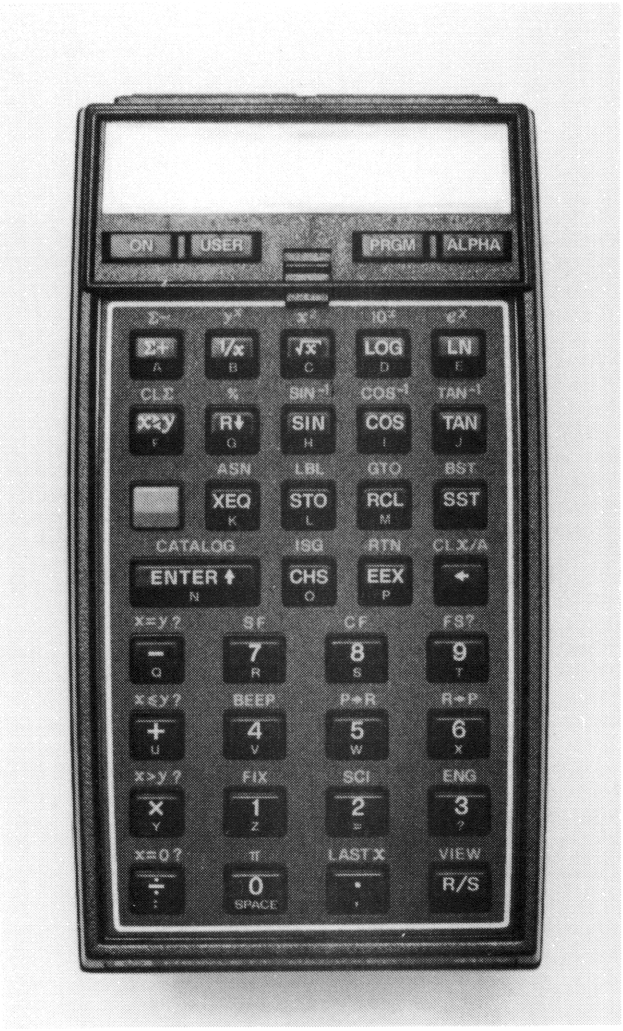
PROGRAMMING THE HP-41C/CV/CX

A Step-by-Step Introduction to Using a Personal Portable Computer

- Second Edition -



Thomas Adams

Information about this book can be obtained from:

Thomas Adams, Ph.D.
Department of Physiology
Michigan State University
East Lansing, Michigan
48824

Table of Contents

List of Figures

Preface

Not everyone successfully rides the waves of today's computer technology. For many people it is complicated to the point of being intimidating, and the beginner often finds it hard even to find readable and reliable manuals. Many are swept away, overwhelmed by the complexity, jargon and detail. More out of desperation than insight, too many invest thousands of dollars in computers and software which not only do they not know how to use but also which are inappropriate for their application.

But there is an alternative. Many models of handheld programmable computers cost less than $200, are battery-operated, portable, inexpensively expandable and actually out-perform most tabletop microcomputers for making complex numerical calculations. They offer unique low-cost opportunities to learn how to use a computer, how to program one, and allow exploring its many applications in areas of personal and professional interest. Once a few basic skills are acquired, these computers can provide many years of valuable and reliable service and can be effective springboards to learning alternative and more complicated computer technologies.

This book gives step-by-step instructions for operating and programming the least expensive and most powerful of the handheld computers, those in the Hewlett-Packard HP-41 series. The book assumes that the reader has little or no prior experience with computers. It requires no more than basic arithmetic skills in order for the reader to work the sample problems and to become proficient in writing one's own computer programs.

There will be at least two pay offs in progressing through this book methodically and with care. First, one can learn generally how all computers work. The inexpensive computers in the HP-41 series input, process and output data generically the same and with analogous operations as does any other computer. Second, the book provides opportunities to achieve a permanent and sure cure for math phobias. One will learn not only how to solve quickly and easily even complex equations of any type but also discover the satisfaction and pleasure of successfully meeting the challenge of computer use. Nothing else will help whittle down to manageable size even complex mathematical operations like the reader's skillful use of one of the HP-41 series computers. They offer unique opportunities for teaching and learning mathematical and statistical procedures.

My hope in writing this book is that the reader will benefit from becoming proficient with the HP-41 series computers, learn to use profitably their many features, and enjoy acquiring these skills as much as my many students and I have over the past years.

Foreword

"Why bother taking the trouble to learn how a small computer works, especially one which looks like a calculator and costs less than a couple of hundred dollars?"

Despite their unpretentious appearance and low cost, the HP-41 computers are surprisingly powerful for making sophisticated and complex calculations. Not only do they have many built-in functions, but also they use hundreds of readily available programs in dozens of technical fields such as medicine, engineering, statistics, mathematics and many other areas, including home-management and games. Perhaps most important, though, you can program these computers yourself to solve problems of your own personal and professional interest. These programs can be either stored in the computer indefinitely, or recorded on inexpensive magnetic cards. Although this book will introduce you to some of these programs for solving common everyday math problems, its emphasis will be on helping you learn basic skills for writing programs of your own. It's a good bet that soon you will be writing complex programs once you have acquired a few basic skills for writing some of the simpler ones, and once you see how easy it is to write programs for your application.

* * * * * * * * * *

"But, why get a book to learn just one type of computer?"

Considering cost, computational power, ability to be expanded and to interface with printers, plotters, magnetic card readers, cassette data storage devices and many other accessories, there are no other computers anything like the HP-41C/CV/CX. They virtually stand alone as a portable, programmable alphanumeric computer. There is no other machine that even comes close to doing what they can do for the price and their reputation for outstanding reliability. Nonetheless, the program design and writing skills you will learn here are readily transferable for use with other computers even though they may use a different programming language. Computers, after all, function basically in very similar ways when it comes to the logic of how they input, process and output information.

* * * * * * * * * *

"Well, if these computers are all that good, how come everybody doesn't have one?"

Maybe part of the answer is that for all their many attractions and computational power, many people find the HP-41C/CV/CX computers a challenge to learn how to use. In many ways they are much less complicated than other more powerful machines, but admittedly, they are compact. For example, many computations they make are not identified by just looking at the keyboard. They have to be "called up" from the computer's memory in a special way, as you will learn how to do to run ready-made programs, as well as those of your own design. This book is designed to help you attain proficiency in this and many other related skills. It's possible that more

people don't use these computers because they haven't had a good introduction to using them.

* * * * * * * * * *

"OK, I'll buy that, but why should I get a special book to learn how to use this computer? Why don't I just use the instruction book which comes with the HP41C/CV/CX?"

You certainly could, and many people do just that with a great degree of success. In addition, there are many other excellent self-instructional books besides the manuals that come with the computer when you buy it. Many of them are cited here. Just starting out, though, you may have to go to several different sources of information to get specific questions answered, and that takes considerable time and sometimes trial-and-error searching for the "right" source. This book is designed to bring together enough basic information about the HP-41 series of computers so you can start using and programming them and then be prepared to use more specialized references.

* * * * * * * * * *

"With so many manuals and books available about computers, what's so special about this one?"

A common problem for people just starting to explore the HP41C/CV/CX is that many of the best instructional books for this excellent machine are written for a reader with a higher level of computer sophistication than many beginners have. It's frustrating, intimidating and discouraging not to be able to take the next step quickly and easily in one's own learning process, particularly in the beginning when one does not yet have even the basics. This is a special problem in learning to operate such a complex tool as a computer. What is special about this book is that it is designed to anticipate the most commonly encountered problems for the beginner. If you already know, for example, how to program indirect addressing statements, how to interface your machine with digital multimeters for on-line data collection, and how to write programs of a publishable quality for the HP-41C/CV/CX, this book may not be for you. If, on the other hand, you are striving to get on the first step of the ladder in learning how to use your HP-41C/CV/CX, then it most likely will help you establish basic concepts, develop key entry skills and learn fundamental program writing strategies, as well as provide you with an information base so you can then go to appropriate advanced instruction manuals and texts. The main purpose of this manual is to get you started, not to teach you everything.

* * * * * * * * * *

"Yeah, all right, but so what? What do I get out of it after I've learned how to use this type of computer?"

You may be the best person to answer that question once you've gained some computer skills and start to see how they meet your particular computational needs. What many people find is that once they have even entry level proficiency with the HP-

41C/CV/CX computers, they no longer have to be concerned about solving rapidly and accurately any equation, no matter how complex. That is a big step for those who are a little uncomfortable with complex mathematical operations often encountered in statistics and other analytical procedures. The skills you will learn in becoming proficient with the HP-41C/CV/CX go a long way in dispelling math phobias, and might even help you overcome apprehensions for learning about other computers. If you are at all like most other people, you'll soon find that once a degree of comfort is attained in using a computer, it becomes fascinating and a constant challenge to increase your proficiency with it.

* * * * * * * * * *

"Well, maybe. Anyway, how do I get started?"

Keep reading.

Introduction

This book will familiarize you in a step-by-step way with the fundamentals of computer operations and use. It is written for the beginner and focuses specifically on the use of one type of handheld programmable computer, the HP-41 series manufactured by Hewlett-Packard Company. There are 3 models in this series, the HP-41C, the HP-41CV and the HP-41CX. Each is compatible with one another in programming and in the selection and use of most peripherals, but there are important differences among them. Unless otherwise specified, all instructions will apply equally well to any of the HP-41 models.

There are several reasons for selecting the HP-41 series of computers as an example in this manual. The two most important are that they are excellent computers themselves and are superb devices with which to learn general computer skills for using commercially available programs and for writing programs of one's own. Even though this book relates exclusively to one type of computer, it contains information applicable to the use and programming of any other, regardless of its design and the program language it uses. The examples and exercises are constructed, however, to aid your becoming proficient in the use and programming of a specific inexpensive battery operated computer which is small yet powerful enough to function as a useful portable record keeper and as a valuable device for solving equations.

Even people who have access to tabletop and mainframe computers find the HP-41 invaluable as a portable computer which supplements the functions of the more expensive and cumbersome machines. The value of the HP-41 series of computers is not that they take the place of other computers, but that they provide unique features these other instruments do not. Knowing how to use and program one of the HP-41 series of computers importantly supplements a tabletop microcomputer.

There is an important secondary gain to progressing step-by-step to learn how to use the HP-41. Besides becoming more proficient in using a computer, you will gain additional confidence in dealing with mathematical procedures. Considering how inhibiting, uncomfortable and self-limiting inadequate mathematical skills are, overcoming them is an important step ahead for many people. This book is intended to help make that step and present basic skills for computer use.

A few suggestions: You will get the most out of this book if you try each step on your own computer as it is introduced. Key in each exercise and program, to understand the topic being presented. Then modify the examples to make them run more easily for you and to make them suit better your own specific needs and interests. Half the fun in learning how to use your computer will be in exercising creativity in modifying existing programs and in writing those of your own design so they operate exactly in the way you want. Most of the programs presented here illustrate programming principles, not necessarily the most efficient program construction. You will learn as much drawing from your own ability to tailor these examples to your own needs, as you will in running them just as they are listed to explain a particular principle.

A last suggestion: Presented Information will be learned more easily, thoroughly and enjoyably if it is taken in measured amounts. Learning how to use the HP-41 computer, or any others, for that matter, cannot be done in just a few sessions, no matter how heroically long each is. Try to set aside half an hour, for example, each day to cover only a few pages or a single section in the manual, to try its examples, and to experiment with related programs of your own. You will retain the computer skills you'll learn much better and longer this way, and you'll enjoy the whole process considerably more.

Chapter 1

Step One

A review is presented first for what you can and cannot reasonably expect from the HP-41 computer. The purpose is to give a broad view of the many possible uses of the HP-41C, HP-41CV and HP-41CX models in order to show what functions and features they have in common, and to describe how each is unique. All of the features of the HP-41 series reviewed in this section are either built-in to each model, or can be obtained by adding-on components. It is unlikely that the general user will ever want to take advantage of all of the potential applications which will be described, but there is value, nonetheless, in knowing what the options are for developing a customized computer system of your own in which the HP-41 series computer is a central element.

Section 1.1. What To Expect

Even the first glance at the HP-41 keyboard (Figure 1) reveals many readily recognized functions of a basic calculator, and, of course, these computers will work easily and well in that way for you. But chances are, you bought it to do much more. An outstanding feature of this computer is that it is virtually unparalleled in its class for being able to make numerical solutions of even the most involved equations, using when necessary strings of built-in decision making steps and series of complex mathematical operations. Whether you are interested in using this machine for basic arithmetic and mathematical computations, for working through statistical procedures, financial calculations, business computations or for solving equations in engineering, biology, medicine or other professional fields, you couldn't select for the money a more powerful, flexible and easily expanded instrument.

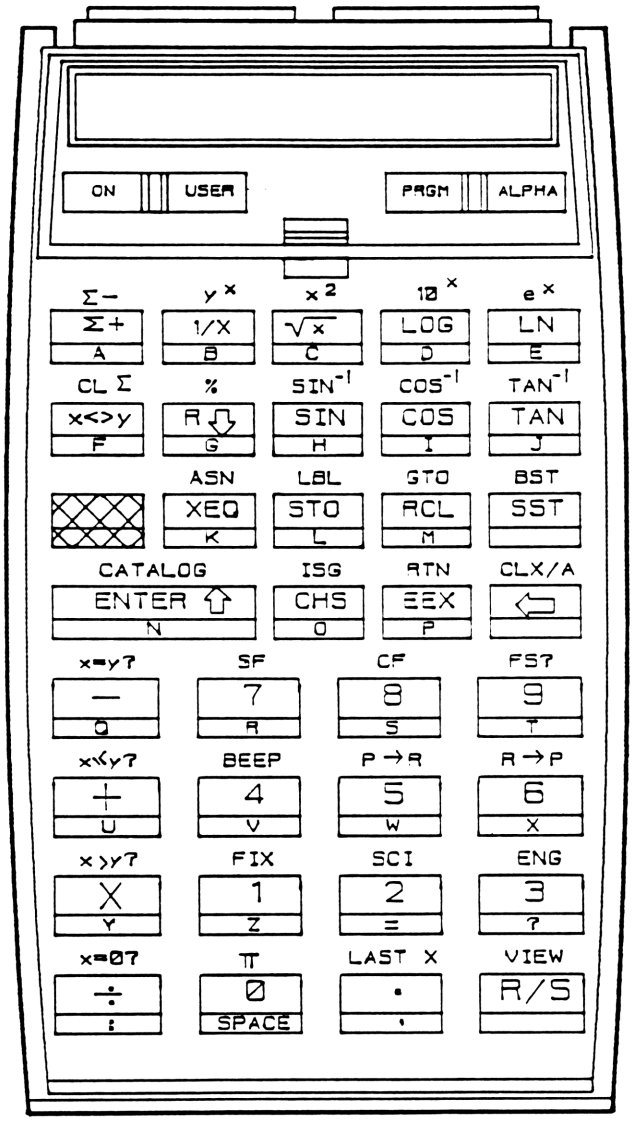
There are 4 ways for you to use the HP-41 computer to solve equations in whatever your area of interest. The simplest and most direct way is for you to enter programs of your own design into the machine yourself. A major purpose of this manual is to help you learn how to design such programs and use them with ease and accuracy. Despite what you may have heard about computer programs, you will soon see that getting started writing your own is quite easy and interesting.

Another way for you to use programs in your HP-41 computer is to obtain those written by others. These programs will come from several sources, but no doubt you will find a gold mine of references in Hewlett-Packard's User's Library, no matter what your interests are. For years, HP-41 owners have submitted deftfully organized and imaginatively constructed programs to be reviewed by Hewlett-Packard personnel for publication in their User's Library. A frequently revised catalog lists descriptions of the ever-growing number of available programs.

A third resource to solve equations is in the use of "application modules". These are small, reasonably inexpensive plug-in additions to your machine which have been constructed to solve the most frequently used equations in engineering, statistics and mathematics. The fourth way to help you solve problems with your HP-41 is by using "application manuals" in which are listed not only appropriate programs for commonly used equations in different areas, but also sample solutions and instructions for program use. The main point to be made at this stage is that you have many different reference sources to solve problems using your new machine. Although it is intriguing to learn how to write programs of your own, you are not limited just to that.

Figure 1

The HP-41 C/CV/CX Keyboard



Not only do you have a computer which will allow you to write and use programs to solve mathematical problems, the HP-41 is designed to retain whatever programs you enter into it after you have turned it off. The "C" in the model designation reminds you that your machine has a continuous memory. Most likely you'll be as intrigued as are others to learn that the HP-41C/CV/CX will remember programs (at least for a while) even though you have disconnected it from its internal power source by removing its batteries - a handy and time saving feature when batteries have to be replaced.

As you progress through this book, it will be only a short time before you have written enough programs of your own, or learned how to use those designed by others, to exceed the memory available even in the HP-41CX, the model with the greatest storage capacity. You will be introduced to the technique by which you can use magnetic cards as an inexpensive and permanent way to store programs and data to free memory space in your machine. These cards function in miniature like the larger floppy discs you may have seen used with tabletop computers. You can store programs and data on small cassettes or encode them into bar codes as a permanent record.

Your HP-41 can do a lot more for you, though, than provide basic computer functions by running specially constructed programs either of your own or someone else's design. Attachments are available for your computer to allow you to use it "on-line" in data collection, not only to "read" information from temperature, pressure and other types of sensors, but also for it to send signals to control switches, solenoids, recorders and other devices. These are rather complex uses of the HP-41 computer and most likely are beyond not only the immediate needs, but also the interests of most users. Even though many people will never use them in this way, it's interesting, nonetheless, to realize these are potential ways for you to expand your computer. If nothing else, such recognition adds to the the growing respect you will undoubtedly build for your new machine.

Although you may never need to expand your computer's interactive scope to include on-line data acquisition and storage devices, using one of the several printers designed for the HP-41 series will increase its usefulness. Whether it's a printer which works with only the computer itself (HP-82143A), one which functions in an interface loop with other peripherals (HP-82162A), or the HP-ThinkJet printer, or some other printer, being able to get copies of program lists and program solutions saves a lot of time and effort. A printer is not necessary, though, in just getting started to learn your computer.

The HP-41's alphanumeric ability adds greatly to its ease of use because it will allow you to use letters, numbers and a wide range of other symbols in any combination to display requests for data input and to read data output. Messages and data are directly read and understood thanks to the alphanumeric feature of this machine which allows it to operate for data entry and retrieval much like larger and more expensive tabletop microcomputers. You will be free of having to remember number codes to index what is required next in a calculation, or what has just been calculated, as you have to in using other similar instruments. The more you use the extremely flexible and attractive alphanumeric capability of the HP-41C/CV/CX and learn to draw from its extensive library of symbols, the more you will appreciate this feature.

Although the HP-41 computer has only a small memory (about 2.2K byte) compared to larger machines, most users soon learn that with judiciously and economically designed programs, this is more than enough space for even quite complex series of calculations and displays. Whatever limitations are presented by their small memory, the HP-41 models compensate greatly by having a minimum of 128 built-in frequently used functions which are available with no cost in memory space. Many of these will be reviewed in detail.

For all of the glowing praises that the HP-41 machines rightly deserve, and for all of their exciting and useful features you can look forward to learning and

using, there are some things you cannot expect your new computer to do. One is that even though these machines have an alphanumeric ability, they are not designed to serve as word processors. They outdo themselves in being able to make long series of intricate and complex numerical calculations, but a portable typewriter they're not. Nor can they be expected to perform computations and program execution with the speed that larger microcomputers can. For many practical applications, most people find the extra few seconds it may take one of the HP-41 models to make a calculation really doesn't make any difference at all. In all fairness, though, speed is not one of the better features of these otherwise outstanding instruments.

The new owner of an HP-41 series computer needs to be prepared to face some difficulties in learning how to use this instrument, no matter what model was purchased. These are compact, efficient, complex, powerful, expertly designed and constructed devices which demand patience and sometimes forbearance in learning how to use them. Few people have the technical background to gain immediate proficiency with this instrument. Unfortunately, many instruction manuals for these computers are difficult for some people to use because of the high level of mathematical, computer, engineering or technical skills presumed by their authors.

This book is written with that problem firmly in mind. It assumes the reader has little if any understanding of how computers work, but probably doesn't need to gain any more than a few superficial insights in order to learn how to use one of the HP-41 series computer to its full advantage in the context of one's own application. For every reader who is offended by the step-by-step, sometimes overly simplified suggestions and examples in this book, the bet is there are a dozen others who are delighted to have such detail and depend heavily on it. Those are the ones for whom this manual is intended.

The general user might have to face some hard work and maybe even frustration in the first stages of learning how to use the HP-41, but there are very few who will not in retrospect find the effort to be more than worthwhile. The gratification one finds in being able to deal easily and competently with complex mathematical operations, the ingenuity one soon discovers to construct computer programs of one's own, and the ever present sense of discovery as one's proficiency with the instrument grows all soon overshadow and more than compensate for any initial discomfort in getting the learning process started.

It will be a bonus to realize that even when you have mastered the HP-41 enough to write whatever programs you need personally and professionally, most likely there are many new and unexplored features of the machine yet to be learned. This instrument provides many of the practical services of a portable programmable computer, but also it is a constant challenge and opportunity for growth in learning to use computing machinery.

Familiarity with the following nomenclature will help as you progress through this book and others, and as you expand your interests and skills to include other computers:

FOCAL (or FOCOL): Forty One computer (or calculator) language.

RPN: Reverse Polish Notation: Mathematical statements as used in the HP-41.

HHC: Handheld computer

LCD: Liquid crystal display. The 14 elements of LCD in the HP-41 computers allows for the formation of 16,384 different characters.

LED: Light-emitting diode. The technique by which displays were created in many earlier handheld computers, for example the HP-65 and HP-67 computers, and in others too.

ROM: Read only memory. The kind of memory which contains the many built-in functions of the HP-41 computers, as well as that contained in the application modules which can be connected to them.

RAM: Random access memory: The kind of memory contained in the HP-41 computers' **MAIN MEMORY** and that contained in the **MEMORY MODULES** which can be connected to them.

Section 1.2. Which Model to Choose?

If you have not yet bought a HP-41 series computer, you are undoubtedly interested in knowing which one will best meet your needs. If you already have one of them, you may be interested in knowing how your model compares to the others. The first part of this section will describe similarities and differences among the HP-41C, HP-41CV and HP-41CX models, and show how features of the HP-41CV can be added to the HP-41C, as well as describe how features of the HP-41CX can be added to the HP-41CV.

The physical similarities among the 3 models of computers in the HP-41 series are striking and provide few clues to their different built-in functions. Were you to have each of the HP-41 model computers in front of you, you'd be hard put to see any differences among them unless you knew exactly what to look for. They are the same size, and the keyboards are configured identically (see Figure 1). Without turning each of them on and performing different keystroke functions, the only way to distinguish among the 3 models is to note the model number unobtrusively printed at the lower right front of the machine.

Even though you will soon see the many ways in which each model is different, it's important also to know how they are similar and to recognize their common functions. For example, it's reassuring to know that all HP-41 models use the same plug-in peripherals, such as the card reader, different models of printers, the optical wand and other devices, and that each has the same functions addressable from the keyboard. Also, each uses the same type of magnetic cards for data and program storage, and each is programmed identically using the same computer language.

These similarities are very important if you decide to move up to the more sophisticated and expensive HP-41CX after getting started with either the HP-41C or HP-41CV models. Programs developed for one's first computer and peripherals bought for it can be used directly in the newer and more complex model without modification of either the programs or the computer itself. Similarities among the HP-41C series computers will also be important for users who own different models but want to share programs and peripherals.

Despite the utility of each model in the HP-41C series being so similar, there are important ways in which they are different, and discriminating users surely need to know exactly what features are included in the model they choose. The HP-41C is the least complicated and least expensive of the 3 models. It comes standard with 63 built-in storage registers to give from about 200 to 400 program lines which can be used either to store data or programs. The major difference between the next more complex model, the HP-41CV, and the HP-41C is the size of its built-in memory. The HP-41CV comes with 319 data storage registers (about 2,000 program statements) to provide considerably more data storage space and room to store programs than the HP-41C.

The HP-41C model is no longer manufactured by Hewlett-Packard. Many of them were made earlier, though. It was a popular and powerful HHC when it was first introduced, and there are many of them available from previous owners. If you don't see one advertised for sale, try running a small ad of your own indicating your interest in

buying one. If you buy the HP-41C model, become proficient with it but find it has too few storage registers, you can expand its memory by using individual "memory modules" (HP-82106A). Each contains an additional 64 registers, and each is easily and conveniently inserted for use into 1 of the 4 accessory ports on the top of the machine. The HP-41C can have its useable memory expanded by having as many as 4 memory modules added to the original model. The cost is not only the price of the modules themselves, but also accessory ports become filled. Using 4 memory modules in the HP-41C gives all of the data and program storage space of the HP-41CV, but with its ports filled, one cannot use a printer, a card reader or any other peripheral device, including application modules.

An alternative for memory expansion of the HP-41C is to use not 4 individual memory modules but 1 "quad memory" module (HP-82170A) which contains 256 registers. The obvious advantage of choosing the quad memory module is that one reaches up to the memory capability of the HP-41CV but at the cost of using only 1 of the accessory ports. Printers, plotters and other devices can now be used by connecting them to one or more of the other available ports. The option for using memory modules is valuable for the HP-41C owner who needs more program and data storage space, but the simplest and least expensive way to get the memory capacity of the HP-41CV is, of course, to buy that model in the first place. This is not only less expensive than buying the HP-41C and adding memory modules, it leaves all of the accessory ports free for peripherals to be plugged-in. It also provides an opportunity to add components to obtain functions unique to the HP-41CX.

The forward thinking reader may have already considered that if the storage registers of the HP-41C can be expanded by using memory modules or a quad memory module, why not buy an HP-41CV, have access to its built-in 319 registers but expand them by adding either memory modules or a quad memory? What a good way to get a handheld programmable computer with a large memory capacity with little extra cost! It doesn't work that way. Although the memory of the HP-41C can be expanded to match that of the HP-41CV, the 319 built-in registers of the HP-41CV are all you're going to get. They cannot be expanded by using memory modules. Some important features can be added to the HP-41CV by using add-on modules of different types and inserting them into the computer's accessory ports, but its main memory is as big as it's going to be when you take the machine out its box for the first time.

If the HP-41CV has all of the memory of any of the HP-41 series, what does one get for the extra expense of buying an HP-41CX, the most sophisticated, expensive and complex of the models in this series? The HP-41CX has the same memory space of 319 storage registers as does the HP-41CV (or the appropriately expanded HP-41C), but it has built-in 124 registers of "extended memory". How "extended memory" differs from data and program storage memory will be described in detail later. For now, though, the point to remember is that the main memory space of the HP-41CV and HP-41CX are the same (319 registers). This is especially important because it is within the main memory space where programs will be written and stored for immediate access from the keyboard, and it's the memory space where both numeric data (numbers) and alpha data (letters, numbers and non-numerical symbols) will be stored for use within a program. The HP-41CX offers other advantages, though.

The HP-41CX has quite useful built-in timer, calendar, alarm and stopwatch functions which are not immediately available in the HP-41CV. In a similar way, though, that the extra memory of the HP-41CV can be obtained by using plug-in memory modules in the HP-41C, the timer and stopwatch functions that come with the HP-41CX can be obtained by plug-in modules for the HP-41CV. Also similarly, they are added to the basic machine by using the accessory ports to provide some of the same advantages and disadvantages as when one chooses to expand the HP-41C. For reasons of expense and diminished availability of accessory ports, it's convenient to get important features of the HP-41CX, including its "extended memory" by adding-on modules to the basic HP-41CV, but it's not the best choice if one has not yet bought

any one of the models in the HP-41 series.

How does one know which model to choose? The HP-41C is an excellent first computer for the beginner, but its limited memory presents a problem as one's library of programs grows. The HP-41CV provides the same directly addessable data and program storage memory of the HP-41CX but does not come with its extended memory and timer functions. For most users, the HP-41CV functions quite well in all computational applications, with its larger built-in memory giving a considerable advantage over the HP-41C. The HP-41CX's extended memory and built-in timer functions strengthen substantially the power of this computer, and is the choice for the experienced user.

If you are concerned about committing too much money to a computer, perhaps because you are not sure you are going to be interested and successfully challenged in learning about such things, the HP-41C is the safest of the choices. Comparatively little money will be invested, and it can always be sold at a later date if you discover your interests are elsewhere. If your curiosity is tickled by what you can do with the HP-41C, but your imagination in programming has outgrown the capacity of its built-in 63 registers, they can be expanded easily and inexpensively, or you can trade-up to the HP-41CV or HP-41CX. All-in-all, the venture is relatively free of financial risk if the many available options are explored.

On the other hand, if you already know that you have applications for a programmable handheld computer, feel at least minimally comfortable in wanting to learn programming, and are intrigued with the adventure of it all, the HP-41CX might very well be the machine for you. As a general rule, there is considerable sense in buying a computer that allows you comfortable growth room, not only for use, but also for learning. I imagine there are many more people who wish they had bought a more powerful computer once they have a year or so experience with whatever was their choice, than there are those who are sorry they bought more than they can handle.

Whatever one's choice is in investing in an HP-41 series computer, it shouldn't be overlooked that one is buying a powerful computational device which with experience in learning how to use it, rivals many features of the tabletop computers at a small fraction of their cost. With selected accessories, one can built an HP-41 series system which can perform virtually any mathematical procedure, as well as use hundreds of computer programs designed by others in many, many financial and professional fields.

Chapter 2

Features and Functions

This section describes the major physical features of the HP-41 computers, introduces how the instrument's toggle and function keys are used, and shows how to control the way numbers are displayed. After you understand the basics of these control operations and have attained some proficiency in using them, you will be ready to put them to work to perform complex calculations and to write programs. This is a similar stage to that of the beginning driver who first learns where the accelerator, brake, light switches and other control devices are and what they do before putting them into action.

Partly because of its small size and tightly compacted designations of keyboard functions, it's hard not to be a little intimidated by any one of the HP-41 models when you first take it out of the box. Many people admit being initially concerned that they might damage the machine by innocently pressing the wrong key in an attempt to perform some half-understood function. Be assured there is no way any of the HP-41 models can be damaged in normal use by activating any key in any order at any time. You might not perform a desired calculation correctly if you mistakedly use the wrong keystrokes, but the machine will not be injured.

Short of gross abuse, the only way you might damage your HP-41 computer is by attaching or removing accessories while it is turned on. **BE SURE** your computer is turned off before inserting or removing any peripheral devices. Otherwise, spurious electrical signals generated at contact surfaces may either introduce unwanted messages and signals into your programs and calculations, or perhaps even damage computer components. As you know if you've glanced through the back sections of your instruction manual, repair procedures are simple and usually inexpensive, but they require returning the instrument to the manufacturer and incurring unnecessary costs. Get into the habit soon of checking to be sure the computer is off when you connect or disconnect its components.

You'll find it helpful to have your computer in front of you as you read the next few sections. There are many examples in the following sections of the book to help you understand major points. The keystrokes to complete these exercises are **highlighted**.

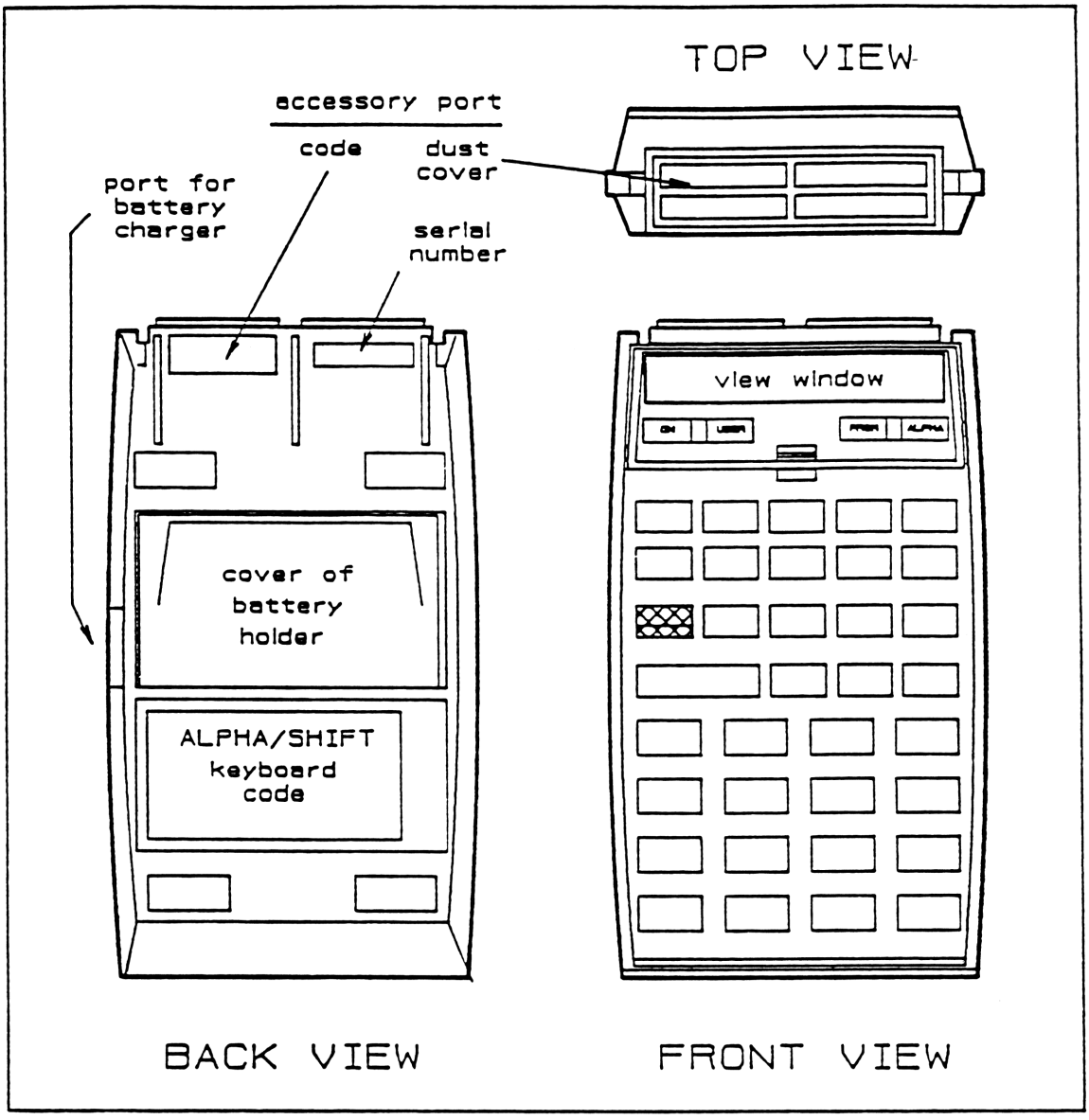
Section 2.1 Physical Characteristics

As shown in Figure 2.1, conspicuous features of the face of a HP-41 series computer are the rectangular view window at the top of the instrument, the 4 keys immediately below it, and the 8 rows of keys which take up the largest room on the instrument's front surface. As you will soon learn, each key acts either as a toggle switch to turn on and then off some operating feature or serves as a function key to initiate a specific operation. You've undoubtedly noticed that most of the keys on the bordered keyboard have a white designation on their face, a gold-colored imprint immediately above them, and a blue letter or symbol on the slanted bottom of the key. The suspicion that each key has different uses is quite right, and you will learn how to execute each of them by the end of this chapter.

As also shown in Figure 2.1, there are 4 large rectangular openings on the top of the HP-41 computers. These are the accessory ports into which are inserted the peripheral devices you decide to use. When you first unpack your computer, each

Figure 2.1.

HP-41 Physical Features



of these ports will be protected by a snap-on dust cover. It's a good practice to leave these protecting covers in place until you are ready to insert a peripheral device or memory expansion unit. Not only does the cover keep out dust, it also reduces the chance of mechanical damage to the delicate contacts which are set deeply in the base of each port.

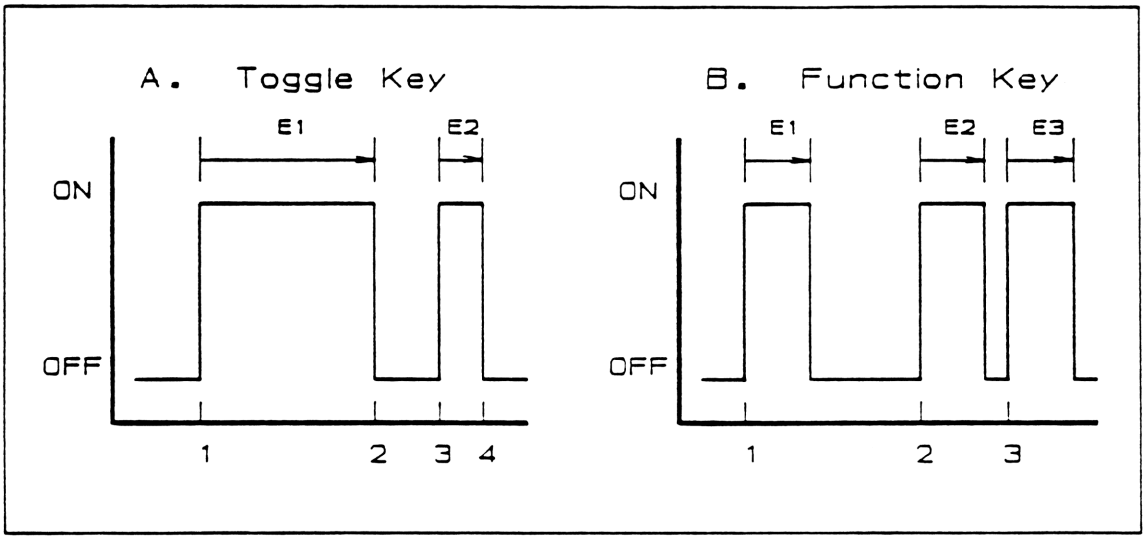
At the center on the right side of the computer is another dust cover which protects the electrical contacts for the battery charger (HP-82059B) used with a rechargeable battery pack (HP-82120A). The computer comes with 4 non-rechargeable batteries, and unless you decide to invest in a rechargeable battery pack, there is no need at all to use the recessed contacts on the computer's right side. It's a good idea, though, to keep the cover in place except when the battery charger is actually in use.

There are 2 more surfaces of the computer to examine. The bottom front edge of the computer lists on its right side the model number, and the back of the machine also contains important information. The serial number of the computer is imprinted in the top right hand corner, and the number code for the accessory ports is shown just to the left. A reduced view of the instrument's front side is displayed at the bottom of the computer's back surface. This reduced view reveals how the keyboard is defined when the instrument is in the ALPHA-SHIFT mode, one of its 5 possible configurations you will soon learn about. Remembering that this keyboard is noted on the back of your computer as a ready reference will save having to leaf through the owner's manual as you use the instrument.

You may have already discovered for yourself the removable rectangular center section on the back which contains its batteries. Gently pushing upward on this battery holder frees it at its lower edge so it can be removed for battery insertion or replacement. Reinserting it requires first aligning the top edge, then gently pressing it downward. The spring action of the battery contacts re-engage the retaining pegs at the lower edge and hold the battery case securely in place.

Figure 2.2.

Toggle and Function Operations



Section 2.2. Keyboard Operations

Even before turning the computer on, it's helpful to know what to expect when you press any of its many keys. This section describes the 2 ways in which these keys operate. It will be confusing if you are unfamiliar with their 2 quite different functions when you start to use the computer. The keys in the bordered section of the keyboard and those immediately under the rectangular view window at the top of the computer's face (see Figure 2.1), operates in one of two ways: either as a toggle switch or as a function key. Figure 2.2 shows what happens in response to the activation of either toggle or function keys.

Section 2.2.1. Toggle Keys

When a toggle key (or switch) is pressed for the first time, as shown in Figure 2.2, an event is initiated, and remains in effect until the toggle switch is pressed again. The length of time the triggered event remains active depends on the time interval between the first and second times the toggle is pressed. Whatever event is triggered when a toggle switch is activated is not affected by holding your finger on the switch. In fact, many toggle switches in daily use remain in the position you place them when they are pressed, whereas others, like those of the HP-41 computers, return to a neutral position to await your next action. Whatever event is initiated when the toggle switch is first pressed is terminated when the switch is pressed a second time. The event is started again, of course, by pressing the toggle switch a third time, and terminated a second time with the 4th press. As with the toggle switches on the HP-41 computers, events occur sequentially with pairs of "on" and "off" activations.

There are many more or less obvious examples of toggle switches in everyday life. The wall switch which turns on (and off) room lights operates as a toggle switch, as does the on-off control of a radio or TV set, a water tap, and a car's ignition. Each of these controls operates in a similar fashion: an event is intiated when the switch is first activated, and the event is terminated when the switch is activated again. The event remains in action between these two operations.

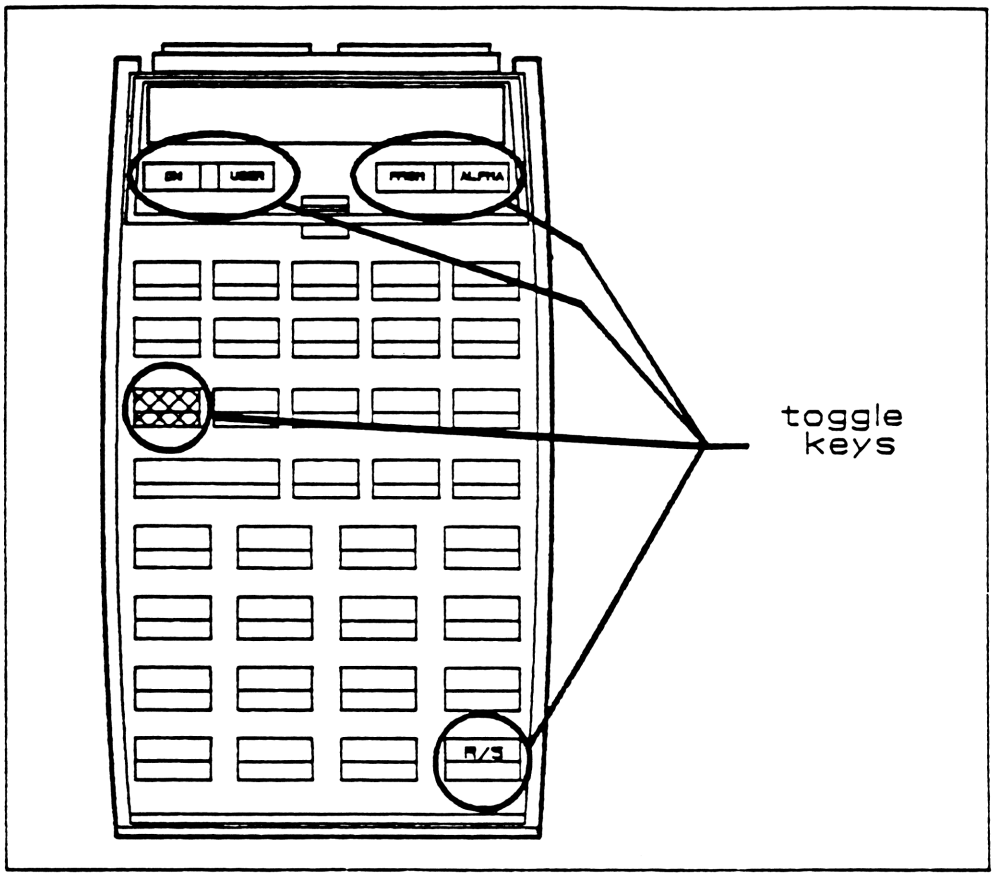
Figure 2.3 shows the location of the 6 keys on the HP-41 series computer which behave as toggle switches. All of the 4 keys immediately below the view window are toggle switches. The one to the far left turns the computer on when it is first pressed, returns to a neutral position, and then turns it off when it is pressed a second time. In most operations, either a set of numbers, words or symbols will appear in the view window when the computer is toggled on, and the window will go blank when the machine is toggled off. Just to the right of the toggle marked ON, and coupled with it is a toggle marked USER. Pressing the USER toggle has no effect unless the ON toggle has been activated first to turn the instrument on. When the computer has been turned on, the USER toggle places the computer into and takes it out of a "user" mode, as indicated by the word which appears then disappears in sequential operations just above the fulcrum of the ON-USER rocker switch.

