
110 INPUT N
120 S=0
130 S2=0
140 PRINT "ENTER VALUES, ONE AT A TIME"
150 FOR I=1 TO N
160 INPUT V
170 S=S+V
180 S2=S2+V↑2
190 NEXT I
200 PRINT "MEAN= ";S/N
210 V1=S2-S↑2/N
220 PRINT "VARIANCE = ";V1
```

8.

```
100 INIT
110 PRINT "INPUT PERIOD"
120 INPUT P
130 F=1/P
140 PRINT "FREQUENCY = ";F
150 PRINT
160 PRINT "DO YOU WANT TO RUN THIS AGAIN?"
170 INPUT A$
180 IF A$="YES" THEN 110
190 END
```

9.

```
100 PRINT "SET UP FRONT PANEL, THEN PRESS RETURN"
110 INPUT A$
120 PRINT @16:"SET?"
130 DIM B$(150)
140 INPUT @16:B$
150 PRINT B$
```

10. WBYTE @ 17:

11. a. line numbers, order or sequence

    b. variables

    c. expression

    d. averages 3 numbers

12. ↑ * / + -

13. PRINT @ 18:"FREQ A"

14. Waits for interrupts and prints the address and status when an interrupt occurs. Line 100 enables the 4050 to respond to interrupts. Line 500 determines which device caused the interrupt and gets the status byte.

APPENDIX C

GLOSSARY

address

A unique number (between 0 and 30 inclusive) identifying a specific instrument in a GPIB system.

AH

Accepter Handshake. The GPIB interface function providing a device with the capability to guarantee proper reception of remote multiline messages.

ALGOL

Algorithmic Language, or Algebraically Orientated Language. A high-level language of European origin designed principally for scientific calculations.

ANSI

American National Standards Institute. An organization chartered with the development of standards.

ASCII

American Standard Code for Information Interchange. A character set of 128 binary codes (seven bit positions) representing the numerals, upper and lower case alphabet, punctuation and other symbols and control functions, used in communications between computers and other digital devices. Also termed USASCII and ANSCII.

assembly language

A low-level, machine oriented system of coding in which programs may be written with mnemonics and labels instead of instruction codes and memory locations. There is usually a one-for-one relation between mnemonics and machine instructions.

ATN

Attention. One of five GPIB interface signal lines used by a controller to specify how data on the eight data lines are to be interpreted and which devices must respond to the data.

BASIC

Beginners' All-purpose Symbolic Instruction Code. A "conversational" high-level language developed at Dartmouth College for ease-of-use and ability to perform calculating machine tasks. Usually implemented on general purpose machines as an interpreter.

BCD

Binary Coded Decimal. The representation of decimal numbers by groups of four binary bits.

binary

The lowest level of data representation in digital logic consisting of two-state values, i.e., ones and zeroes.

bit

Binary digit. A unit of information content, the smallest unit in the binary system having logic states true and false.

BREAK

One of the keys on the 4050 alphanumeric keyboard allowing a user to interrupt, or stop the program being executed.

byte

A set of binary digits usually six to nine bits long, considered and operated on as a unit. It is most frequently applied to eight bit units as character representations.

codes

The form in which numbers, data, and messages must appear.

codes and formats standards

A Tektronix standard which augments the IEEE 488 (1978) standard by specifying in rigorous detail the codes and formats of device-dependent messages between instruments and/or the controller.

compiler

A program which accepts as input another program written in a high-level language (e.g., FORTRAN) and produces, as output, instructions in machine code for a particular computer.

controller

A device in a GPIB system which can address other devices to listen or to talk. In addition, this device can send interface messages to command specified actions within other devices. A device with only this capability neither sends nor receives device-dependent messages. (A computer is often used for this task.)

CR

Carriage Return. The ASCII code normally used to terminate input, usually followed by a line feed (LF). CR literally returns the cursor back to the left column.

DAV

Data Valid. One of three GPIB interface signal lines used to effect the transfer of each byte of data on the eight data lines from a talker or controller to one or more listeners. DAV is used to indicate the condition (availability and validity) of information on the eight data lines.

device dependent message
A message used by the devices interconnected via the interface system that are carried by, but not used or processed by, the interface system directly.

digitize
To convert analog data to binary bits (digital data) so it can be stored, processed, or transmitted by a digital device. Only digital data can be transmitted on the GPIB.

disk
A storage device in which binary information is stored on one or both sides of rotating disks by selective magnetization of portions of the surface.

distributed processing
The ability of computers and other intelligent devices to share processing tasks.

DUT
Device Under Test. In a GPIB system, the device on which the measurements are made.

EBCDIC
Extended Binary Coded Decimal Interchange Code. An eight-bit character code used mainly in IBM equipment and giving 256 possible characters.

EOI
End Or Identify. One of five GPIB interface signal lines used (by a talker) to indicate the end of a multiple byte transfer sequence or, in conjunction with ATN (by a controller), to execute a polling sequence.

firmware
Software programs stored in ROM which cannot be dynamically altered.

floppy disk
A moving head disk storage device whose magnetic surface is removable and flexible.

formats
The order, syntax, and control protocol used by device-dependent messages.

FORTRAN
Formula Translation. A high-level language of U.S. origin, designed mainly for scientific and engineering applications.

GPIB General Purpose Interface Bus. The IEEE Standard 488 (1978), IEEE Standard Digital Interface for Programmable Instrumentation, which deals with systems that use a byte-serial bit-parallel means to transfer digital data among a group of instruments and system components.

handshake The process by which an acknowledgment signal is sent from acceptor to source to indicate receipt of information.

high-level language A generalized method of writing a computer program which allows the programmer to express problems in a simple and convenient fashion in forms similar to mathematical expressions or natural (English) language.

HP-IB Hewlett-Packard Interface Bus. Hewlett-Packard's implementation of IEEE 488 (1978).

IEC International Electrotechnical Commission. An international standards organization.

IEC 625-1 The international counterpart to IEEE 488 (1978).

IEEE The Institute of Electrical and Electronics Engineers, Inc. U.S. sponsor of GPIB.

IFC Interface Clear. One of five GPIB interface signal lines used (by a controller) to place the interface system, portions of which are contained in all interconnected devices, in a known quiescent state.

interface messages GPIB messages used to manage the interface system itself.

interpreter A program which converts instructions written in a high-level language to machine code and causes the computer to check these for syntactical errors and/or execute the instructions immediately. BASIC is a language which is normally implemented as an interpreter.

I/O Input/Output. Data transferred between devices, or the process of performing this transfer.

keystroke programmable
The ability of a device to store a sequence of instructions, perform I/O, and choose between alternative actions based on input or processed information.

LEARN mode
The capability of using the front panel of an instrument for developing a set-up program in the controller. The SET? query provides this capability to Tektronix' customers.

LF
Line Feed. The ASCII character used by some devices to terminate a transmission.

listener
A device on the GPIB which can be addressed by an interface message to receive device-dependent messages from another device connected to the interface system.

machine code
Instructions expressed in binary form and capable of being processed by a computer.

mass storage
A device using media such as magnetic tape or disk for permanent storage of large quantities of data.

multiplexer
A switching network for analog or digital signals which selects one of several inputs for onward transmission from its single output.

NDAC
Not Data Accepted. One of three GPIB interface signal lines used to effect the transfer of each byte of data on the eight data lines from a talker or controller to one or more listeners. NDAC is used to indicate the condition of acceptance of data by device(s).

NRFD
Not Ready For Data. One of three GPIB interface signal lines used to effect the transfer of each byte of data on the eight data lines from a talker or controller to one or more listeners. NRFD is used to indicate the condition of readiness of device(s) to accept data.

operating system
A program which controls the execution of other programs, often providing scheduling, debugging, I/O control, accounting, storage management and related functions.

PAGE One of the keys on the 4050 which erases the screen and positions the cursor at the upper left corner of the screen.

parallel Simultaneous handling of all bits in a group of bits. GPIB transmits eight bits in a parallel manner.

PASCAL An ALGOL-based high-level language containing many data handling constructs.

peripheral Various devices such as hard copy units, plotter, magnetic tape drives, etc., used to input data, output data, and store data.

poll Invite other devices to transmit signals.

programmable instruments Those instruments whose functions or settings can be controlled via the GPIB.

query A command requesting information from an instrument.

RAM Random Access Memory. Usually applied to semiconductor memory into which programs and data can be written for temporary storage.

REN Remote Enable. One of five GPIB interface signal lines used (by a controller in conjunction with other messages) to select between two alternate sources of device programming data.

RETURN One of the keys on the 4050 which is used to terminate input. This key issues a CR.

ROM Read Only Memory. Any type of storage to which data cannot be written by the system in which it exists. Usually applied to non-volatile semi-conductor memory used for permanent program (firmware) storage.

serial Mode of operation in which items are treated sequentially. Opposite of parallel. GPIB transmits bytes serially.

SH Source Handshake. The GPIB interface function providing a device with the capability to guarantee the proper transfer of multiline messages.

software
A program, or sequence of instructions written in a programming language for execution on a computer.

software maintenance
Debugging and enhancing software programs after they have been "completed."

SRQ
Service Request. One of five GPIB interface signal lines used by a device to indicate the need for attention and to request an interruption of the current sequence of events.

talker
A device on the GPIB which can be addressed by an interface message to send device-dependent messages to another device connected to the interface system.

UDK
User Definable Key. Ten keys on the 4050 keyboard which allow the user to branch to any of twenty specific BASIC program locations.

waveform digitizer
An instrument which converts an analog waveform to its digital representation. This is required to transmit any waveform over the GPIB.