The third and fourth toggle switches (PRGM and ALPHA) under the view window at the far right similarly have no effect unless the ON toggle has been activated first. When the computer has been turned on, these 2 switches place the computer in a program mode and/or in an alpha mode, respectively, as designated by the abbreviations (PRGM and ALPHA) which appear above each of them in the view window when they are activated. Note that when the computer is on and either the PRGM or ALPHA toggle switches is activated, either of these statuses can be turned off in one of two ways. One way to turn them off, of course, would be to press the corresponding toggle switch. Notice that turning off the PRGM toggle when ALPHA is on, turns off both actions. Another way to turn off both the PRGM and ALPHA toggles is to press the ON toggle. It might come as a surprise that the USER toggle switch is not turned off by turning off the computer with the ON-OFF switch, as are the PRGM and ALPHA switches. The only key that turns off the USER mode is the USER toggle itself. There are excellent and ingenious reasons why these switches have been configured to operate in such an interactive way, as you'll discover shortly in using these features of your computer.

Of the HP-41's 6 toggle switches, 4 of them (ON, USER, PRGM and ALPHA) reside just below the view window. The other 2 are within the bordered keyboard. One is the gold-colored key which turns on and off a SHIFT function. Similar to the actions of the 4 toggle switches outside the bordered keyboard, the SHIFT toggle turns on a word (SHIFT) in the view window when the computer is on as a reminder of the

Figure 2.3.

Toggle Key Locations



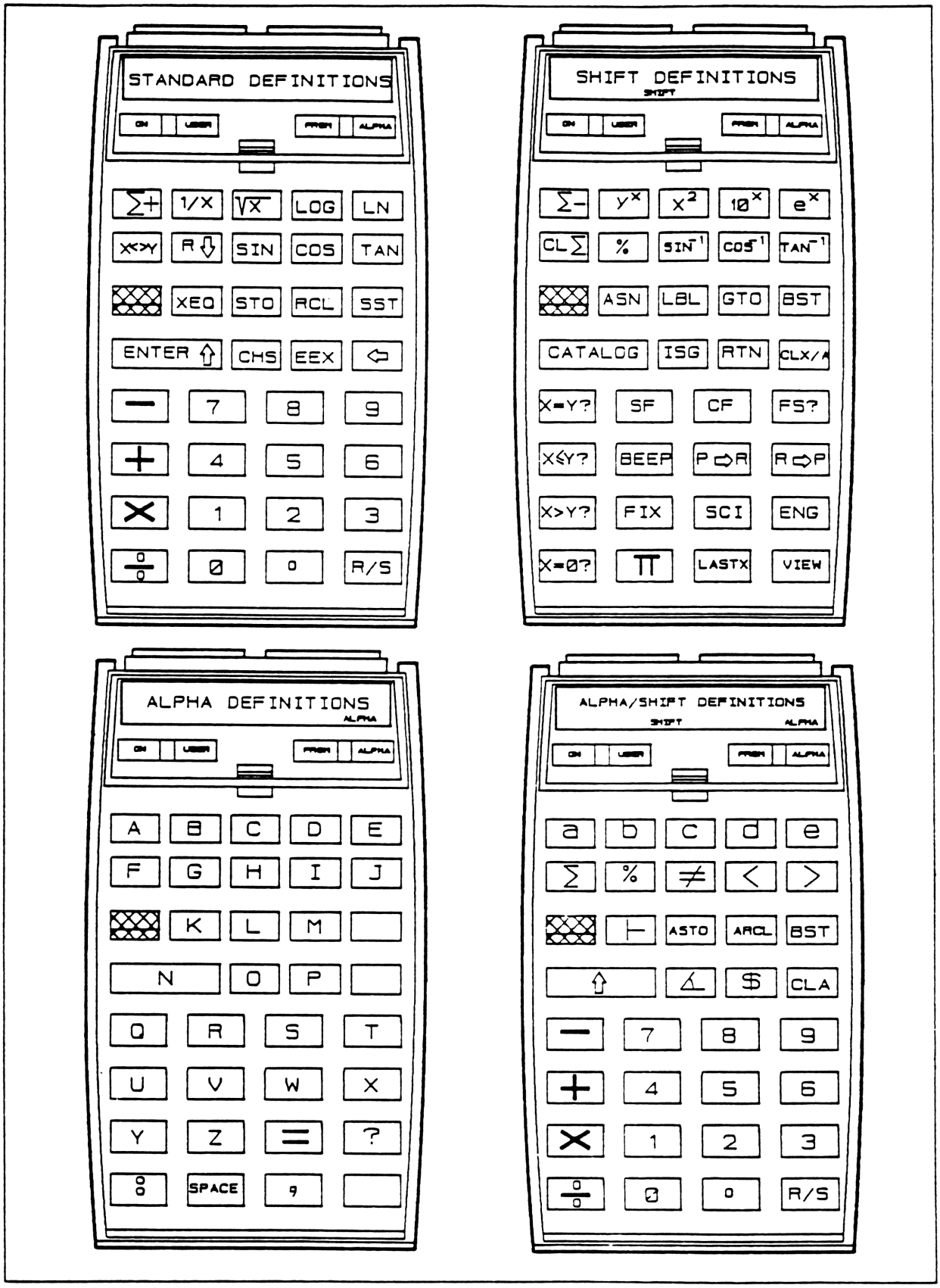
current status of the machine. The last of the 6 toggle switches is the key in the lower right corner of the keyboard marked R/S. This key operates only when the computer is on, but there is no reminder of its status in the view window as there is for the other toggle switches. You will see how the R/S key is used in programming to "run/stop" a program, and learn also the other uses it provides. The event the R/S key initiates when it is pressed is different in many ways from that of the ON, USER, PRGM, ALPHA and SHIFT keys, but it functions as a toggle switch nonetheless.

Section 2.2.2. Function Keys

Once you have located the 6 toggle keys on your computer, it's easy to find the function keys. All of the other keys on the HP-41 computers are function keys. The way in which they produce an action is different from that initiated by the toggle keys. As shown in Figure 2.2, an event is triggered by the activation of a function key, and in contrast to how a toggle switch operates, the event continues independently and normally without interruption to its end with no further operation of the function key itself. The event can be produced again by pressing the function key a second time, but it is not turned on and off as it would be were it controlled with a toggle

Figure 2.4.

Key Definitions



switch. The length of time in which the event takes place once a function key is pressed is an intrinsic property of the event itself and is not determined by sequential activation of the function key which only initiates it. The function keys on the HP-41 computers return to a neutral position after being activated and are primed to trigger the same action when pressed again.

There are many examples of the general use of function switches in everyday life. The trigger of a gun, for example, behaves as a function switch. When it is activated, a fixed sequence of events is initiated as the cartridge powder is ignited and the bullet is sent on its way. The event is produced a second time after machinery has cycled to place a new cartridge in the chamber and the trigger is once again activated. Similarly, in flushing a toilet, using the brake on a car, or blowing its horn, an event is initiated when a function switch is pressed and can be produced again when the function switch is pressed a second, third, etc. time after an interim cycle for recharging or resetting.

The action produced by most of the HP-41's function switches is either obvious from their designation (the function name printed on, above or below each key), or easily discovered simply by turning the computer on and pressing the key in question. **Pressing** the function key marked "**4**", for example, places the number 4 in the view window. It does some other important things too, which are described later. Then pressing the **TAN** function key places the tangent equivalent of that number in the view window. **Pressing** the **erase** key, the 4th key in the 4th row, the one marked "←" deletes the contents of the view window and lists zeros there. The operations performed by the function keys which are not so self-explanatory, such as for the SST, CHS, BST and other keys will be described in detail later.

Most of the keys on the HP-41 computer activate different operations depending on the sequence in which other keys are pressed. An easy way to understand how these different operations are made is to envision that the computer has 4 different sets of keyboard definitions, as depicted in Figure 2.4.

One of the functions each key brings about is designated in white letters on its upper surface (the "Standard Definitions" of the keyboard). These actions are triggered simply by pressing the key when the computer has first been turned on. Another function each key can activate is designated in gold immediately above the key (the "Shift Definitions" of the keyboard). The gold-colored function is selected by first pressing the gold-colored SHIFT key and then pressing the selected key which has now had its operation redefined. A quick example: **pressing** the key designated in white as "**4**" enters the symbol "4" in the view window. If the **SHIFT** key is toggled **before** the key designated "**4**" is pressed, the shifted function of that key will be executed, that is, a sequence of 4 tones, the BEEP, will be sounded. Each time the SHIFT key is pressed, followed by a press of the "4" key, the same BEEP tones will be heard.

It's too early to be concerned about learning what each of the standard and shifted operations are for all of the keys; these will be learned in the context of many examples to come. The important points to remember now are the similarities and differences between operations of the toggle and function keys and how each function key can be used to produce different actions.

The function designated in blue on the slanted bottom surface of each function key is selected by placing the computer in an ALPHA mode (the "Alpha Definitions" of the keyboard). Just to try it, turn the computer **ON**, then **press** the **ALPHA** key (note the word ALPHA in the view window), then press in sequence the now redefined function keys to **spell a word** using the blue symbols to guide your choice for the **correct keys**. When the word is complete as displayed in the view window, press the **ALPHA** toggle switch a couple of times to see the word turn on and off as you cycle the machine between the keyboard and ALPHA modes. Note the word is retained in the ALPHA memory even after you have turned the computer off and then on again with the ON toggle. You erase the word by placing the machine in an ALPHA

mode, then pressing the erase key ("←").

You learned earlier that key functions are redefined when the SHIFT key has been pressed, and that they are defined in yet a different way when the machine is in an ALPHA mode. There is still another way to redefine key operations by the sequence in which keys are activated. First a review. When the computer is first turned on (and not in a USER mode), each key will perform the function designated in white on its upper face. If the SHIFT key is depressed first, though, the function displayed in gold above the key will be activated. If the ALPHA key is first pressed, the function displayed in blue will occur when appropriate keys are activated.

The new point is that each key has yet another function by first pressing the ALPHA toggle, then the SHIFT toggle, and then the selected function key (the "Alpha/Shift" Definitions of the keyboard). Each key will then activate its ALPHA-SHIFTed function, none of which, unfortunately, is designated in any way on the face of the keyboard. You can tell the computer is in an ALPHA-SHIFTed mode by looking at the words ALPHA and SHIFT displayed in the view window, but to know which operation will be triggered when a key is pressed will depend either on your memory, the use of the owner's manual, looking at the list of ALPHA and ALPHA-SHIFTed definitions shown on the back of your computer, or referring to Figure 2.4.

Although the multiple uses of function keys may be inconvenient and provide initial confusion, they offer extremely valuable options in programming and using the instrument. With practice their use will become second nature. There will be plenty of help in selecting the right key for different operations in reviewing the many exercises to come.

Figure 2.5.

Timed Key Operations

For a function key when there is a number (N) in the X STACK REGISTER and when the key is pressed for:

Less than 1 second, then the function is executed for N.

More than 1 second, but less than 2 seconds, then the function name is displayed before it is executed for N.

More than 2 seconds, then NULL is displayed and the function is not executed for N which remains unchanged in the X STACK REGISTER.

There is another characteristic of the function keys and their operation which is important in normal use of the computer. Not only is each function key redefined by a SHIFT or ALPHA key operation, some behave differently depending on the length of time they are pressed, as summarized by Figure 2.5. Each of the 10 function keys in the upper 2 rows of the keyboard will perform either its function designated in white or the SHIFTed function of that key when pressed for approximately less than 1 second. Holding the key down for a longer time either first displays in the view window what function is going to take place, and then does it, or cancels the initiation of the function if the key is depressed for longer than about 2 seconds. In normal operation, however, pressing one of the function keys is such a brief event only the function itself will be executed.

Some other keys also operate with the timed key operations described in Figure 2.5. For example, the SHIFTed function of the arithmetic operation keys + − × ÷ which activate the conditional tests X=Y?, X=0?, etc. will display NULL if held down long enough. So will the PI key (SHIFTed function of the zero key), the SHIFTed functions of the "5" and "6" keys, the LASTX and CLX/A functions and keys to which a USER function has been assigned (see Section 2.3).

Besides having several different functions, each key in the bordered keyboard can be configured to provide customized service. This offers a time saving feature in selecting a particular function which is used quite frequently. The USER toggle switch and how to use the options it provides are described next in Section 2.3.

Many readers will find that learning how to work with the USER feature of their HP-41 computer is much easier after they have more experience with the operations of the machine. If you are one of these people, skip Section 2.3. and plan to come back to it later. For now, go to Section 2.4. to learn how to control the number display of your computer.

Section 2.3. The USER Mode

It was explained in Section 2.2.1. that the USER key is physically coupled to the ON toggle, and together they form a rocker switch. It was also described earlier that pressing the USER key when the computer is ON places the word "USER" in the view window in small letters immediately above the USER key. This operation is planned, of course, to provide a reminder that the computer is now in a USER mode and a signal to expect its function keys to be redefined accordingly. If, however, none has been redefined, then each key operates in exactly the same way whether the computer is in a USER mode or not.

The only value for placing the computer in a USER mode is to gain access to a function key whose operation has been modified to save time in making calculations. The cost for making such a shortcut available is that one must either remember how each key has been USER defined or construct a keyboard overlay with the newly designated functions printed on it. Another alternative to knowing the function of a USER-defined key is to take advantage of the "timed key operation" features summarized in Figure 2.5.. This make take a little more time and effort, but they are relatively low costs to obtain the increased utility provided by the USER functions. An example will help show how to exploit this feature of your computer.

Even a cursory review of the HP-41 computer manual shows there are built-in programs to calculate simple statistics such as taking the average of a series of numbers, determining their standard deviation and similar numerical relationships. These computations could be performed directly from the keyboard, of course, without using any of the built-in programs. For example, one could easily calculate the average of a series of numbers using the computer as a 4 function calculator, by adding the numbers, then dividing by the number of entries. Also, one could solve an equation to calculate standard deviation in a similar way by making the correct string of arithmetic operations in pressing keys in an appropriate sequence.

A much easier way to calculate the average and standard deviation of a series of numbers is to use the built-in programs for MEAN and SDEV. To do this, though, it is necessary first to set aside selected data storage registers in which the interim results of calculations can be stored automatically. This is easily done using a SIZE function. A later part of this book (Section 3.3.) describes how the SIZE function operates, but for this example, just press **XEQ**, then **ALPHA**, spell **SIZE**, then press **ALPHA** again. **Key "017"** in response to the ALPHA prompt "SIZE _ _ _". After a moment, the view window regains its original display, indicating the SIZE operation was performed successfully .

To calculate the average and standard deviation of a series of numbers, first press **SHIFT**, then **CLΣ** (to clear the statistical registers), **key** the **first number** in your list, then press **Σ+** . The view window will display a "1" indicating you have entered the first number in the series. **Key the second number** and press **Σ+** again to see a "2" in the view window, showing the second number has been entered. Continue entering the remainder of the numbers in the same way: keying them first, then pressing **Σ+** . When all have been entered, press **XEQ**, **ALPHA**, spell **MEAN**, then press **ALPHA** again. MEAN is the GLOBAL LABEL of the program using to calculate an average. The average of your series of numbers is now displayed in the view window. To calculate the standard deviation of the series, press **XEQ**, **ALPHA**, spell **SDEV**, then press **ALPHA** again to see the answer displayed in the view window.

Activating the operations MEAN and SDEV directly from the keyboard as you have just done is simple enough but becomes laborious if you have many different series of numbers for which you need to calculate these statistics. It's time to assign the MEAN and SDEV operations to USER defined keys, so you can obtain each function more simply with just the press of a single key, the one whose function you have redefined.

First, identify 2 keys to which you wish to assign the MEAN and SDEV operations. You can select any key to be USER defined other than either the 4 toggle keys immediately below the view window (ON, USER, PRGM, or ALPHA), or the SHIFT key. You will not lose the regular keyboard, SHIFT, ALPHA or ALPHA-SHIFT functions of the key(s) you select to be redefined for the USER mode since they will continue to perform their designated operations when your computer is not in the USER configuration. For this example, the key marked LOG will be chosen for the MEAN redefinition in the USER mode, and the one marked LN will have the SDEV function assigned to it. Figure 2.6. summarizes how to assign a USER function to a key and then return the key to its original definition afterward, if you so choose.

Figure 2.6.

Designating USER Mode Operations

A. To assign a key a USER function:

A.1. Press **SHIFT**, then **ASN**
A.2. Key the program **GLOBAL LABEL**
A.3. Press key to which function is to be assigned
A.4. Use newly assigned key function in USER mode

B. To return a key to its original function:

B.1. Press **SHIFT**, then **ASN**
B.2. Press **ALPHA**, **ALPHA**
B.3. Press key to which function had been assigned
B.4. Key now operates the same in USER and KEYBOARD modes

The computer need not be in a USER mode to assign a USER function to a key. To assign the MEAN function to the LOG key, press **SHIFT**, then **ASN** (this is the SHIFTed function of the XEQ key), then **ALPHA**, spell **MEAN**, and press **ALPHA** again. Next, press the **key** to which you want to assign the displayed function name. For this example, press the LOG key. Notice that for a short time the phrase

"ASN MEAN 14" appears in the view window. The number "14" indicates you have made an assignment to the 4th key in the first row.

Be sure your computer is not in a USER mode, press and hold down the **LOG** key. You will read "LOG" in the view window which, as reviewed in Figure 2.5., indicates the function of this particular keys when it is held down for longer than a second or so. Now press the **USER** toggle, and press the **LOG** key again. This time you read "MEAN" in the view window, identifying the USER redefinition of the **LOG** key whose function is now available to you any time USER appears in the view window.

To assign the SDEV function to the LN key, press **SHIFT**, then **ASN**, next **ALPHA**, spell **SDEV**, and press **ALPHA** again. Press the **LN** key to complete the assignment series, and read in the view window "ASN SDEV 15", indicating you have assigned the SDEV function to the 5th key in the first row. Holding down the **LN** key when the computer is in the USER mode shows in the view window "SDEV" as the re-definition of that key. Holding down the **LN** key when the computer is not in a USER mode shows the function name LN in the view window.

Calculating the average and standard deviation of a series of numbers is now even easier than it was before. First press **SHIFT CLΣ** to clear the statistical registers, then **enter each number** in the series using **Σ+** as you did before. These entries can be made whether the computer is in its regular keyboard mode or in a USER mode. After all entries have been made, place the computer in a **USER** mode (if it is not already in that configuration), and press the **LOG** key to calculate MEAN, then **LN** to calculate SDEV for your series. The USER definitions of the LOG and LN keys will be retained even when the computer is turned off. Follow the steps in Figure 2.6. to return these keys to their original definitions when you no longer need them to be USER defined.

Section 2.4. Display Options

There are several choices for how numbers are grouped, punctuated and displayed in the view window. This section describes how to control the computer to display numbers in exactly the format you wish and how to change the display at any time, even in the middle of a calculation. Figure 2.7. shows how numbers of different magnitude are displayed using either a STANDARD, SCIENTIFIC, or ENGINEERING format.

STANDARD notation for a number probably displays it in its most familiar form for the largest number of people. SCIENTIFIC and ENGINEERING displays are of advantage in dealing with very small or very large numbers. It's worth even the general user's time to become at least familiar with them. Both SCIENTIFIC and ENGINEERING notations display numbers in the decimal form you designate, multiplied by the power of 10 shown to the right of the view window. SCIENTIFIC notation displays these numbers with unit steps in the designated power, whereas ENGINEERING notation shows them in power steps of 3. STANDARD notation presents most numbers in a familiar display format but will display them in SCIENTIFIC NOTATION for either very large or very small numbers. Regardless of either the format or number of decimals chosen to display numbers or to use them in calculations, the computer uses their full 10 digit expression in all calculations.

The display format and the number of decimals one wants to read when using one of the HP-41 series computers are designated with the appropriate SHIFT functions of the keys numbered "1", "2" or "3". For an example, first **key** the number **123.4567** into the X STACK REGISTER by pressing these numbered keys, then pressing **ENTER**. To control the computer's display so that these numbers are viewed in

Figure 2.7.

Display Options: Digits

	View Window Display		
ENTER	Standard Notation FIX2	Scientific Notation SCI2	Engineering Notation ENG2
0.01	0.01	1.00 -02	10.0 -03
0.12	0.12	1.20 -01	120. -03
1.23	1.23	1.23 00	1.23 00
12.34	12.34	1.23 01	12.3 00
123.45	123.45	1.23 02	123. 00
1234.56	1234.56	1.23 03	1.23 03
12345.67	12345.67	1.23 04	12.3 03
123456.78	123456.78	1.23 05	123. 03
1234567.89	1234567.89	1.23 06	1.23 06
12345678.90	12345678.90	1.23 07	12.3 06
123456789.0	123456789.0	1.23 08	123. 06

STANDARD notation with 2 decimals, first press **SHIFT**, then **FIX**, and respond to the alpha prompt "FIX _" by pressing the key numbered "2". The view window now displays the contents of the X STACK REGISTER which at the moment is "123.4567" in your selected format of 2 decimals to show "123.46". The computer has rounded the numbers to the right of the decimal to conform to the display format you selected, but will use "123.4567" in any subsequent calculations you make.

To change the display to show a larger number of digits to the right of the decimal, key **SHIFT**, **FIX**, and respond to the prompt "FIX _" by pressing the key numbered "5", for example. The view window now shows "123.45670". To display "123.4567" in SCIENTIFIC notation with 2 decimals, press **SHIFT**, **SCI** then "2" (to see "1.23 02" in the view window), or to view it with 4 decimals, press **SHIFT SCI** "4" (to see "1.2346 02"). To display this number in ENGINEERING notation with 2 decimals, press **SHIFT**, **ENG** "2" (to see "123. 00"), and to see it in a 4 decimal display, press **SHIFT**, **ENG** "4" (to see 123.46 00). Figure 2.8. summarizes how changing the decimal display for a number affects how it is read in each of the notation formats. As described later in this book, the number of decimals with which a number is displayed can be designated within the body of a computer program or you can determine it from the keyboard.

There are still other options for how a number is displayed. For example, you can see the number 1234567.89 displayed as either "1234567.89", or "1,234,567.89", or even as "1.234.567,89", if that is more familiar to you. Information in Chapter 8 will help you make those selections. Because it is more than likely the most familiar way for the general reader to see numbers, displays for all data entries and solutions to calculations in this book will use the STANDARD notation.

The next chapter describes in more detail what happens to data when toggle and function keys are pressed, where it is stored, how you can review it, and how you can use it in mathematical operations.

Figure 2.8.

Display Options: Decimals

Step 1: **Key** the number 1234.567890, then press **ENTER**
Step 2: To see the number displayed in:

Notation:	with decimal of:	Key SHIFT then:	Display	
STANDARD	6	FIX 6	1,234.567890	
	2	FIX 2	1,234.57	
	0	FIX 0	1,235.	
SCIENTIFIC (or ENGINEERING)	6	SCI 6 (or ENG 6)	1.234568	03
	2	SCI 2 (or ENG 2)	1.23	03
	0	SCI 0 (or ENG 2)	1.	03

Chapter 3

Memory Location and Use

At this point, you have the basics of how toggle and function keys operate and know how to use many of the computer's keys to perform different operations by controlling their SHIFT, ALPHA, and ALPHA-SHIFT definitions. If you read Section 2.3. you know also how to take advantage of the USER key definition. This information is important in understanding the different ways in which data (numbers, words and symbols) are entered into the computer. Now is a good time to examine how that information is stored, retrieved, displayed and used in calculations.

This chapter reviews the way in which the HP-41 computers use their memory locations. There are minor differences in the way to use some of the memory in the HP-41CX, but for the most part, what is covered in this chapter applies to all HP-41 models. Beginning with the next chapter, you will learn how to use each of these memory locations for different types of data processing and numerical calculations.

Figure 3.1. summarizes the categories of memory in the HP-41. Not all models have all of these locations, but some can have them added using plug-in modules. Each of these variations and options will be explained in detail as this section develops. The STACK REGISTERS will be discussed first. Remember as you progress through the following pages, the keystrokes recommended to complete each exercise are **highlighted**.

Section 3.1. The STACK REGISTERS

The STACK REGISTERS are the most frequently used memory locations in your computer. Their 5 memory sites, denoted X, Y, Z, T and L, are used mainly to store numerical data, but with special storage instructions you will learn later, they can contain letters and non-numerical data as well. The next section describes how the X, Y, Z and T STACK REGISTERS operate individually and how they interact among themselves for data storage. How the L STACK REGISTER operates is illustrated in a separate section (Section 3.1.7.) later in this chapter.

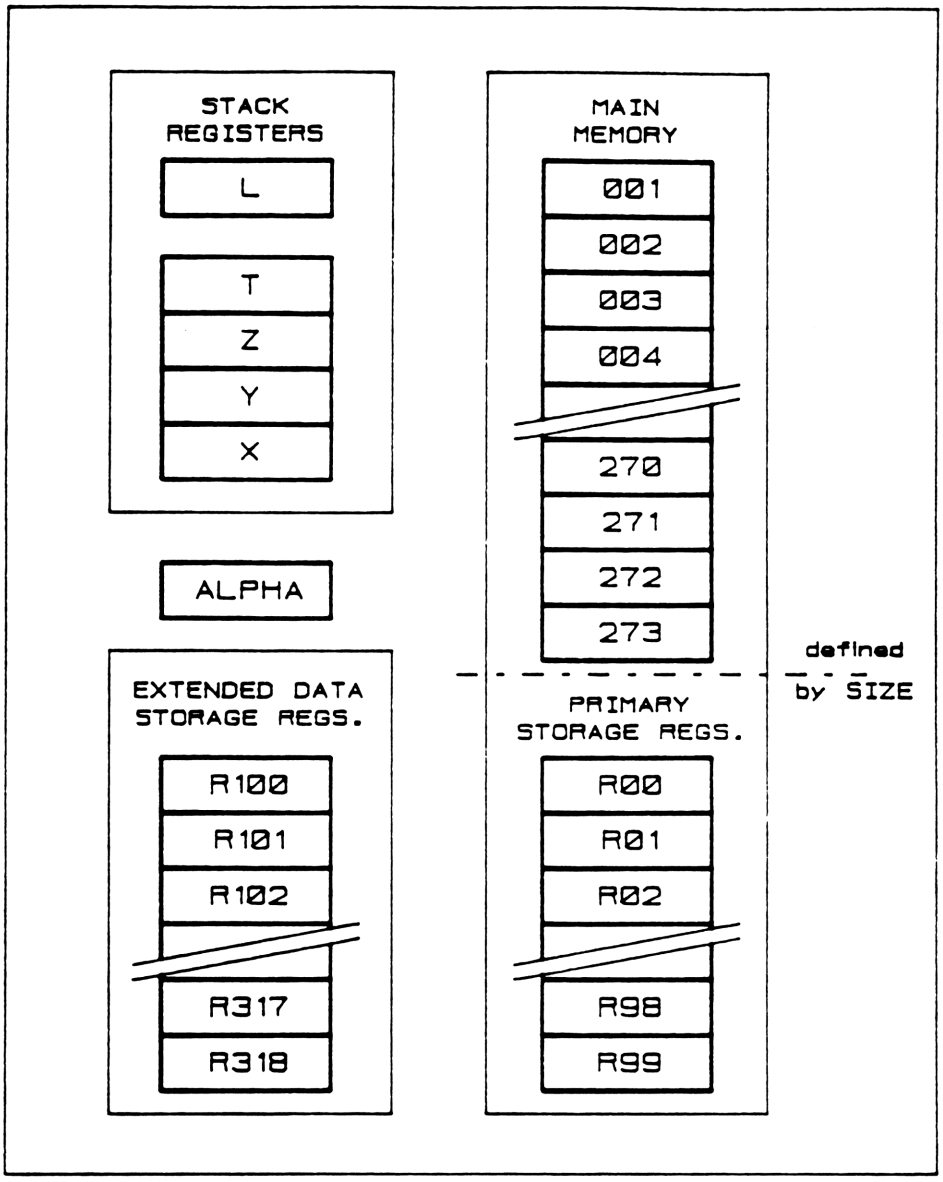
Section 3.1.1. The X, Y, Z and T STACK REGISTERS

If you've even turned on your new computer, you've already seen the contents of one of the STACK REGISTERS. In normal operation, the contents of the X STACK REGISTER is brought to the view window as soon as you activate the ON toggle switch of the machine. Since the HP-41 computer benefits from having a continuous memory, whatever number was stored in the X STACK REGISTER when the computer was last used is displayed there again when it is turned on the next time. It is unnecessary to have a zero in the X STACK REGISTER to perform accurately the next series of calculations, but you can clear the view window if you wish simply by depressing the **erase** function key (the key marked "←").

The X STACK REGISTER is going to handle a lot of the information traffic flow as you use your computer. It will function not only as a location for numerical storage itself, but also it will be the register into which you place a number before storing it anywhere else. Even more, it will be the site for viewing the contents of other memory locations, not just those of the STACK REGISTERS. The X STACK REGISTER is linked both by function and by display to the other STACK REG-

Figure 3.1.

Memory Locations



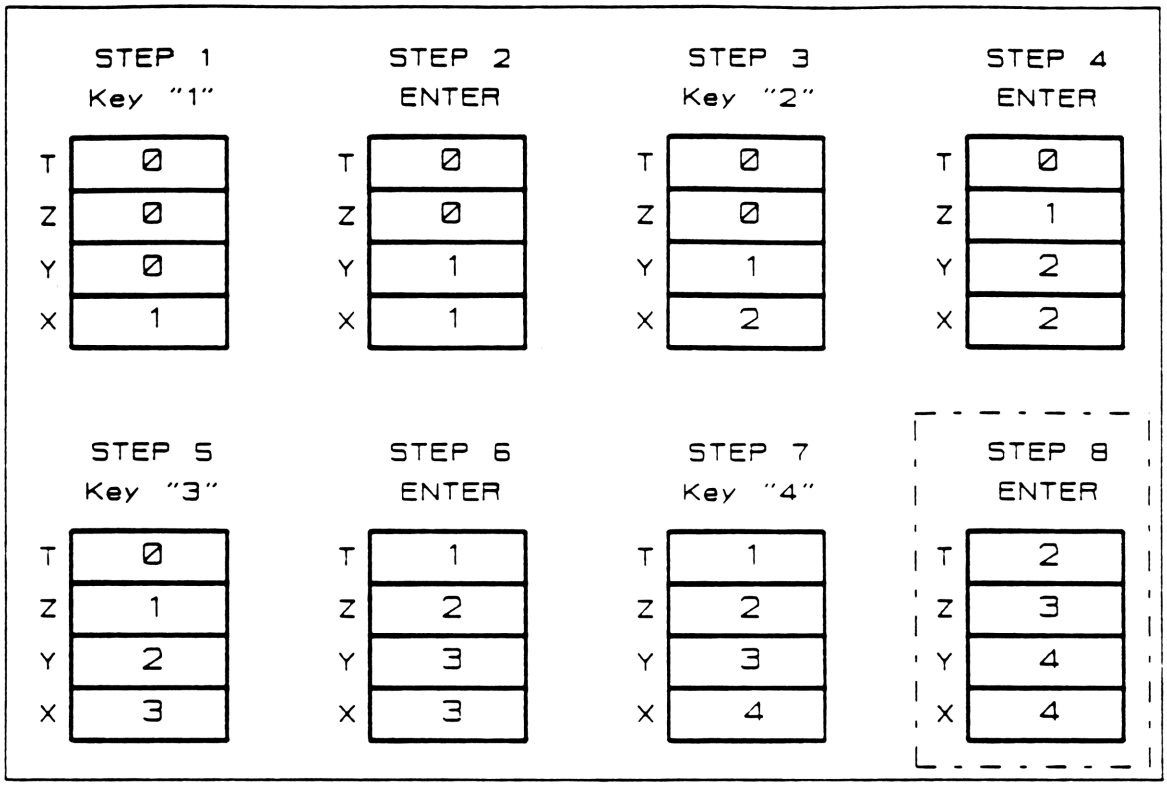
ISTERS as is explained in the next section.

Section 3.1.2. Entering Data into STACK REGISTERS

This section first explores the functional relationships among the X, Y, Z, and T STACK REGISTERS and then describes how their contents can be viewed either in specific sequence or individually. The simple exercise shown in Figure 3.2 demonstrates the functional and display relationships among the STACK REGISTERS.

Figure 3.2.

Loading The STACK



To start this exercise, turn your computer on by depressing the **ON** toggle. If it is already on, just continue with the exercise. Ignore whatever number is displayed in the X STACK REGISTER, and (step 1 in Figure 3.2.) press the key marked "1". You have placed the number 1 in the X STACK REGISTER and will see it displayed in the view window. Press **ENTER** to copy that number (step 2) into the Y STACK REGISTER. Notice that it is still in the view window. Also, it is still in the X STACK REGISTER. Now for step 3, press the key numbered "2". You will see a 2 displayed in the view window. You have replaced the number 1 which was just previously in the X STACK REGISTER with the new number. The number 1 remains unchanged in the Y STACK REGISTER.

When you now press **ENTER** (step 4 in Figure 3.2.), the number 2 is raised to the Y STACK REGISTER (it still remains in the X STACK REGISTER and is still displayed in the view window just the same as the number 1 did at step 2), but the previous contents of the Y STACK REGISTER (the number 1) has been raised into the Z STACK REGISTER. If you now (step 5) press the key marked "3", the sequence is repeated in which the number 3 takes its place in the X STACK REGISTER, is displayed in the view window, and when you again press **ENTER** (step 6), the stack is raised once again, this time to place the number 1 in the T STACK REGISTER, the number 2 in the Z STACK REGISTER, and the number 3 in the Y STACK REGISTER. The number 3 is still in the X STACK REGISTER, of course, and is still displayed in the view window. As a last step (step 7) press the key marked "4", but don't press ENTER.

The sequence shown in Figure 3.2. filled each of the STACK REGISTERS with numbers which can now be operated upon with many different mathematical functions to complete a series of calculations. Were there a "step 8" to press the ENTER key again, the stack would be raised once more to place the number 4 into the Y STACK REGISTER. The number 1 would be irretrievably lost in this operation when the number 2 took its place in the T STACK REGISTER.

The steps in Figure 3.2. show how to "load" the STACK REGISTERS using a sequence in which a number first is keyed into the X STACK REGISTER, then raised into the Y STACK REGISTER using the ENTER key. You can store a number in any single stack register, though, without having to perform a sequence of key-strokes similar to those shown in Figure 3.2. In order to demonstrate this, first clear the STACK REGISTERS by entering 4 zeros (key a **zero**, then press **ENTER** 4 times). Next enter a number into the X STACK REGISTER by pressing the appropriate key (or keys).

If you wish to store this number in the Z STACK REGISTER, for example, first press **STO** which indicates you want to "store" a number, then press the **decimal key** to signify you want to store your number in one of the STACK REGISTERS, and last, respond to the prompt "STO ST __ " by pressing the "Z" key. As you probably noticed, you didn't need to press the ALPHA toggle in order to enter the symbol "Z". Pressing the decimal key after STO signaled your computer to accept next either the letters X, Y, Z, T or L. At the end of the sequence STO ST Z, your view window will once again display the contents of the X STACK REGISTER to show the number you stored in the Z STACK REGISTER. Two examples of using STO ST__ operations are shown in Figure 3.3.A.

There are two ways to verify a number was stored in a STACK REGISTER correctly after the STO ST __ operation. One is to use a recall function (RCL), and the other is to use a stack "roll down" function. These operations are introduced next.

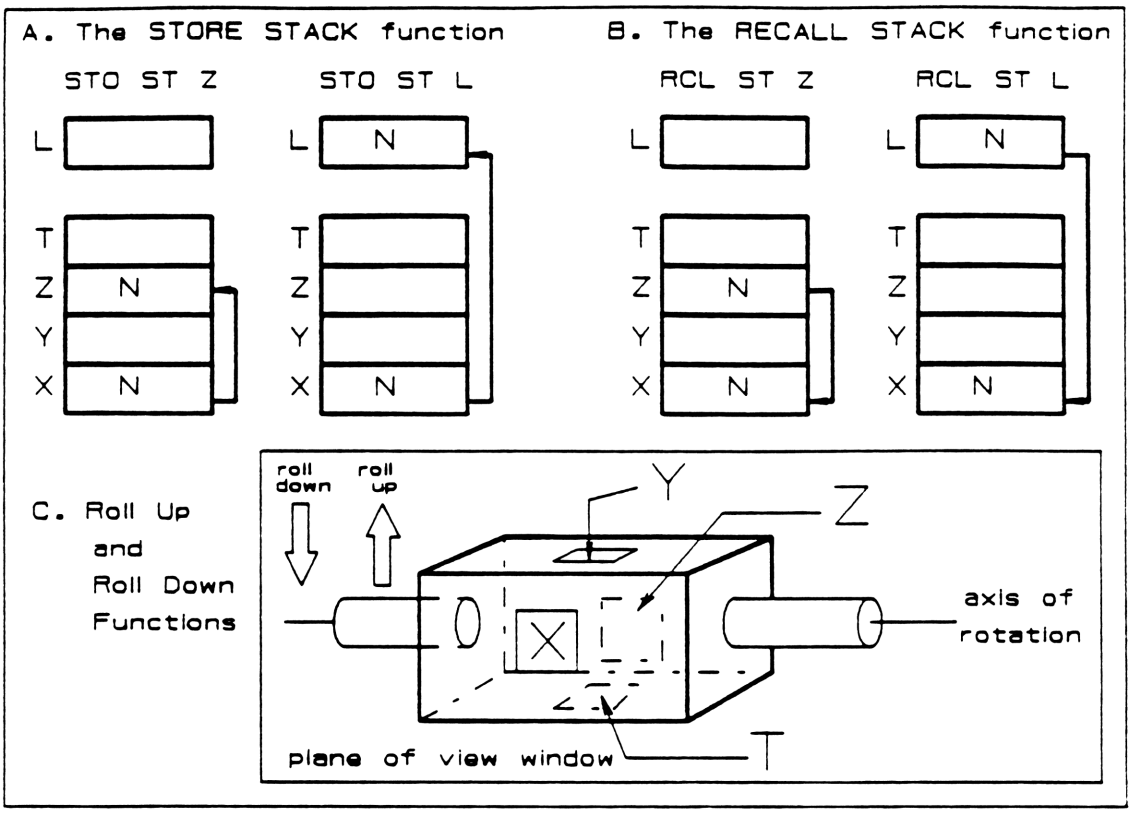
Section 3.1.3. Reviewing the Stack

If you performed correctly the sequences described in the previous section, you know how to load data into the memory locations of the STACK REGISTERS by sequential key strokes. If for no other reason than just a way to check that all the numbers are placed in correct memory locations, it's useful to know how to review the contents of the STACK REGISTERS. There are several ways to examine the contents of each of the memory locations of the STACK REGISTERS, the most direct of which is to "roll-down" the stack so that the contents of each register is displayed in turn in the view window. Figure 3.3.C. conceptualizes how the STACK REGISTERS are manipulated in this way.

The key marked "R↓" (2nd row, 2nd key) controls the ROLL-DOWN function. Using this key ("R↓"), review the contents of the STACK REGISTERS. As shown in Figure 3.3.C., **pressing it once** rolls the top of the stack toward you one position so you can see in the view window the contents of the STACK REGISTER immediately above the one at which you were just looking. If you first saw the contents of the X STACK REGISTER, pressing the ROLL-DOWN key once shows you next the contents of the Y STACK REGISTER. **Pressing it again** shows the contents of the Z STACK REGISTER, and **pressing** it the **third time**, reveals the number stored in the T STACK REGISTER. **Pressing** the ROLL-DOWN key **once again** rotates the stack another position to complete its cycle, so that the X STACK REGISTER is again displayed. Continuing to activate the ROLL-DOWN function key displays in order the X, Y, Z and T STACK REGISTERS in endless sequence.

Figure 3.3.

Reviewing the STACK REGISTERS



Pressing the ROLL DOWN function key reviews the contents of the STACK REGISTERS in the order of X, Y, Z, and T. To review the STACK REGISTERS in a reverse sequence (that is, X, T, Z, Y), use a "roll-up" function. There is no single key to perform this operation, and you will need to use one of the HP-41's built-in programs. First, press **XEQ**, then **ALPHA**, next **R**, then **SHIFT**, then **ENTER** and finally **ALPHA**. Figure 2.4 helps explain how the upward pointing arrow is shown in the display by using the ALPHA-SHIFT definition of the ENTER key. Don't be concerned at this point if you find executing the "roll-up" function a little confusing. Remember you can see the contents of the STACK REGISTERS in sequence any time more simply by using single presses of the ROLL-DOWN key. Remember also that if you lose track of which of the STACK REGISTERS you see in the view window, simply turning the computer off, then on again displays automatically the contents of the X STACK REGISTER.

Section 3.1.4. Recalling Stack Data

Using either the ROLL-DOWN key, or the keystrokes to perform the roll-up function are two ways of displaying in sequence the contents of each of the STACK REGISTERS. Neither will show, however, the contents of the L STACK REGISTER. The number stored in any of the STACK REGISTERS, including that in the L STACK REGISTER, is readily viewed out of sequence by using another one of the built-in functions of your computer, the recall (RCL) function. Two examples are shown in Figure 3.3.B.

To try it, first press the **RCL** key (second from right in the third row). You will

see "RCL _ _" in the view window indicating the computer is prompting you for a 2 digit number to designate the memory location whose contents you want to see. To indicate you want to recall a number from one of the STACK REGISTERS, press the **decimal** key. "RCL ST __" is now displayed, prompting you for the single letter (X, Y, Z, T or L) to designate the STACK REGISTER whose contents you want to be brought forward to the X STACK REGISTER to be read in the view window.

If you'd just turned your computer on, it wouldn't make much sense to press the X key in response to RCL ST __ since it's the contents of the X STACK REGISTER which is displayed in the view window anyway under this circumstance. In response to the prompt RCL ST __ now in your view window, press the appropriate **Y**, **Z**, **T** or **L** key; it is unnecessary to place the computer into an ALPHA mode to obtain the correct letter. You will then see immediately the number stored in the stack register you designated.

Recalling the contents of any of the STACK REGISTERS in this way does not alter the contents of the register itself; it merely copies to the view window the number stored in the STACK REGISTER you indicated. Similarly, recalling the contents of any data storage register, as shown later on, does not alter the number stored in it. Recalling a number from any memory location and placing it into the X STACK REGISTER for viewing changes, of course, the previous contents of the X STACK REGISTER itself. The number it originally contained is displaced and permanently lost by the one which is recalled.

In order to place a new set of numbers in the STACK REGISTERS, it is unnecessary to perform any special operation to clear them, although you will soon read about a single operation which will do that if you want. The new set of numbers will be stored in place automatically as they are entered from the keyboard, just the same as you did when you completed the data entry exercises in Section 3.1.2. As each number is first placed in the X STACK REGISTER by pressing the appropriately marked key, it is then entered successively into the higher (Y, Z and T) STACK REGISTERS when ENTER is pressed. It might be helpful to enter different sets of numbers, and then review the contents of the STACK REGISTERS just to be sure this point is clear. Being comfortable with the procedures for storing data in and seeing the contents of any of the STACK REGISTERS will help a great deal when you come to use them either in complex calculations, as described in the next chapter, or in programming, as outlined later in this book.

Section 3.1.5. Clearing the STACK REGISTERS

There will be times when starting a new calculation, either by using the keyboard to enter data, or in the execution of a program that you will need to have only zeroes stored in the STACK REGISTERS. That is, they need to be cleared of previously stored data. One way to achieve this status is to enter a series of zeroes, much the same as when you loaded the STACK REGISTERS with other numbers earlier (Figure 3.2.). Another way is to use the built-in function in the HP-41 called CLST (for "clear STACK REGISTERS"). To see how this operation is used, first **place numbers** in each of the STACK REGISTER as you did earlier, then use the **ROLL DOWN** function key to assure yourself you did this correctly. Now perform the recommended steps in the next paragraph.

The following sequence of keystrokes automatically places a zero in each of the X, T, Z and T STACK REGISTERS, but not in the L STACK REGISTER. Since you are going to use a built-in program in your computer to clear the STACK REGISTERS, it will be necessary first to address it in its storage location in the machine's memory. This is easily done by pressing the **XEQ** function key which is located just to the right of the SHIFT key. You will see "XEQ _ _" displayed in the

view window. This is a prompt for you to indicate with your next keystroke(s) which program you'd like to "execute".

The program you want to use, "CLST", is an alpha statement. In order to write this in the view window, you first need to press the **ALPHA** toggle switch. You are now ready to spell out "**CLST**" using the keys marked in blue with these letters, and then terminate the request to use this program by once again pressing the **ALPHA** toggle switch. After a short pause, you will see the contents of the X STACK REGISTER which is now zero, of course, since you have cleared this register along with all the other STACK REGISTERS when you executed the program CLST. Use the **ROLL DOWN** key to review the current contents of the STACK REGISTERS which will be all zeros.

Just to be thorough about all of this, you can see that the use of CLST did not clear the L STACK REGISTER by recalling its contents. First press the **RCL** key, then the **decimal** key, then the **L** key. If you see a zero in the view window, it's because there was a zero in the L STACK REGISTER. Try all of this again after you have stored a number other than zero in the L STACK REGISTER following the instructions summarized in Figure 3.3.B. The main point to be made is that the CLST operation clears only the X, Y, Z and T STACK REGISTERS.

By now you are able to place numbers sequentially into the STACK REGISTERS using ENTER (Figure 3.2.), selectively load a specific STACK REGISTER using the STO ST _ function (Figure 3.3.A.), review the STACK REGISTERS in order using the ROLL-DOWN function key (Figure 3.3.C.), and review the contents of any one of them by responding to the "RCL ST __" prompt (Figure 3.3.B.). You also know two ways to clear the STACK REGISTERS (enter a series of 4 zeros, or use the CLST operation). There are a couple of other ways to manipulate the contents of the STACK REGISTERS which are valuable to know. They are described next.

Section 3.1.6. Stack Storage Arithmetic and Exchange

You have the power to manipulate the contents of the STACK REGISTERS in important ways other than storing, recalling and clearing numbers in them. You can perform mathematical operations in any one of them and rearrange their contents. These are valuable operations not only for making calculations directly from the keyboard, but also they are especially powerful in program design.

You can add, subtract, multiply or divide a number in any of the STACK REGISTERS (including L) using a number in the X STACK REGISTER. This next exercise demonstrates this useful operation. First **clear** the STACK REGISTERS **using** the **CLST** operation (Section 3.1.5.). Next, place a number in the Z STACK REGISTER (Figure 3.3.A.). Using the number "**24.5**" as an example, **key it** into the X STACK REGISTER, then press **STO**, next press the **decimal** key, and in response to "STO ST _ _", press **Z**. Now add, for example, 12.3 to 24.5 using stack arithmetic. First press the **erase** key. Next, **key** "12.3" into the X STACK REGISTER, then press **STO**, **+**, **DECIMAL**, **Z**" in order. Recalling (using **RCL ST Z**) or reviewing (using the **ROLL-DOWN** key) the contents of the Z STACK REGISTER shows it now contains "36.8", the sum of 12.3 and 24.5.

The other arithmetic operations (-, X and /) are performed in an analogous way for any of the STACK REGISTERS, even to perform an arithmetic operation on the contents of the X STACK REGISTER itself using the number stored in the X STACK REGISTER. It may take a few moments, though, for most people to figure out why you can only double the number you wish to add to the X STACK REGISTER in this way. (Hint: What happens to the contents of the X STACK REGISTER when a new number is keyed into it, and what does the operation "ST+ ST _ _" do?)

Being able to rearrange the contents of the different STACK REGISTERS is handy when completing a mathematical sequence either from the keyboard or in a program. Fortunately, it's easy to do. Exchanging the contents of the X and Y STACK REGISTERS is especially straightforward. For a demonstration, first **place a number** in the X STACK REGISTER, press **ENTER** to copy it into the Y STACK REGISTER, now **key a different number** into the X STACK REGISTER by pressing the appropriate key(s), but don't press ENTER. To exchange the contents of these 2 registers, simple press the **X<>Y** function key which is just above the SHIFT key. Sequential activation of the X<>Y key switches back and forth the contents of the X and Y STACK REGISTERS. You will see several applications for this maneuver when you start writing programs.

Exchanging the contents of other than the X and Y STACK REGISTERS is a little more involved, but since it is so useful a technique for making calculations and running programs, it's worth the trouble to learn it. You'll need to execute a built-in program, much the same as you did earlier for CLST and RCL ST _ operation.

The contents of the X and Z STACK REGISTERS will be exchanged in this example. You could exchange the contents of any one of the STACK REGISTERS including the L STACK REGISTER with any other one using appropriately similar steps. To begin with, **clear** the contents of the STACK REGISTERS. Then, **key a number** into the X STACK REGISTER, and store it in the Z STACK REGISTER using **STO ST Z**. Next, **erase** the view window, then **key a different number** into the X STACK REGISTER.

To exchange the number now in the X STACK REGISTER with the one in the Z STACK REGISTER, press **XEQ**, then place the computer in an ALPHA mode (press **ALPHA**). Press the **"X"** letter key (not the key with the multiplication symbol X). Add the symbol "<" to the display in the view window by pressing the **SHIFT** key, then the **COS** key. Add the symbol ">" by pressing the **SHIFT** Key, then the **TAN** key. Now press the **ALPHA** key and respond to the prompt in the view window (X<> _ _) by pressing the **decimal** key and then the **Z** key. You have now exchanged the contents of the X and Z STACK REGISTERS. Use the **ROLL DOWN** function to be sure it all went as planned. If you were surprised at how the symbols "<" and ">" were written into the statement in the view window, review Figure 2.4. to remind yourself of ALPHA-SHIFT key definitions.

Section 3.1.7. The L STACK REGISTER

Careful readers will see that even though the L STACK REGISTER is listed in Figure 3.1. as a member of the group of STACK REGISTERS, it doesn't function either in data storage or in stack roll-down operations as do the X, Y, Z and T STACK REGISTERS. The L STACK REGISTER has a special function. It is coupled to the other 4 STACK REGISTERS, but it functions in a different and quite useful way. It is used to store the number entered immediately before an arithmetic operation. The function and use of the L STACK REGISTER are valuable for performing a special type of calculation as described next.

A short exercise will help describe how valuably different the operation of the L STACK REGISTER is from that of the X, Y, Z and T STACK REGISTERS. First, **key a number** into the X STACK REGISTER, and remember what it is. Then store this number in the L STACK REGISTER by pressing the **STO** key, then the **decimal** key, and respond to the alpha prompt "STO ST _ " by pressing the **L** key (see Figure 3.3.A.). Your number is once again displayed in the view window and, of course, it remains in the X STACK REGISTER with a copy of it now in the L STACK REGISTER. Clear the contents of the X, Y, Z and T STACK REGISTERS by **CLST**. You will see a zero in the view window (since the X STACK REGISTER was cleared),

and if you press the **ROLL DOWN** key **3 times**, you will see that the Y, Z and T STACK REGISTERS have been cleared also, as expected. The L STACK REGISTER still contains your original number, as you can easily verify by recalling the contents of that register (**RCL**, **decimal**, **L**; as shown in Figure 3.3.B.).

Another way to review the contents of the L STACK REGISTER is to bring its contents to the X STACK REGISTER using the SHIFTed function ("LAST X") of the decimal key. To try it, **erase** the current contents of the X STACK REGISTER. Next press **SHIFT** and then the **decimal** key, and you'll see again the number previously stored in the L STACK REGISTER. As you will learn from the next chapter, recalling the contents of the L STACK REGISTER into the X STACK REGISTER allows you to exchange it with the contents of any of the other STACK REGISTERS, store it directly or indirectly in other memory locations in your computer or use it in a specific calculation.

If the operation CLST is ineffective in clearing the L STACK REGISTER, how can it then have a number eliminated from it? One way, of course, would be to store a zero there using the sequence illustrated in Figure 3.3.A. But quite reasonably, why bother? Any new number stored in the L STACK REGISTER automatically displaces its contents, just the same as when a number is stored in the other STACK REGISTERS and alternative memory locations. Furthermore, the contents of the L STACK REGISTER do not become involved in a calculation unless you designate its recall specifically by RCL ST L or LAST X.

Since the L STACK REGISTER functions as a "quiet" memory location, controlling its contents is not as critical as it is for the other STACK REGISTERS whose contents are lifted into higher memory locations with ENTER, for example, or combine with those in lower STACK REGISTERS during arithmetic operations, as you'll see in the next chapter. A short exercise presented at the end of the next chapter (Section 4.5.) demonstrates how the unique operational features of the L STACK REGISTER are valuable in keyboard calculations and in the program you will soon learn to write.

Section 3.2. The ALPHA REGISTER

Similar to the STACK REGISTERS, the ALPHA REGISTER (Figure 3.1.) is a feature common to all computer models in the HP-41 series. It has only one memory location and is used to store letters and alpha symbols. It cannot store numbers to be used as such in calculations, but it can store the symbols for numbers, as in the alpha statement "ENTER X1". If you see the number 123.45 in the view window when your computer is in its ALPHA mode, you know that is a numerical statement like a word, but it is doesn't designate a quantity, as it would were it displayed in the X STACK REGISTER, for example.

The contents of the ALPHA REGISTER is viewed simply by turning the computer on and pressing the **ALPHA** toggle switch. The view window may remain blank except for the word "ALPHA" in small letters over the ALPHA toggle switch which shows the computer has been placed in an ALPHA mode. The window is still displaying the contents of the ALPHA REGISTER, but is showing that it is empty of alpha statements.

You can place symbols in the ALPHA REGISTER, of course, just by **pressing the appropriately labeled keys** (symbols are shown in blue on the slanted lower face of keys when the computer is an **ALPHA** mode). Pressing **ALPHA** after these keys have been used stores the entire string of symbols into the ALPHA REGISTER where it can be viewed any time the ALPHA key is pressed or if the instruction AVIEW is used.

As for the STACK REGISTERS and all other memory locations in the HP-41

series, the content of the ALPHA REGISTER is retained when the computer is toggled off. Just to reassure yourself that this is true, follow through a simple exercise. Turn the computer **ON**, place it in an **ALPHA** mode, **enter** a sequence of **alpha symbols** and then turn the computer **off**. Turning it **ON** again shows the contents of the X STACK REGISTER, as you might expect, but the symbols in the ALPHA REGISTER also have been preserved which you can easily verify by pressing the **ALPHA** key to see the current contents of that register.

The ALPHA REGISTER has a special characteristic for displaying whatever symbols have been stored in it. Even though the view window has only enough spaces to display a string of 12 ALPHA characters, the ALPHA REGISTER itself holds twice that many. The computer "scrolls" to the left the contents of the view window to display the entire contents of the ALPHA REGISTER when more than 12 characters are in its storage. To demonstrate this feature, turn the computer **ON**, press **ALPHA**, then press the **erase** key to clear the ALPHA REGISTER. Next **key** in alphabetical order the letters **A** to **X**. Note that a tone is sounded when you make the last entry, signaling that the ALPHA REGISTER is now full. Next press **ALPHA**. This stores the 24 character string you just generated in the ALPHA REGISTER. Press **ALPHA** again to display with "scrolling" the letters A to X. The sequence will be repeated for every second press of the **ALPHA** toggle (as you'd predict from the information in Figure 2.2.A.).

Although there is a simple way of clearing the ALPHA REGISTER by executing a built-in program (CLA), as described in the next paragraph, there is a more selective way of getting rid of unwanted ALPHA symbols. If you are spelling out a word or phrase and press the wrong key, you can easily erase just the unwanted symbol by pressing the key with the left pointing arrow (the erase key). If you have completed the ALPHA statement and have pressed ALPHA to enter the entire sequence of symbols into the ALPHA REGISTER, you can erase the entire entry by placing the computer again in the ALPHA mode, and then pressing the erase key, as you saw in the previous exercise. Entering a new ALPHA symbol also erases the existing contents of the ALPHA REGISTER.

Using one of your computer's built-in programs quickly and easily clears the contents of the ALPHA REGISTER either when it is executed by keystrokes or when it is executed as a line in a program. To try it, first **place** a set of **alpha statements** in the ALPHA REGISTER. Then turn the computer **off**, then **ON** again. This will place your alpha set into the ALPHA REGISTER, but it will also return the view window to show the contents of the X STACK REGISTER. To clear the alpha statement, press **XEQ**, then **ALPHA**, then spell out **CLA** (for "clear alpha"), then press the **ALPHA** key a last time. The ALPHA REGISTER is now clear, which you can check by pressing **ALPHA** to see a blank view window.

Even though the ALPHA and STACK REGISTERS will be the most frequently used storage registers, you will soon come to depend on your computer's MAIN MEMORY, PRIMARY DATA STORAGE REGISTERS, and EXTENDED DATA STORAGE REGISTERS, and the EXTENDED MEMORY, if you have appropriately expanded the functions of your HP-41CV or own a HP-41CX. These are reviewed next.

Section 3.3. MAIN MEMORY and Data Storage Registers

For most calculations, even quite complicated ones made with direct keyboard entries by pressing its keys, you will depend primarily on the use of the STACK REGISTERS. Once you write a program to make calculations, however, you will want to know how to manipulate your computer's memory beyond that of the STACK REG-

ISTERS. The power of your HP-41 computer lies in its ability to use algorithms (lists of step-by-step instructions) to solve problems and in its considerable memory for storing interim and final calculations, as well as alpha statements and symbols. This section only introduces these memory locations. You will learn more about how to use them later on in this book when sample programs are being written and used. If by the end of this section you have a general idea about where data are stored in your computer, that's more than good enough for the moment. This would be a good time to take another look at Figure 3.1. to remind yourself of your computer's different memory sites.

All of the individual statements which make up the body of programs will be stored in the computer's MAIN MEMORY. This memory space will be used also for PRIMARY STORAGE REGISTERS in which you can place either alpha or numeric data. A valuable feature of the HP-41 computer is that you have control in allocating the memory storage space you alone decide is needed for program construction and data storage. A built-in program called SIZE gives you full control over the available memory space of the MAIN MEMORY. It will be used when you decide how much memory you need for data storage, and how much for saving program lines.

There is always a degree of concern when one first begins to write programs as to the number of program lines, where they have been stored in the MAIN MEMORY and how many memory locations are left. The basis of the concern is that one might run out of memory space in writing the program. These are reasonable thoughts, especially in using a computer which has relatively little memory, but much of the worry is unjustified. The HP-41 computer has a built-in subroutine (PACKING) which stores program lines efficiently and economically in terms of space in the MAIN MEMORY, and a number of messages which will appear automatically in the view window if you run out of either program writing or data storage space. You will learn more about PACKING and its several functions later. In the great majority of applications, there will be plenty of program writing space for any single program, although there will be limits to the number of such programs which can be stored simultaneously.

Both the storage and retrieval of numerical and alpha data are quite straightforward for the HP-41 computer, as a simple example will help to illustrate. Turn the computer **ON**, and ignore any number displayed in the view window. Whatever number is there, you know was the contents of the X STACK REGISTER the last time the computer was used. First, adjust the size of the MAIN MEMORY to have enough data storage registers for this exercise by executing the SIZE subroutine. You do this by first pressing the **XEQ** key, then the **ALPHA** key, then spell **SIZE** and press the **ALPHA** toggle again. You will now see in the view window "SIZE __ __ __", as a prompt to indicate next (by **keying a sequence of 3 numbers**) the size of the data storage space you require. Press the keys "011" to indicate you'd like 10 PRIMARY STORAGE REGISTERS set aside for data storage. The view window will again display the previous contents of the X STACK REGISTER at the end of this sequence. You have now adjusted your MAIN MEMORY to hold 10 PRIMARY STORAGE REGISTERS.

You are now set to store and recall a number using the PRIMARY STORAGE REGISTERS. To try it, **enter a number** in the X STACK REGISTER by pressing the appropriately numbered key(s). Now, press the key designated **STO**, and you will see the statement "STO __ __" in the view window, asking you which of the PRIMARY STORAGE REGISTERS you wish to use. You previously set aside 10 of them with the SIZE subroutine. Put your test number in PRIMARY STORAGE REGISTER 07 by pressing **07** in response to STO _ _. Once you have pressed the keys "0" and "7", the view window again displays the contents of the X STACK REGISTER, which is, of course, the number you just stored. **Erase** that number using the erase key, and then recall the stored number by pressing the "**RCL**" key, and then designating in response to "RCL __ __" the register whose contents you want to see, which

is **07** in this example. The number you had stored in PRIMARY STORAGE REGISTER 07 is brought to the X STACK REGISTER and displayed in the view window. That number still remains stored in PRIMARY STORAGE REGISTER 07, you just extracted a copy of it by the instruction RCL 07 to see what it was.

As shown in Figure 3.1, there are 100 PRIMARY STORAGE REGISTERS available for data storage (designated R00 to R99), and you can use any of them if you have correctly set them aside in the MAIN MEMORY using the SIZE operation, just as you did for 10 of them in the previous exercise. Suppose you had tried to store your test number in PRIMARY STORAGE REGISTER 50, for example. If you try it, you will see that your computer is unable to find that PRIMARY STORAGE REGISTER in response to your keyed command STO 50, because you did not create a space for it in the MAIN MEMORY with SIZE, and will give you the message "NONEXISTENT" in the view window. What is "nonexistent" is the memory location you tried to use. You could make it exist, of course, by **XEQ ALPHA SIZE ALPHA 051**.

You can store alpha data in the PRIMARY STORAGE REGISTERS just as readily as you can numerical data, assuming you have generated the appropriate space using SIZE. A simple example will help. Turn the computer **ON**, clear the ALPHA REGISTER (by **XEQ ALPHA CLA ALPHA**), then place it in the ALPHA mode (press the **ALPHA** toggle), and **create a word** or phrase you can easily recognize. Next press **ALPHA** again to store this string in the ALPHA REGISTER. To store this ALPHA statement in a PRIMARY STORAGE REGISTER of your choice, press **ALPHA**, next press **SHIFT**, then **STO** (the key STO has now been redefined to be ASTO, as shown in Figure 2.4. since the computer is in its ALPHA mode) and **respond** to the prompt "ASTO __ __" **with a 2 digit number** indicating which PRIMARY STORAGE REGISTER you wish to use. If you were surprised that you had to use the shifted function of the STO key for alpha data storage, refer either to the chart on the back of your computer, review Figure 2.4., or look in your instruction book to see how the keys are defined when the computer is in an ALPHA mode. As a next step in this exercise, press the erase key to clear the ALPHA display and turn your computer off.

To recall your alpha statement from the PRIMARY STORAGE REGISTER in which your stored it, turn the computer **ON**, place it in an **ALPHA** mode, press the **SHIFT** key, then the **RCL** key and **respond** to "ARCL __" **with the number** of the PRIMARY STORAGE REGISTER you used. If all is well, you should now read the test ALPHA statement in the view window.

The combined use of the STACK REGISTERS and the PRIMARY STORAGE REGISTERS with program statements in the MAIN MEMORY provides considerable power and flexibility for writing even quite complicated, long and involved programs which use both alpha and numeric data. There is, however, yet another data storage register for you to use with your programs. As shown in Figure 3.1, it is the EXTENDED DATA STORAGE REGISTERS (numbered R100 to R318). Be careful not to confuse this register with the EXTENDED MEMORY feature built-in the HP-41CX and which can be added to the HP-41CV. It is unfortunate that each is called "EXTENDED ...", because they are quite different in function and use.

When you stored and then recalled a number and later an alpha phrase using the PRIMARY STORAGE REGISTERS in the 2 previous examples, you used the statements STO, RCL, ASTO and ARCL. You used these DIRECT ADDRESSING statements to manipulate your data for storage and recall from the PRIMARY STORAGE REGISTERS, but you would not have been able to address the EXTENDED DATA STORAGE REGISTERS in this way. The EXTENDED DATA STORAGE REGISTERS give you valuable memory storage space, but data are placed into and retrieved from them using the technique of INDIRECT ADDRESSING. This technique is sufficiently complex that it is described in a chapter of its own (Chapter 9). You will see in that chapter how data are stored and recalled using the PRIMARY DATA STORAGE REGISTERS by both DIRECT ADDRESSING and INDIRECT ADDRESSING

techniques, but that data are entered and retrieved from the EXTENDED DATA STORAGE REGISTERS only by INDIRECT ADDRESSING.

Section 3.4. The CATALOGs

Most likely it's already apparent that there are many different places to store information of one type or another in the HP-41 computer. It would be easy to lose track of the contents of these memory locations were it not for the CATALOG organization and operation of the machine. Like other general catalogs (Sears, L.L. Bean, library card files, etc.), those in your computer are a compiled list of available items. The HP-41 CATALOGs list GLOBAL LABELS and function names you can access when each is properly addressed. You will learn about GLOBAL LABELS in Chapter 5. For now, though, knowing that a GLOBAL LABEL designates the title of computer program may be enough to see how a CATALOG review is a valuable function. The computer's CATALOG system provides a convenient way to review what is stored where in its many memory locations.

The HP-41C and HP-41CV models have 3 CATALOGs, numbered 1, 2 and 3. CATALOG 1 lists the GLOBAL LABELS for all the programs written into the computer's MAIN MEMORY by keyboard entries or transferred there from magnetic cards, cassettes or other storage devices. CATALOG 1 has its contents reviewed by using the SHIFTed function of the ENTER key (first press **SHIFT**, then **ENTER**), then answering the prompt "CAT _" by pressing the key marked "1". Since you have not yet stored any programs in the computer, you will see no GLOBAL LABELS listed in the CATALOG 1 review. Were any stored in the MAIN MEMORY each would be then displayed in rapid succession, with the contents of the X STACK REGISTER held in the view window at the end of it all. CATALOGs 2 and 3 are reviewed in a similar way.

You will see later on that the display of GLOBAL LABELS is quite rapid in a CATALOG review. If you wish to stop the display series of GLOBAL LABELS in the CATALOG review, press R/S immediately after pressing the numbered key designating the CATALOG to be seen. Each press of the key marked SST advances the CATALOG review sequence one step at a time to show the next listing which is then held in the view window. Pressing R/S at any time reinstates the rapid display of the CATALOG. When a GLOBAL LABEL is held in the view window during a CATALOG 1 review, pressing PRGM provides entry into the program at its first line. Pressing SST then presents successive lines in the program. If PRGM is pressed again, you exit the program, cancel the CATALOG review, and the view window returns to display the current contents of the X STACK REGISTER.

CATALOG 2 lists the GLOBAL LABELS of programs contained in the application modules plugged into one or more of the accessory ports at the top edge of the computer. Similar to the review of CATALOG 1, sequential display of CATALOG 2's contents is stopped when R/S is pressed and taken step-by-step with successive pressing of the SST key, or the rapid display is reinstated if R/S is pressed a second time. How the R/S key operates in a CATALOG review is a good example of its "run-stop" feature. Unlike the review of CATALOG 1, pressing PRGM when a GLOBAL LABEL is displayed in a CATALOG 2 review does not provide entry to the program.

Reviewing CATALOG 3 displays in alphabetical order the STANDARD FUNCTIONS of the computer. CATALOG 3 contains many built-in function used either by keyboard operations, or when a program is executed. These functions do not take space in the MAIN MEMORY, and they are not erased when you clear programs and other stored information from your computer. They are stored in ROM and remain in the computer even if it has had its batteries removed and it has remained

unused indefinitely. For reference, the STANDARD FUNCTIONS contained in CATALOG 3 are listed in Figure 3.4.

Figure 3.4.

Catalog 3

+	CLD	FS?C	PROMPT	ST/
-	CLP	GRAD	PSE	STO
*	CLRG	GTO	R^	STOP
/	CL	HMS	R-D	TAN
1/X	CLST	HMS+	R-P	TONE
10^X	CLX	HMS-	RAD	VIEW
ABS	COPY	HR	RCL	X=0?
ACOS	COS	INT	RDN	X≠0?
ADV	D-R	ISG	RND	X<0?
AOFF	DEC	LASTX	RTN	X≤0?
AON	DEG	LBL	SDEV	X>0?
ARCLX	DEL	LN	SCI	X=Y?
ASHF	DES	LN1+X	SF	X≠Y?
ASIN	END	LOG	Σ +	X<Y?
ASN	ENG	MEAN	Σ -	X≠Y?
ASTO	ENTER	MOD	Σ REG	X>Y?
ATAN	E^X	OCT	SIN	X<
AVIEW	E^{X-1}	OFF	SIGN	X<Y
BEEP	FACT	ON	SIZE	XEQ
BST	FC?	P-R	SQRT	X^2
CAT	FC?C	PACK	SST	Y^X
CF	FIX	%	ST+	
CHS	FRC	%CH	ST-	
CLA	FS?	PI	ST*	

The HP-41CX has the same 3 CATALOGs as the other 2 models, and each is reviewed in the same way (SHIFT, CATALOG, (1, 2 or 3)), although the view window display is slightly different. When CATALOG 1 is reviewed, for example, first each program's GLOBAL LABEL is displayed as in the other models, but with the display of the associated END statement, the number of bytes in that program is also shown at the far right of the view window. Also differently, at the end of the review there is shown the number of unused REGISTERS remaining in the MAIN MEMORY. A review of CATALOG 2 displays function groups of programs in plugged-in application modules. Reviewing CATALOG 3 presents the same information as does the other models (Figure 3.4.).

The HP-41CX has 3 CATALOGs which are not in the HP-41C/CV models. A review of CATALOG 4 displays a list of files in the computer's EXTENDED MEMORY. CATALOG 5 contains a list of all the alarms set for the machine, and CATALOG 6 lists the GLOBAL LABELS and functions designed for the USER mode of the computer.

The next chapter will put into practice the skills you have learned so far for data allocation to the STACK REGISTERS. Subsequent chapters on programming will draw upon your abilities to store and recall alphanumeric data from the PRIMARY DATA STORAGE REGISTERS. Also, they will demonstrate the utility of CATALOG review.

Chapter 4

Stack Register Arithmetic

The previous chapter introduced the important concept that the computer's STACK REGISTERS will contain data entered from the keyboard, as well as that recalled from many locations in the HP-41's different storage registers. You will use these data in the STACK REGISTERS for many types of calculations, even for quite complicated step-by-step chain operations to solve complex equations. This chapter shows you how to use the STACK REGISTERS in these ways.

In order to use the STACK REGISTERS accurately and efficiently, you will need to know in some detail how calculations are performed with them using the built-in logic of your computer. This logic is probably different from the way in which you now perform arithmetic, but you'll soon see that the strategy your HP-41 uses to make calculations is easy both to learn and to use. The first section of this chapter introduces the logic of RPN ("Reverse Polish Notation"), then you will learn how to use it in performing calculations within the STACK REGISTERS. The phrase, "Reverse Polish Notation" designates a special sequence for performing arithmetic operations invented by a Polish mathematician (Lukasiewicz) more than half a century ago.

Section 4.1. RPN Logic

RPN logic has an undeserved bad reputation. Some people claim a primary reason for their not becoming proficient with the HP-41 computers is their inability to use RPN logic. Hard to believe!! By the time you have read the next couple of pages and tried some of their exercises, not only will you see for yourself how straightforward this system is for calculations, but also you will have learned all you need to about how RPN functions in order to use your HP-41 series computer expertly. The investment made in learning the information presented in the next few pages will pay off time and time again in becoming progressively more proficient with your HP-41 computer. It's not hard, but it does require some willingness to think about doing arithmetic in a slightly different way than most of us were taught in elementary school.

Because it's important to understand thoroughly how the STACK REGISTERS are used for mathematical solutions using RPN, there are many examples to try in the next pages. Even if you can "see through" the solution intuitively, take the time to key in each step. The point is to develop a sense of what's happening to the STACK REGISTERS, not to learn arithmetic.

Figure 4.1. summarizes the similarities and main differences between the way we have all learned to perform arithmetic ("standard notation") and the alternative type of notation ("Reverse Polish Notation"; RPN) used with the HP-41 computers. As outlined in the example shown at the top in Figure 4.1., a calculation is made using "standard notation" by stating the first number (N1), then the arithmetic symbol (+, -, X, or /) for the calculation to be made, then stating the second number (N2), and finally identifying an "equals" in order to get the answer. Easy enough. We have all done it this way for so long, it has become habit. If you'd want to step through adding the 2 numbers 5 and 6, for example, you'd say to yourself, "5 plus 6 equals 11" to complete the operation. You would perform a series of 5 analogous steps for the other arithmetic procedures of subtraction, multiplication and division.

RPN logic arrives at an answer in an arithmetic process in a much shorter and

Figure 4.1.

RPN Logic

A. Standard Notation:

	Step				
	1	2	3	4	5
Indicate:	N1	function	N2	equals	answer
	examples				
addition:	5	+	6	=	11
subtraction:	10	-	12	=	-2
multiplication:	8	x	4	=	32
division:	15	/	3	=	5

B. Reverse Polish Notation (RPN):

	Step			
	1	2	3	4
Indicate:	N1	N2	function	answer
	examples			
addition:	5	6	+	11
subtraction:	10	12	-	-2
multiplication:	8	4	x	32
division:	15	3	/	5

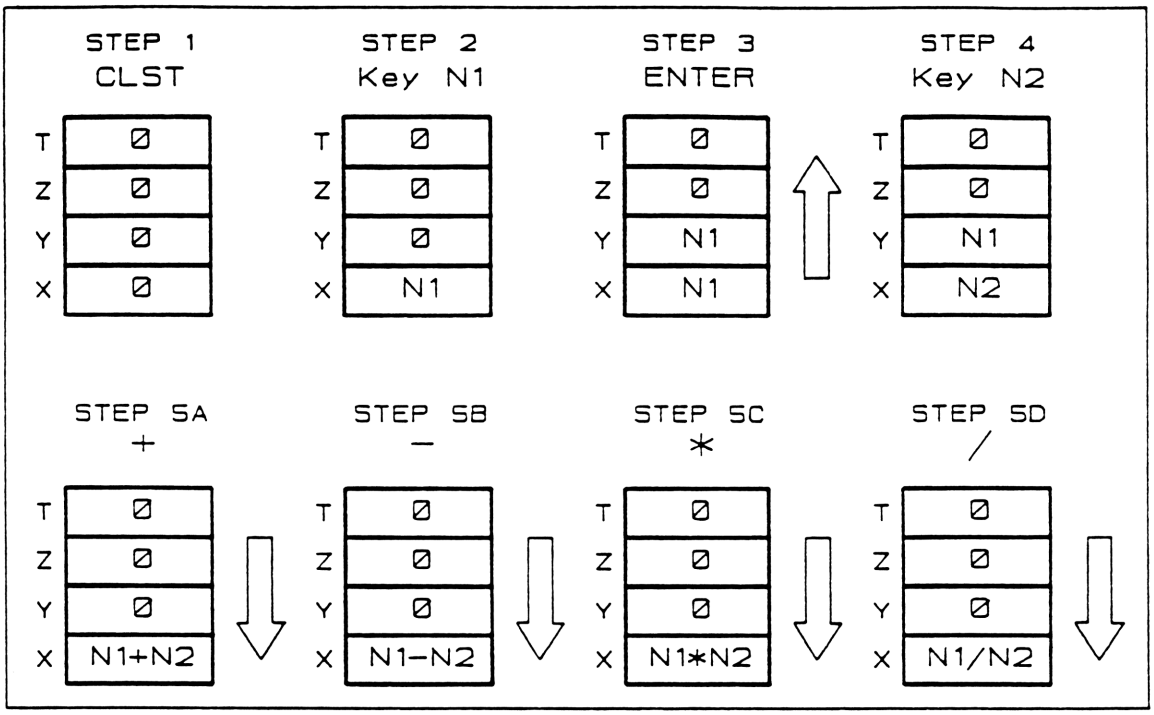
more direct way. Being able to shorten the sequence of statements in an arithmetic operation is of considerable advantage when these instructions are written into a computer program, since each step occupies at least one valuable memory location. The shorter the statement sequence can be made with no loss of accuracy, the better. As shown in the lower section of Figure 4.1., RPN shortens the process of stating arithmetic operations by 20%, compared to using "standard notation". This is much like having a computer with 20% more memory at no cost other than learning how to do arithmetic in a slightly different way. That's a good trade-off for just having to learn a new way to state the steps in the process.

There is another advantage to using RPN. It will become apparent as you use RPN for more and more complicated calculations, that not only is this form of mathematical statement shorter, it also eliminates the need to use parentheses and brackets to separate groups of numbers in a sequence of mathematical operations. These symbols are not on your HP-41 keyboard simply because you won't need them.

In contrast to the sequence followed using "standard notation" to perform an arithmetic calculation, RPN requires identifying the first number (N1), then the

Figure 4.2.

STACK REGISTER Operation



second number (N2), and finally simply indicating the arithmetic operation you want to perform on the 2 numbers. Following this sequence using appropriate keystrokes for the HP-41, the calculation is automatically performed once the symbol (+, -, X or /) for the operation is stated. The answer is quickly displayed in the view window and resides in the X STACK REGISTER. In this process, as shown in the next section, each number goes into a specific memory location in the STACK REGISTERS as it is keyed into the computer, and then the contents of those registers are combined arithmetically in the way you specified. This simple process is the same for all arithmetic functions and it is the same for all models of computers in the HP-41 series.

That's all there is to it, really. It's hard to understand why learning RPN is seen by some people to be so difficult. What happens step-by-step to the contents of the STACK REGISTERS during RPN arithmetic operations is described next. It might be a good idea for you now to review quickly the main points in Section 3.1. Just glancing over those pages ought to be good enough to remind you of the basics of STACK REGISTER operations in preparation for the next part of this chapter.

Section 4.2. STACK REGISTER Content

This section reviews how each of the STACK REGISTERS has a number entered into it during an RPN sequence and how the contents of these registers change for multiple data entries during each of the 4 basic arithmetic operations of addition, subtraction, multiplication and division. This information is essential for conceptualizing how the STACK REGISTERS change their contents when you make calculations in conjunction with the entry of data from the keyboard, or when data are

recalled from different storage registers to make a calculation. Becoming used to thinking in terms of STACK REGISTER configuration and content will make learning to write programs considerably less complicated.

Figure 4.2. shows how each of the STACK REGISTERS changes during the process of entering data and performing arithmetic operations. Step 1 in this example places a zero in each of the STACK REGISTERS by executing the **CLST** subroutine you learned to use in Section 3.1.5. Step 2 places the first number, N1, into the X STACK REGISTER by making an appropriate keyboard entry. This is done, of course, simply by pressing the key(s) for the number(s) you want. Pressing **ENTER** in Step 3 copies this number into the Y STACK REGISTER. It also "raises the stack" so that the content of the Y STACK REGISTER is moved into the Z STACK REGISTER and its content is lifted into the T STACK REGISTER. These last 2 processes are not as easy to see in this example because the Y and Z STACK REGISTERS contained zeros at the beginning of the exercise, and, of course, one zero looks about the same as any other.

It's easier to verify that the contents of the X STACK REGISTER have been copied into the Y STACK REGISTER, since you can check the contents of these registers simply by using the "X<>Y" function key (as you learned in Section 3.1.6.) after N2 has been keyed (Step 4; Figure 4.2.), assuming you didn't make the choice that N1=N2. As long as you placed 2 different numbers in each of these 2 registers, you can watch them switch back and forth as you press in sequence the "**X<>Y**" function key. If you had placed the same two sets of numbers in the X and Y STACK REGISTERS, using the "X<>Y" function key still exchanges the contents of these registers, but, of course, the display in the view window would not appear to change.

Step 4 in Figure 4.2. shows that when the second number, N2, is keyed, it replaces the contents of the X STACK REGISTER, as you saw earlier in the example described in Figure 3.2. N1 is now in the Y STACK REGISTER, and N2 is in the X STACK REGISTER. Step 5A shows that when the function key "+" is pressed, the contents of the Y and X STACK REGISTERS are added, the stack is dropped, and the sum (N1+N2) is displayed in the X STACK REGISTER. Steps 5B, 5C and 5D show how the other arithmetic operations operate in a very similar manner.

Try these simple exercises several times with different numbers until you are quite sure of the different steps in the process shown in Figure 4.2. When you succeed in seeing how the STACK REGISTERS are "raised" with successive data entries, and "dropped" with successive arithmetic operations, there will be few surprises for you in the remainder of this chapter. You will have grasped an important central idea not only for performing all types of mathematical procedures with your computer, but also for programming it. This is a major step.

It's obvious that you'd get the same correct answer in Figure 4.2. whether N1 was in the Y STACK REGISTER and N2 was in the X STACK REGISTER, or vice versa, when you pressed the "+" function key in Step 5A, since N1+N2=N2+N1. It's probably equally obvious that you would not get the same answer in Step 5B in Figure 4.2. if the contents of the X and Y STACK REGISTERS had been reversed when you pressed the "-" function key, since N1-N2 is not equal to N2-N1. The digits of the answer are the same, but the sign is different. It's important to remember that, "... it's the contents of the Y STACK REGISTER minus the contents of the X STACK REGISTER" when you subtract numbers in this way. A little practice will soon make this a routine judgment.

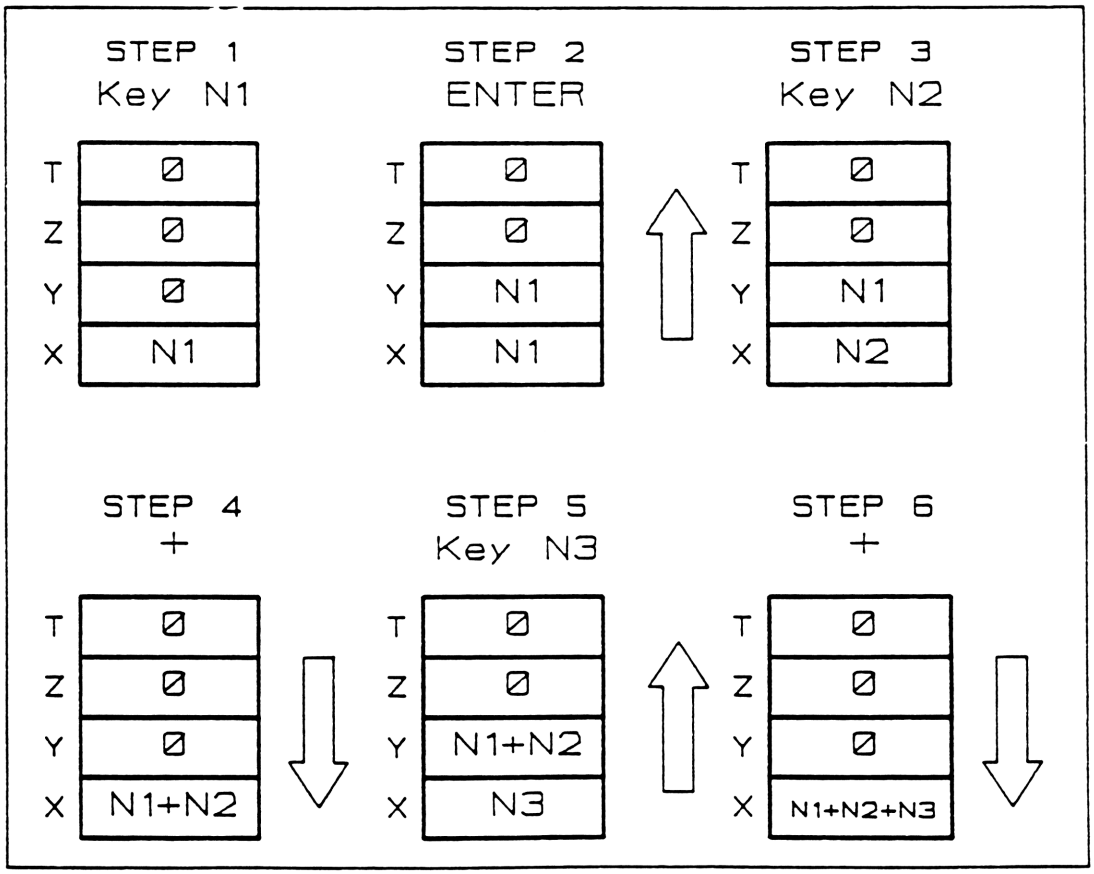
For addition, the ordering of data in the X and Y STACK REGISTERS does not affect the accuracy of the answer, but it does for subtraction. In a similar way for multiplication, the contents of the X and Y STACK REGISTERS can be reversed without error, since N1*N2 = N2*N1. This not true for division, since N1/N2 is not equal to N2/N1. It's necessary to remember that, "... it's the contents of the Y STACK REGISTER divided by the contents of the X STACK REGISTER."

As a point of review, perhaps you recall that you can perform any of the arithmetic operations using the STACK REGISTERS without entering data as described in Figure 4.2. Section 3.1.6. described how to use the function key "STO" to add (ST + ST __), subtract (ST - ST __), multiply (ST X ST __; the symbol "X" designates multiplication, not the letter "X"). or divide (ST / ST __) the contents of any of the STACK REGISTERS (or PRIMARY STORAGE REGISTERS) using the contents of the X STACK REGISTER. Remembering the keyboard entries for arithmetic described in this chapter and knowing how to use the STO (+, -, X or /) ST __ operations for STACK REGISTER arithmetic will be handy in writing programs.

Using the STACK REGISTERS for adding, subtracting, multiplying or dividing more than 2 numbers in any order is not much more difficult than the simple procedures shown in Figure 4.2. Figure 4.3. shows the steps followed for one of the ways to add more than 2 numbers. The data entry steps in Figure 4.3. are the same as corresponding ones shown in Figure 4.2. Once the sum N1+N2 is in the X STACK REGISTER, though, keying N3 (Step 5 in Figure 4.3.) raises the stack

Figure 4.3.

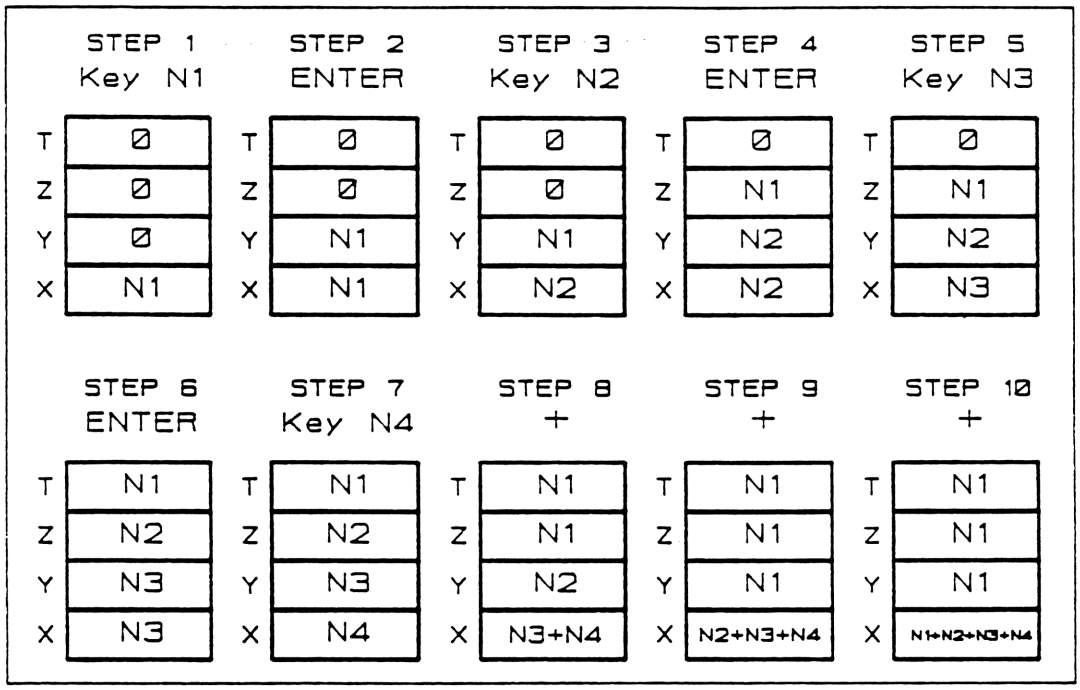
STACK REGISTER Operation:
Adding 3 or More Numbers
(Version 1)



(similar to step 2) to place the sum N1+N2 into the Y STACK REGISTER and enters N3 into the X STACK REGISTER. Pressing the function key "+" again (Step 6) "drops the stack" and combines the contents of the X and Y STACK REGISTERS to show the sum N1+N2+N3 in the view window as a display of the contents of the X STACK REGISTER. Additional numbers in an unending sequence can be added, of course, by repeating steps 5 and 6.

Figure 4.4.

STACK REGISTER Operation:
Adding 3 or More Numbers
(Version 2)



A different way of adding more than 2 numbers using the STACK REGISTERS is shown in Figure 4.4. The steps for entering the first 2 numbers are the same as matching ones in Figure 4.2. Once N1 and N2 are in the Y and X STACK REGISTERS, though, keying **ENTER** (Step 4) raises the stack and moves the contents of the Y to the Z STACK REGISTER. Keying N3 (Step 5) replaces the contents of the X STACK REGISTER. Pressing **ENTER** once again raises the stack, and keying N4 (Step 7) replaces N3 in the X STACK REGISTER. If the function key "+" is pressed once (Step 8), the "stack drops" and the contents of the X STACK REGISTER is displayed in the view window as N3+N4. Pressing the key "+" again (Step 9), drops the stack a second time, the contents of the X and Y STACK REGISTERS are additively combined, and the sum N2+N3+N4 is brought to the X STACK REGISTER. Pressing "+" once more drops the stack again to complete the solution. Loading the stack in this way and then stating two or more arithmetic operators one after the other from the keyboard (as you did in this example) would provide identical results as had you executed these steps as lines in a computer program.

It is perhaps already clear that the contents of the STACK REGISTERS can be loaded with any 4 sets of numbers and they can be combined in any order of arithmetic operations in a chain calculation, each "dropping the stack" to combine the contents of the X and Y STACK REGISTERS in the stipulated way. As a general description of this process, numbers in the X and Y STACK REGISTERS are combined in the manner designated by the arithmetic operators +, -, X and /, and a sequence of calculations can be made in any order by loading, raising and dropping the STACK REGISTERS with the appropriate function keys time after time until the final solution is displayed. All of this takes place, as described so far, from the keyboard in a specific operator-defined sequence of key operations. Be prepared to learn in Chapter 6 that an identical sequence of calculations can be made automatically from the computer programs you will write.

Most of the examples presented up to this point have shown that the STACK REGISTERS above those containing numbers for a calculation retained zeros from their having been cleared initially using the CLST program. As the stack dropped with each operation, a zero was entered into each progressively lower register, except for the X STACK REGISTER which contained the final answer of whatever arithmetic sequence was concluded. It was shown quite correctly in Figures 4.2. and 4.3. that the contents of the T STACK REGISTER (a zero) was dropped to the Z STACK REGISTER, and that of the Z STACK REGISTER dropped to the lower Y STACK REGISTER. That's OK for when indistinguishable zeros are replaced by other indistinguishable zeros, but what happens when the contents of the T STACK REGISTER is some number other than zero?

Figure 4.4. shows the contents of the STACK REGISTERS in which a number different from zero is placed (at step 6) into the T STACK REGISTER during a calculation sequence. As in the earlier examples, the contents of the T STACK REGISTER are copied into lower registers as the stack is dropped in a calculation. But this time, the same number repeats itself in a copy for each of the lower STACK REGISTERS. A potential disadvantage of this characteristic of STACK REGISTER operation is that calculation errors are unavoidable if one doesn't keep a clear mental note of the status of the STACK REGISTERS in a calculation sequence. An advantage of this characteristic is that the contents of the X STACK REGISTER can be added to, subtracted from, multiplied or divided by a constant that at one time was placed in the T STACK REGISTER (either by a stack lift or a "STO ST T" procedure (see Section 3.1.6.)) during sequences of "dropping the stack" using an arithmetic operations.

In any event, keeping close tabs on the contents of the different STACK REGISTERS is certainly a good habit, and it's one acquired with some practice. Conceptualizing what's where in the STACK REGISTERS soon becomes automatic. If you get lost in remembering what is in which STACK REGISTER, there are several ways to review their contents at any time (see Section 3.1.3.).

Section 4.3. Solving Complex Equations with the STACK REGISTERS

If you followed the examples in the previous sections, this last part of this chapter will be quite easy. You might have guessed correctly that if simple arithmetic is performed in such a straightforward and uncomplicated way using the STACK REGISTERS (Figures 4.2. to 4.4.), performing even a long sequence of arithmetic and mathematical steps can't be all that hard.

Although making a paper-and-pencil solution of the equation shown in Figure 4.5. would probably take some time for most people, especially if N3 were a fractional number, for example. But the HP-41 computer readily comes to the mathematical rescue if one executes the correct sequence of keystrokes. Figure 4.6. shows one of the several ways in which such a calculation can be done with reliable accuracy in just a few seconds with a little practice. When you are writing programs for your computer in the next couple of chapters, you'll see how such a calculation is done even more easily and quickly.

It is suggested you follow the sequence of keystrokes shown in Figure 4.6. enough times for you to be quite comfortable with chain calculations using the STACK REGISTERS, work the "trial solutions" in Figure 4.5., and then construct a few similar exercises of your own to test your skills.

Exercises up to now have been designed to give practice in executing functions from the keyboard and learning how the STACK REGISTERS are used in calculations. If you have made any number of mistakes in performing these exercises, you may have seen one or more of the messages described in the following section.

Figure 4.5.

A Complex Equation

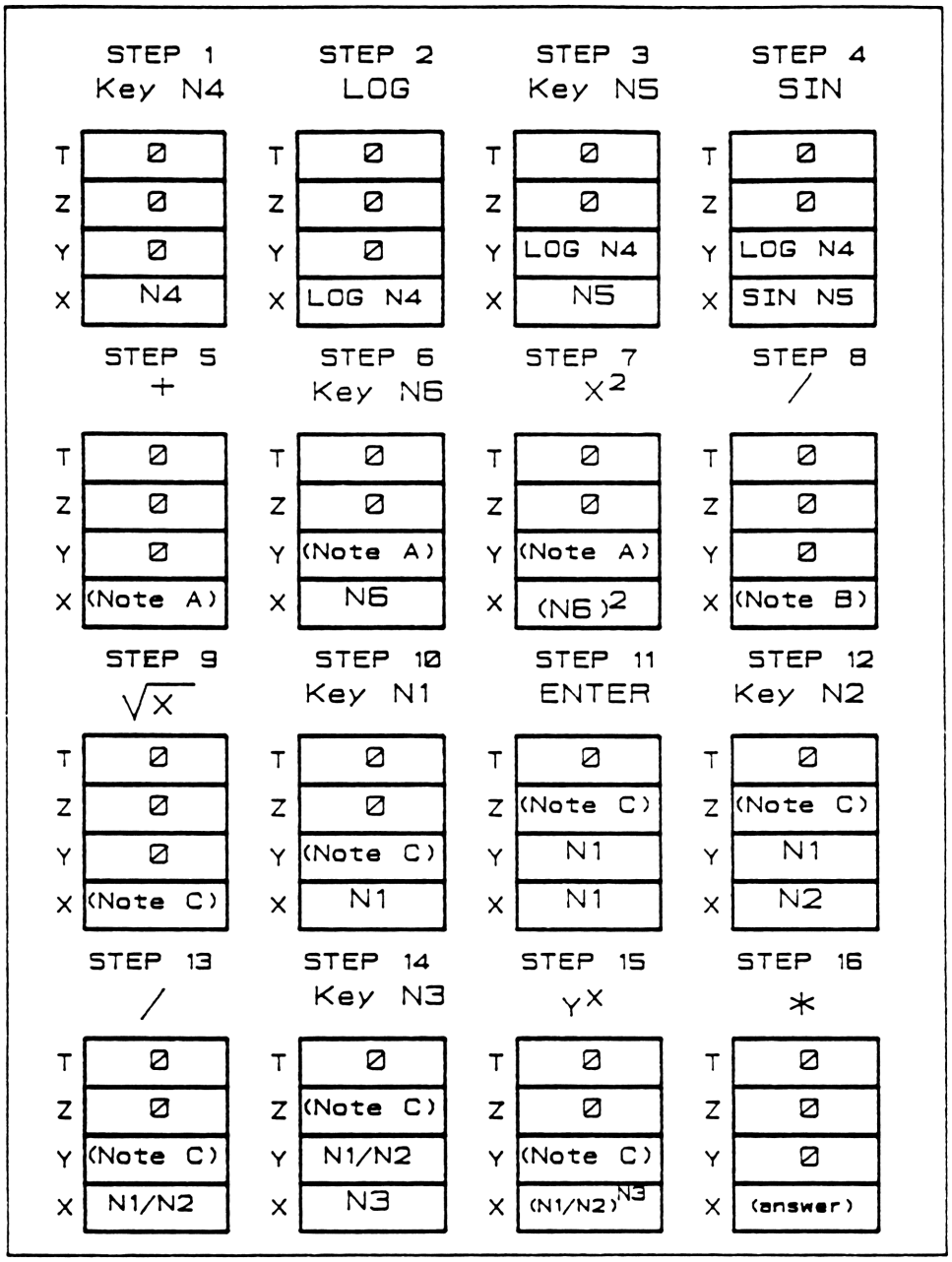
$$X = \left(\frac{N1}{N2}\right)^{N3} \sqrt{\frac{LOG\ N4 + SIN\ N5}{(N6)^2}}$$

Trial Solutions

	Test 1	Test 2	Test 3
N1	5	0.916	7.2
N2	6	0.301	4.8
N3	3	0.987	-0.876
N4	2	5.16	1.58
N5	3	4.92	1.010
N6	5	0.013	4.3
X =	0.069	206.166	0.076

Figure 4.6.

Solution of Complex Equation
Using the STACK REGISTERS



Notes:

$A = \text{LOG N4} + \text{SIN 5}$ $\quad B = \dfrac{\text{LOG N1} + \text{SIN N5}}{(\text{N6})^2}$ $\quad C = \sqrt{\dfrac{\text{LOG N1} + \text{SIN N5}}{(\text{N6})^2}}$

Section 4.4. Error Messages

A valuable feature of the HP-41 computer is that its logic system for processing data constantly "looks over your shoulder" to be sure you have not inadvertently asked the computer to perform some function which will result in a calculation logic error. You will see later on in this book that similar solid-state supervision is automatically provided when you "read" and "write" magnetic storage devices, store or load programs or data, and run programs. Other "error messages" will be explained in the context of computer use in later sections, but ones you may have already seen are described here.

The error message "DATA ERROR" will be automatically displayed in the view window if you attempt an invalid mathematical operation. Trying to divide by zero, either attempting to take the logarithm of a negative number, or using the MEAN built-in program to get the average of a series of numbers when, in fact, none has been entered, are all impossible calculations involving data. It's easy to determine what you asked the computer to do immediately prior to seeing the "DATA ERROR" message by reviewing either the contents of the STACK REGISTERS, or entering a program (as you'll see) at the point of message display. "DATA ERROR", as for all of the other error messages is easily overcome by erasing, if you wish, this alpha statement from the view window (use the erase key) and executing the attempted calculation again, but this time correctly.

The error message "ALPHA DATA" will be displayed in the view window if you try to attempt a numerical operation on non-numerical data. You may see this error message more frequently when you are using alpha statements and numerical data in a programmed calculation.

The error message "MEMORY LOST" will be displayed if you have removed a memory expanding plug-in module, have removed the batteries from your computer for a considerable length of time, or if you have had the computer turned off, pressed the erase key and held it down when you turned the machine on. Don't do this unless you want to remove all of the keyed-in programs and data you've been using. This procedure will not erase the built-in memory stored in CATALOG 3.

Your understanding and insight into how the STACK REGISTERS are used in mathematical procedures, and your knowledge of selecting the keystrokes you need for different keyboard, USER, ALPHA and ALPHA-SHIFT operations will now be put to good use in writing programs. There is another operation to be examined, though before progressing to program the computer. Section 3.1.7. described some of the unique features of the L STACK REGISTER. The next section shows how it is used in calculations.

Section 4.5. Calculations Using the L STACK REGISTER

Section 3.1.7. introduced how the L STACK REGISTER "stood alone" from the X, Y, Z and T STACK REGISTERS in its retaining a stored number when the operation CLST was performed. It is this protected feature which gives the L STACK REGISTER its special value in performing a series of calculations using the same number, and in recovering from calculation errors. Figure 4.7. illustrates how the L STACK REGISTER holds whatever number was in the X STACK REGISTER immediately prior to an arithmetic operation. This number can then be recalled either by keying the sequence RCL, decimal, L (see Figure 3.3.B.), or by performing the operation LAST X. It is then copied into the X STACK REGISTER, raising the number previously stored there into the Y STACK REGISTER.

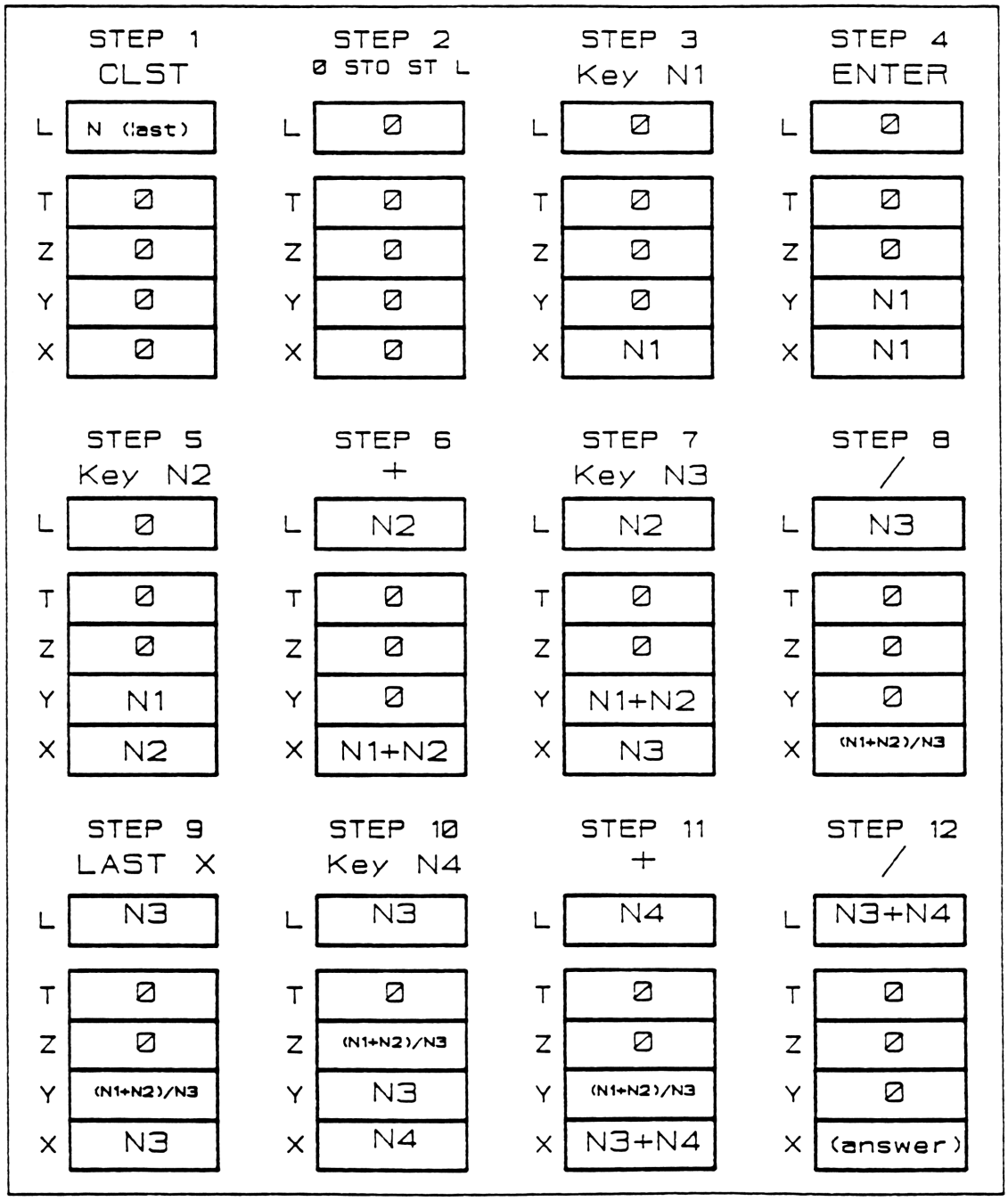
Figure 4.7.

L STACK REGISTER Operation

A. Equation:

$$X = \frac{(N1 + N2)/N3}{N3 + N4}$$

B. Solution:



In any calculation sequence, performing the function LAST X recalls, as its name implies, the last contents of the X STACK REGISTER.

Solution of the equation shown in Figure 4.7. follows manipulation of the STACK REGISTERS similar to that used in the examples presented earlier in this chapter. A difference is that in the example in Figure 4.7. the function LAST X is used. In Step 1, CLST results in there being a zero in the X, Y, Z and T STACK REGISTERS, but not in the L STACK REGISTER. This register retains the last number (N (last)) that was stored there. The L STACK REGISTER is cleared in this example by storing a zero in it at step 2.

Steps 3 to 6 are familiar from earlier exercises. They first enter 2 numbers into the X and Y STACK REGISTERS, then add them together, with the sum (N1+N2) brought to the X STACK REGISTER and displayed in the view window. As soon as the "+" key is pressed (Step 6), however, N2 (the "last X value") is copied into the L STACK REGISTER, and remains there intact as N3 is keyed (Step 7). The contents of the L STACK REGISTER is displaced by N3 when the next mathematical operation is performed (step 8) to place the solution of the equation's numerator into the X STACK REGISTER.

The contents of the X and Y STACK REGISTERS are altered with the entry of N3 at Step 9. N3 is entered using the LAST X function, drawing from storage the contents of the L STACK REGISTER. The STACK REGISTERS change again with the final process of division (Step 12).

Since the contents of the L STACK REGISTER is changed to contain whatever number was in the X STACK REGISTER immediately prior to an arithmetic operation, it is a handy repository from which to recall a number to recover from an incorrectly performed function. For example, if you wanted to add N1 and N2, had keyed them into the X and Y STACK REGISTERS (similar to steps 1 to 5 in Figure 4.7.), but pressed "-" instead of "+" at Step 6, the error is easily recovered, since the "last X" value (N2 in this example) has been automatically stored in the L STACK REGISTER as soon as you incorrectly pressed the "-" key. Pressing LAST X recalls N2 into the X STACK REGISTER, pressing "+" adds it to the difference (N1-N2), pressing LAST X again gets N2 once more into the X STACK REGISTER, and now pressing "+" gives the required calculation, the sum N1+N2.

You are adequately accomplished now in keying data, and know enough about STACK REGISTER, data storage and recall operations to put these skills to use in writing computer programs. What you have learned so far has set the stage for you now to explore the real power of your computer. The next chapter introduces basic concepts about computer programs, looks at their construction requirements, and offers some suggestions for program design. Armed with that information, actually writing programs, beginning in Chapter 6, to perform specific types of calculations will be easy. With just a little more experience and information, you will be surprised and pleased not only with the complexity of program construction you can command, but also how logical, straightforward and interesting it is.

Chapter 5

Introducing Programming

Being able to write programs into the HP-41 and retain them there intact indefinitely even when the machine is turned off are unique and powerful features of this computer. This sets them apart from the handheld portable calculators they resemble and with which, unfortunately, they are often confused by uninformed people. This chapter introduces basic definitions and concepts for writing computer programs. It summarizes some of the rules one is obliged to follow in programming the computer. Also it shows how data are entered and processed, and then how answer outputs are displayed as a program is executed. Following chapters provide many examples to illustrate these general principles.

Some of the programs presented are very simple. There are several, for example, which only add 2 numbers and others which perform other uncomplicated arithmetic calculations. But one certainly doesn't need a computer to carry out such simple tasks, so program examples must be presented in this way for other reasons.

There are 4 reasons for the construction and presentation of sample programs. The major goal is to introduce the structure of computer program logic. Since it takes practice to attain proficiency with it, the first programs are designed to make simple calculations so programming features themselves are seen more clearly. The assumption is that learning basic program writing techniques would be considerably harder to do if the mathematical examples themselves were also complex. Other programs are presented later which are more complex. All of the programs, however, are organized so that if confusion arises at a particular point, one needs only to go back to the immediately previous example to review a pertinent skill.

A second reason for presenting programming examples is for the reader to exercise key entry skills. It's not easy in beginning to use the HP-41 computers to remember all of the SHIFT, ALPHA and ALPHA-SHIFT functions of the keyboard (see Figure 2.4.), and even experienced users get confused once in a while. Practicing keying-in programs, though, will go a long way in reducing this confusion by providing familiarization with the keyboard and the many operations it controls. Writing simple programs first, then others with increasing complexity is an excellent way to obtain these skills.

The third reason why programs are presented here is that some of them will be useful for everyday calculations. For example, since most people have a checkbook, a simple program to keep it in balance is judged to be a useful programming example. Also, many people use their HP-41 series computer for simple statistical calculations like taking averages and evaluating standard deviations and standard errors for groups of numbers. You'll see a homemade program to do that also. You'll see one for getting areas under curves, and others for solving similar common problems. These examples are presented to help you get started in custom designing programs for your own use.

The most valuable reason, though, for presenting sample programs is to challenge your own design instincts. If you find "faults" (but not errors) in the style of the programs listed here, act on your judgment to change them to make the calculation(s) more the way you would like. All programs are presented as raw material for you to do just that. After you've become proficient in editing these examples to meet your needs, you will be in a much better position to write imaginatively designed programs of your own.

In the following sections, you will learn what a computer program is, and how one is designed and tested.

Section 5.1. What is a Computer Program?

A computer program is a list of step-by-step instructions designed either to solve a numerical problem, or reach a required goal. You've constructed a "computer program" of sorts whenever you've given directions to someone. Were you asked, "Can you tell me how to get to the National Bank building?", you'd generate a fixed sequence of instructions to answer the question, and may state them something like: "Sure. To get to the National Bank..." (you've identified the title for your "program"), "...go to the next traffic light, turn right, go 3 blocks to the stop sign, then turn left,..." (you've stated the sequence of instructions to follow), "...and it's the large building on the left in the middle of the next block." (and you've ended the "program" with a specific statement of the goal). Writing a successful computer program is not much more difficult than generating these directions. In the same way that you had to be precise, explicit, accurate and logical in giving your instructions to the person trying to find the bank building, you'll have to follow similar guidelines in constructing a computer program for your HP-41 computer.

Probably without thinking much about it, you followed some important self-imposed rules for providing the requested instructions to find the bank building. For example, if your directions were to be valuable, you needed to respond to the person's question in the same language in which it was asked. If they spoke to you in English, you answered in English. It is unlikely you responded in some other language, but if you did, chances are the information could not be "processed" by whomever you were talking to. Similarly in giving instructions to your HP-41 computer, you'll have to use the RPN logic and language of this machine. Fortunately, it's easy to learn, as you've already seen from the examples shown in Figure 4.1.

To help the person get to the bank, you could not have reasonably told them: "Sure, go to the next corner, turn right, go to the next corner, turn right, go to the next corner, turn right, go to the next corner, turn right...". In a similar way, if your programs are to be of value as a string of instructions for making a particular calculation, they cannot contain instructions which cause to the computer to go in computational circles. You will learn in the next few pages how to outline your program and draw a logic diagram for it so you can direct your computations to go precisely where you want them to with ease to yourself and value to the computer which has to follow these directions.

Whether embodied in a sequence of verbal directions about how to get to a bank building, or written as steps in a computer program, useful instructions need to follow some rules, not only in how individual statements are made, but also in how these instructions are placed in a sequence. For good or for bad, you'll soon see that what comes out of your pencil in writing a computer program is sometimes an uncomfortable indicator of what's going on in your head. If you're confused, your program will be confused. On the other side, if you've thought it all out in a rational way, writing and using your computer program is quite simple. Learning the structure and discipline required to generate a successful computer program is an excellent exercise in establishing logic patterns for thinking about how many problems may be solved, not just those you present to your computer. Some people are better at this in the beginning than are others, but everyone improves with practice.

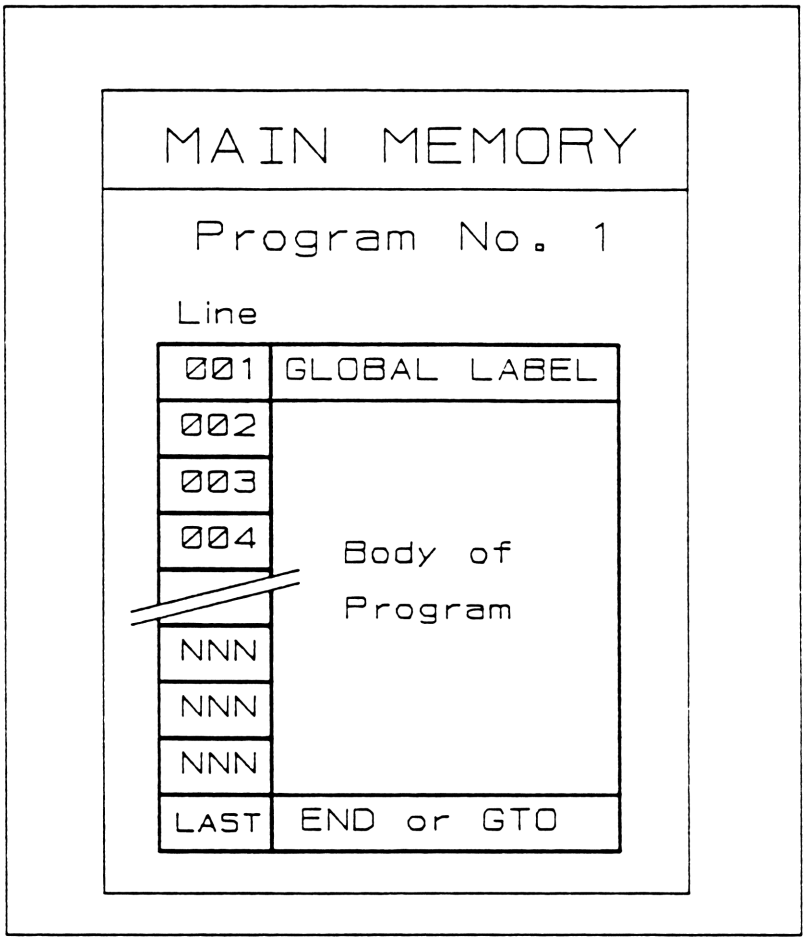
Section 5.2. Where are Programs Stored?

Figure 3.1. and the discussion in Section 3.3. introduced the concept that programs are stored in your computer in its MAIN MEMORY. Whether these programs are ones you have written yourself and entered from the keyboard, or whether they have been loaded into memory from magnetic cards, a cassette, or a set of

bar codes, each will occupy specific locations in the MAIN MEMORY. Also, each will have the same general format, as summarized in Figure 5.1.

Figure 5.1.

Storage of a Single Program



Every computer program has 3 basic elements. It will have its own unique title (defined as a GLOBAL LABEL), its set of program solving steps (in the body of the program), and an END or a GTO statement. Not only are all computer programs constructed in generically the same way, each is executed in a similar way. Program initiation begins with the GLOBAL LABEL, the program methodically passes through each of the step-by-step instructions in its body, and then program flow is either terminated (with an END instruction), or directed elsewhere (by a GTO instruction).

Figure 5.1. shows how a single program is structured to reside in your computer's main memory. Unless it is large enough to occupy all of the storage locations in the MAIN MEMORY, which is quite unlikely, other programs can co-exist there. Each can be run independently of the others. As shown in Figure 5.2., if there is

Figure 5.2.

Storage of More Than One Program

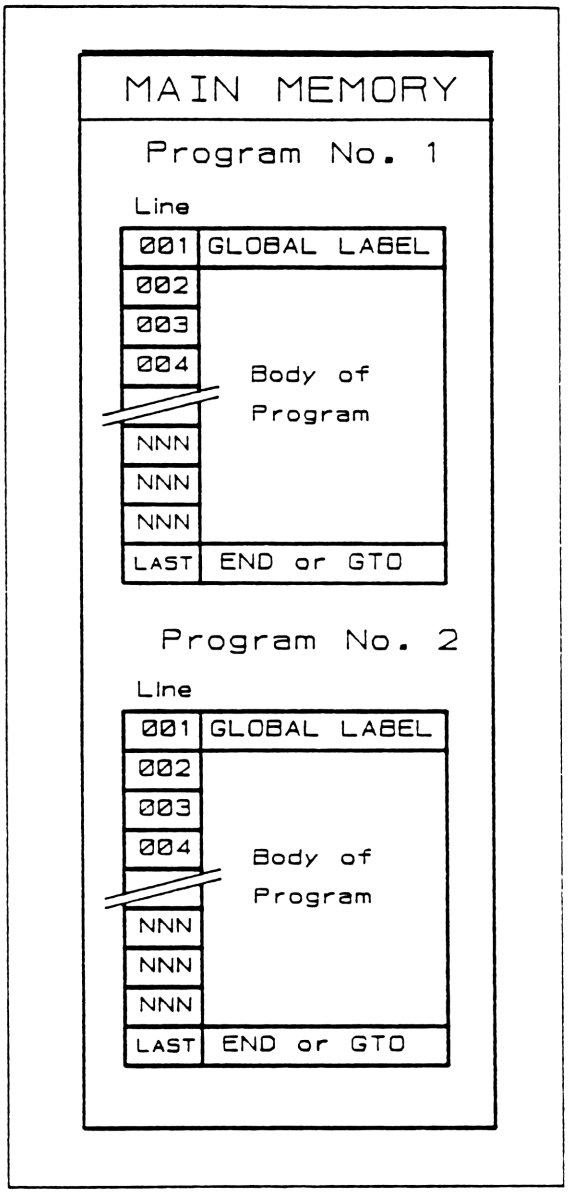
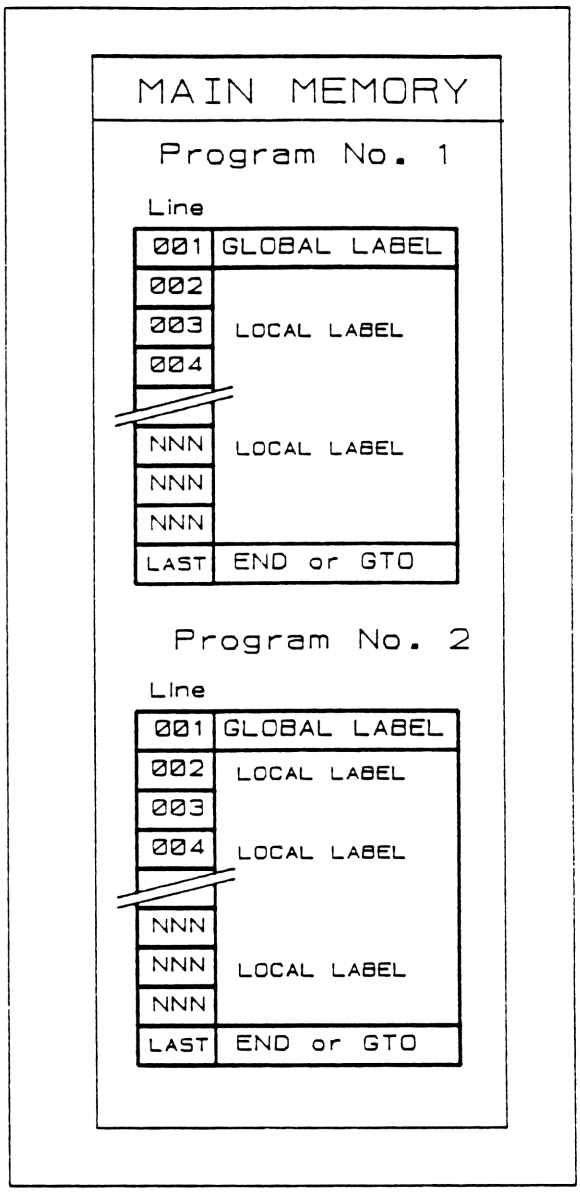


Figure 5.3.

Organization of Local Labels



more than 1 program in the MAIN MEMORY, each will have the same general organization, no matter how many of them there are, and no matter how complex any one of them is. Each has the same 3 elements: a distinguishing GLOBAL LABEL, a body of instructions, and an END or a GTO statement.

You will use 2 types of labels in writing programs for your HP-41 computer. One of them is the GLOBAL LABEL. It gives the program a unique title and distinguishes it from all other programs, not only from those stored in the computer, but also from all the others you will eventually have in your library of magnetic cards, cassettes and sets of bar-codes. No two different computer programs can have the same GLOBAL LABEL. Each GLOBAL LABEL serves as a definable entry point for your program. It tells the computer where to start the sequence of instructions for a specific calculation.

A second type of label, the LOCAL LABEL, also indicates a starting place for calculations, but operates within the body of the program, not at its very beginning, as does the GLOBAL LABEL. Figure 5.3. shows 2 programs in the MAIN MEMORY, each of which has 1 GLOBAL LABEL, but a different number of LOCAL LABELS. Some programs you will write will have no LOCAL LABELS at all, whereas others will have many of them. All programs, though, will have 1 (but only 1) GLOBAL LABEL.

Figure 5.4

Label Rules

I. A GLOBAL LABEL:

1. Can be up to any 7 ALPHA characters, except the single letters A to J, or a to e.
2. Can be any 2 digit number.
3. Cannot include commas, periods or colons.

Examples:

Good:	Bad:
LBL GRADES	LBL GRADES.1
LBL TEXT 1	LBL TEXT:2
LBL BOOK 3	LBL BOOK,3
LBL 01	LBL 1
LBL 82	LBL 123
LBL 00	LBL 0

II. A LOCAL LABEL:

1. Can be any ALPHA character(s)
2. Can be any 2 digit number

Examples:

LBL A, LBL AA, LBL AAA, LBL 00, LBL 99

Section 5.3. Label Rules

One of the first sets of rules encountered in writing a computer program are those defining how GLOBAL LABELS and LOCAL LABELS must be stated. The rules are not hard, but they are fast. Figure 5.4. summarizes required guidelines for constructing GLOBAL LABELS and LOCAL LABELS for your HP-41 programs.

It is wise, of course, to generate a GLOBAL LABEL for a program not only within the simple rules summarized in Figure 5.4., but also one which clearly indicates the calculation the program is designed to make. For example, the

GLOBAL LABEL for a program designed to balance a checkbook could reasonably be selected to be CHECKS. Using the word "checks" for the GLOBAL LABEL provides an easy reminder of the kinds of calculations this program will make. Using a GLOBAL LABEL named HORSE for this program would meet all the rules for writing a GLOBAL LABEL, and the program would run quite well under such a title, but it is hardly a good reminder of what the program was designed to do. Remembering what the program called CHECKS does, though, is quite simple. The GLOBAL LABEL for the CHECKS program meets all the requirements stipulated in Figure 5.4. It is less than 8 alpha characters, it does not consist of the single letters A to J or a to e, and it contains no punctuation symbols. Also, it is descriptive and easy to remember.

As shown in the examples of legitimate GLOBAL LABELS in Figure 5.4., spaces are allowed between words and symbols, but you need to be careful in using them. For example, the GLOBAL LABEL "TEXT 1" will work quite well, but if you forget the space and try to address a program by using the GLOBAL LABEL "TEXT1", it won't run. Unless you have a different program in your computer named "TEXT1", you will see "NONEXISTENT" displayed in the view window.

Writing LOCAL LABELS meets fewer restrictions than writing GLOBAL LABELS, as shown in Figure 5.4. They can consist of just a single alpha character, but can have more than 1 if you'd like. Also, they can be any 2 digit number. As soon as you start reviewing simple programs in the next few chapters, you'll soon see where LOCAL LABELS are placed in a program and what they do. If you can remember how they differ in location, construction and general use in a program, that's more than good enough for now.

Section 5.4. Program Construction

It might come as a surprise to discover that relatively few errors arise in writing a computer program because of the misuse of RPN logic and incorrect manipulation of the STACK REGISTERS. As already seen (Chapter 4), these procedures are simple enough. They are seldom the source of errors. What then accounts for all of the time which sometimes has to be spent in "de-bugging" a program? Most computer programs don't run accurately because of the design of their logic flow, not because of the attempted execution of specific calculations. Designing the sequence of steps in even uncomplicated programs is often tricky, and overlooking a single step which is simple in itself can be very costly. What then is a useful strategy for keeping it all straight? You learn about one in the next section.

Section 5.4.1. Designing a Program

Figure 5.5. lists steps to follow for designing a computer program. The construction of all program examples presented in this book follows these suggestions. Even though it is tempting to omit a step or two in building a simple program, it's a poor decision. The little extra time invested to get into good program writing habits pays off big in the long run when you're building complex programs with many instructions, places for data inputs, branches and loops.

This section reviews the steps to follow in designing a program. The next section will describe steps for keying your new program into the computer and testing it. Don't expect at this stage to be able to execute all of these suggested steps. You'll learn how to do that when specific programs are presented in the next chapter. For now, just familiarize yourself with what needs to be done, not how it is done.

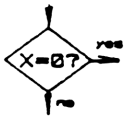
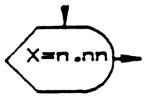
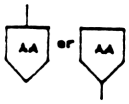
Figure 5.5

Suggestions for Designing a Program

1. Identify the program's goal or calculation.
2. Write all related equations.
3. Define all equation symbols with their units.
4. Draw a logic flow diagram.
5. Write the program steps in rough draft form.
6. Key the program steps into the computer.
7. Test the program for logic flow and accuracy
8. Rewrite the program using subroutines.
9. Add additional features of TONES, alarms, etc.
10. Write User Instructions.
11. Document program in detail with sample problems.

Figure 5.6

Sample Flow Chart Symbols

Symbol	Designated Function	Note
	START (beginning of program) or STOP (end of program)	1 output only 1 input only
	PROCESS (for example: "Add", "Calculate X", etc.)	1 input and 1 output
	MANUAL OPERATION (for example: "ENTER N1", "ENTER DATA", etc.)	1 input and 1 output
	DECISION (for example: "X=0?", "FS?06", etc.)	1 input and 1 of 2 outputs
	DISPLAY (for example: "X=25.67", "GRADE = A minus", etc.	1 input and 1 output
	OFF-PAGE CONNECTOR (for example: "To page 4","From Sect. B", etc.)	1 input or 1 output only

The first step in writing a successful computer program is to state precisely what you want that program to do. This is done in 2 stages (Steps 1 and 2 in Figure 5.5.). The first step is to write a simple sentence or two describing WHAT will be calculated or accomplished by the program. The second step is to express HOW this will be done by stating the equation(s) the program will use. All symbols need to be identified, and each needs to have its appropriate units (if any) listed with it (Step 3 in Figure 5.5.) The next step is to construct a logic flow diagram. It makes less difference that you use the symbols defined in Figure 5.6., than it does you draw a programming map you understand which indicates where you want to start the program, what you want to have it do at each step, and where you want it to end or branch.

Armed with accurately stated equations and the details of a flow diagram, writing the program itself is a straightforward step-by-step process. In contrast, trying to write a program without the equation and flow diagram references is at

Figure 5.7.

Sample Program Records

Title: _____ Page: _____ of _____

Date: _____ Version: _____ SIZE: _____

Step	Entry	Note	Step	Entry	Note
01			50		
02			51		
03			52		
04			53		
05			54		
06			55		

Title: _____ Page: _____ of _____

Date: _____ Version: _____ SIZE: _____

REG.	Data Stored	FLAG	If Set	If Clear
00		00		
01		01		
02		02		
03		03		
04		04		
05		05		

best chancy. It's where most people get into trouble in program writing. They simply loose track of what they need to do next and where they need to go as their program becomes longer and more complicated. Their tangled logic and program trip-ups get them into deeper and deeper trouble as they try to hack their way out of the confusion. You can predict how they might describe their unsuccessful attempt to complete a program without careful prior planning once it has mired down: "I can't write a program using RPN, that language is too difficult". That's not the real reason at all.

It is unnecessary when writing the first draft of a program to be concerned either with how efficiently the program will run, or whether it contains all the many features you'd like it to have eventually in its final draft form. It is much more important that you have a working record of each of the steps in the program, and know where you have stored numbers, alpha statements and interim calculations. There are many different ways to document this information. With a little experience, you'll want to design your own programming record sheets, but to get started, try using the format similar to the ones shown in Figure 5.7. You'll see how useful these are in the next couple of chapters.

As another suggestion, have on hand many blank copies of the program record sheets so that subsequent versions of the program can be listed on new pages, not by erasing entries from earlier drafts. The advantage is that the earlier drafts provide a valuable set of notes and a record of how your program evolved.

If you find the next section confusing because it presents too much detail too soon, don't bother to struggle through all of it now. Just look at Figure 5.8., read the last paragraph in this chapter, then go on to Chapter 6. You will be referred to Figure 5.8. several times later. You will be reminded to come back to read Section 5.4.2. carefully after you've gained some practical experience in writing simple computer programs.

Figure 5.8

Suggestions for Keying-in
A Program

1 Turn on HP-41 computer and key **GTO ..** to initiate **PACKING** function.
2. Determine if there is enough room in the **MAIN MEMORY** to receive the new program (Hint: either press **PRGM** to see **REG nn**, or note number displayed at end of a **CATALOG 1** review). If there is not enough room:

   2.1. **XEQ "SIZE" nnn** to redefine memory space.
   2.2. Clear programs from **MAIN MEMORY** which are no longer needed
   HINT: use XEQ "CLP" (GLOBAL LABEL), or
   2.3. If HP-41C is being used, add Memory Modules.

3. Press **PRGM** to place computer in program edit mode and key-in **GLOBAL LABEL**.
4. Key-in sequential program steps.
5. Press **PRGM** to return computer to keyboard mode
6. Key **GTO ..** to initiate **PACKING** function.
7. Use **CATALOG 1** to check for program's **GLOBAL LABEL**
8. Test program for logic and accuracy.

Section 5.4.2. Keying and Testing a Program

Once the program written in rough draft form (Step 5 in Figure 5.5.), it's time to key it into the computer and give it a try. The suggestions presented in Figure 5.8. will help avoid errors at this critical next step. Before starting to enter program steps, it is necessary to verify there is enough room in the MAIN MEMORY of the computer for the number of program steps. It's easy to do. First make sure that whatever programs are already in the computer are stored as efficiently as possible. All the work will be done for you using one of the built-in programs ("PACK") in your machine. The easiest way to execute this program is first to press the **SHIFT** key, next press **GTO**, then press the **decimal key twice**. You'll see the word "PACKING" displayed in the view window, and then a quick display of the remaining number of registers you have in the MAIN MEMORY.

If you miss seeing the number of available registers displayed when you performed the PACKING instruction, there's another way to get it. Press the **PRGM** toggle switch to place the computer in a program writing mode, and note the register number which is displayed. If there is inadequate MAIN MEMORY space for your new program, follow the instructions in step 2 of Figure 5.8. to generate more memory locations.

Once you press the **PRGM** key, you are then ready (Step 3 in Figure 5.8.) to key-in first the **GLOBAL LABEL** for your program, then key-in each of the individual program steps in order. After the last one has been keyed, press the **PRGM** toggle, then execute the PACK program again ("**GTO..**") so your new program steps are stored efficiently. You are now ready to see if your program runs as you intend.

There is no need to be concerned in the first check of program construction whether calculations are made accurately or not. First see whether the program flows logically from one point to another as intended in the program's design. The logic flow diagram (Step 4 in Figure 5.5.) is indispensable at this point. After any programs errors have been corrected which inhibit logic flow, it's time to check if calculations are being made with numerical accuracy.

More than half the battle is over once it's been determined that the program runs correctly through the logical sequences designed in its flow diagram. Checking for numerical accuracy is quite simple. First, solve a variety of sample problems by hand using the equations you wrote for the program (Step 2 in Figure 5.5.). You will need to solve at least one sample problem for each type of calculation you wish to make, and double check that calculations are made correctly no matter how the flow diagram branches. If you encounter calculation errors when you run your sample problems on your program, they are most likely due to your misinterpreting how the STACK REGISTERS are loaded with numbers, how mathematical procedures rearrange the contents of these registers, and how they are raised and lowered with successive calculations. A review of Section 4.2. and 4.3. will undoubtedly help.

When convinced the program follows the logic flow you intended and it performs the mathematical calculations correctly, it's time to put the finishing touches on your creation. Now is the time to review the lines of the program to identify redundant steps which can be placed in subroutines (definitions, explanations and examples will come in the next chapter) and to add alerting tones and cleverly constructed alpha prompts for data input and display of answers. There will also be lots of examples for this too as you read ahead.

You are now at a point when you are most familiar with how your program is initialized, how data are entered into it, how it runs, and how answers are displayed in sequence. Such detail might not come to mind quite so readily later after you have written other programs and done a lot of other things. Step 10 in Figure 5.5. suggests

now is the best time to write step-by-step "user instructions" for your program. These will not only help you when you come to use your program again, but also they will be invaluable for others who wish to use your program. They will not have had, of course, the benefits of familiarization with your program that you have as its author.

The last step is for you to document your new program (Step 11 in Figure 5.5.) and review in detail what each step does. No doubt you will be able to use your program without this last piece of documentation, but your memory will soon fade about the details of how you performed different calculations. It will be very difficult to decipher specifics of program flow after you have written your program to include multiple subroutines (Step 8 in Figure 5.5.). Now while your program's construction is still fresh in your mind is the easiest and best time to bring together all of the careful documentation (equations, symbol definitions and flow diagrams) you've developed for writing your program. You'll find them to be valuable records.

As you "grow into" your computer and write more and more programs for it, it's only a matter of time before you are faced with the decision to abandon some of the ones you already have in your machine in order to make room in the MAIN MEMORY for new ones. It's unlikely you'll want to take the time and trouble to key-in the older ones that had to be erased when you need them again, and you'll have to consider how to expand your computer to interface it with program and data storage devices of one kind or another.

If you decide it's time to start saving programs you have written, the least expensive and simplest device to obtain is the HP-82104 card reader. With the inexpensive magnetic cards it uses, you have virtually unlimited ability not only to store programs of your own by "writing" them on the cards, but also to "read" magnetic cards written by others for programs you want to use. The card reader plugs into the top of the computer, through one of its accessory ports, and is light weight and small enough to remain connected to it while the machine is in general use. The protective vinyl case you received with your computer is made to fit even with the card reader in place. Both encoding a program onto magnetic cards and reading one from them is

Figure 5.9.

"Writing" a Magnetic Card

1. Enter program to be recorded into the HP-41 computer using keystrokes, an optical wand, pre-recorded cards, a cassette drive or other device.
2. Turn computer **OFF** and connect (HP-82104) card reader into port 4.
3. Turn computer **ON** and key **GTO..** to initiate the PACKING function.
4. Key **GTO."GLOBAL LABEL"**, then press **PRGM**.
5. Insert blank magnetic card into right side of card reader to "write" side 1. "Write" side 2 in a similar manner.
6. Repeat Step 5 until all cards have been processed, then turn computer **OFF**.

Figure 5.10.

"Reading" a Magnetic Card

1. Be sure the HP-41 computer is **OFF**, then connect card reader (HP-82104) to port 4 and turn computer **ON**.
2. Key **GTO..** to initiate Packing function.
3. Be sure the computer is in a **keyboard**(not PRGM) mode.
4. Insert first magnetic card to be read into right side of the card reader to "read" side 1. "Read" side 2 in a similar manner.
5. Repeat Step 4 until all cards have been processed. After last card, repeat Step 2.
6. Turn computer **OFF** before removing card reader.

simple, if the proper steps are followed. Suggestions for these procedures are offered in Figures 5.9. and 5.10.

More complex and sophisticated data and program storage devices, such as HP's digital cassette drive (HP-82161), or optical wand (HP-82153) for reading bar codes are also options, as is a portable 3 1/2 inch disc drive to be used with HP's Interface loop. These and many other valuable peripherals, such as printers, a video interface, as well as an ever-growing library of programs will help you obtain the HP-41 computer system you need for your own special interests and applications.

There are many commercial sources for equipment, supplies and instruction manuals for your computer, the most obvious of which is your regional Hewlett-Packard distributor. Another one, however, which offers not only accessories for the HP-41 made by Hewlett-Packard, but also those manufactured by other companies is EduCALC (27953 Cabot Road, Laguna Niguel, CA 92677). Their frequently updated catalog is a reliable source of what's new for the HP-41 series computers.

This chapter has focused on what needs to be done in the development of a computer program. You are now prepared to see how each step is actually accomplished. You will write your first program after reading a few pages of introduction in the next chapter.

Chapter 6

Elementary Programming Techniques

Beginning with this chapter, first a series of simple, but then progressively more complicated programs will be presented to demonstrate basic programming techniques. Each example will introduce one or more new ways in which program flow can be controlled. These techniques will be used again in other exercises, and you'll soon come to see how to construct your own programs to accomplish complex solutions. Now is the time, though, to begin introducing these techniques in their simplest forms. The reader is encouraged to key in each exercise into the computer, try the program to be sure it runs correctly, but then modify it to make it run in some way that might better suit your needs.

Regardless of how complicated the logic flow is in any program, there will always be a requirement to enter data from which calculations can be made, and to format the output of solutions so they can be read and recorded. Data input into programs developed for the HP-41 may come eventually from external monitoring sources through digital multimeters, from magnetic cards and from cassettes as your use of the computer becomes more sophisticated. It is likely, though, that for the programs you will write at first, data will be entered directly from the keyboard.

In order to do this, you will need some signal from your computer to indicate not only when it is appropriate to enter data, but also to designate what kind of data is required. Once calculations are made in the body of the program, some other technique is necessary to display answers in a readable format. A simple scheme for controlling data input and output is shown in the program outlined in Figure 6.1. and discussed in the next section.

Section 6.1. SUM1

Following the suggestions offered in Figure 5.5. for designing a program, the upper section of Figure 6.1. states explicitly what this first simple program is expected to do. The appropriate equation is shown immediately below, even though it is a very simple one. No units are listed since the calculation deals with numbers only, not those representing a particular physical measurement. The flow diagram shows the sequence for data input, calculation and output, and indicates where the program will be directed once it has made and displayed the results of a calculation in a single pass through the program. The program code is listed on the right.

Since this is the first program to be presented, all instructions for keying it into the computer will be given in detail. Programs presented later on will omit some of this information, assuming it has become familiar. If later on you forget how to enter one or more of these instructions from the keyboard, you may want to come back and review one or more of these first exercises.

Before starting to key-in program lines for this exercise, it will be useful to know how to recover from errors of either pressing a wrong key or entering a wrong line. It is inevitable these mistakes will be made from time to time. They are only minor annoyances, though, if one knows how to correct them without endangering the remainder of the program which has been entered correctly. The erase key (the one marked "←" helps you correct both errors of key entry and program line listing.

If you have mis-entered only a single symbol, pressing the erase key once eliminates just that symbol and moves the prompt one space to the left. If you press it

again, it erases the next symbol to the left, and moves the prompt once again to the left. If you mis-entered an entire program line, pressing the erase key eliminates the entire line and brings into the view window the program line immediately preceding it. Pressing the erase key again eliminates the displayed line and takes another backward step in the program. Key entry errors are easy to make, but fortunately they are easy to correct.

Figure 6.1.

SUM1

Problem Statement:

This program is designed to add 2 numbers, N1 and N1 using the STACK REGISTERS.

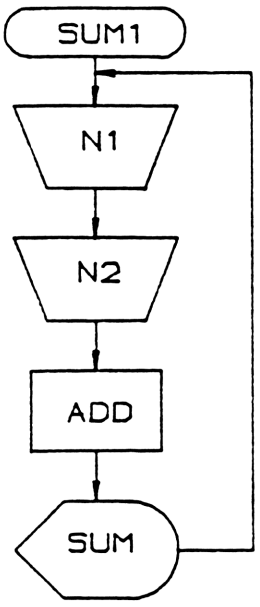
Equation:

$$SUM=N1 + N2$$

Where:

N1 = first number
N2 = second number

Flow Diagram:



Program Listing:

```
01 LBL"SUM1"
02 "ENTER N1"
03 PROMPT
04 "ENTER N2"
05 PROMPT
06 +
07 "N1+N2="
08 ARCL X
09 AVIEW
10 STOP
11 GTO "SUM1"
12 END
```

Section 6.1.1. Keying the Program

Now is a good time to go back and familiarize yourself with Figure 5.8., even if you haven't read Section 5.4.2. in detail yet. You will be reminded to go back and look at parts of Figure 5.8. from time-to-time in the next sections and chapters.

The exercise for SUM1 shown in Figure 6.1. starts with a computer which is completely free of data, instructions and entered programs. To attain this status, toggle the computer **OFF**, press the **erase key** and hold it down. Now turn on the computer **ON** and release the **erase key**. When executed correctly, the view window displays the phrase "MEMORY LOST" to indicate all REGISTER memory locations have been emptied, as well as entries in the MAIN MEMORY which may have previously contained programmed instructions. You have cleared all RAM.

Because the computer has just been cleared of all stored information, you can start at step 3 of the suggestions listed in Figure 5.8. for keying in the SUM1 program. After pressing the **PRGM** toggle key (and seeing "PRGM" appear in the view window as well as seeing also a statement of the available MAIN MEMORY registers), the GLOBAL LABEL is ready to be written in at the top of the program as its first line. The GLOBAL LABEL "SUM1" is selected for this program since it is the first of a series of programs to demonstrate how numbers can be added by using an HP-41 program. It is written as the beginning step in the program by first pressing the **SHIFT** key, and then pressing the **LBL** key (this is the shifted function of the key marked "STO"). The statement "01 LBL _ _" appears in the view window.

Since the GLOBAL LABEL of the SUM1 program is an alpha statement, next press the **ALPHA** toggle, then spell "**SUM1**" using the appropriate keys marked in blue (the ALPHA definition of the key) and finally press **ALPHA** to complete the entry of the alpha statement "SUM1". You get the "1" for SUM1 by using the SHIFTed function of the "Z" key while the computer is an ALPHA mode (press SHIFT then the "1" key). If getting the "1" was a problem, review the definitions in Figure 2.4. and look at the diagram on the back of your computer. You will need to combine numbers with alpha statements in many of the exercises to come, so you might as well be sure how to do it right now.

You should now see "01 LBL SUM1" displayed in the view window. The next line of the program is the alpha prompt which requests the entry of the first piece of data (N1). It is written into the program by first pressing the **ALPHA** key, then spelling, "**ENTER N1**", and finally pressing the **ALPHA** key. The word "ENTER" must be spelled by using the ALPHA definitions of the keys, not by attempting to use the rectangular key marked "ENTER ↩". The third program line is necessary to hold the alpha statement "ENTER N1" in the view window until you have actually entered a number using R/S when the program is run. This next line (program line 03) is written by first pressing the **XEQ** key (you are going to execute a built-in program of CATALOG 3 called "PROMPT") then pressing the **ALPHA** key, next spelling "**PROMPT**", and finally pressing the ALPHA key. You will see in the view window "03 PROMPT" if you did all of that right. If you didn't, use the erase key to get back to your last correct key or program line entry, and try it again.

The PROMPT function may appear many times in a program. It is required to hold a displayed word or phrase until requested data have been entered. When you find it tiresome to have to key-in each letter of the word PROMPT, consider activating this function by a single step after assigning it to a USER defined key. Section 2.3. describes how to do this and Figure 2.6. outlines the steps of the operation.

You are now ready to construct the 4th line of the program, the one which will request the second data entry. You do it by first pressing the **ALPHA** key, then spelling "**ENTER N2**", and finally pressing the **ALPHA** key. Write the 5th line of the program the same way you did the 3rd line, then for the 6th line, simply press

the "+" key. You have completed the instructions to enter data and to perform the required calculation on them. You are now ready to construct how you want the answer to be displayed.

In order to write line 07 of the program, first press **ALPHA**, then spell "N1+N2=" (hint: you get the "+" in this statement by first pressing **SHIFT** then the "+" key), but don't press the ALPHA toggle again just yet, even though that was the next step in writing lines 02 and 04 which appear similar to line 07. For line 07 to be able to display the numerical solution to the calculation made at step 6, not only does this number have to recalled from the X STACK REGISTER, but it also has to be recalled as an ALPHA STATEMENT in order to appear with the phrase "N1+N2=".

It's simple to do. Next press **SHIFT** then the **RCL** key (ARCL is the SHIFTed function of the RCL key when the computer is in an ALPHA mode), then press the **decimal** key (in order to indicate you are going to recall data from a STACK REGISTER), and finally press the "X" key to indicate the letter "X". You'll see "08 ARCL X" in the view window if you did all of that correctly. In order to write the next line of the program which will allow you to see the display constructed in steps 7 and 8 of the program when it is run, press **SHIFT**, then the **VIEW** key which is the SHIFTed function of the R/S key. Now press the **ALPHA** key. You will see "09 AVIEW".

Writing step 10 is simple. Just press the **R/S** key and you'll see "10 STOP" in the view window. The last program line to write requires first pressing the **SHIFT** key, then the **GTO** key, then **ALPHA**, next spell out the GLOBAL LABEL for this program (SUM1), and finally press **ALPHA** for the last time. The program is now complete. Press **PRGM** to get the computer out of a program writing mode.

Following the suggestions listed in Figure 5.8. for keying in a program into your computer, you are now ready (Step 6 of Figure 5.8.) to execute the PACKING instruction. You do this by first pressing the **SHIFT** key, then **GTO**, then pressing the **decimal** key **twice**. You should see "PACKING" displayed in the view window for a couple of seconds before the contents of the X STACK REGISTER is once again displayed. It will display zeros, of course, since all registers and memory locations were cleared at the beginning of this exercise, as indicated when you saw the MEMORY LOST statement in the view window.

Section 6.1.2. Testing the Program

You can verify that the program SUM1 is in your computer by reviewing the GLOBAL LABELs of programs in the MAIN MEMORY. Press the **SHIFT** key, then **CATALOG** (which is the SHIFTed function of the ENTER key), then the key numbered "1". You will see rapidly displayed the GLOBAL LABEL SUM1, then an indication of the remaining unfilled registers in the MAIN MEMORY, and finally the contents of the X STACK REGISTER again. Had there been other programs in the MAIN MEMORY, you would have seen their GLOBAL LABELs displayed in sequence.

If you had difficulty reading this information because it was displayed so quickly in the CATALOG 1 review, you have several options, as described in Section 3.4.. One is to stop the display sequence at any point by pressing the R/S key. Pressing it a second time re-establishes the CATALOG review. Rather than pressing R/S a second time, pressing the SST ("single step") function key advances the display a step at a time under your control after SHIFT CATALOG 1 has been keyed.

It is now time to see if the SUM1 program runs correctly. According to step 7 in Figure 5.5., it is recommended you first test the program SUM1 for logic flow. Easy to do. First press the **XEQ** key, then **ALPHA**, next spell out the GLOBAL LABEL of the program ("SUM1"), then press **ALPHA**. You should see the first of the

data entry prompts "ENTER N1". Press any **numbered** key, it doesn't matter which one since we're checking only for correct program flow at this stage, then enter that number by pressing **R/S**. After the second data entry prompt "ENTER N2" is displayed, enter a **second number** and press **R/S** again. If you see "N1+N2= (sum of the 2 numbers you entered)", everything is fine.

Next press the **R/S** key to check the last step of program flow, that is, to see if the program goes back to its own GLOBAL LABEL. If you see "ENTER N1", you know the flow of your program corresponds to that shown in Figure 6.1. You don't see the GLOBAL LABEL displayed, of course, since that is the address from which the program is initiated, not a displayed phrase. For the program outlined in Figure 6.1., the alpha prompt "ENTER N1" is the first message seen once the SUM1 GLOBAL LABEL has been addressed.

As a final check, see if the program makes calculations accurately, as suggested in Figure 5.5. Now that "ENTER N1" is in the view window, enter a **number** (using R/S) and remember what it was. Enter **another number** you can remember in response to "ENTER N2", and check if the answer displayed is correct. If so, you've written your first successful program for the HP-41 computer! This program will now remain in the computer's MAIN MEMORY indefinitely, even when the machine has been turned off. Your using the SUM1 program again, or any other stored program, is a simple matter of first turning the computer on, then performing the keystrokes XEQ ALPHA (name of program's GLOBAL LABEL) ALPHA.

Section 6.1.3. Program Analysis and Review

Before abandoning the simple program SUM1 outlined in Figure 6.1., it will be useful to examine what happens to the contents of the STACK REGISTERS when this program runs. It is in the STACK REGISTERS, after all, where calculations are made. The program is just a mechanism for controlling how data went into it and how they were manipulated there. The program "SUM1" didn't do anything fundamentally different from what you did with individual keyboard entries when you went through the exercise shown in Figure 4.2. for adding 2 numbers. The program you have just written made very quickly and automatically the steps you had to do manually in the earlier exercise. That's not a small thing, but it is all the program did nonetheless.

In running the SUM1 program, when you responded to the data prompt "ENTER N1" by keying a number and then pressing R/S, you placed N1 in the X STACK REGISTER. When you responded to the alpha prompt "ENTER N2" and pressed R/S, you entered N2 into the X STACK REGISTER and "raised the stack" to place N1 in the Y STACK REGISTER. The program then automatically combined the contents of the X and Y STACK REGISTERs arithmetically in the way you designated at step 6 in the program ("+"), "dropped the stack" and placed the sum of N1+N2 in the X STACK REGISTER. This is exactly what you did manually in working through the example in Figure 4.2. Your computer program SUM1 then combined the contents of the ALPHA REGISTER (which at step 7 was "N1+N2=") with the contents of the X STACK REGISTER (which was converted to an alpha statement by the instruction listed in program line 08), and displayed the entire alpha string in the view window. You were really seeing the contents of the ALPHA REGISTER, of course. When you pressed R/S, the program followed your instruction to seek the GLOBAL LABEL "SUM1" to start the program over again.

Just the same as when you sought the GLOBAL LABEL "SUM 1" in the very beginning of testing the program by the keystrokes XEQ ALPHA SUM1 ALPHA, the program was initialized and stopped its execution when it got to the first data

prompt. It was the program instruction at line 03 ("PROMPT") which held the program flow at this step until you entered data and pressed R/S to cause the program to run again. Something quite similar occurred also at Step 5.

There is one last thing to do with the SUM1 program while it's handy and you are familiar with it. You will benefit greatly in future program development and use by knowing how to enter programs at points other than their GLOBAL LABEL, and knowing how to move around in them for editing and correcting errors. You already know how to get into a program through its GLOBAL LABEL (key "XEQ ALPHA (spell the GLOBAL LABEL) ALPHA). But, you can display any line in a program you want to just from the keyboard, as you'll see next. Also, you can go automatically to any program LOCAL LABEL directly from the keyboard, which you'll learn how to do when we get to that section later.

Using SUM1 for experimentation, go to line 09 (for example) directly from the keyboard. First turn the computer **OFF**, then **ON** again to be sure you're on the keyboard, not in a PRGM mode. Just pressing PRGM accomplishes the same thing, if you are, in fact, in the PRGM mode. First press **SHIFT**, then **GTO**, then press the **decimal** key once, and in response to the prompt "GTO._ _ _", key "009". The view window will go back to display the earlier contents of the X STACK REGISTER, whatever that was. How do you know your have successfully located program line 09? Press **PRGM** to see that your keyboard instructions have indeed taken you to the desired program line.

While the program line 09 ("09 AVIEW") is being displayed is a good time to show how to move around in the program while the computer is in a PRGM mode. Press the **SST** key (which is an abbreviation for "single step") once, and note the program steps ahead one line to display now "10 STOP". Press the **SST** key once again to see "11 GTO SUM1", the last line you wrote for the program. Any bets as to what you'll see if you press SST again? Press **SST** to see a line you didn't key in directly from the keyboard. You generated "12 END" when you performed the PACKING instruction with "GTO.." earlier. The END program line protects your SUM1 program from either being inadvertently overwritten, or erased when you enter other programs into the MAIN MEMORY.

Press **SST** once again. You see the first line of the program (that is, its GLOBAL LABEL) because you have reviewed the program SUM1 as though it were written on a complete loop of paper. Going from the last step in the program ("12 END") with SST took you to the next step in the program which appears on the loop as the first line (the GLOBAL LABEL) of the program. You would have returned to the first line in this program after having pressed SST at step 12, even if there had been other programs (with different GLOBAL LABELS, of course) in the MAIN MEMORY. Each program could have been reviewed autonomously in the same fashion once you were reviewing or executing lines in it. Section 6.5. presents a model to help you visualize to processes of locating GLOBAL LABELS and reviewing program lines.

While you are still looking at the program line which contains the statement of the GLOBAL LABEL ("SUM1"), Press **SST twice** more to advance the view window to display the third line of the program ("03 PROMPT"). Now for something different. Press **SHIFT**, then press the **SST** key which has now, of course, been converted to execute its SHIFTed function BST. Its not hard to remember that BST stands for "back step" since you now see displayed the immediately preceding program line, line 02 ("02 ENTER N1"). Pressing **SHIFT** again, then **BST** to take you one more step back, so you now see the first line of the program again. You are able to review any program in the MAIN MEMORY line-by-line with either forward (SST) or backward (BST) steps by first entering into the program from the keyboard using XEQ and the appropriate GLOBAL LABEL, then pressing PRGM. You may want to try these maneuvers later when you have more than 1 program in the MAIN MEMORY and

after you've read Section 6.6.

The program SUM1 has been useful for introducing some of the basic techniques for data entry, storage in the STACK REGISTERS, and displaying a final answer, but it's outlived its usefulness. It will now be erased, and another similar program will be written, but one with different features. You know from the instructions presented at the beginning of this exercise how to clear your computer of programs by generating the MEMORY LOST statement. Clear SUM1 from the MAIN MEMORY using a different technique, though, one which is used just to clear a single program by addressing its GLOBAL LABEL. First press **PRGM** to get back to the keyboard if the computer is in the program writing mode, PRGM. Press **XEQ**, then **ALPHA**, spell "**CLP**" (to indicate you are going to "clear a program"), then press **ALPHA**. You should see "CLP _ " in the view window which is prompting you to designate the program you want to clear. Since the program you want to clear has an alpha statement ("SUM1") as its GLOBAL LABEL, you need to respond to the prompt with this alpha statement. To do this, press **ALPHA**, spell "**SUM1**", then press **ALPHA**. After a short pause, you will see PACKING, and after a second or so more, once again see the contents of the X STACK REGISTER. The program SUM1 no longer exists.

You can verify your having eliminated successfully the SUM1 program by reviewing the GLOBAL LABELs of programs in the MAIN MEMORY using the CATALOG 1 review technique. Since SUM1 was the only program you have written so far, and have now eliminated it, you'll see no GLOBAL LABELS displayed in this review. Now, on to writing programs a little more complicated and interesting.

Section 6.2. SUM2 and SUM3

Figure 6.2. outlines the program SUM2 which is similar to the SUM1 program in that it adds 2 numbers using the STACK REGISTERS. It is different by having features which control for digit display, and tones to signal data input and display. It is also different from SUM1 in pausing to display an answer, rather than having the program stop at that point. The purpose of presenting SUM2 is to show how to add these features to programs of your own.

The GLOBAL LABEL SUM2 is keyed into the computer in a similar way to that of the SUM1 program. First turn the computer on, then press **SHIFT GTO..** and when the display is stable after briefly showing PACKING, press the **PRGM** toggle to set the machine to receive the program lines you are about to write. The keystrokes **SHIFT LBL ALPHA SUM2 ALPHA** write the GLOBAL LABEL as line 01 of the program, and "01 LBL SUM2" will be displayed. **SHIFT FIX 2** defines line 02 which controls the decimal display in the view window to show numbers with 2 digits to the right of the decimal. Line 03 is written by **SHIFT BEEP**.

Lines 04, 05, 07 to 09, 12 to 14 and 16 are written in the same way as were corresponding instructions in SUM1. Lines 06, 10 and 11 are written by **XEQ ALPHA** spell **TONE**, then **ALPHA**, and then responding to the ALPHA prompt "TONE _" by pressing the **numbered** key from 0 to 9 to indicate which tone you want to hear at each of these points in the program. The instruction at line 15 will cause the displayed answer to remain in the view window for only a moment or two before program flow is directed to the program's GLOBAL LABEL to start the next calculation. Line 15 is written by **XEQ ALPHA** spell PSE then **ALPHA**. When you get tired of having to spell TONE, remember you can assign a USER function for this operation, as you may have done earlier for the PROMPT function.

Although the SUM2 program does nothing computationally different from SUM1, it is a more interesting program because of its added features. Once you have the lines for SUM2 keyed into your computer, press **PRGM** to return to the keyboard

Figure 6.2.

SUM2

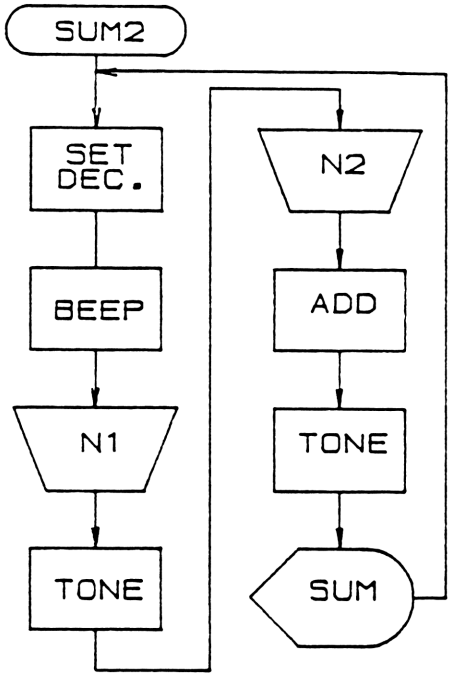
Problem Statement:

This program is designed to add 2 numbers using display and signal features.

Equation:

(Same as for Figure 6.1.(Sum1))

Flow Diagram:



Program Listing:

```
01 LBL "SUM2"
02 FIX 2
03 BEEP
04 "N1?"
05 PROMPT
06 TONE 5
07 "N2?"
08 PROMPT
09 +
10 TONE 9
11 TONE 9
12 "N1+N2="
13 ARCL X
14 AVIEW
15 PSE
16 GTO "SUM2"
17 END
```

mode, then press **SHIFT GTO..** to secure SUM2 into your machine. Review **CATALOG 1** to be sure the SUM2 GLOBAL LABEL is listed in the MAIN MEMORY. Next, run the SUM2 program (**XEQ ALPHA SUM2 ALPHA**) to see and hear its features, and test it for logic flow and calculation accuracy.

Execution of the SUM1 and SUM2 programs manipulated the STACK REGISTERS in an identical way to how you used them with key entries directly at the keyboard for the exercise in Figure 4.2. You might easily imagine how similar programs could be written to perform other arithmetic and mathematical operations, even those involving more than 2 numbers (Figures 4.3. and 4.4.). It is suggested you take a few minutes to write those programs now using the programming features of the SUM2 program. The benefit will be in gained confidence and experience for keying program lines, even though the programs themselves are simple. These new programs will be easy to check for logic flow and calculation accuracy, as suggested in Figure 5.5.

Figure 6.3.

SUM3

Problem Statement:

This program is designed to add 4 numbers, N1 to N4, using the PRIMARY STORAGE REGISTERS, then display their sum.

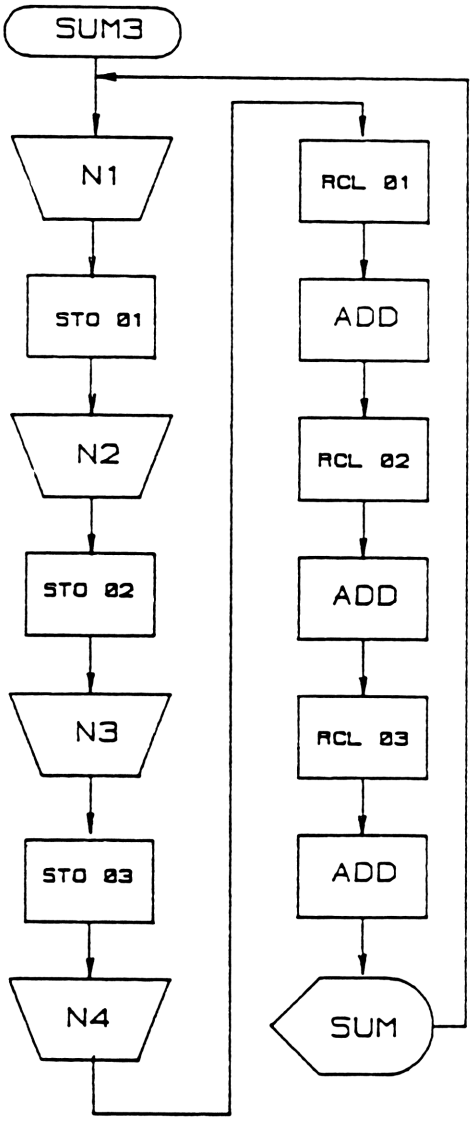
Equation:

$$SUM = N1 + N2 + N3 + N4$$

where:

N1 to N4 = numbers 1 to 4

Flow Diagram:



Program Listing:

```
01 LBL "SUM3"
02 FIX 2
03 BEEP
04 "N1?"
05 PROMPT
06 STO 01
07 TONE 5
08 "N2?"
09 PROMPT
10 STO 02
11 TONE 5
12 "N3?"
13 PROMPT
14 STO 03
15 TONE 5
16 "N4?"
17 PROMPT
18 RCL 01
19 +
20 RCL 02
21 +
22 RCL 03
23 +
24 TONE 9
25 TONE 9
26 "SUM="
27 ARCL X
28 AVIEW
29 PSE
30 GTO "SUM3"
31 END
```

If it is clear how the SUM1 and SUM2 programs deal with 2 numbers arithmetically, and if you understand how the STACK REGISTERS are manipulated in a computational sequence, then it should be staightforward to write a program to complete automatically a calculation as involved as that shown in Figures 4.5. and 4.6. One design for such a program would be to follow exactly the sequence of operations listed in Figure 4.6. to raise and lower data in STACK REGISTERS as each new piece of information is keyed. Another way to perform this calculation would be to take advantage of the opportunities your computer offers for storing entered data in the PRIMARY STORAGE REGISTERS before making a calculation, as you had introduced in Section 3.3. The program SUM3 shown in Figure 6.3. illustrates this technique for the simple process of adding 4 numbers.

None of the program logic in SUM3 is new. Steps are keyed into the computer in the same way as you have in earlier exercises. SUM3, though, handles data in a different way than seen before. Rather than place sequentially entered data into the STACK REGISTERS, each data entry is stored in a correspondingly numbered PRIMARY STORAGE REGISTER. N1 was stored in REG 01, N2 in REG 02, etc. Each data entry could have been stored just as well in any of the 99 PRIMARY STORAGE REGISTERs, or (using the INDIRECT ADDRESSING technique described in Chapter 9) in any of the EXTENDED DATA STORAGE REGISTERS.

After all data have been entered with N4 now in the X STACK REGISTER at program line 17, previously entered data are recalled in their numerical order from storage and added (lines 18 to 23) to the growing sum in the X STACK REGISTER. By line 24, all data are added, and their sum is brought to display (lines 26 to 28), held there momentarily (line 29), then program flow is directed to the program's GLOBAL LABEL to start data entry, storage and addition of the 4 numbers in the next series. The BEEP TONEs (line 03) provide an audible reminder that the program is starting over again. When you've keyed all of the steps for SUM3, press **PRGM**, then **GTO..** to secure this new program in the MAIN MEMORY. Next, try the program for logic flow and accuracy using a sample problem.

The programs SUM1 and SUM2 handle well the process of adding only 2 numbers, and SUM3 shows how 4 data entries can be added. It's obvious how the SUM3 program could be modified to add more than 4 numbers. Simply increasing the number of data prompts and program lines to store the new numbers in the series are basically all that is required. The type of program construction shown in SUM3, however, obligates the user to know in advance how many numbers there will be in the series for addition. SUM3, for example, can handle only 4 data entries. The program SUM4 described in Figure 6.4. shows how an indefinite number of data entries can be added using the STO+ nn function introduced in Section 3.1.6.

Section 6.3. SUM4

The last program in this sequence (SUM4) is constructed, as shown in Figure 6.4., to add an indefinite number of data entries, rather than add just 2 (SUM1 and SUM2) or 4 (SUM3) numbers, as have the other examples presented so far. SUM4 also introduces 2 new programming concepts: the use of the conditional statement "X=0?" and the use of LOCAL LABELS. The flow diagram for SUM4 omits elements showing where TONEs and other minor non-operational steps are in the program. Future program flow diagrams will not show these steps either.

The equation for SUM4 is simple enough. It indicates that the sum will be calculated for numbers in a series with an unspecified length. When you run this program, you will enter numbers which will be added in a sequence until you designate with a code number you have finished the data entry series. Zero is used for that code number in this example.

Figure 6.4.

SUM4

Problem Statement:

This program is designed to add a series of numbers of an unspecified length, then display their sum.

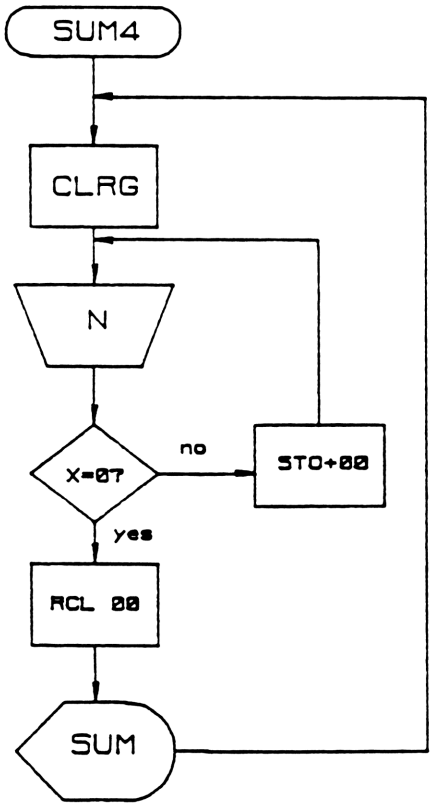
Equation:

$$SUM = N1 + N2 \ldots + Nn$$

where:

N1, N2, . . . Nn = sequentially entered numbers

Flow Diagram:



Program Listing:

```
01 LBL "SUM4"
02 FIX 2
03 BEEP
04 CLRG
05 LBL 00
06 TONE 5
07 "ENTER N"
08 PROMPT
09 X=0?
10 GTO 01
11 STO+ 00
12 GTO 00
13 LBL 01
14 RCL 00
15 TONE 9
16 TONE 9
17 "SUM="
18 ARCL X
19 AVIEW
20 PSE
21 GTO "SUM4"
22 END
```

The flow diagram in Figure 6.4. shows only one of many ways to construct a program for the HP-41 to add numbers in an indefinite series. It indicates that once the program SUM4 is started, the prompt statement for the data entry ("ENTER N") is the first thing to be displayed. Once the user has keyed a number, then pressed R/S, the program first determines if this entered number is zero by using the conditional test "X=0?" at line 09. As shown in the flow diagram, if it is not zero, line 10 is

skipped and the number is placed in REG 00 additively using STO + 00 (line 11). If an entered number is a zero, the instruction at line 10 is executed and (at line 14) the contents of REG 00 is recalled and displayed as the sum of all the numbers previously entered. Program execution pauses momentarily, then SUM4 is started over again, as announced by the BEEP instruction at line 03.

The instruction at step 4 clears all of the computer's registers when the program is started. Certainly, taking the program from line 12 back to its GLOBAL LABEL for the entry of the 2nd, 3rd and subsequent numbers in the sequence to be added would not be a good idea, because the PRIMARY STORAGE REGISTER 00 would be cleared each time the program completes this loop. Generating a LOCAL LABEL at step 5 (LBL 00), though, provides an internal address in the program to complete the loop for the addition of subsequent data entries. Lines 06 to 08 are familiar from the previous programs you have written.

The conditional statement at line 09 is easily written. Press **SHIFT** then the "/" key to get the "X=0?" expression, which is the SHIFTed function of that key. What this statement does in the program deserves some explanation. The expression "X=0?" is a conditional statement, as are "X>Y?", "X<=Y?" and "X=Y?" which appear as other SHIFTed functions on the keyboard. There are other conditional statements you can use too, but you'll see these later. Conditional statements simply ask a question, and then direct flow in the program depending on the answer to that question. It is important to remember that all conditional statements operate in the same way. They will "DO" the immediate next line of the program if the answer to the statement is "TRUE", but will skip the immediate next line in the program if the answer is "NOT TRUE".

At line 09 of the SUM4 program, the conditional statement asks "is the contents of the X STACK REGISTER equal to zero?" If it is, the program's flow is directed (by GTO 01 at step 10) to the LOCAL LABEL 01 (at step 13) which begins the display of the contents of the PRIMARY STORAGE REGISTER 00. There would be a zero in the X STACK REGISTER, of course, had you entered one from the keyboard in response to the data prompt at step 7. If you had entered any positive or negative number in response to "ENTER N", then the instruction in step 10 of the program (GTO 01) would be skipped, and the line after it (ST + 00) would be executed (adding your keyed number to the sum in REG 00). The next program line (GTO 00) would direct program flow back to the LOCAL LABEL 00 which would sound the TONE 5, then ask for the next number in the sequence.

There are several important concepts to gain from reviewing the SUM4 program. It shows a simple case in which a conditional statement is used to direct program flow according to a characteristic of entered data. It shows also how LOCAL LABELs are placed in a program to provide entry points for program flow. The organization of SUM4 is similar to the general program structure shown in Figure 5.3. which illustrates where LOCAL LABELS are placed in programs. SUM4 shows what LOCAL LABELs do in a program. It is hoped that even with such a simple example as SUM4, many ideas will soon come to mind about how the combined effects of conditional statements and LOCAL LABELs can be used to advantage in programs of your own interest.

After you have keyed all of the steps for SUM4 into your computer, check the program's integrity by first reviewing it for logic flow, then use the program to add a sequence of numbers to see if it did it correctly. Remember you terminate the data entry sequence by entering a zero. If there is a problem either for program flow, or with its accuracy of calculation, press **PRGM** and use **SST** and/or **BST** to find the erroneous program statement, erase it, and enter the correct one. If all is well with your new program, though, use **SHIFT GTO..** to add SUM4 to your inventory. Before you proceed to the next section, try SHIFT CATALOG 1 again to see displayed in sequence the GLOBAL LABELs of the programs you now have in your computer's MAIN MEMORY.

Section 6.4. Conditional Tests

The previous section introduced how the conditional test X=0? could be used to control program flow in SUM4 either to direct the program to LOCAL LABELS for the entry of additional data, or to make a calculation. It's probably obvious not only how the other conditional tests X=Y?, X≤Y? and X>Y? could be used in a similar way, but also how they would be keyed into a program. As for keying the X=0? conditional statement, the SHIFTed functions of the appropriately labeled keys would be selected to write these program lines. They would function in a program in exactly the same way as the X=0? conditional test, that is, if the contents of the X STACK REGISTER is tested to be true in the context of the conditional statement, then the next line of the program is executed, but skipped if the conditional test is untrue.

There are many other conditional tests which are invaluable in programming, but these are keyed in different ways than simply using a SHIFTed keyboard function. Some of these conditional tests compare the contents of the X STACK REGISTER in different ways, and others are used for testing the status of FLAGS. Conditional testing for FLAGS is described in Chapter 8. Additional conditional tests using the STACK REGISTERS are discussed next.

The other conditional tests, and those for which there are no SHIFTed key function labels on the keyboard are: X>0?, X<Y?, X<0?, X≤0?, X≠Y? and X≠0? and tests comparing the contents of the X STACK REGISTER to that of other registers. These function in a program in a similar way to the conditional tests for which there are SHIFTed keyboard labels, but they are written into a program differently. These are keyed by using an XEQ function and depends on the definitions of the keyboard when it is an ALPHA SHIFT mode (see Figure 2.4.). For example, to write the program line for the conditional test X<0?, first turn the computer **ON** and press **PRGM**. Next press **XEQ**, then **ALPHA**, next press the key numbered **6** to get the letter "X". Then press **SHIFT**, next the **COS** key (in order to write its ALPHA SHIFTed function "<"), then **SHIFT**, next press **zero**, then press the key with the symbol **?** and finally **ALPHA**. If you followed this sequence correctly, you will see "X<0?" as a program line.

In running a program in which X<0? is a statement, when program flow comes to this conditional test, the next line of the program is executed if the numerical content of the X STACK REGISTER is less than zero. The line immediately following this conditional test would be skipped if the content of the X STACK REGISTER contained zero or any number larger than zero, and program flow would begin with the very next line.

The conditional tests X=Y? and X≠Y? are used not only to test the numerical contents of the X and Y STACK REGISTERS, but used also to compare ALPHA statements and strings stored in these registers. If the ALPHA phrase stored in the X STACK REGISTER is exactly the same as the one stored in the Y STACK REGISTER, then the immediately next program line after X=Y? would be executed in program flow, for example. Being able to compare not only numbers but also ALPHA statements in this way with conditional tests offers valuable alternatives in program design and execution, as you'll see in the next couple of chapters. Other conditional tests like X<Y? and X≤Y?, of course, have no significance when attempting to compare ALPHA statements in the X and Y STACK REGISTERS.

The first sentence in Section 5.1. defined a computer program as a list of instructions to bring about either the solution of a numerical problem, or to reach a desired goal. the programs SUM1, SUM2, SUM3 and SUM4 are examples of algorithms (lists of step-by-step instructions) constructed to solve numerical problems, for them the simple task of adding 2 or more numbers. The SUM4 program was more interesting than the others in that it used a conditional test (X=0?) to direct program

Figure 6.5.

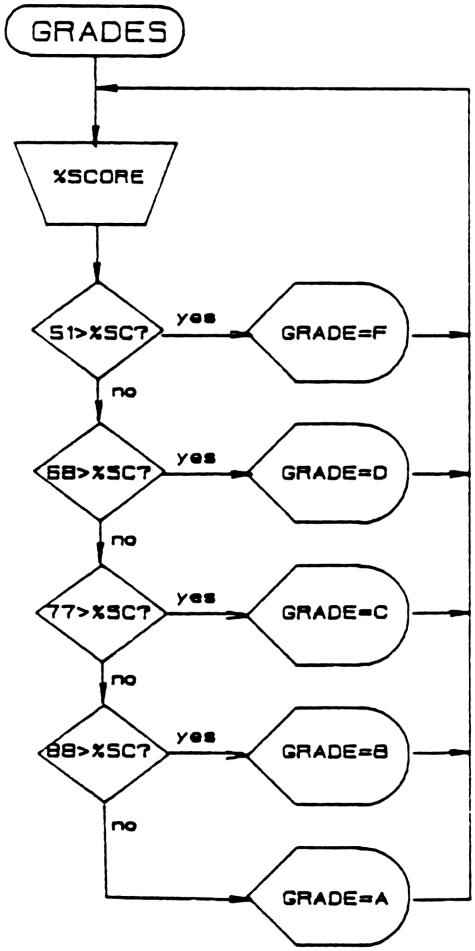
Grades

Problem Statement:

This program is designed to select and display a letter grade for an examination based on a student's percent score using the criteria:

Minimum % Grade	Letter Grade
88	A
77	B
68	C
51	D
50 or less	F

Flow Diagram:



Program Listing:

```
01 LBL "GRADES"
02 BEEP
03 "ENTER %SCORE"
04 PROMPT
05 STO 00
06 51
07 X>Y?
08 GTO 00
09 RCL 00
10 68
11 X>Y?
12 GTO 01
13 RCL 00
14 77
15 X>Y?
16 GTO 02
17 RCL 00
18 88
19 X>Y?
20 GTO 03
21 TONE 9
22 "GRADE=A"
23 GTO 04
24 LBL 00
25 TONE 9
26 "GRADE=F"
27 GTO 04
28 LBL 01
29 TONE 9
30 "GRADE=D"
31 GTO 04
32 LBL 02
33 TONE 9
34 "GRADE=C"
35 GTO 04
36 LBL 03
37 TONE 9
38 "GRADE=B"
39 GTO 04
40 LBL 04
41 AVIEW
42 PSE
43 GTO "GRADES"
44 END
```

flow dependent on the status of entered data. Conditional tests are important in programs like this which make a numerical calculation, but they are also of use in programs constructed to reach a required goal, not to solve for something. It is in its ability to made decisions and do low-level "thinking" and sorting for you that computers generally, and your HP-41 specifically, return many fold both your financial and time investments in it.

Figure 6.5. describes a program designed to reach a goal, not make a calculation. The GRADES program operates to display a letter grade of A, B, C, D or F after a student's percent score on an exam has been entered. This program uses LOCAL LABELS in a similar way to the SUM4 program, but with an important difference. In SUM4, a LOCAL LABEL was used to return program flow to a preceding line to form a loop asking for sequential entries of data. In the GRADES program, LOCAL LABELs are used to direct program flow around instruction steps to a point further on in the program. A similar strength for both programs, though, is that they use LOCAL LABELs in conjunction with conditional tests to give flexibility and generate many options in how each program is executed.

Since you've already practiced all of the types of key entries, none of the program lines in the GRADES program will be difficult to write into your computer, and the program itself should be simple enough to follow. Once the program's GLOBAL LABEL is addressed (by XEQ ALPHA GRADES ALPHA), a tone introduces the alpha prompt for data entry ("ENTER %SCORE"). After a number has been keyed and R/S is pressed, a sequence of comparisons is made between the entered grade and that minimally required for a particular letter grade. Once a correct match is made, an appropriate statement is selected to display the student's deserved grade.

For example, whatever percent score is entered in response to the alpha prompt at step 3, it is stored in REG 00. Steps 6 and 7 determine if the entered score is minimally adequate to receive a grade of D, and if it is not, the program branches at step 8 to LBL 00 where a grade of F is displayed. After a pause (step 42), the program returns to its own GLOBAL LABEL to start the comparison series for the percent grade of the next student. If, however, the percent score of the first student is greater than 50 (so a grade better than F is deserved), program flow continues to step 9 where a second comparison (steps 10 and 11) determine if the student will receive a grade of D.

If the conditional test at step 11 is determined to be true, the next line of the program is executed which results in presenting the letter grade "D" as a display in the view window. Similar comparisons are made through the remainder of the program to match the display of a grade for the appropriate percent score criterion level.

The GRADES program would be useful for an instructor who always used the same criteria for grading exams. If the cut-off points for grades were different for the next exam, though, the GRADES program would have to be rewritten at lines 06, 10, 14, and 18). This could become a laborious process if there were many different exams to score. The GRADES program would be more useful if grading criteria were entered in response to data prompts at the beginning of the program. The reader is encouraged to modify the GRADES program in this way and then later compare this new program with the EXAM program introduced in Chapter 9.

Conditional tests are valuable for directing program flow depending on a single condition in a program, as shown in the SUM4 and GRADES programs. Using a sequence of conditional tests gives considerable additional power for making decisions in a program. This technique is introduced next. Figure 6.6. describes the BUMP program which contains features of the SUM4 and GRADES program. It combines the use of conditional tests which made a decision leading to a goal with others which lead to a numerical calculation. The program listing and trial solutions for BUMP are shown in Figure 6.7. The arrangement of conditional tests which provide the sorting routine in this program can be expanded, of course, to allow Mr. Crunch to select the closer 2 of more than 3 estimates, or rearranged to allow him to choose the average of

Figure 6.6.

BUMP

Problem Statement:

Charley Crunch manages an insurance agency which specializes in automobile collision coverage. He's learned that to obtain a reliable indicator of a city-wide charge for a particular type of repair, he needs to take an average of the closest 2 of 3 estimates (not necessarily the 2 lowest) he gets from different shops. The BUMP program is designed to make this calculation for him after he has entered the 3 estimates.

Flow Diagram:

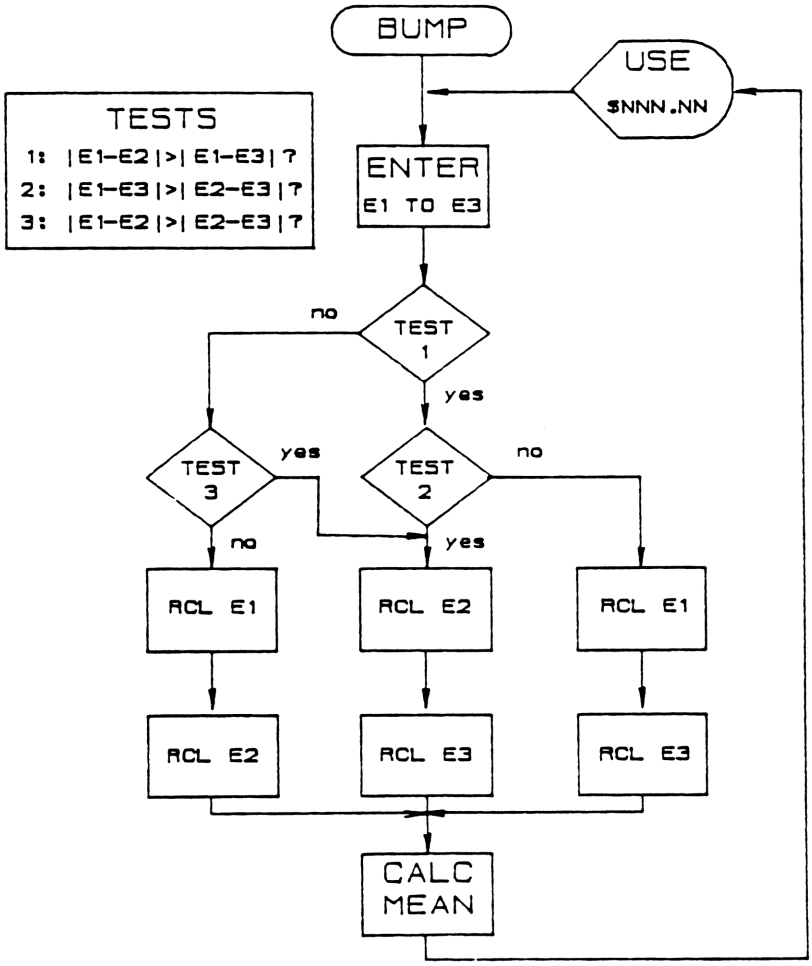


Figure 6.7.

Program Listing: BUMP

Program Listing:

```
01 LBL "BUMP"
02 BEEP
03 FIX 2
04 "ENTER E1"
05 PROMPT
06 STO 01
07 TONE 5
08 "ENTER E2
09 PROMPT
10 STO 02
11 TONE 5
12 "ENTER E3"
13 PROMPT
14 STO 03
15 RCL 01
16 RCL 02
17 -
18 ABS
19 STO 04
20 RCL 02
21 RCL 03
22 -
23 ABS
24 STO 05
25 RCL 01
26 RCL 03
27 -
28 ABS
29 STO 06
30 RCL 04
31 X>Y?
32 GTO 00
33 RCL 05
34 RCL 04
35 X>Y?
36 GTO 01
37 RCL 01
38 RCL 02
39 GTO 02
40 LBL 00
41 RCL 05
42 RCL 06
43 X>Y?
44 GTO 01
45 RCL 01
46 RCL 03
47 GTO 02
48 LBL 01
49 RCL 02
50 RCL 03
51 LBL 02
52 +
53 2
54 /
55 TONE 9
56 "USE S"
57 ARCL X
58 AVIEW
59 PSE
60 PSE
61 GTO "BUMP"
62 END
```

Solutions:

	Type of Repair		
Estimate	A	B	C
1	$110.50	$340.10	$964.80
2	180.78	369.50	1032.10
3	140.39	282.30	985.85
average of 2 closest	125.45	354.80	975.33

the 3 highest of however many estimates he wants to get, etc. It would be good practice to modify the BUMP program in one of these ways, or in some other way you might find interesting. Once you have keyed the BUMP program into the computer's MAIN MEMORY, test it with the sample problems shown at the bottom of Figure 6.7.

Because the logic flow in the BUMP program is more complex than in the others presented so far, a brief explanation of how Mr. Crunch designed this program is in order. The basic task of this program is to select the 2 closest of 3 numbers. These numbers are first requested as keyboard entries (E1 to E3), and then stored in REGs 01, 02 and 03 in the first 14 lines of the program. The next job is to start a series of comparisons among their differences which will serve as the basis for determining which 2 will be used for an average.

Instructions at lines 15 and 16 recall the first and second numbers from storage, and determine (at line 17) the difference between them. Since the critical comparison in this program is which 2 numbers are the closest together, not which one is larger than the other, only the absolute difference between them (not the sign of their difference) needs to be considered. The instruction at line 18 calculates the absolute difference between the first and second entries. The instruction is keyed as XEQ ALPHA spell ABS ALPHA, and uses one of the built-in functions (ABS) of CATALOG 3 (see Figure 3.4.). This difference is then stored in REG 04. Instructions in lines 20 to 24 calculate the absolute difference between the second and third data entries, and stored it in REG 05. A similar calculation is made for the first and third entries in lines 25 to 29.

Starting at program line 30, each of the differences between data entries is withdrawn from their locations in the PRIMARY STORAGE REGISTERS and compared using the conditional statement X>Y?. The flow diagram in Figure 6.6. shows how the final selection is made for the 2 pairs of numbers which are the closest. Once this decision is reached, their average is calculated (lines 51 to 54), and an appropriate display is constructed to inform Mr. Crunch of the best estimate of the city-wide repair cost he is looking for. Reviewing the BUMP program in detail is a good test to determine whether you understand data entry, STO and RCL functions, as well as have insight into how conditional tests direct program flow to LOCAL LABELS. It's important to have a sound working knowledge of these operations.

Section 6.5. Where Is It and How Do I Get To It?

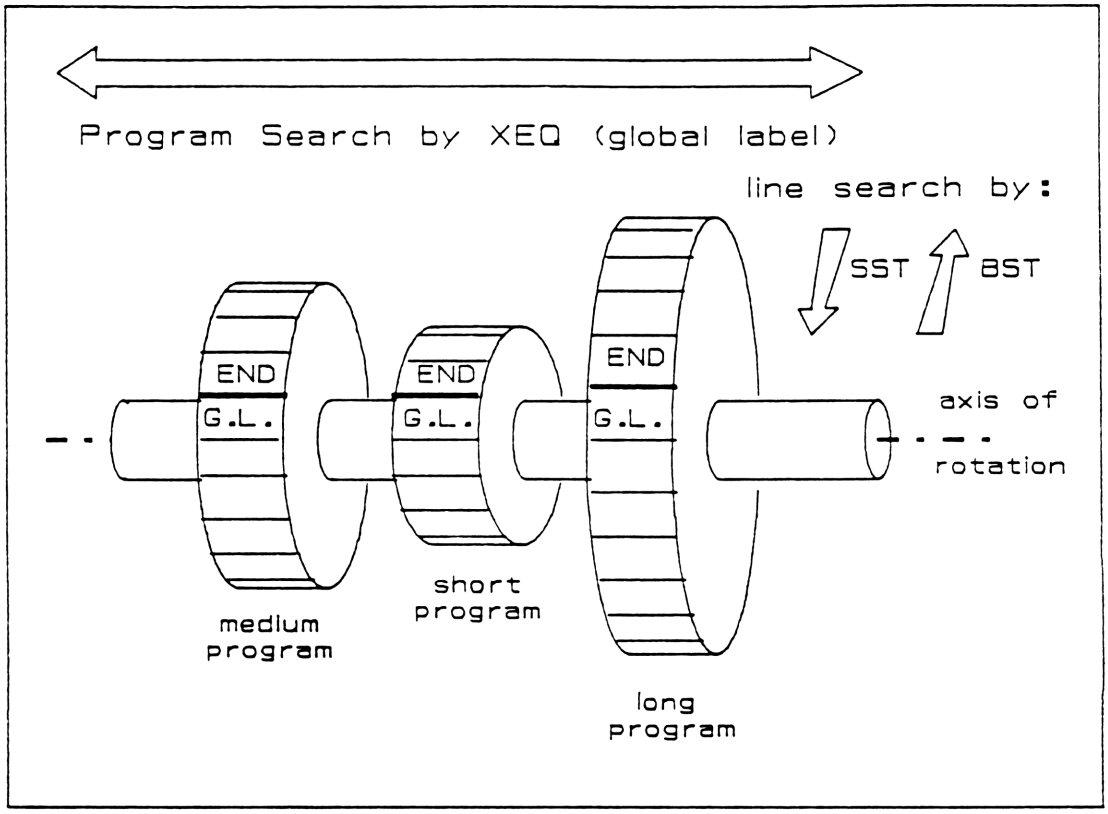
You have already seen some of the complexity with which the HP-41 computer stores data, operates upon it mathematically and makes decisions with it. You've also seen how it displays answers and decisions at the end of a program. It is confusing especially at first to know what is going where in the process of data entry and knowing what is where in the process of data storage at any time during a program's execution. The simple model shown in Figure 6.8. might help. Characteristic of models, they do not show realistically how things operate, but they are valuable in presenting a simple scheme for thinking about how a system works.

Imagine that each program you write is represented by a wheel which can be fitted onto a spindle or an axle around which it can rotate. If you have only 1 program in your computer, there is (in the model) only 1 wheel on the spindle. If you have stored 3 programs, there are 3 wheels on the spindle. There can be as many wheels on the spindle (there can be as many programs in your computer) as you have space on the spindle (as you have room in the MAIN MEMORY).

Each program must have a GLOBAL LABEL (symbolized as "G.L." in Figure 6.8.) as its first line and have an END statement as its last line. You discovered in the analysis of the program SUM1 (Section 6.1.3.) that you could step through the program line by line either forward (for example, step 10, step 11, step 12, etc.) using SST or backward (for example, step 12, step 11, step 10, etc.) using BST. This line by line review of the program is symbolized in the model by rotating the wheel, which represents a single program, either forward or backward and reading around its rim each of the MAIN MEMORY storage spaces in which individual program lines are listed.

Figure 6.8.

Where Is It? (A Model)



You discovered also in the analysis of SUM1, that were you looking at the END statement in the computer's view window, then pressed SST, you next saw the program's GLOBAL LABEL listed as line 01. Similarly, if you were looking at the GLOBAL LABEL in the view window, pressing BST showed you next the END statement at the program's last line. For a short program, the END statement would appear at a lower numbered line that would the END statement for a long program. This feature is symbolized in Figure 6.8. by having programs of different length drawn as wheels with different radii. The long program is shown as a wheel with a large diameter; the short program is shown as a wheel with a small diameter. Regardless of the length of the program, however, each has a juxtapositioned GLOBAL LABEL and END statement around the rim of the wheel which represents it.

Searching for a program either to review it line by line or to operate it, though, cannot be achieved by using the SST and BST keys. They are good for reviewing program lines, but do not function in locating the GLOBAL LABEL of other programs. The XEQ key, as you have already learned, is the function which initiates locating a program by addressing its GLOBAL LABEL. You used the program SUM1, for example, by entering the instructions from the keyboard XEQ "SUM1". This operation is symbolized in the model by having the view window of the computer move along the axis of the spindle until it locates the requested GLOBAL LABEL.

Once the program is located by the XEQ "GLOBAL LABEL" commands, pressing PRGM gives you entry into that program, but not at the level of the GLOBAL LABEL. Seeking the GLOBAL LABEL of the program and displaying it when PRGM is pressed is achieved by the operation GTO."GLOBAL LABEL". You may have discovered for yourself that once the PACKING operation initiated by GTO.. is complete, there is no program line to be seen when PRGM is pressed. This would be symbolized in the model by looking at the space between adjacent wheels on the spindle. The point of presenting the model in Figure 6.8. is to help visualize the differences among the operations SST, BST, XEQ "GLOBAL LABEL", GTO."GLOBAL LABEL" and GTO..

If you understand the principles presented in these first few programs, you are ready to examine some others in the next section which will introduce more complex programming techniques. If you are a little uncertain either about some of the keystrokes shown so far, or about some of the construction details of the programs, look back over this section again in some detail before going to the next chapter. Having a good grasp of these fundamentals will make the next sections much easier to understand.

Chapter 7

Intermediate Programming Techniques

None of the programs presented so far has been mathematically challenging, nor were any of them they meant to be. They got you started, though, in seeing how programs are structured, and gave you a chance to practice some important key entry skills. You have enough background in the fundamentals of programming now to be able to write programs which handle more complicated mathematical tasks. The next section outlines a program to solve the complex equation introduced earlier (Figure 4.5.). When this program is run, it executes automatically the same step-by-step changes (Figure 4.6.) in the contents of the STACK REGISTERS performed manually to solve this equation earlier. You may want to review these operations before reading further.

Section 7.1. SOLVX

When asked earlier (Section 5.1.) for directions to the National Bank, not only did you have to think to yourself what you were going to say in your response, but also you had to decide the order in which you were going to make statements. Even if you were caught off balance initially by being asked to provide this assistance, once you had it all straight in your mind, it was easy to be of help in giving accurate directions. It is much the same when you are presented with the challenge to solve an equation, like the one, for example, shown in Figure 7.1. (this is the same equation as that in Figure 4.5). Despite seeing initially a statement of complex relationships among 6 variables (N1 to N6), once you'd thought about it for a moment, more than likely some ideas come to mind about the sequence of steps you could take to reach its solution.

It is not as important that someone else might come up with a different sequence of steps, as it is that both of you can solve the equation accurately using as few steps as necessary. Similarly, a good case can be made for the concept that the design of one computer program is just about as good as any other, as long as both do the job accurately without wasting memory space and program running time. The fact that you think about a problem solution differently than some other person is an excellent opportunity for each of you to express your individual creativity, insight and ingenuity in designing the appropriate computer program to accomplish it.

One way to approach the solution of the equation shown in Figure 7.1. would be to think, "First, I'll take the Log of N4, then the Sine of N5, add them together, square N6, divide it into that sum, then take the square root of that ratio. Next I'll divide N1 by N2, raise it to the N3 power and multiply that times the results of my earlier calculation." If you thought about the solution of X in a way similar to this, you have taken the most important step in writing your computer program to make this calculation. You've determined the general route you are going take to reach a final solution of the problem. Your part in the alliance with your computer is to decide what steps have to be taken in solving a problem and then communicate them to the machine in a language it understands. Your computer's job is to perform each step accurately, quickly and always in an identical manner.

Writing the program for your HP-41 to solve X in Figure 7.1. is not much more involved than placing into the RPN language the sequence of steps for the solution which you thought of in your own language. Making as clear a statement of sequence

Figure 7.1.

SOLVX

Problem Statement:

This program is designed to solve the following equation:

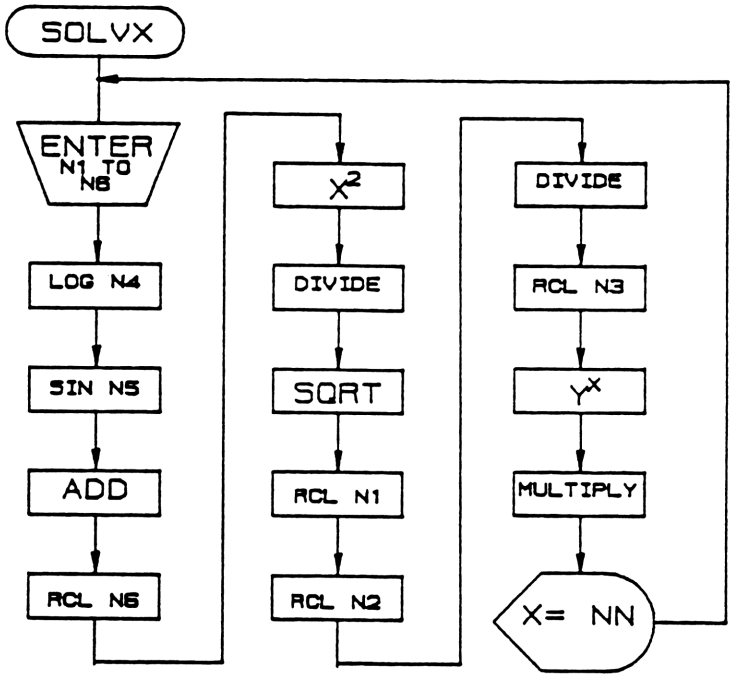
Equation:

$$X = \left(\frac{N1}{N2}\right)^{N3} \sqrt{\frac{LOG\ N4 + SIN\ N5}{(N6)^2}}$$

where:

N1 to N6 = entered data

Flow Diagram:



and intent as quoted in the previous paragraph is the heart of a good computer program, no matter what computer and what computer language is used. One way to make that statement in the language of the HP-41 series computers is shown in the program steps listed in Figure 7.2. which follow the flow diagram shown in Figure 7.1.

The first 26 steps in the SOLVX program are familiar initialization and data entry statements. Entering data for the 6 variables in this style, though, is cumbersome, and you'll see in Chapter 9 some simple procedures for doing the same job with fewer program steps using an INDIRECT ADDRESSING technique. The calculation of X begins at step 27 when N4 is recalled from its memory storage location (REG 03), its LOG is calculated, then N5 is recalled from REG 04, its SIN is calculated and they are added together at step 31. N6 is recalled from REG 05, squared

and divided into (LOG N4+SIN N5). After the square root of the calculation is taken at Step 35, N1 and N2 are recalled, and (N1/N2) is calculated, raised to the power of N3, and the current contents of the X and Y STACK REGISTERs are multiplied together to complete the calculation.

The rest of the program for the tones, the answer display and the branch back to the beginning of the program after a pause is similar to that of the earlier programs SUM1 to SUM4. The program listed in Figure 7.2. manipulates the STACK REGISTERS in the same way as you did when the calculation was made by hand using direct keyboard operations (Figure 4.6.).

Figure 7.2.

Program Listing: SOLVX

Program Listing:

```
01 LBL "SOLVX"
02 CLRG
03 BEEP
04 "N1?"
05 PROMPT
06 STO 00
07 TONE 5
08 "N2?"
09 PROMPT
10 STO 01
11 TONE 5
12 "N3?"
13 PROMPT
14 STO 02
15 TONE 5
16 "N4?"
17 PROMPT
18 STO 03
19 TONE 5
20 "N5?"
21 PROMPT
22 STO 04
23 TONE 5
24 "N6?"
25 PROMPT
26 STO 05
27 RCL 03
28 LOG
29 RCL 04
30 SIN
31 +
32 RCL 05
33 X^2
34 /
35 SQRT
36 RCL 00
37 RCL 01
38 /
39 RCL 02
40 Y^X
41 *
42 TONE 9
43 TONE 9
44 FIX 3
45 "X="
46 ARCL X
47 AVIEW
48 PSE
49 GTO "SOLVX"
50 END
```

After you have tested the SOLVX program for logic flow, test its arithmetic accuracy by going back and recalculating the sample problems shown in Figure 4.5. You will realize immediately the advantage of having written a program to solve this equation when you see how much more rapid this process is than keying in each step manually, as you did in Chapter 4. You will recognize the value of your HP-41 series computer even more when you consider how long it would take to do this calculation using only paper and pencil.

Figure 7.3.

Operations Which Cannot Be Programmed

Operation	Description	Operation	Description
CLP	Clear program	←	Erase entry
BST	Back step	SST	Single step
DEL	Delete	ASN	Assign
USER	USER toggle	SIZE	Size operation
PRGM	Program mode	GTO.	Line designator
GTO..	Packing instr.	Catalog	Catalog review
ON	Cont. power	ON	Initiate power
COPY	Copy program		

You have seen in the sample programs presented so far, the many functions of your computer which can written directly into an algorithm for automatic execution in solving an equation. There are many other functions which also can be programmed, but there are some which cannot be used in this way, but must be executed by manual keystrokes. These are shown in Figure 7.3.

Section 7.2. Subroutines

A minor restriction of the HP-41 computer is its limited memory compared to tabletop computers. So, any programming technique which makes efficient use of available memory is of special value. Using subroutines is one such technique. This procedure is introduced next.

The problem SUBS lists 3 simple equations to solve for X, Y and Z respectively, which vary as functions of the variables A to F. The flow diagram in Figure 7.4. suggests one way to pattern data input, have required calculations made and display solutions for each of the equations. The program SUBS1 listed in Figure 7.5. is a straightforward way of completing these processes. It uses no techniques which are unfamiliar to you. Data for variables A and B are keyed into the computer in response to the alpha prompts "A?" and "B?", respectively, X is calculated using equation 1, and the solution is displayed. Calculations for Y and Z follow a parallel pattern after corresponding data are entered. This program is well constructed from the perspectives of logic flow and accuracy, but it can be written using fewer program lines if the technique of constructing subroutines is used as shown in the program SUBS2.

When you keyed the program lines for SUBS1 into your computer, you probably noticed the identity among steps in lines 07 to 13, those in lines 24 to 30, and those in lines 41 to 47. You may have seen also that lines 16 to 18, 33 to 35 and 50 to 52 are identical too. The second program, SUBS2, shows how program lines which are identical can be used in subroutines executed by "XEQ" statements inserted at appropriate lines in the main body of the program. There are several points to see in understanding this technique.

In both SUBS1 and SUBS2, the program begins after the GLOBAL LABEL is addressed and progresses in the same way through the first 2 alpha prompts for data inputs A and B. Also similarly, line 06 in each program begins the calculation of equation 1, but in SUBS2, program flow is directed (by XEQ 01) to a subroutine (LBL 01) for the execution of these steps, rather than have them sequenced in the main body of the program, as occurs in SUBS1. After X is calculated in SUBS2 (lines 27 to 35), program flow is returned (by RTN) to line 8 which immediately follows the XEQ 01 statement, but is then directed (by XEQ 02 at line 09) to the second subroutine (LBL 02) at line 37.

Once program flow has progressed through the steps listed as lines 37 to 40 in LBL 02, it returns (by an RTN statement at line 41) to the main part of the program at line 10, and program flow continues to request data inputs for the variables D and E. Follow the remainder of program flow as it is directed to the same 2 subroutines (listed under LBL 01 and LBL 02) and then returns to the main body of the program in the calculation series for Y and Z.

The 2 subroutines used in SUBS2 are not only similar to one another, but also they share characteristics with all other subroutines. Each subroutine: 1. is addressed by an XEQ statement in the body of the program, 2. returns to the immediately next step in the program beyond the corresponding XEQ instruction, 3. ends with a RTN statement to redirect program flow back to the main body of the program, and 4. is a sequence of identical program instructions used in different places in the program. These 4 features are common to all subroutines.

Figure 7.4.

SUBS

Problem Statement:

This program is designed to calculate X, Y and Z as functions of the variables A, B, C, D, E and F using equations 1, 2 and 3.

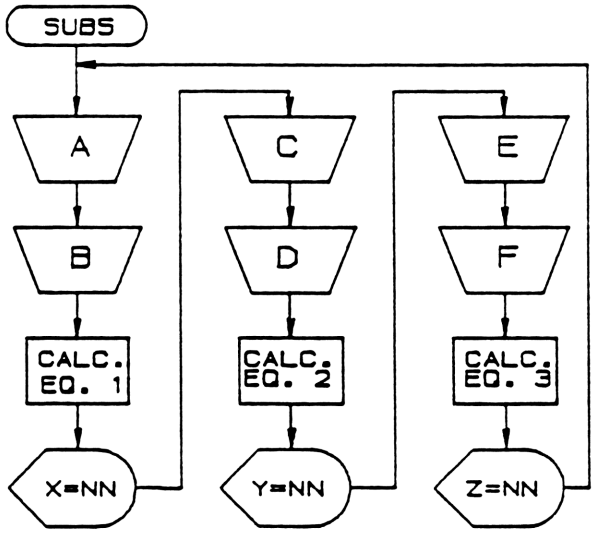
Equations:

$$X = \frac{(A + B)^2}{PI} - 2 \qquad (1)$$

$$Y = \frac{(C + D)^2}{PI} - 2 \qquad (2)$$

$$Z = \frac{(E + F)^2}{PI} - 2 \qquad (3)$$

Flow Diagram:



The RTN instruction is essential in a subroutine. For example, the statement at line 26 in SUBS2 obligates program flow to return to the GLOBAL LABEL. Subroutines listed under the LOCAL LABELS LBL 01 and LBL 02 are constructed in higher numbered program lines than step 26. They are addressed in the main body of the program only by the statement "XEQ _ _". After the steps in any subroutine have been executed, program flow must be redirected to the main body of the program using an "RTN" statement.

Were RTN not to exist at line 36 in SUBS2, for example, program flow would continue from step 34 (-) through line 35 and into the steps listed under LBL 02. This would have no rationale in the construction of the program, and calculations would not be made correctly. Each subroutine outside of the main body of the program which has been addressed with an XEQ _ _ statement needs to have, without exception, an RTN statement at its end to keep calculation sequences ordered properly. After you have keyed SUBS1 and SUBS2 into your computer, be sure to test both their logic flow and numerical accuracy, as suggested in steps 7.1. and 7.2. in Figure 5.5.

Figure 7.5.

Program Listing:
SUBS1 and SUBS2

```
01 LBL "SUBS1"
02 BEEP
03 "A?"
04 PROMPT
05 TONE 5
06 "B?"
07 PROMPT
08 +
09 X^2
10 PI
11 /
12 2
13 -
14 TONE 9
15 "X="
16 ARCL X
17 AVIEW
18 PSE
19 TONE 5
20 "C?"
21 PROMPT
22 TONE 5
23 "D?"
24 PROMPT
25 +
26 X^2
27 PI
28 /
29 2
30 -
31 TONE 9
32 "Y="
33 ARCL X
34 AVIEW
35 PSE
36 TONE 5
37 "E?"
38 PROMPT
39 TONE 5
40 "F?"
41 PROMPT
42 +
43 X^2
44 PI
45 /
46 2
47 -
48 TONE 9
49 "Z="
50 ARCL X
51 AVIEW
52 PSE
53 GTO "SUBS1"
54 END
```

```
01 LBL "SUBS2"
02 BEEP
03 "A?"
04 PROMPT
05 TONE 5
06 "B?"
07 XEQ 01
08 "X="
09 XEQ 02
10 TONE 5
11 "C?"
12 PROMPT
13 TONE 5
14 "D?"
15 XEQ 01
16 "Y="
17 XEQ 02
18 TONE 5
19 "E?"
20 PROMPT
21 TONE 5
22 "F?"
23 XEQ 01
24 "Z="
25 XEQ 02
26 GTO "SUBS2"
27 LBL 01
28 PROMPT
29 +
30 X^2
31 PI
32 /
33 2
34 -
35 TONE 9
36 RTN
37 LBL 02
38 ARCL X
39 AVIEW
40 PSE
41 RTN
42 END
```

Section 7.3. Mixing Uses of LOCAL LABELs

So far you have seen 2 uses of LOCAL LABELS. One was in the program SUM4 (Figure 6.4.) to direct flow within the main body of the program. The sequence of calculation was diverted to LBL 00 at line 05 in SUM4 by the unconditional branch statement GTO 00 at line 12. A similar type of branch was directed in the GRADES program (Figure 6.5.) by the statement GTO 01 (line 12) to take program flow to LBL 01 (line 28). This use of LOCAL LABELs is quite different from that in the program SUBS2 where they are used to identify subroutines. The directing statements to the LOCAL LABELS in SUM4 are "GTO _ _", whereas in SUBS2 they are "XEQ _ _". Also, the end of statements in each LOCAL LABEL in SUBS2 required a "RTN" state-

ment to return program flow. No such statement was required in the statements listed under LOCAL LABELS in SUM4, and program flow continued through progressively higher numbered lines in the program.

The program SUBS2 is just a little more than 22% shorter than SUBS1, because of the use of subroutines. You can imagine how there would be an even greater shortening of a program were there more equations to be solved which had identical subcomponents. As the programs you write on your own become more and more complex, you will undoubtedly encounter many opportunities to use subroutines to save lines in the MAIN MEMORY of your computer which can then be used for data storage or provide more room for other programs.

It would take a very special talent for someone to be able to write a long program to solve several equations and see as the program's first draft where all of the subroutines would fit in and know how they would be numbered. Suggestions for program design listed in Figure 5.5. indicate there is a more practical way to put such an involved program together.

Figure 5.5. recommends that after you have stated all of the equations you are going to use, defined their symbols and constructed a flow diagram, you proceed to write the first draft with no consideration at all for the use of subroutines or the efficiency of calculations. Just go ahead and complete the statements for each of the steps in the flow diagram much the same as SUBS1 was written, and don't be concerned with program length as the program is written in this first draft. This would not be good advice were you using a mainframe computer or one whose time you had to share with others. You'd be expected to work out all the details of your program ahead of time to be sure it would run efficiently and accurately the first time, since access to the computer and operation time on it are precious.

But what an advantage you have in owning your own HP-41 personal computer! You have the opportunity to work out the problems in your program, modify it to suit your own needs and change it in any way you want by actually running it on the machine as many times as you need. Such a quick turn-around in correcting your mistakes has a powerful effect on how rapidly and fully your learn about your computer. Also, it is of great value in developing your programs quickly.

Once you know the program runs correctly and accurately, it's then time to go back and seek out those program lines which are repetitive and construct appropriate subroutines to take care of them . You followed this process in the development of the program SUBS2 from its earlier version SUBS1. Then add the tones, alarms and special features to your creation. As a last procedure, be sure you have not accidentally altered your program in the process of adding subroutines to it so it no longer runs the way you designed it.

The next section in this chapter examines the construction of a program which uses LOCAL LABELS both as subroutines, as well as to direct the flow of calculations in the main body of the program. It will show also how subroutines can be built into other subroutines to save even more program space.

The program entitled, "AREA" is described in Figure 7.6., along with its equation and flow diagram. Both the equation and the program steps to solve it (Figure 7.7.) are more complex than others presented so far. Read the following description of the program's design, though, if for no other reason than to learn how to use your computer to calculate areas of irregular surfaces. There are many applications for such a program. If you find that understanding the construction of the program doesn't come to you right away, perhaps you can see through it easier when you've had more programming experience.

Understanding how the AREA program is constructed will be easier if you first see a practical demonstration of it. Despite the program's complexity, it is quite easy to run. First, key in the program steps listed in Figure 7.7., perform the sequence **GTO..**, then use the program in the following test. Draw a square on a piece of graph paper, and define the X and Y coordinates for each of its corners. Start the

Figure 7.6.

Sample Problem: AREA

Problem Statement:

This program is designed to calculate the area of an irregular polygon defined by an indefinite number of 3 or more X and Y coordinate pairs. All coordinate points must be equal to or greater than zero. Coordinates are entered in response to alpha prompts for data progressing successively either clockwise or counterclockwise around the polygon's perimeter. Data entry is terminated by entering a negative number in response to the alpha prompt "X_n?".

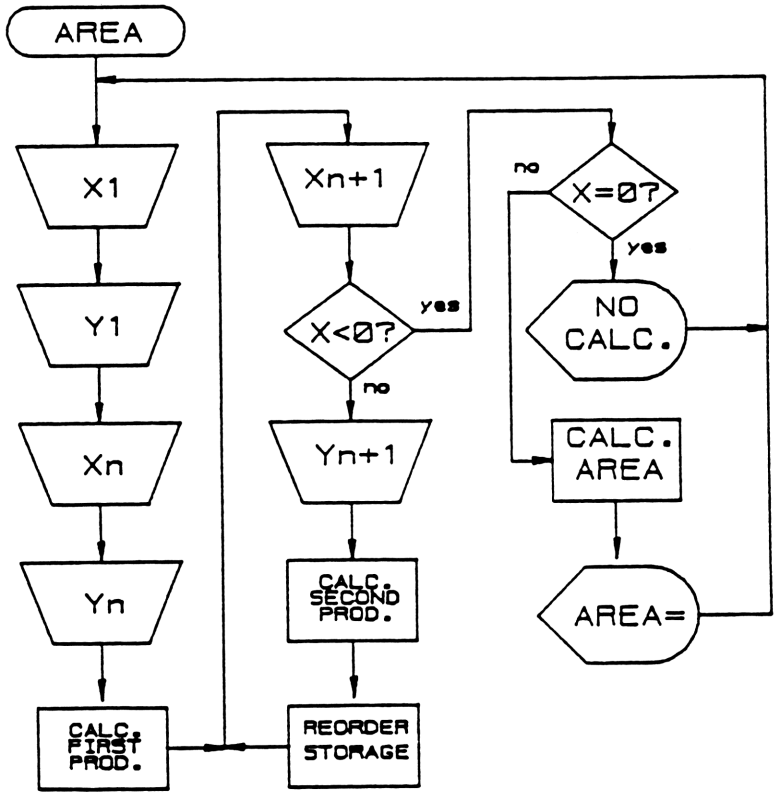
Equation:

$$A = \frac{(X_1 + X_n)(Y_1 - Y_n) + (X_n + X_{(n+1)})(Y_n - Y_{(n+1)}) \ldots + (X_L + X_1)(Y_L - Y_1)}{2}$$

where:

A = area in $(\text{units})^2$ of entered data
X_1, Y_1 = first X and Y coordinate pair
X_n, Y_n = next X and Y coordinate pair
$X_{(n+1)}, Y_{(n+1)}$ = successive X and Y coordinate pairs
X_L, Y_L = last X and Y coordinate pairs

Flow Diagram:



program by **XEQ "AREA"**. You will first hear the 4 tones of the BEEP function, then see the alpha prompt "X1?" displayed in the view window. **Enter** using R/S the **X** value for the first of your coordinate pairs. As stated in the program description in Figure 7.6., you can start at any point of the square. **Enter** the corresponding **Y1** value using R/S in response to the alpha prompt "Y1?". When "X2?" is displayed, **enter** the **X** value for the next coordinate pair, and **then** the **Y** value in response to "Y2?".

Continue **entering** data for the **other** 2 coordinate **points**. After the fourth pair have been entered, **key** any **number**, change its sign to be negative (using **CHS**), then press **R/S**. The area of the square is calculated, then after a pause, you will be asked for the first piece of data for the calculation of a second area. You can determine if the AREA program calculated the area of your test square accurately simply by counting the number of smaller squares contained within the larger one you drew. A more comprehensive test would be to draw a variety of irregular polygons on the graph paper, calculate their areas, then verify the accuracy of each calculation.

The flow diagram in Figure 7.6. shows the sequence in which data are entered and how interim and then final calculations are made. The list of program steps in Figure 7.7. gives more detail about the AREA program. Even though its flow is convoluted, it's worth following it through to see how LOCAL LABELS are used both to direct the sequence of operations in the main body of the program (steps 1 to 68), but also to see how they are used to identify subroutines (steps 69 to 94).

Figure 7.7.

Program Listing: AREA

```
01 LBL "AREA"
02 BEEP
03 CLRG
04 FIX 0
05 XEQ 01
06 XEQ 04
07 STO 01
08 XEQ 05
09 STO 02
10 XEQ 01
11 XEQ 04
12 STO 03
13 XEQ 05
14 STO 04
15 XEQ 01
16 RCL 02
17 RCL 04
18 -
19 RCL 01
20 RCL 03
21 +
22 *
23 STO 07
24 LBL 00
25 XEQ 04
26 X<0?
27 GTO 06
28 STO 05
29 XEQ 05
30 STO 06
31 XEQ 01
32 RCL 04
33 RCL 06
34 -
35 RCL 03
36 RCL 05
37 +
38 *
39 RCL 07
40 +
41 STO 07
42 RCL 05
43 STO 03
44 RCL 06
45 STO 04
46 GTO 00
47 LBL 06
48 RCL 05
49 X=0?
50 GTO 03
51 RCL 01
52 +
53 RCL 06
54 RCL 02
55 -
56 *
57 RCL 07
58 +
59 2
60 /
61 TONE 9
62 TONE 9
63 FIX 2
64 "AREA="
65 ARCL X
66 AVIEW
67 PSE
68 GTO "AREA"
69 LBL 01
70 1
71 ST+ 00
72 RTN
73 LBL 02
74 ARCL 00
75 "⊢?"
76 PROMPT
77 RTN
78 LBL 03
79 TONE 5
80 TONE 8
81 "NO CALC"
82 AVIEW
83 PSE
84 GTO "AREA"
85 LBL 04
86 TONE 5
87 "X"
88 XEQ 02
89 RTN
90 LBL 05
91 TONE 5
92 "Y"
93 XEQ 02
94 RTN
95 END
```

Data Storage:

REG	Data
00	counter
01	X_1
02	Y_1
03	X_n
04	Y_n
05	$X_{(n+1)}$
06	$Y_{(n+1)}$
07	first product $(X_1+X_n)(Y_1-Y_n)$

After the program is initialized (by XEQ "AREA"), the BEEP is sounded, PRIMARY STORAGE REGISTERS and STACK REGISTERS are cleared (step 3). Next, the decimal point is set (step 4), program flow is directed at step 5 to the first of the subroutines (LBL 01) by XEQ 01. LBL 01 (step 69) first stores the number 1 additively in REG 00, and program flow returns to step 6 where the next subroutine is executed. LBL 04 contains steps which sound a tone (TONE 5), places an "X" into the ALPHA REGISTER, then executes a subroutine (XEQ 02) which recalls (ARCL 00) into the ALPHA STACK REGISTER the number which had been previously stored in REG 00. As program flow passes this point for the first time, the contents of REG 00 is the number 1 placed there by the steps listed in LBL 01, as is the symbol "?". Step 77 lists a RTN which returns the program to the step immediately after the last XEQ statement, which in this case is line 89 where another RTN command is encountered to take the program back to line 07.

The number entered in response to the alpha prompt "X1?" is stored in REG 01 at line 07. A value for Y2 is requested by the alpha prompt "Y2?" which is constructed in the view window by statements listed in the subroutine LBL 05 and those in LBL 02. Note that the program lines in LBL 02 did double duty. They provided the way to display the latter half of both alpha prompts "X1?" and "Y1?". You'll see that all subsequent data input requests are structured in a similar way using LBL 02. Once the program has been brought back to step 09, the value entered for Y1 is stored in REG 02.

As the program progresses, the number in REG 00 then has 1 added to it (LBL 01), and the appropriately next higher "X2?" and "Y2?" are displayed in sequence. After data have been entered for them, they are stored in REG 03 and REG 04, correspondingly. The first product in the equation shown in Figure 7.6. is calculated in steps 16 to 22, and stored in REG 07 at step 23. The conditional statement in step 26 (X<0?) tests if the number entered in response to the next and each subsequent entry of the X coordinate is a negative number. If it is, data entry is terminated, and the program is directed to LBL 06 where the remainder of the equation is calculated unless the number stored in REG 05 is a zero.

REG 05 would contain a zero if only 2 pairs of X and Y coordinates had been entered. Some number greater than zero would be stored in REG 05 if a minimum of 3 coordinate pairs had been entered, which is the minimum number of data required to calculate an area. If the contents of REG 05 is tested to be zero (step 49), an unconditional branch (at step 50) takes the program flow to LBL 03 which sounds a 2 tone sequence, displays "NO CALC", and directs the program to its GLOBAL LABEL from which a new set of data are requested for the next calculation. "NO CALC" was displayed, of course, because an area cannot be calculated when only 2 points are entered into the AREA program. A minimum number of 3 points is required to define an area.

If as tested at step 49, REG 05 does not contain a zero, at least 3 coordinate pairs have been entered, the remainder of products and sums are calculated for the equation shown in Figure 7.6., and the solution is displayed beginning at step 64. The answer is displayed for a second or two (PSE; step 67), and a new set of data are requested for the next calculation after program flow has been directed to the GLOBAL LABEL AREA.

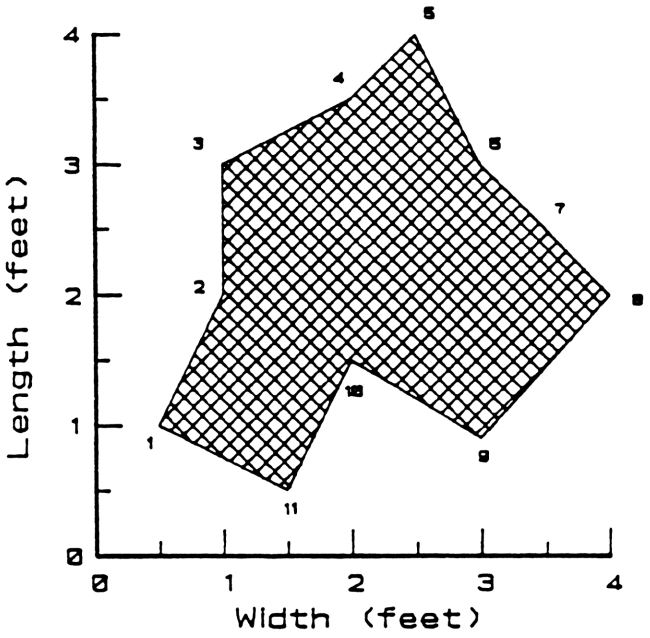
Even though AREA is somewhat involved, the program itself is easy to operate. Also, the program is reasonably short and runs quickly despite the complexity of the calculation, thanks to the HP-41's feature for using LOCAL LABELS. Not only are LOCAL LABELS used in the main body of the program as addresses for unconditional branches (like GTO 06 at Step 27, GTO 00 at Step 46, etc.), they also function to identify subroutines for repetitively used operations (like LBL 04, LBL 05, etc.). LBL 04 (the subroutine which establishes the letter "X"), in fact, has its own internally executed subroutine (XEQ 02 at Step 88) to direct program flow from one subroutine (LBL 04) to another (LBL 02), each with its own RTN statement. Stacking subroutines inside of one another provides additional opportunities to

Figure 7.8.

Sample Problem: AREA

Problem Statement:

Sam Snip owns a leather shop where he sells handcrafted items. Every once in a while, customers ask him to sell irregularly shaped pieces of his stock for their own projects. He knows that each piece of leather is most accurately priced based on its area, and he wrote the AREA program to make this calculation. He has placed a piece of leather on his cutting board and has defined coordinate points for it using the board's scales. What is the area of this item?



Point No.	(X) Width (ft.)	(Y) Length (ft.)
1	0.5	1.0
2	1.0	2.0
3	1.0	3.0
4	2.0	3.5
5	2.5	4.0
6	3.0	3.0
7	3.5	2.5
8	4.0	2.0
9	3.0	1.0
10	2.0	1.5
11	1.5	0.5

(answer: 6.13 ft^2)

shorten a program and make it run more quickly. All models of the HP-41C series computers allow for the construction of 6 levels of subroutines in this way.

Except for the instruction at step 75, keying the program AREA into your computer should present no problem if you have practiced the keystrokes for other programs presented earlier. Step 75 uses an APPEND function which is keyed using the ALPHA SHIFT definition of the XEQ key (see Table 2.4.). The APPEND function allows you to add to an alpha string already stored in the ALPHA REGISTER. It is necessary in this program associated with the efficient use of the subroutines in LBL's 02, 04 and 05 which constructed the alpha prompts for data entries for X1 to Xn, and Y1 to Yn. An easy way for you to see what the APPEND function does in program step 75 is to first write the program with it, run a sample problem or two, then erase line 75 and rewrite it to contain only the question mark. Running the program in such an amended fashion will soon demonstrate how the alpha strings "Xn?" and "Yn?" are formed and displayed.

A sample problem for the AREA program is shown in Figure 7.8. After you have made this calculation with the program you've already keyed into your computer,

try editing it so that not only does it calculate the area of a piece of leather for Sam, but also it will automatically calculate its price. Figure 7.9. lists the steps for the program entitled, "PRICE" which shows one way this calculation can be built into the AREA program. The next couple of paragraphs will demonstrate that it is unnecessary to start all over again to key in the rather long PRICE program. Its features are easily added to the AREA program you now have in the MAIN MEMORY of the computer.

Figure 7.9.

Program Listing: PRICE

```
01 LBL "PRICE"
02 BEEP
03 CLRG
04 FIX 0
05 XEQ 01
06 XEQ 04
07 STO 01
08 XEQ 05
09 STO 02
10 XEQ 01
11 XEQ 04
12 STO 03
13 XEQ 05
14 STO 04
15 XEQ 01
16 RCL 02
17 RCL 04
18 -
19 RCL 01
20 RCL 03
21 +
22 *
23 STO 07
24 LBL 00
25 XEQ 04
26 X<0?
27 GTO 06
28 STO 05
29 XEQ 05
30 STO 06
31 XEQ 01
32 RCL 04
33 RCL 06
34 -
35 RCL 03
36 RCL 05
37 +
38 *
39 RCL 07
40 +
41 STO 07
42 RCL 05
43 STO 03
44 RCL 06
45 STO 04
46 GTO 00
47 LBL 06
48 RCL 05
49 X=0?
50 GTO 03
51 RCL 01
52 +
53 RCL 06
54 RCL 02
55 -
56 *
57 RCL 07
58 +
59 2
60 /
61 TONE 9
62 TONE 9
63 FIX 2
64 "AREA="
65 ARCL X
66 AVIEW
67 PSE
68 TONE 5
69 "COST/SQFT?"
70 PROMPT
71 *
72 TONE 9
73 TONE 9
74 "PRICE=S"
75 ARCL X
76 AVIEW
77 PSE
78 GTO "PRICE"
79 LBL 01
80 1
81 ST+ 00
82 RTN
83 LBL 02
84 ARCL 00
85 "⊢?"
86 PROMPT
87 RTN
88 LBL 03
89 TONE 5
90 TONE 8
91 "NO CALC"
92 AVIEW
93 PSE
94 GTO "PRICE"
95 LBL 04
96 TONE 5
97 "X"
98 XEQ 02
99 RTN
100 LBL 05
101 TONE 5
102 "Y"
103 XEQ 02
104 RTN
105 END
```

Editing a program either to add new lines, or to remove unwanted ones is easy. It involves basically the same procedures you followed in reviewing the SUM1 program in Section 6.1.3. Generating the new program PRICE by editing the AREA program will be used to demonstrate these basic steps. Two general changes need to be made in the AREA program if it is to calculate the price in addition to the area of a piece of leather for Sam. First, program lines need to be added which will calculate the price of the piece and then display an answer, and second, the program's GLOBAL LABEL and LOCAL LABELS which direct flow to the GLOBAL LABEL need to be re-phrased.

Since it is necessary to know what price per square foot Sam wants to charge for his stock, another data request must be added to the program in order to calculate the price of a piece of leather once its area is known. First, review steps 68 to 70 in Figure 7.9. to see how this data request appears in the PRICE program, then examine steps 71 to 78 to see how the "COST/SQFT?" information is used to calculate the price of the piece. Up to step 67, the area of the piece had been calculated and the answer remains in the X STACK REGISTER. Once the unit price for the leather has been entered (step 70), the contents of the X and Y STACK REGISTERS are multiplied together to calculate the price for the entire piece of leather (step 71). This is displayed (step 74), and after a pause the program is directed to its GLOBAL LABEL for the next calculation (step 78). You will now see how steps 68 to 78 were added to the AREA program.

One way to get the PRICE program into your computer, of course, would be to key in all of its steps shown in Figure 7.9. starting at the very beginning of the program. On the other hand, if you have already entered the AREA program into your computer, only a few segments of it need to be changed to convert it to the PRICE program.

Assume that you already have the AREA program in your computer, but now want to edit it to be the PRICE program. Place your computer in a keyboard mode (by turning it **OFF**, then **ON**, for example), then call the AREA program from memory by **XEQ "AREA"**. You will hear the BEEP tones and see the alpha data prompt "X1?" in the view window. Rather than continuing to run the program, you will now edit sections of it beginning at step 68 where Steps 68 to 78 of the PRICE program need to be inserted. You first need to find step 68 of the AREA program, and do this by keying **SHIFT GTO.068**. When the view window displays "1", pressing the **PRGM** toggle changes the display to show step 68 of the AREA program, which is GTO AREA. This unwanted step is edited out simply by pressing the erase key. The display window now shows step 67 of the program, PSE.

Step 68 of the PRICE program is entered just by keying **XEQ "TONE"** and then after a pause, keying "5" to designate which tone will be sounded. Key the remainder of the **steps 69 to 78**, remembering that the "S" symbol is written by using the ALPHA SHIFTed function of the EEX key (see Figure 2.4.). After you have entered step 78 of the PRICE program, you are ready to amend the GLOBAL and LOCAL LABELS of the AREA program. Press the **PRGM** toggle to get out of the PRGM mode, then key **SHIFT GTO 04**. This instruction will take you to the LOCAL LABEL 04 which is step 85 of the AREA program. To see this line of the program, press **PRGM** to get back into the program mode. Keying SHIFT **BST** displays the immediately previous program step which is "GTO AREA" in the AREA program. This unwanted step is edited out by pressing the **erase** key, and the new step is entered by keying **GTO PRICE**.

The last change to made in the AREA program is to rewrite its GLOBAL LABEL. While the computer is still in the PRGM mode, key SHIFT **GTO. 001** to take the program to its GLOBAL LABEL. Eliminate the GLOBAL LABEL AREA by pressing the **erase** key, and enter the new GLOBAL LABEL by keying SHIFT **LBL "PRICE"**. You have now made all the changes required to convert the AREA program to the PRICE program. To secure your new program, take the computer out of the PRGM mode and key **GTO..** to initiate the PACKING function. The last thing to do is to check the PRICE program for logic flow and for numerical accuracy of its calculations.

Try amending the AREA program in a different way just to be sure you understand the important points introduced in the last few pages. Imagine, for example, that you have been asked to write a computer program for a forest ranger who needs to determine the surface areas of lakes in different sections of his region. As you look at a map showing these many land sections and the lakes they contain, you decide your

Figure 7.10.

COUNT

Problem Statement:

As a laboratory technician, Bob Beaker often has to prepare serial dilutions and samples of various kinds with different levels of precision. He wrote the program COUNT to help him with this job. The program allows him to start counting from any number, increment by any amount, end at or near any number and control the level of precision for these calculations by defining the program's digit display.

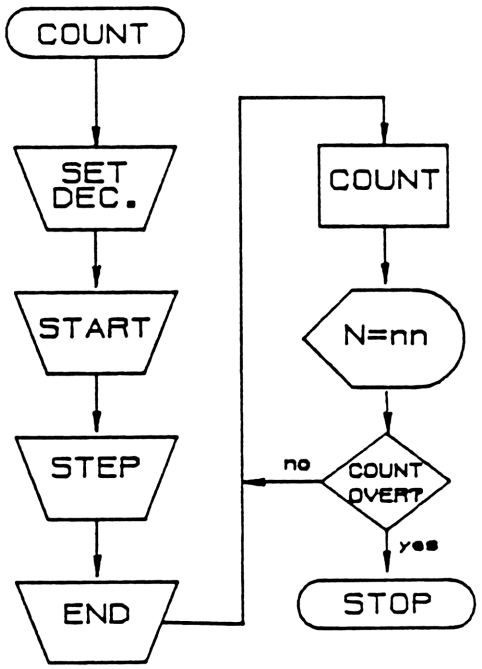
Equation:

$$N_C = N_I + I$$

where:

N_C = displayed next count
N_I = initial number
I = count increment

Flow Diagram:



Program Listing:

```
01 LBL "COUNT"
02 BEEP
03 "PRECISION?"
04 PROMPT
05 STO 00
06 FIX IND 00
07 TONE 5
08 "START?"
09 PROMPT
10 STO 01
11 TONE 5
12 "STEP?"
13 PROMPT
14 STO 02
15 TONE 5
16 "END?"
17 PROMPT
18 STO 03
19 LBL 00
20 RCL 01
21 RCL 02
22 +
23 STO 01
24 TONE 7
25 "N="
26 ARCL X
27 AVIEW
28 STOP
29 RCL 03
30 X>Y?
31 GTO 00
32 TONE 9
33 "COUNT OVER"
34 AVIEW
35 STOP
36 END
```

computer program needs to be able to calculate and display: 1. the area of any single lake, 2. the combined areas of many lakes in a single section, 3. the total area of all lakes in the entire region.

What additional changes would the AREA program require so that it could also calculate and display the length of shore line for each lake and for the total number of lakes? If you knew the average number of fish per cubic yard for the region, how could your program estimate the fish population for each lake? for all lakes in a section? for all lakes in the region? The forest ranger is willing to pay $1,000 for a program which will make these calculations, so it's worth a little effort and time to write it.

Section 7.4. Four Easy Pieces

A bonus from learning to use a portable computer like the HP-41 is not just being able to solve complex equations, but to provide a tool for completing everyday on-the-job tasks. Many of these jobs are simple, yet they take precious human

Figure 7.11.

Sample Problems: COUNT

Sample Problem No. 1:

Bob often needs to prepare solutions in a series which have different initial and final volumes after being supplemented by a constant amount. He used the COUNT program to provide the data for the example shown in Table 1. For this batch, he started with a 14.35 cc volume, added successive volumes of 1.87 cc. and ended with a last sample being at least 25.00 cc. Precision was defined by making calculations with 2 digits to the right of the decimal.

Sample Problem No. 2:

Bob needed to weigh out a series of samples of dry chemical used in an analytical procedure. He started with a mass of 1.000 gram, added 0.932 grams at each step, and wanted the last sample to be at least 7.000 grams. He used the COUNT program with a precision of 0.000 to provide the information in Table 2.

Table 1

Sample No.	Vol. (cc)
1	14.35
2	16.22
3	18.09
4	19.96
5	21.83
6	23.70
7	25.57

Table 2

Sample No.	Vol. (gm)
1	1.000
2	1.932
3	2.864
4	3.796
5	4.728
6	5.660
7	6.592
8	7.524

time to complete unless one takes advantage of the HP-41. This section uses 4 examples of time-savings programs to introduce several new programming techniques.

The simple task of having to make counts and keep track of steps in preparing chemical solutions or chemical samples is made much easier with the COUNT program shown in Figure 7.10. This straightforward program allows for a selected level of precision to be maintained in a controlled counting procedure, and it keeps track of the last element in the count, as shown in the sample problems in Figure 7.11. A new programming technique is introduced in the COUNT program, that of INDIRECT ADDRESSING. This technique is adequately complicated that it is discussed in detail in a chapter of its own (Chapter 9), but its use in COUNT is simple enough.

The precision with which answers are to be shown in the COUNT program is determined by how the user decides to display digits to the right of the decimal (see Figure 2.7.). This information is stored in REG 00 at line 05. The actual setting of the display digit count is controlled by the INDIRECT ADDRESSING statement in the next line, FIX IND 00. The instruction reads, "set the display to have as many digits to the right of the decimal as is the number in PRIMARY STORAGE REGISTER 00". For example, were the number 2 stored in REG 00, answers would be displayed as "N.NN"; were REG 00 to contain a 4, the display would be "N.NNNN".

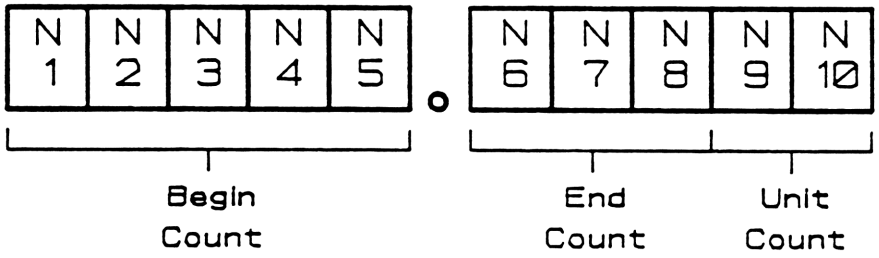
The reader is encouraged to rewrite the COUNT program so that only a designated number of serial dilutions would be controlled, rather than have the program operate between user-defined counting limits. Also, how could the program be modified so that Bob Beaker could display decremental intervals in his count?

Figure 7.12.

ISG and DSE Functions

ISG: Increment and skip if greater than

DSE: Decrement and skip if equal to



There is a technique for automatic counting built into the HP-41 computer which performs a similar operation as that in the COUNT program. The DSE and ISG functions operate in reference to a stored 10 digit control number. The code for this control number is described in Figure 7.12. This number is easy to interpret with a little practice. For example, were the number 0.01001, then counting would begin at zero, increase in a step of 1 and end at 10 when the ISG function was activated. Similarly, were the code number 50.10002, the count would begin at 50 and end at 100, and made in steps of 2. The program AUTOC in Figure 7.13. shows how the ISG instruction is listed in a program and shows one way in which the control number can be constructed.

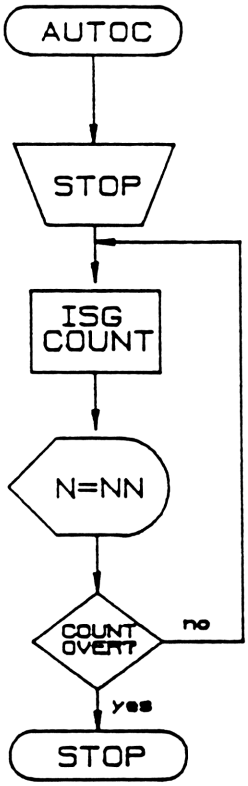
Figure 7.13.

AUTOC

Problem Statement:

Bob, the laboratory technician, needs to keep track of the number of times he completes a repetitive operation which is involved and sometimes distracting. He wrote AUTOC so that once the program is initialized, he needs only to press the R/S toggle each time he's completed the task. The HP-41 keeps tally for him and tells him when the sequence is over.

Flow Diagram:



Program Listing:

```
01 LBL "AUTOC"      17 1
02 BEEP             18 ST+ 01
03 FIX 0            19 RCL 01
04 0                20 TONE 5
05 STO 00           21 "N="
06 STO 01           22 ARCL X
07 1 E-5            23 AVIEW
08 ST+ 00           24 STOP
09 "STOP?"          25 ISG 00
10 PROMPT           26 GTO 00
11 1                27 TONE 9
12 -                28 "COUNT OVER"
13 1 E3             29 AVIEW
14 /                30 STOP
15 STO+ 00          31 END
16 LBL 00
```

But, why write a program like COUNT (Figure 7.10.) when the HP-41 computer already has an automatic counting operation like that shown in Figures 7.12. and 7.13.? For 1 reason, the COUNT program allows for steps in a count to be made in fractional units for either decrementing or incrementing a number. For example, Bob Beaker was able to use the COUNT program to increment dry samples by 0.932 grams in Sample Problem No. 2 (Figure 7.11.). The AUTOC program counts only in whole numbers.

But the ISG and DSE counting functions allow you to do something important which the COUNT program will not. Not only will they provide automatic counting, more important, they can be used to control the number of times a designated

function is operated. This is an important distinguishing feature of the ISG and DSE operations. A simple example for controlling the TONE function is shown in the program TONEC described in Figure 7.14.

Although the user has control over which TONE will be sounded, a decision must be made (line 07) about the number entered in response to the alpha request at line 03, because the HP-41 has TONEs numbered only 0 to 9. The control number for the ISG function is stored in REG 01 by instructions in lines 10 to 21.

Note how the controlled sounding of the designated tone is made by only the few lines of instructions in LBL 00. Note also how INDIRECT ADDRESSING is used in deciding which TONE will be sounded at line 23. This instruction reads, "sound the tone whose number is stored in REG 00." More about INDIRECT ADDRESSING in Chapter 9.

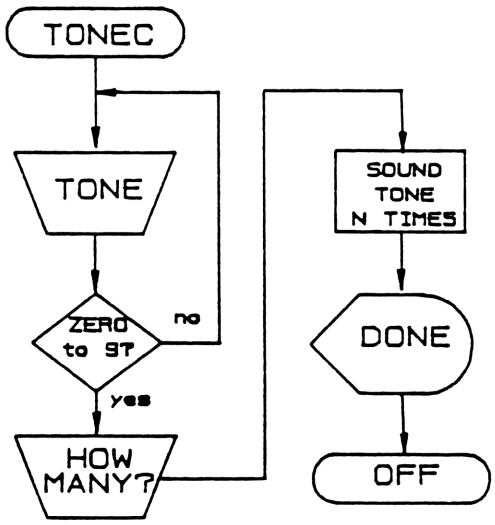
Figure 7.14.

TONEC

Problem Statement:

This program uses ISG (and is easily modified to use DSE) to control repeating a function, for which TONE is used as an example.

Flow Diagram:



Program Listing:

```
01 LBL "TONEC"      16 1 E3
02 BEEP             17 /
03 "TONE?"          18 ST+ 01
04 PROMPT           19 1 E5
05 STO 00           20 1/X
06 10               21 ST+ 01
07 X≤Y?             22 LBL 00
08 GTO "TONEC"      23 TONE IND 00
09 TONE 5           24 ISG 01
10 0                25 GTO 00
11 STO 01           26 "DONE"
12 "HOW MANY?"      27 AVIEW
13 PROMPT           28 PSE
14 1                29 OFF
15 -                30 END
```

Many times on-the-job tasks are awkward because they involve one or more calculations of simple proportions. The point to be made is that these calculations are easy enough to do, but they take time and can be confusing unless one does them many times frequently. But for someone who has not made these computations for a while, it can take more time than it's worth to have to work through the problem with paper and pencil: "OK, we've got a thermal conductivity value of 1.25 watts per cm. per degree Kelvin, what is that in units of BTU's per hour per ft. per degree Fahrenheit? Or would it be better to use units of calories per second per centimeter per degree Celsius? Damn, I used to be able to do this in college".

Figure 7.15.

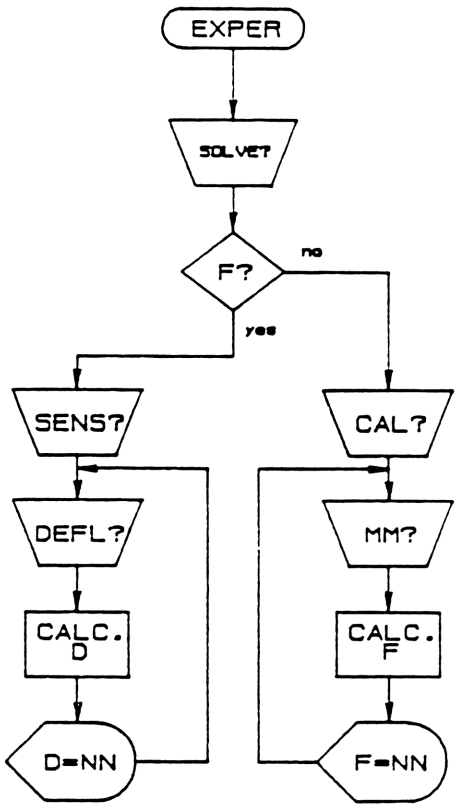
The Experiment

David Digit conducts tests which measure the small amplitude and low frequencies at which a small machine operates. He is using a sensor for measuring the machine's movements which generates an analog voltage which he records. He determined ahead of time that the sensor's calibration is: Voltage output (mV) = -0.5492 + 0.9167(distance (mm)). The amplifier on his recorder is set at 0.2 mV/cm. of deflection on his recording paper. Paper speed is 5 mm/sec. He is interested in calculating how much and how rapidly the machine moves using data he has recorded. He obtained the information in Table 1 using the EXPER program.

Table 1

	Recorded		Calculated	
Movement No.	Recording Amplitude (cm)	Distance of peaks (mm)	Distance moved (mm)	Movement frequency (per min)
1	3.4	-	1.34	-
2	6.4	2.5	2.00	120.00
3	0.3	0.9	0.66	333.33
4	7.2	2.1	2.17	142.86
5	2.4	2.0	1.12	150.00

Flow Diagram:



Program Listing:

```
01 LBL "EXPER"
02 BEEP
03 FIX 2
04 AON
05 "SOLVE(D/F)?"
06 PROMPT
07 AOFF
08 ASTO X
09 "F"
10 ASTO Y
11 X=Y?
12 GTO 01
13 LBL 00
14 TONE 5
15 "SENS(MV/CM)?"
16 PROMPT
17 STO 00
18 LBL 02
19 TONE 5
20 "DEFL(CM)?"
21 PROMPT
22 RCL 00
23 *
24 0.5492
25 +
26 0.9167
27 /
28 TONE 9
29 TONE 9
30 "D="
31 ARCL X
32 "⊢MM"
33 AVIEW
34 PSE
35 GTO 02
36 LBL 01
37 TONE 5
38 "CAL(MM/SEC)?"
39 PROMPT
40 STO 01
41 LBL 03
42 TONE 5
43 "MM?"
44 PROMPT
45 RCL 01
46 /
47 60
48 /
49 1/X
50 TONE 9
51 TONE 9
52 "F="
53 ARCL X
54 "⊢/MIN"
55 AVIEW
56 PSE
57 GTO 03
58 END
```

The time spent in writing a simple program for the HP-41 to make these calculations is often well invested, even if only a dozen or so computations are required. The program, EXPER, described in Figure 7.15. shows one such application for the HP-41. EXPER shows how a linear equation expressing the calibration of an instrument and information about data recording are built into a program to help with the calculation of (in this example) distance and frequency of movement of a device under study. EXPER shows also how program flow is controlled on the basis of comparing ALPHA statements (lines 04 to 11), and it gives 2 examples of how the APPEND function is useful (lines 34 and 56). Data in the table at the bottom of Figure 7.15 show the results of sample calculations with EXPER.

Section 7.5. Keeping Track

It is often necessary in constructing a computer program to include features in it which will keep track of the number of data which have been entered. Branching, looping and continuing with program flow frequently depend heavily on the data entries themselves. For example, the options for what the program does next may be decided on the basis of what kinds and how many data have already been entered.

This is a similar necessity to checking during program operation to be sure that entered data are within designated and legitimate ranges. Many programs will require an internal check for a minimum or maximum number of data entries to run correctly besides testing whether any single datum entry is in an acceptable range. The program PVAR described in Figure 7.16. demonstrates one of the many ways in which the computer can police program flow depending on the number of entered data.

Standard deviation (a built-in function of the HP-41 computer called SDEV) and standard error are commonly used statistics. They require evaluating mathematically the relationships among a set of data in a single group. Sometimes, though, it is useful to calculate a single standard deviation and standard error for several groups which for reasons unique to a series of tests can be justifiably considered to represent a single population.

If for no other reason than the different groups may have different numbers of observations in each of them, calculating a "pooled standard deviation" or "pooled standard error" is not as simple as just adding up the standard deviations or standard errors of all the groups and dividing by the number of groups. A legitimate calculation of these pooled statistics is not the same as only getting the average of the individual variability in the groups. The program PVAR shows one of the ways in which standard deviations of different groups of data can be mathematically combined for a representation of pooled statistic.

The program includes several useful programming features. One of them is the combined use of arithmetic operations both in the STACK REGISTERS and in the PRIMARY STORAGE REGISTERS. Another is the use of a single number stored in a PRIMARY STORAGE REGISTER to serve the functions of display control and program control.

Once the program is initialized, the number "1" is stored automatically and additively in a PRIMARY STORAGE REGISTER (REG 00; lines 07 and 08). To assure that REG 00 has a zero in it at the beginning of the program, the function CLRG is listed at program line 02. CLRG automatically clears (puts a zero in) all PRIMARY STORAGE REGISTERS. The standard deviation for the first numbered group is then requested as a data entry at lines 09 to 11. Notice how an APPEND function is used to provide corresponding numbers in the data request for the sequence of data entry. The number stored in REG 00 serves for the display counter when "N" is

Figure 7.16.

PVAR

Program Description:

This program calculates a "pooled standard deviation" and a "pooled standard error" for 2 or more groups of data for each of which standard deviation and the number of samples are known.

Equations:

$$PSD = \sqrt{\frac{[(SD_1)^2(N_1 - 1)] + [(SD_2)^2(N_2 - 1)] \ldots + [(SD_n)^2(N_n - 1)]}{(N_1 + N_2 \ldots + N_n) - 1}}$$

$$PSE = \frac{PSD}{\sqrt{N_1 + N_2 \ldots + N_n}}$$

where:

$SD_1, SD_2, \ldots SD_n$ = standard deviations of groups
$N_1, N_2, \ldots N_n$ = number in each data group
PSD = pooled standard deviation
PSE = pooled standard error

Flow Diagram:

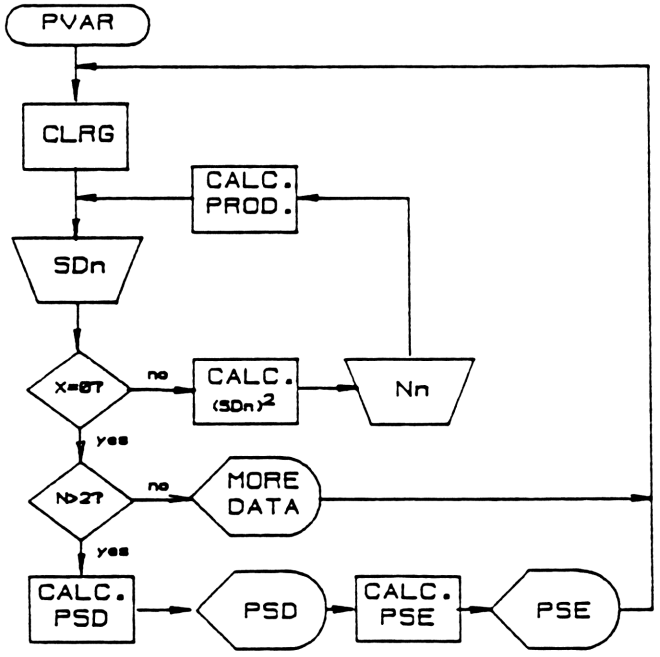


Figure 7.17.

Sample Problem: PVAR

A research team obtained the data in Table 1 for a group of patients who had been given a drug in a controlled test. The decision was made to calculate a standard deviation and a standard error considering that all observations were for a single test group. Using the PVAR program: PVAR = 13.37; PSE = 1.25.

Table 1

	Group 1	Group 2	Group 3	Group 4
St. Dev.	14.96	12.81	15.92	11.01
N	23	56	18	17

Program Listing:

```
01 LBL "PVAR"
02 CLRG
03 FIX 0
04 BEEP
05 LBL 00
06 TONE 5
07 1
08 ST+ 00
09 "SD"
10 ARCL 00
11 "⊢?"
12 PROMPT
13 X=0?
14 GTO 02
15 X²
16 TONE 5
17 "N"
18 ARCL 00
19 "⊢?"
20 PROMPT
21 ST+ 02
22 1
23 -
24 *
25 ST+ 01
26 GTO 00
27 LBL 02
28 RCL 00
29 3
30 X>Y?
31 GTO 03
32 RCL 01
33 RCL 02
34 1
35 -
36 /
37 SQRT
38 FIX 2
39 TONE 9
40 TONE 9
41 "PSD="
42 ARCL X
43 AVIEW
44 PSE
45 TONE 9
46 TONE 9
47 RCL 02
48 SQRT
49 /
50 "PSE="
51 ARCL X
52 AVIEW
53 PSE
54 GTO "PVAR"
55 LBL 03
56 TONE 5
57 TONE 9
58 TONE 9
59 TONE 5
60 "MORE DATA"
61 AVIEW
62 PSE
63 GTO "PVAR"
64 END
```

requested for the data group at lines 17 to 20. Notice also how the program instructions controlled as a loop under the LOCAL LABEL 00 (lines 05 to 26) construct both the numerator and denominator for equation 1 by taking advantage of PRIMARY REGISTER arithmetic involving REG's 01 and 02.

Entering a zero when the data request "SD nn?" is displayed signals to the computer that all available data have been entered, and the calculations of PSD and PSE (not to be confused with the HP-41 operation PSE which produces a "pause" in program flow) are to be made. The conditional test "X=0?" at line 13 determines whether the next stage of numeric operations for the equation's numerator and denominator are to be made, or whether the calculation of equations for PSD and PSE are to be calculated (beginning at line 27).

If the calculation of pooled variance is to have any sense, a minimum of 2 sets of data must be used for the calculation. This contingency is tested as a first step under the LOCAL LABEL 00. First, it is determined if the number stored in REG 00 (which has so far served as a counter for display control) is at least 2. If it isn't, no meaningful calculation can be made. The user is informed of this by instructions in the program lines 56 to 63 and is requested to provide more data, then incurs the minor penalty of being taken back to the beginning of the program and having to start over with data entry. Since only 1 set of data have been entered anyway, this is a minor, but necessary delay. If data from 2 or more sets have been entered, PSD and PSE are calculated and sequentially displayed, and the program begins anew for a subsequent calculation. Data in Figure 7.17 show the results of a sample calculation using the program PVAR.

You've seen in this chapter that often more program lines are devoted to controlling sequence in program flow than are involved in actual computations. Even for simple programs, considerable attention must be paid to designing how data are entered, where calculations are to be made, and how answers are to be formatted, announced and displayed. Also, you have seen how program flow is controlled using conditional tests, unconditional branches and LOCAL LABELS in the main body of the program, as well as in subroutines. The next chapter introduces yet another programming technique, and a powerful one, for efficiently controlling program execution, defining where data are required to be entered, and affecting how answers to computations are displayed. You will learn next how to use FLAGS.

Chapter 8

Flags

The basic idea of how FLAGS function in a computer program is implied by the word itself: flags are signaling devices. With this basic concept in mind, learning how to control FLAGS in the HP-41 computer and how to use them in a program will be much easier. A general example will help.

Imagine you're driving along a road and come upon a stretch where construction is underway. You slow down as you see a workman standing by the side of the road holding a signal flag. Just off to one side behind him is a large roadgrader which every once-in-a-while needs to back into your traffic lane. When the grader is operating on the shoulder of the road, the workman lowers his flag and traffic proceeds down the unobstructed roadway. When the grader needs to move onto the road surface, though, the workman raises his flag, signaling traffic to move into an alternative lane. Whether the detour needs to be taken or not depends on the current state of the roadway signaled by the workman using his flag.

The workman's signal flag is always in 1 of 2 positions. It is either "set" (when it is raised) to direct traffic into a side-route, or it is "clear" (when it is lowered) indicating the road is open and traffic flows down its proscribed lane. Either flag status (SET or CLEAR) signals a specific condition important to the motorist. The "either-or" decision signaled by the flag's status depends on a specific condition to be encountered at the next stage of the trip. FLAGS work that way in a computer program too to direct both information and program traffic flow.

There is another feature of how the workman uses his flag which will help understanding the use of FLAGS in a computer program. Imagine you've pulled your car off the road for a while to watch the many different operations at the road construction site. You'd see that at one time the workman would use his flag to signal traffic flow around the operations of the roadgrader. After that phase of the project had been completed and the roadgrader was shut down, the workman might move to a different location and use his flag to stop traffic so a truck could cross the road. Later, he might use it to signal directions for loading a piece of heavy equipment on a trailer. It's the same flag, but it is used to signal different situations at the judgment of the workman.

In the same way that the workman uses his flag at different times to control many different operations, any FLAG available to you in the HP-41 can be used in program writing to signal any condition you choose. Even though some FLAGS are specified to be used for only one function, there are many others which can be used in connection with just about anything you want.

A FLAG in a computer program operates exactly as does the flag used at the construction site. Each FLAG in your HP-41 computer is either in a SET, or in a CLEAR status depending on a specific condition in a program, or on that of the computer itself. You have direct control over the status of many of these FLAGS, but others will be automatically SET or CLEARed for you depending on what accessories you have plugged into your computer, which mode the computer is in, its battery charge, and many other conditions. This is all done for you quietly and unobtrusively by internal operations of the computer. In most cases, you don't have to be concerned or even aware of these changes in FLAG status as you go about the business of using the machine.

Having control over the status of FLAGS, though, give the HP-41 owner considerable power in writing computer programs which are efficient and which run quickly. This chapter emphasizes how the FLAGS over which you have direct control

are used in programming. Figure 8.1. summarizes how your computer's FLAGS are numbered, what function they monitor, and how their status is automatically adjusted when the machine is first turned on. Information in Figure 8.1. will be useful for choosing which FLAGS you decide to use.

FIGURE 8.1.

FLAG Functions

FLAG No.	FLAG Function	Comment
00 to 10	General Purpose User Defined	Status maintained
11 to 20	Special Purpose User Defined	Cleared with ON
11	AUTO On	Cleared with ON
21	Printer Enable	Same as FL55 with ON
22	Numeric Input	Cleared with ON
23	ALPHA Input	Cleared with ON
24	Range Error Ignore	Cleared with ON
25	Error Ignore	Cleared with ON
26	Audio Enable	SET with ON
27	USER Mode	Status maintained
28	Decimal	Status maintained
29	Digit Group	Status maintained
30	CATALOG	
31 to 35	Peripheral	
36 to 39	Digit count	Status maintained
40	FIX	Status maintained
41	ENG	Status maintained
42	GRADS	Status maintained
43	Radians	Status maintained
44	Continuous ON	
45	Data entry	
46	Partial Key Sequence	
47	SHIFT SET	
48	ALPHA	Cleared with ON
49	Low Battery	
50	Message	
51	SST	
52	PRGM	Cleared with ON
53	I/O	
54	PSE	
55	Printer	SET if connected

Section 8.1. Setting, Clearing and Testing FLAG Status

The driver coming to the construction site looked at the status of the workman's flag and then decided what to do next depending on whether it was raised or lowered. A FLAG in a computer program serves a similar function. It is a symbol whose status is tested, and depending on the results of the test, program flow is appropriately controlled. Much as the workman SET or CLEARed his signal flag depending on the position of the roadgrader, program FLAGS are either SET or CLEARed depending on what is happening to whatever function the numbered FLAG is designated to symbolize. For example, which calculation will be made next in a program may depend either on the status of the program itself, or on the results of a previous calculation. A FLAG can be used to determine either of these circumstances and will be used to direct subsequent program flow accordingly.

It is important to know first how FLAG status is controlled and tested before examining specific examples of how FLAGS are used. Any FLAG has only one of 2 statuses, it is either SET, or it is CLEAR. You SET any FLAG by using the SHIFTed function ("SF") of the key marked "7". To SET a FLAG, first turn the computer **ON** (the view window will display the current contents of the X STACK REGISTER), press the **SHIFT** key, then SF. Answer the ALPHA PROMPT "SF _ _" by **keying a 2 digit number** to indicate which FLAG you want to SET.

If you have correctly SET the FLAG, the view window will return to the display of the current contents of the X STACK REGISTER. If you respond to the ALPHA PROMPT "SF _ _" with a number which is not a correct designation for one of your computer's FLAGS (see Figure 8.1.), you will see the error message "NONEXISTENT". As in the sample programs shown later in this chapter, the SF function is not only activated directly from the keyboard, but also it can written into a program to be used in conjunction with either DIRECT or INDIRECT ADDRESSING techniques. INDIRECT ADDRESSING is described in the next chapter.

To CLEAR a FLAG is just as easy as to SET one. CLEARing a FLAG requires using the SHIFTed function ("CF") of the key numbered "8". First turn the computer **ON**, press the **SHIFT** key, then **CF**. Answer the ALPHA PROMPT "CF _ _" with a 2 digit number designating which FLAG you wish to CLEAR. If the operation was performed correctly, the contents of the X STACK REGISTER will be displayed again, if not, you will see "NONEXISTENT" in the view window. Similar to the SF function, CF can be activated either from the keyboard, or built into a DIRECT or INDIRECT ADDRESSING program statement.

Since FLAGS function to direct program flow automatically depending either on the status of your computer, or on a previous computation, it may be critical to know at any one time whether a specific FLAG in a program is either SET or CLEAR. The computer gives a direct indication in the VIEW WINDOW of the status of its first 5 FLAGS numbered 00 to 04. The digits 0, 1, 2, 3 or 4 appear in the view window just above the PRGM toggle switch when FLAGS 00, 01, 02, 03 and 04 have been SET. Each of these digits disappears when a corresponding FLAG is CLEARed. Different procedures are required to determine the status of any of the other FLAGS, but it's easy, and involves simply using a SHIFTed function of the key numbered "9".

To determine the status of any FLAG, including FLAGS 00 to 04, turn the computer **ON**, press **SHIFT**, then **FS?**, and respond to the ALPHA PROMPT "FS?_ _" with a 2 digit number indicating the FLAG whose status you which to determine. Keying a 2 digit number for which there is no FLAG (see Figure 8.1.), will result in "NONEXISTENT" being displayed in the view window. Otherwise you will see "YES", if your designated FLAG is SET, or "NO", if it is not. In keying FS?nn, you are in fact asking, "Is FLAG nn set?".

When the workman used his flag at the construction site, the signaling device itself had no intrinsic meaning. But it was valuable as a symbol for designating what was happening in the operation he was monitoring at the time. Similarly in a computer program, a FLAG by itself will not directly trigger a particular function, but it is valuable as a signaling device to designate the status of whatever operation is chosen for it to monitor. It will serve to indicate a condition. To a great extent it's up to you what condition you want any FLAG to indicate.

A decision is made as to what happens next in the program depending on whether a certain FLAG is either SET or CLEAR. The status of a FLAG is tested, and program flow is directed exactly in the same way that you have learned to use for the conditional test X=0? (see Figure 6.4.). As you may recall from the SUM4 program, the condition of whether the number stored in the X STACK REGISTER is equal to zero is tested, and if it is, the next line of the program is executed. If it is not, the next line of the program is skipped, and the following line is executed. It was in this way that program flow in SUM4 was directed either to request additional data for calculation, or for the program to display the result of its calculation. Other conditional tests operated in a similar way in the other programs presented so far.

The conditional tests for FLAG status (SET or CLEAR) are performed in a very similar way to those described in Section 6.4., and program flow is directed in exactly the same way depending on the results of the test. For example, in response to the conditional test FS?00, if FLAG 00 is set, then the next line of the program is executed, but it will be skipped if FLAG 00 is not SET. This operation is demonstrated for you in examples to follow. You will see that the "Do-if-True" rule works in connection with the conditional tests determining whether a FLAG is SET or CLEAR, just the same way as it does for X=0?, X<Y?, and the other conditional tests involving the X STACK REGISTER.

Some of the conditional tests described in Section 6.4., (such as, X=Y? and X=0?) were keyed as program lines simply by using the SHIFTed functions of appropriate keys shown of the keyboard. Others (such as, X<Y? and X≠Y?) required entering the program line using an XEQ ALPHA (symbols for the conditional test) ALPHA statement. Similarly, FLAG status can be tested in another way than the FS?nn designation on the keyboard above the key numbered "9". You can test whether a FLAG is CLEAR (by FC?nn), in addition to testing whether it is SET (by FS?nn). As shown by examples presented later, whether FS?nn or FC?nn is the appropriate conditional test to make for a program condition is determined by the context of the program itself. Several examples to come will illustrate this.

Asking the question if a particular FLAG is set directly from the keyboard, or writing a program line to construct the conditional test FS?nn required only using the SHIFTed function of the key numbered "9". Writing a program line for FC?nn is quite different. To try it, turn the computer **ON**, press **XEQ**, then **ALPHA**, next key "**FC?**" and press **ALPHA** again. **Key 2 numbers** in response to the ALPHA PROMPT "FC?_ _" to indicate the FLAG number whose status you want to check. You will receive either the message "YES", indicating your designated FLAG is CLEAR, or "NO", showing it is not. If a particular FLAG is tested to be not CLEAR, then, of course, it must be SET, and vice versa.

Special attention is required to use FLAGS accurately. As shown in Figure 8.1., the status of some FLAGS is retained even though the computer is turned off, whereas that for others is altered when you press the ON toggle. Also, some FLAGS designate specific information and can be used for no other purpose. For example, FLAG 55 is automatically SET when a printer is connected to your computer, and is CLEAR when no printer is used. Advantage will be taken of the information provided by the status of FLAG 55 in a program designed either to display or print interim and final calculations (a sample program is presented later on), but normally this FLAG can be used for no function other than to indicate whether a printer is an available accessory or not.

The program presented in the next section demonstrates how FLAGS are used not only for generally controlling the flow of calculations, but also for determining when certain data are required to be entered from the keyboard. How program flow will be directed and which ALPHA PROMPTS will be displayed will depend on the status of the computer and what calculations have already been made. Using FLAGS makes this type of conditional testing and control quite straightforward and easy, and provides an extremely powerful tool for writing programs.

Section 8.2. FLDEMO

The program outlined in Figures 8.2. to 8.4. is designed to demonstrate how FLAGS are used to control program flow. The program is described first in general terms, then reviewed for specific numerical calculations. As in many of the earlier examples, the program FLDEMO is presented to be arithmetically simple so that programming points can be made more clearly.

FLDEMO (Figure 8.2.) is constructed to calculate either a value for A using Equation 1, or one for E using Equation 2. Each of these equations involves adding only 3 numbers (symbolized by the letters B, C and D for equation 1, and B, F and G

Figure 8.2.

FLDEMO

Problem Statement:

This program is designed to calculate the values A and E in the listed equations using FLAGs so that constants need to be entered only once in a calculation series.

Equations:

$$A = B + C + D \qquad (1)$$

$$E = B + F + G \qquad (2)$$

Where:

A, E = values to be calculated
B, C, F = constants for single calculation
D, G = variables in all calculations

FLAG Designations:

FLAG	If SET	If CLEAR
00	B Stored	no data
01	C Stored	no data
02	F Stored	no data

Data Storage:

REG	Data
00	B
01	C
02	F

Figure 8.3.

Flow Diagram: FLDEMO

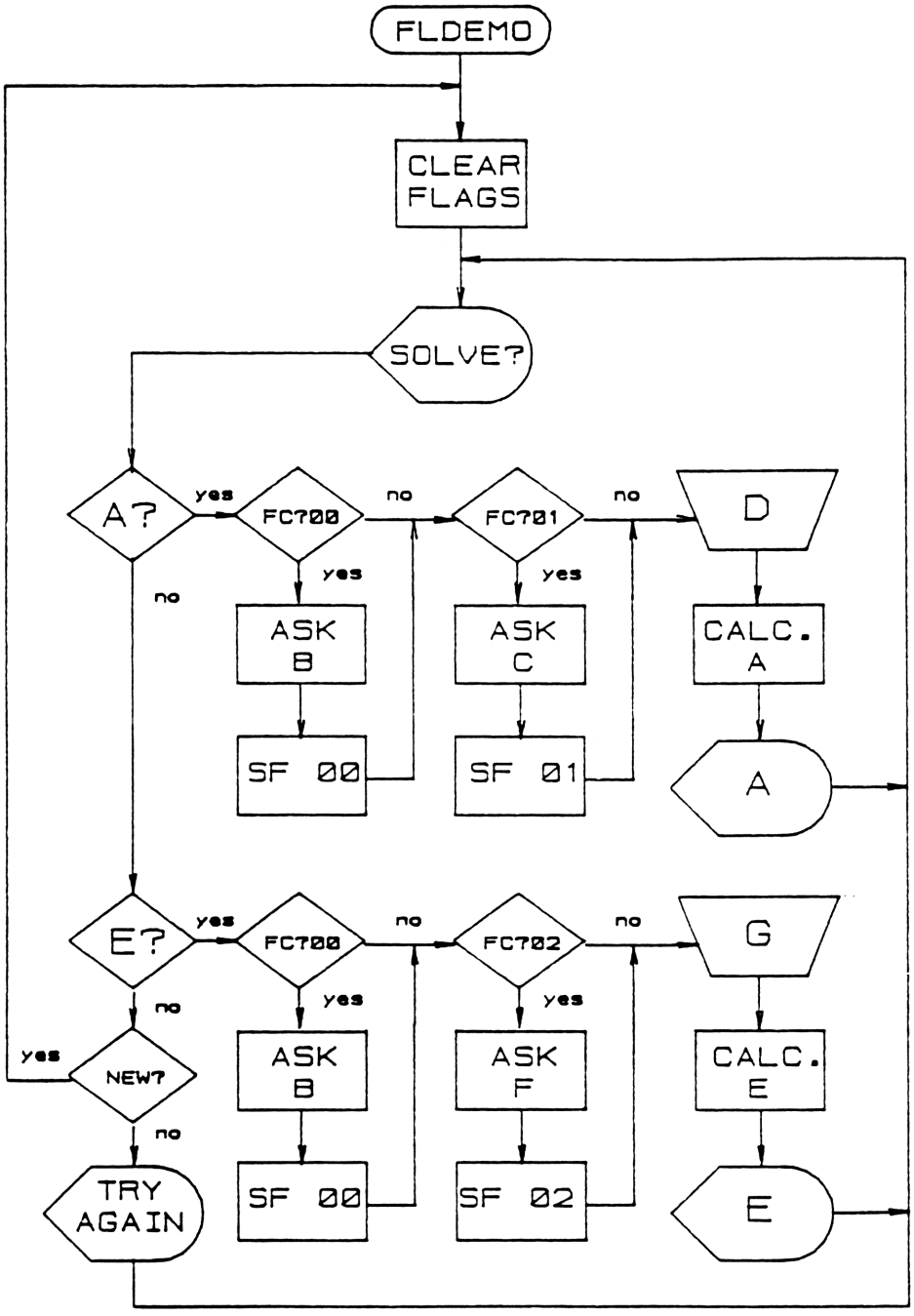


Figure 8.4.

Program Listing: FLDEMO

```
01 LBL "FLDEMO"
02 BEEP
03 CF 00
04 CF 01
05 CF 02
06 LBL 00
07 TONE 5
08 AON
09 "SOLVE?"
10 PROMPT
11 ASTO X
12 AOFF
13 "A"
14 XEQ 01
15 X=Y?
16 GTO 02
17 "E"
18 XEQ 01
19 X=Y?
20 GTO 03
21 "NEW"
22 XEQ 01
23 X=Y?
24 GTO "FLDEMO"
25 TONE 9
26 "TRY AGAIN"
27 AVIEW
28 PSE
29 GTO 00
30 LBL 01
31 ASTO Y
32 RTN
33 LBL 02
34 FC? 00
35 XEQ 04
36 FC? 01
37 XEQ 05
38 RCL 00
39 RCL 01
40 +
41 TONE 5
42 "ENTER D"
43 PROMPT
44 +
45 TONE 9
46 "A="
47 ARCL X
48 AVIEW
49 PSE
50 GTO 00
51 LBL 03
52 FC? 00
53 XEQ 04
54 FC? 02
55 XEQ 06
56 RCL 00
57 RCL 02
58 +
59 TONE 5
60 "ENTER G"
61 PROMPT
62 +
63 TONE 9
64 "E="
65 ARCL X
66 AVIEW
67 PSE
68 GTO 00
69 LBL 04
70 TONE 5
71 "ENTER B"
72 PROMPT
73 SF 00
74 STO 00
75 RTN
76 LBL 05
77 TONE 5
78 "ENTER C"
79 PROMPT
80 SF 01
81 STO 01
82 RTN
83 LBL 06
84 TONE 5
85 "ENTER F"
86 PROMPT
87 SF 02
88 STO 02
89 RTN
90 END
```

for equation 2), and both use the same number (variable B) in making a calculation. The program makes several assumptions about entered data. For example, it assumes in using Equation 1 that once the program is initialized and numbers for B and C have been entered, they are the same for all subsequent calculations until the program is started over again by XEQ "FLDEMO". Variable D, however, will change for each calculation whether the program is started over again or not. Also, it is assumed in using Equation 2 that once the program is initialized and the numbers for C and F have been entered, they are the same for all subsequent calculations. Variable G, however, will change for each calculation. New values for all variables are required when the program is started over again.

Figure 8.2. shows which PRIMARY DATA STORAGE REGISTERS will be used to store data entered for variables B, C and F. It also shows which FLAGS are used to signal whether data have been entered for these variables. It is convenient in designing this program to use similarly numbered FLAGS and REGS for correspondingly numbered data. For example, When variable B is entered, FLAG 00 will be set, and the number for B will be stored in REG 00. Correspondingly numbered data storage locations and FLAGS, of course, are unnecessary. Data can be stored in many different locations and many different FLAGS can be used with no requirement for them to be numerically related. It is essential, though, that one remember what is stored where and what FLAGS are used to signal what program status. Record forms similar to those shown in Figure 5.7. are valuable for keeping all this information straight.

The flow diagram for FLDEMO is shown in Figure 8.3. Since the choice has been made (see Figure 8.2.) to SET FLAGS 00, 01 and 02 to designate when data for variables B, C and F have been entered, it is important these FLAGS are CLEAR at the beginning of a new calculation. A sequence of statements "CF00, CF01, CF03" early in the program assures this initial status. Continuing with the program flow shown in Figure 8.3., an ALPHA PROMPT is next presented to ask whether the calculation of A (using Equation 1) or E (using Equation 2) is to be made, or whether a new calculation is required.

If A is selected for solution, Equation 1 will be used which requires data for B, C and D. The program first determines the status of FLAG 00. If FLAG 00 is clear (as it would be, of course, for the first calculation), then data are requested for variable B. When FLAG 00 is SET for a second calculation (indicated by its not being CLEAR at the conditional test of FC?00), it signals that data have already been entered for variable B and stored in REG 00. Data for B are then not requested.

In the solution of A, program flow next determines the status of FLAG 01. If FLAG 01 is clear, indicating no data have yet been entered for variable C, these data are requested. If FLAG 01 is not CLEAR (and therefore must be SET with data for variable C stored in REG 01), then no data for C are requested, and equation 1 can be solved once a number for variable D has been entered. No FLAG is used to determine if variable D has been entered, since one of the assumptions in this program is that data for variable D need to be renewed for each calculation. After Equation 1 has been solved, the calculation of A is displayed, and the program controls for the ALPHA PROMPT "SOLVE?" to be displayed again.

If Equation 2 is chosen for calculation, the selections for data entry for variables B and F follow a similar pattern as in the evaluation of Equation 1. Also similar to program design for solving Equation 1, no FLAG is used to determine if variable G has been entered, since this is obligated to be a new number each time a calculation is made for Equation 2. After Equation 2 has been solved, the calculation of E is displayed, and program returns to place the ALPHA PROMPT "SOLVE?" in the view window.

If a new calculation is required, "NEW" is keyed into the computer in response to the request "SOLVE?", and program flow is directed to the program's GLOBAL LABEL from which first FLAGS 00, 01 and 02 are CLEARed, and then "SOLVE?" is displayed again. Were some letter, word or symbol other than "A", "E" or "NEW" keyed into the computer in response to "SOLVE?", program design would display "TRY AGAIN", then display "SOLVE?" once more.

Following the flow diagram in Figure 8.3., predict how data requests would be sequenced the first time the program FLDEMO is run. When the program is initialized, "SOLVE?" asks which calculation is to be made. If "A" is keyed, then "ENTER B?" would be displayed, and after a number had been keyed in response to this ALPHA PROMPT and R/S is pressed, "ENTER C" would be displayed. When a number is keyed and entered with R/S, "ENTER D" is displayed. As soon as a number for D is entered, A is calculated and the solution displayed, then after a PAUSE, "SOLVE?" is once again in the view window. If "A" is then keyed again and entered with R/S, only "ENTER D" would be displayed, and when a number has been entered, the solution for A is displayed, and "SOLVE?" appears again. Thanks to the option of being able to use FLAGS, the computer determines for itself what information it has already on hand for a calculation, and can "decide" whether or not to ask the operator for data.

As the solution of A is made the second time, data for B and C are not required to be entered, since they are retained in memory (REGS 00 and 01), and FLAGS 00 and 01 are SET to indicate this status. After A is calculated again, were "E" keyed in response to "SOLVE?", "ENTER F" would appear at the first data request, since data for B have already been entered for the earlier calculation of A. Once data for F (and

then G) have been entered and Equation 2 solved for E, subsequent solutions of Equation 2 would begin with "ENTER G", because data for variables B and F are now in storage (REGS 00 and 02), and FLAGS 00 and 02 are correspondingly SET to indicate this status and to direct program flow accordingly.

Try a numerical example for FLDEMO. To get started, key XEQ "FLDEMO" and in response to "SOLVE?", press A then R/S. When "ENTER B" is displayed, key 13.6, press R/S. When "ENTER C" is displayed, key 12.9, press R/S. Key 10.2 and press R/S in response to "ENTER D". A is calculated as 36.7 and after a pause, "SOLVE?" is displayed again. Press A, then R/S. The computer now asks only for a new value of D. Key 19.6 and press R/S to see the calculation of A as 46.10. Selecting A for a third calculation and keying 14.0 in response to "ENTER D" gives a new solution of A as 40.5. All subsequent calculations of A would require only new values of D.

Keying E in response to "SOLVE?" displays "ENTER F". Key 19.1, press R/S, key 12.0 in response to "ENTER G", then press R/S to see the first calculation of E as 44.7. When "SOLVE?" is displayed again, press E, then R/S and enter a value for G of 18.4 using R/S and see the second calculation of E as 51.1 in the view window. Press E, then R/S, next key 4.5, press R/S and read 37.2 as the next solution of E. All subsequent calculations of E will require only new values of G. FLAGS did your thinking for you about previous data entries and about what ALPHA PROMPTS were required for new data.

The reader is encouraged to develop simple programs using FLAGS which would not only follow the sequence of data entry, but will also make appropriate choices for appended units in, for example, the calculation of an equation solving for temperature which could be displayed as, "T=NN.NN C" or "T=NN.NN F" in the same program. For another example, write 2 programs which would calculate A=B+C if B is less than 50, but will calculate A=B/C if B is greater than 50, and will calculate A=B*C if B=50. Write one program which uses conditional tests to compare numerical values for B, but write the second which uses FLAGS to make the required decisions.

How most of the program lines are keyed for the FLDEMO program (Figure 8.4.) is already familiar from earlier sample programs. Also, it should be clear from Section 7.3. how LOCAL LABELS 04,05 and 06 are structured as subroutines to request the input of numerical data. It may not be apparent, however, how program lines are constructed to receive an ALPHA statement in response to the data request "SOLVE?". Different from a request for numerical data, the flow diagram in Figure 8.3. indicates that the ALPHA PROMPT "SOLVE?" asks that the letters "A" or "E", or the word "NEW" be entered.

In anticipation of the entry of ALPHA, rather than numerical data, the instruction at line 08 places the computer into its ALPHA mode automatically as this step is executed in program flow. The statement "AON" (keyed as XEQ ALPHA AON ALPHA), indicating "Alpha On", performs the same function as were you to press the ALPHA toggle switch, and appropriately redefines each key (see Figure 2.4.). Whatever ALPHA symbol is entered in response to "SOLVE?" by first pressing corresponding keys (then R/S) is stored as an ALPHA statement in the X STACK REGISTER (by ASTO X at line 11), and first tested against the ALPHA symbol "A" (entered at line 13 and stored as an ALPHA statement in the Y STACK REGISTER by the instructions in LBL 01) using X=Y? at line 15. This conditional test compares the contents of the X and Y STACK REGISTERS, whether they contain either numerical data, or ALPHA statements. After the ALPHA symbols A, E, or NEW are stored in the Y STACK REGISTER (using ASTO, of course), the computer is taken out of the ALPHA mode automatically by the instruction "AOFF" at line 34.

If the X and Y STACK REGISTERS contain identical data (alpha or numerical), the next line of the program is executed (which in this example takes the program to

LBL 02 where the calculation of Equation 1 is begun), or if they are not identical, line 16 is skipped. The symbol "E" is then entered into the Y STACK REGISTER and tested against whatever ALPHA statement had been previously stored in the X STACK REGISTER. If "E" had not been entered in response to "SOLVE?", the program tests to see if "NEW" had been the response to the first ALPHA PROMPT. If not, "TRY AGAIN" is displayed for a second or two before "SOLVE?" is again in the view window.

Reviewing the program FLDEMO in algebraic terms is one thing, seeing it in action with the entry of numerical data is something else, and each type of review shows different features of the program's design. After you have keyed the program steps listed in Figure 8.4., try running the program several times with different numbers, note how data are requested, then key "NEW" in response to "SOLVE?" and run the program again. Using such a simple program as FLDEMO to understand how FLAGS are used to control the sequence of data requests, calculations and answer displays will be valuable in understanding how the technique of FLAGS can be of value in designing your own more complicated programs.

Section 8.3. Using "Non-User Defined" FLAGS

As shown in Figure 8.1., FLAGS 00 to 10 are considered to be "general purpose user" FLAGS, and are available for discretionary use. Whether you SET or CLEAR them by keyboard operations or by appropriate instructions in program lines, their status is retained by the computer's continuous memory, and you can depend on their being in the same SET or CLEAR position when you turn your computer ON a second time. In contrast, FLAGS 11 to 20 are "special purpose user" FLAGS, and although you can SET or CLEAR them, they are automatically CLEARed each time the computer is turned ON. Figure 8.1. lists other FLAGS in your computer over whose status you have only indirect control, but which you can use, nonetheless, to your advantage in program design. The following sample programs illustrate uses of these options.

Programs LOCK1, LOCK2, LOCK3 and LOCK4 listed in Figure 8.5. illustrate the use of one of the "special purpose user" FLAGs (FLAG 11) for different ways to prohibit others from using your computer. Each program uses FLAG 11 which controls the automatic execution of a program in the computer's MAIN MEMORY when it is first turned on. Each of these programs lists "SF 11" as an instruction line, and each uses the built-in program "OFF" to turn off the computer automatically, but different flow in each produces quite different results as each program is initialized. LOCK1 is the simplest of these examples.

None of the lines in the short LOCK1 program should be difficult to key-in, if you have been able to enter previous programs. Note that the instruction "OFF" in line 3 (keyed as XEQ ALPHA OFF ALPHA) is not the same as the instruction AOFF used at step 34 in the FLDEMO program (see Figure 8.4.). AOFF brought about triggering off the ALPHA toggle to take the computer out of the ALPHA mode, whereas the instruction OFF turns off the computer itself.

After LOCK1 has been loaded as a program into your computer, execute the program (by XEQ "LOCK1"), and watch the view window. As soon as you have pressed ALPHA as the last instruction for executing the LOCK1 program, the computer automatically is turned off and the view window goes blank. Press the ON toggle in an attempt to use your computer again, and note that except for a brief display of the PRGM annunciator, the view window remains blank, and in fact the computer remains off. The computer has now been disabled by the LOCK1 program.

Figure 8.5.

Program Listing: The LOCK Programs

LOCK1:

```
01 LBL "LOCK1"
02 SF 11
03 OFF
04 GTO "LOCK1"
05 END
```

LOCK2:

```
01 LBL "LOCK2"
02 LBL 00
03 "    LOCKED"
04 AVIEW
05 PSE
06 SF 11
07 OFF
08 GTO 00
09 END
```

LOCK3:

```
01 LBL "LOCK3"
02 SF 11
03 "YOU BROKE IT"
04 AVIEW
05 PSE
06 OFF
07 GTO "LOCK3"
08 END
```

LOCK4:

```
01 LBL "LOCK4"
02 LBL 00
03 "TOUCH CODE"
04 AVIEW
05 123
06 PSE
07 X=Y?
08 GTO 01
09 TONE 9
10 "    LOCKED"
11 AVIEW
12 CLX
13 PSE
14 SF 11
15 OFF
16 GTO 00
17 LBL 01
18 TONE 9
19 CLX
20 "RDY FOR USE"
21 AVIEW
22 PSE
23 CLD
24 END
```

You have 3 choices in being able to get a computer that works again. One way is to buy a new computer. A second way is to send $50 in unmarked bills of any negotiable currency to the address listed at the front of this book to receive a code breaker program. The third way is to read the next sentence. Hold down the R/S key while pressing the ON toggle allows you access again to your keyboard. Either pressing the R/S key again, or executing the LOCK1 program again once more secures your computer.

When the LOCK1 program is operating, having FLAG 11 set is a signal to activate automatically whatever program is on-line when the ON toggle is pressed. In the case of the LOCK1 program, operation is only to turn off the computer. The LOCK2 program operates in a similar fashion to LOCK1, but rather than just turning off the computer, the message "LOCKED" is first displayed in the view window for a short time. The LOCK4 program displays a more intimidating message, and the

program is configured somewhat differently than LOCK2, but its effect is similar.

LOCK4 uses FLAG 11 in a similar way to the other LOCK programs, but adds several new features. When LOCK4 is executed, the message "TOUCH CODE" is displayed, and for a short time the keyboard is open to accept either a numeric or an alpha entry. After a second or two, whatever has been entered into the X STACK REGISTER is compared (X=Y? in line 7) to the contents of the Y STACK REGISTER. In the version of the LOCK4 program shown in Figure 8.5., the number "123" is in the Y STACK REGISTER to serve as the basis for this comparison. If the number "123" has been entered in response to the alpha prompt "TOUCH CODE", then program flow branches to LBL 01, the message "RDY FOR USE" is displayed momentarily, then with the display of the contents of the X STACK REGISTER, the computer is ready for normal use.

If either some number other than "123", or if an ALPHA statement had been entered during the access time following "TOUCH CODE", the message "LOCKED" is displayed for a moment, and then the computer is turned off. Each time the computer's ON toggle is pressed, "TOUCH CODE" will be displayed and the computer will once again automatically turned off until the correct "combination" is entered. Line 5 can be rewritten to contain any number you wish to use for your own "combination lock" of the computer. Adding additional lines similar to line 6 (PSE) provides more time to enter the lock-breaking code. It might be interesting to incorporate AON and AOFF statements in the program so its combination is an entered word rather than a number.

Section 8.4. STAT

The STAT program is presented to demonstrate how functions built into your computer (see Figure 3.4. for a listing of CATALOG 3) are tapped by a program, and how a non-user defined flag (FL 55; see Figure 8.1) is used to control the flow of solutions to equations. A statement of the problem for STAT and its equations are shown in Figure 8.6. Figure 8.7. lists the flow diagram, the program list and a sample problem.

The STAT program not only uses the computer's built-in statistical registers, but also depends on its MEAN and SDEV functions listed in CATALOG 3. In order to use these functions and the STAT program, first provide the storage register space to accept interim calculations (XEQ "SIZE" 017). The instruction at program line 5 (Figure 8.7.) is a necessary first step for clearing the statistical registers. Data are entered from the keyboard in response to the ALPHA PROMPT at line 8, and whatever number is entered, it is first tested (at line 10) to see if it is equal to zero. If it is not, program flow continues at line 12 to include the number in the statistical calculations. (Hint: The statement at line 12 is keyed as: XEQ, "SHIFT, Σ, SHIFT, +". The symbol "Σ" is obtained using the SHIFTED function of the X◇Y key; see Figure 2.4.)

Program flow is then directed back to the LOCAL LABEL 00 at line 6 by GTO 00 at line 13, where a request is displayed to enter the next piece of data. This sequence continues for the entry of each of Karen's measurements, until all of them have been entered. She then enters a zero in response to "ENTER DATA". Since the conditional test X=0? at line 10 is now answered "yes", program flow is directed to LBL 01 where calculations begin using the data which had been previously entered into the statistical registers. The first of these calculations is to use the function MEAN listed in CATALOG 3 (see Figure 3.4.). The statement at line 15 is keyed as: XEQ, "MEAN". The calculation of SDEV (line 19) also uses a function in CATALOG 3, and is written into the program in an analogous way.

The calculation of the statistic "standard error" cannot depend directly on a

Figure 8.6.

STAT

Problem Statement:

Karen Kareful is a production control engineer who needs to determine from time-to-time how uniformly long are pieces of material cut automatically by the many different machines under her supervision. She wrote the program STAT to provide this information. It calculates the average (MEAN), standard deviation (SDEV) and standard error (SE) for an indefinite number of samples, and also displays the number (N) of items she has included in her test batch. Sometimes Karen makes these calculations at her desk where she has an HP-82143 printer, but often she needs to have these data right at the machines themselves where the printer is not available.

Equations:

$$MEAN = \frac{N_1 + N_2 \ldots + N_L}{N}$$

$$SDEV = \sqrt{\frac{\sum X_1^2 - N_X^2}{N - 1}}$$

$$SE = \frac{SDEV}{\sqrt{N - 1}}$$

Where:

$N_1, N_2, \ldots N_L$ = data entries
N = number of data entries
MEAN = sample average
SDEV = standard deviation of sample
SE = standard error of sample

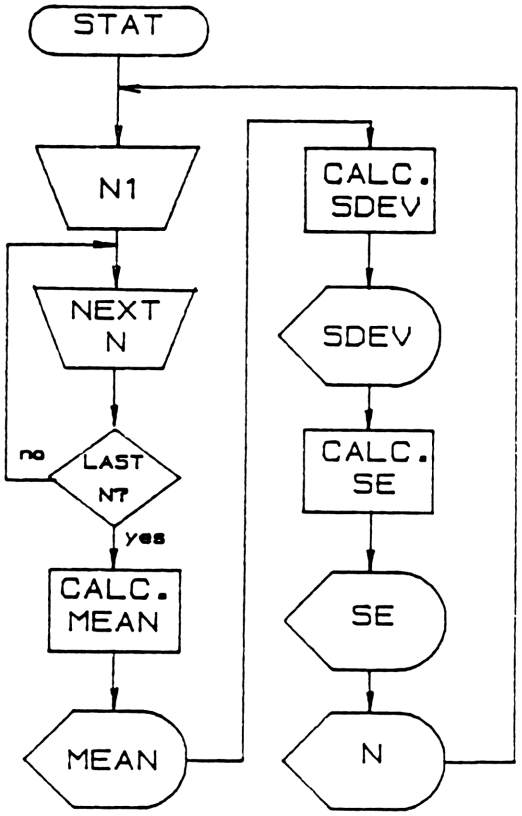
function in CATALOG 3, but it is easy enough to do since SDEV has just been determined. Obtaining data to display the number of test pieces measured (N) extracts information from REG 16 which is used by the built-in statistical program.

As indicated in the problem statement in Figure 8.6., sometimes the STAT program is used in association with a printer, but often one is not available. The program's construction allows for this contingency, and depends on one of the non-user defined FLAGS of the computer to determine how the solution to calculations will be displayed. Figure 8.1. shows that FLAG 55 is either SET or CLEAR depending on whether or not a printer has been connected through one of the computer's

Figure 8.7.

Flow Diagram and Program List: STAT

Flow Diagram:



Program Listing:

```
01 LBL "STAT"
02 FIX 2
03 BEEP
04 CLRG
05 CL Σ
06 LBL 00
07 TONE 5
08 "ENTER DATA"
09 PROMPT
10 X=0?
11 GTO 01
12 Σ+
13 GTO 00
14 LBL 01
15 MEAN
16 XEQ 02
17 "MEAN="
18 XEQ 03
19 SDEV
20 CLA
21 XEQ 02
22 "SDEV="
23 XEQ 03
24 XEQ 02
25 RCL 16
26 1
27 -
28 SQRT
29 /
30 "SE="
31 XEQ 03
32 FIX 0
33 CLA
34 RCL 16
35 XEQ 02
36 "N="
37 XEQ 03
38 GTO "STAT"
39 LBL 02
40 TONE 9
41 TONE 9
42 RTN
43 LBL 03
44 ARCL X
45 AVIEW
46 FS? 55
47 RTN
48 STOP
49 RTN
50 END
```

Sample Problem:

item	length (in)
1	39.21
2	38.98
3	39.12
4	39.89
5	39.00

MEAN = 39.24
SDEV = 0.38
SE = 0.19
N = 5

accessory ports. The conditional statement FS?55 at line 46 of the STAT program (Figure 8.7.) tests for the presence of the printer. If one is connected, each solution is printed, and the program automatically and without stopping progresses to the next calculation in the series. The program stops only at the PROMPT statement (line 9) when it has been reinitialized by the statement at line 38.

If a printer is not connected at the time of a calculation, the conditional statement FS?55 is not true (FLAG 55 is CLEAR), and the STOP statement at line 48 holds the solution of the previous calculation in the view window until Karen has time to read and record it. She then presses the R/S toggle key, and the program starts to run again. The next calculation is then made, and its solution is displayed continuously until R/S is once again pressed. At the end of the last calculation, pressing R/S returns the program to its own GLOBAL LABEL from which a new set of data for a subsequent calculation are requested. After the program STAT is keyed into the MAIN MEMORY, try the sample problem shown in the insert in Figure 8.7.

This is a good time to go back and read Section 2.3. if you skipped it earlier. Among other things, it describes how the USER mode of the computer provides another option for calculating MEAN and SDEV. Using both "general purpose user flags" (FLAGS 00 to 10), "special purpose user flags" (FLAGS 11 to 20) and FLAGS with designated functions (FLAGS 21 to 55) provide considerable power and control in writing programs. Along with INDIRECT ADDRESSING techniques described in the next chapter, they allow not only for substantial flexibility in program design, but also considerable efficiency in program flow.

Section 8.5. FLAGS at Work

FLAGS in a program work at their best when they take over complicated sorting and decision making tasks. The more complicated the set of problems, the more valuable are the functions which FLAGS perform. The sample program entitled, "BILL" described in Figure 8.8. shows how FLAGS might be used to take charge of routine, but involved decision making.

Similar to the FLDEMO program (Figures 8.2. and 8.4.) advantage is taken in the BILL program of both FLAGS and PRIMARY STORAGE REGISTERS as HP-41 features. As an alternative to using the instruction CLRG to clear all data storage registers, a zero is stored (lines 03 to 05) only in those 2 REGISTERS (REGs 00 and 01) used in this program. Early in the program (lines 06 to 08) the initial status of FLAGS 00, 01, and 02 is guaranteed by selectively clearing them.

There is no built-in function in the HP-41 similar to CLRG which will clear all, or even more than 1, FLAGS. Were, for example, FLAGS 00 to 10 used in a program, the initial status of each would have to be set by "CF nn" or "SF nn" statements as individual program lines. The next chapter, though, gives examples of how INDIRECT ADDRESSING can be used to complete such a laborious process quickly, easily and with only a few lines of program instruction. Were FLAGS 11 to 20 used in a program, no specific statements would be required to set them to CLEAR when the computer is first turned on, as described in Figure 8.1.

Two kinds of data are requested in the first 37 lines of the program BILL. One is the familiar request for numerical data which will serve as the basis for calculations. These data are requested by instructions in lines 28 to 35. A new kind of data, though, are also required. These are the conditions under which calculations will be made, and are requested in lines 11 to 23. In contrast to the numerical data which are stored once they are entered into the HP-41 keyboard, data defining conditions are used to set one or more of the 3 FLAGS used in the program. Testing the status of each of these FLAGS serves to direct program flow appropriately for a correct calculation of the customer's costs.

The numerical calculations in BILL are simple. But the decisions to be made for the calculations are not, as described in the problem statement in Figure 8.8. FLAGS free the user of the program, though, from any jobs other than just answering data requests accurately. The job of keeping track of who pays how much under each of the conditions of repair in the sample problems (Figure 8.9.) is handled swiftly and competently by the program's use of FLAGS.

Figure 8.8.

FLAGS at Work

Problem Statement:

Tom Tappet has operated a prosperous automobile dealership and repair garage for 40 years. Part of his business success is because he provides special options for his customers in addition to his quality workmanship. On car maintenance, for example, he charges for labor, but not for parts if any repair is made under his 3 year warranty, regardless of the car's age. Also, he gives a 10% discount to repeat customers (those who've come to his shop more than 3 times in the past 2 years), but he charges a surcharge of 5% if a non-warranted repair is made on a car older than 5 years. If a customer's bill is greater than $450, Tom pays the 4% state sales tax. If it weren't for the HP-41 computer, such a complicated billing procedure would be awkward, but using the BILL program, it's easy, as shown by the examples in Figure 8.9.

Program Listing:

```
01 LBL "BILL"
02 FIX 2
03 0
04 STO 00
05 STO 01
06 CF 00
07 CF 01
08 CF 02
09 BEEP
10 AON
11 "WARR(Y/N)?"
12 XEQ 04
13 X=Y?
14 SF 00
15 TONE 5
16 "REPEAT(Y/N)?"
17 XEQ 04
18 X=Y?
19 SF 01
20 AOFF
21 TONE 5
22 "AGE(YR)?"
23 PROMPT
24 5
25 X≤Y?
26 SF 02
27 TONE 5
28 "LABOR?"
29 PROMPT
30 STO 00
31 FS? 00
32 GTO 06
33 TONE 5
34 "PARTS?"
35 PROMPT
36 ST+ 00
37 LBL 06
38 FS? 01
39 XEQ 01
40 FS? 01
41 XEQ 02
42 RCL 00
43 RCL 01
44 +
45 STO 00
46 450
47 X≤Y?
48 GTO 07
49 RCL 00
50 .04
51 *
52 RCL 00
53 +
54 STO 00
55 LBL 07
56 RCL 00
57 TONE 9
58 "BILL="
59 ARCL X
60 AVIEW
61 PSE
62 GTO "BILL"
63 LBL 01
64 RCL 00
65 .10
66 CHS
67 XEQ 05
68 RTN
69 LBL 02
70 FS? 00
71 RTN
72 RCL 00
73 .05
74 XEQ 05
75 RTN
76 LBL 03
77 RCL 00
78 .04
79 XEQ 05
80 RTN
81 LBL 04
82 PROMPT
83 ASTO X
84 "Y"
85 ASTO Y
86 RTN
87 LBL 05
88 *
89 ST+ 01
90 RTN
91 END
```

FLAG	If SET	If CLEAR
00	warranty	no warranty
01	repeat	new customer
02	age: 5 yrs+	age: <5 yrs

REG	Data
00	Accumulated bill
01	% adjustments

Figure 8.9.

Sample Problems: BILL

Problem No. 1:

Frank and Ann Stein bought a new car 3 years ago and have had a continuing problem with it stalling in traffic in hot weather. They've taken it to several garages, but the problem continues. They finally bring it to Tom who replaces a fuel injector part which costs $82.50 at a labor charge of $53.00. The car now runs fine. The bill was: $140.92. (Hint: "warranty": no; "repeat": no; "age": 3 yrs; customer pays tax)

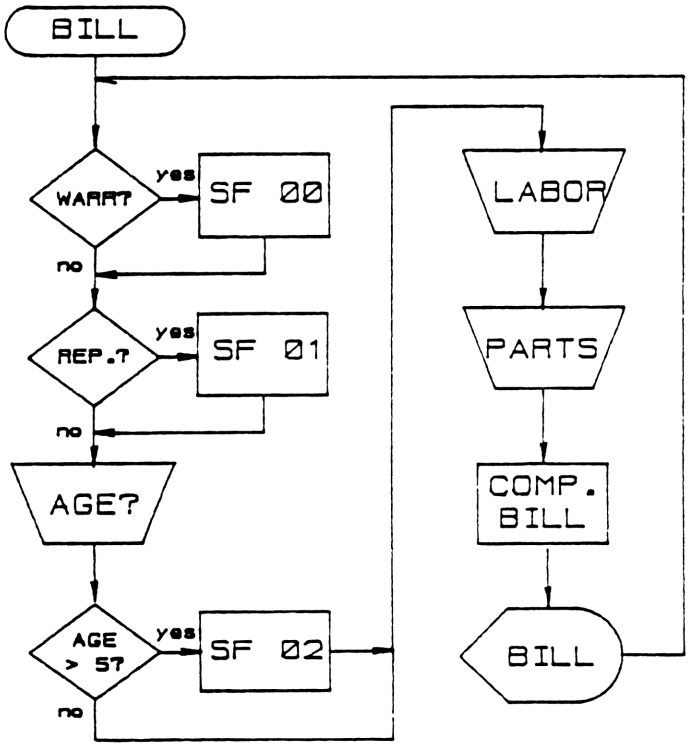
Problem No. 2:

Larry Lender, the town librarian, has taken his 1952 DeSoto to Tom's garage every 6 months for a check-up ever since he bought it new. It usually needs just a tune up, but this time a repair is required. The car is back on the road after replacing $620.10 worth of parts at a labor charge of $419.12. The bill is: $987.26. (Hint: "warranty": no; "repeat": yes; "age": 36 yrs; Tom pays tax)

Problem No. 3:

Mr. Lender's car is still running rough only 4 months after Tom's last repair of it. Another series of tests shows that a defective replacement part costing $65.00 had been used. It requires $32.50 in labor costs to install a new one. The bill is: $30.42. (Hint: "warranty": yes; "repeat": yes; no charge for parts, but customer pays tax)

Flow Diagram:



Chapter 9

Indirect Addressing

The best part has been saved for last. The technique of INDIRECT ADDRESSING is powerful and interesting. It gives many advantages both for the efficient use of program space in the computer and for how rapidly a program can be run. It is, for example, an economical and useful way to input, store, retrieve and perform an operation on a sequence of numbers or alpha data. Also, it can be used to perform arithmetic, branches and flag test operations. This chapter shows how such operations are achieved using a repetitive string of similar instructions in which INDIRECT ADDRESSING statements provide the controlling information.

Some people find INDIRECT ADDRESSING to be complicated and confusing when first encountered. But they find after a little practice that the technique is not all that hard to use and it soon becomes indispensable, especially as program construction grows to be long and complicated.

INDIRECT ADDRESSING would have had a valuable application, for example, in the SOLVX program shown in Figure 7.2. This program required the entry of 6 independent variables (N1 to N6) which were requested by a series of alpha prompts. The way this program was structured for Figure 7.2. obligated using 26 program lines for data entry. You will learn in the next few sections how to input data like this with many fewer program lines using INDIRECT ADDRESSING. Also, you will see how programs involving other sequential operations like setting and clearing flags, storing data in STACK REGISTERS and in PRIMARY STORAGE REGISTERS are shortened by INDIRECT ADDRESSING. By no means least, you will learn, too, how to use the EXTENDED DATA STORAGE REGISTERS. You may recall from section 3.3. that data are stored in and retrieved from the EXTENDED DATA STORAGE REGISTERS only by the procedure of INDIRECT ADDRESSING.

How INDIRECT ADDRESSING works in a program will be easier to see if one first forms a concept of how an analogous process is used in everyday life. A form of INDIRECT ADDRESSING takes place, for example, when a letter is sent to someone with an "in-care-of" statement on the envelope. If the letter is addressed, "Person B, c/o Person A", Person A is the one to whom the letter is sent directly and who is relied upon to relay it secondarily, or indirectly, to Person B. In a similar fashion, Person A's location could serve as a directly addressed site to send letters indirectly to Persons C, D, E, etc. were envelopes labeled, "Person C (D, E, etc.), c/o Person A".

In each example, Person A's address serves as the direct one, and Persons B, C, D, E, etc. are INDIRECTLY ADDRESSED through it. There is a sequence of 2 instructions in each example. The first is to identify the DIRECT ADDRESS ("Person A") and the second is to designate the location for the INDIRECT ADDRESS ("Person B, C, D", etc.). A similar convention will be used in writing program lines for INDIRECT ADDRESSING.

Section 9.1. Manipulating the Contents of the PRIMARY STORAGE REGISTERS using DIRECT ADDRESSING and INDIRECT ADDRESSING

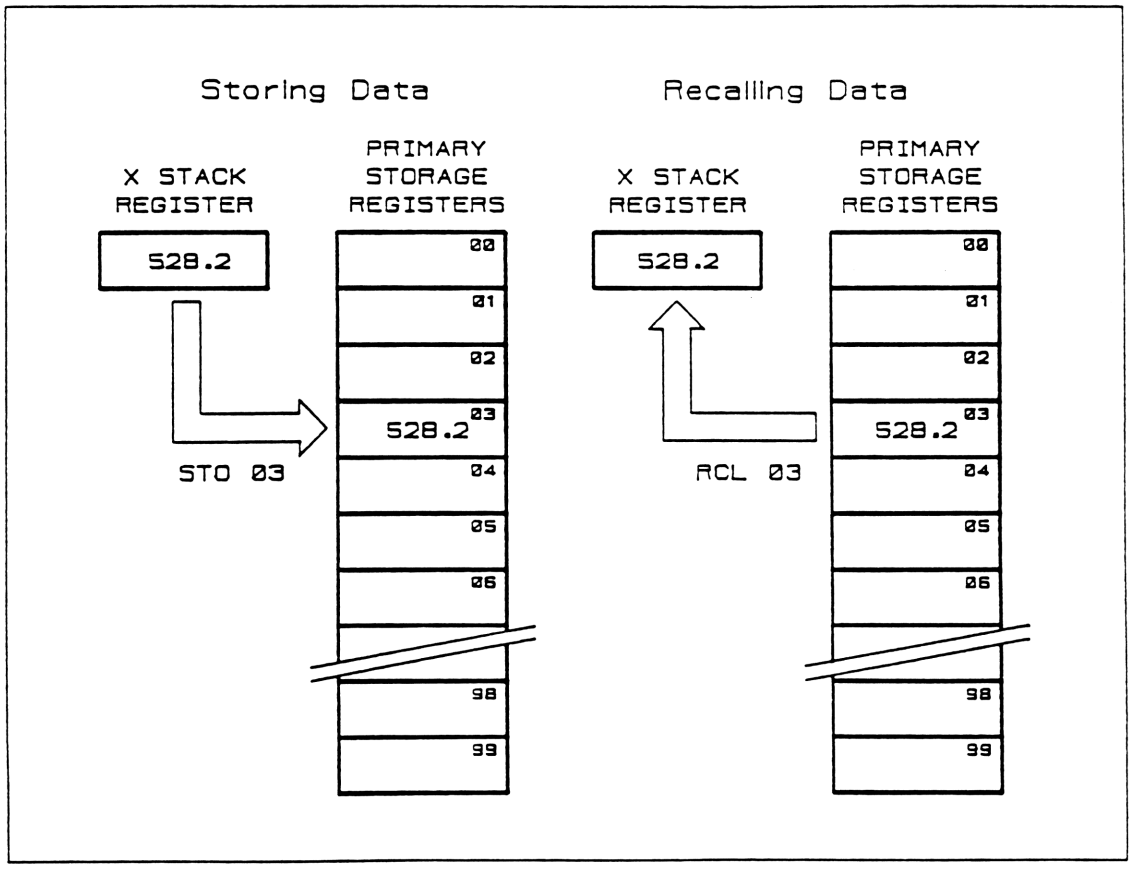
The first part of this section provides a short review of the techniques for storing and recalling data using DIRECT ADDRESSING techniques. You learned from section

3.1.2. how to enter data directly in the STACK REGISTERS by STO ST nn, and learned from Section 3.1.4. how to recall data from those memory locations (using RCL ST nn). Also, you learned how to store and recall data from the PRIMARY STORAGE REGISTERS using STO nn and RCL nn instructions, as well as how to perform arithmetic operations using ST+ nn, ST- nn, ST* nn and ST/ nn. Figure 9.1. summarizes the bases for these operations of DIRECT ADDRESSING of the PRIMARY STORAGE REGISTERS.

A short exercise: First create 10 PRIMARY STORAGE REGISTERS from the MAIN MEMORY by keying XEQ "SIZE" 011. To follow the procedure summarized in Figure 9.1., first **key** a number, "528.2" for example, into the X STACK REGISTER. The next step in Figure 9.1. is the instruction **STO 03** which copies the number from the X STACK REGISTER to store it in REG 03. Erasing the contents of the view window and completing the instruction **RCL 03** copies the number stored in REG 03 back into the X STACK REGISTER. This process has been used several times in the storage and recall of numbers in the PRIMARY STORAGE REGISTERS in many of the previously presented examples. The number manipulated in this way remains, of course, both in X STACK REGISTER during the storage operation and in REG 03 during the data recall operation. Only a copy of it is transferred to the designated new location. The procedures summarized in Figure 9.1. are examples of DIRECT ADDRESSING techniques.

Figure 9.1.

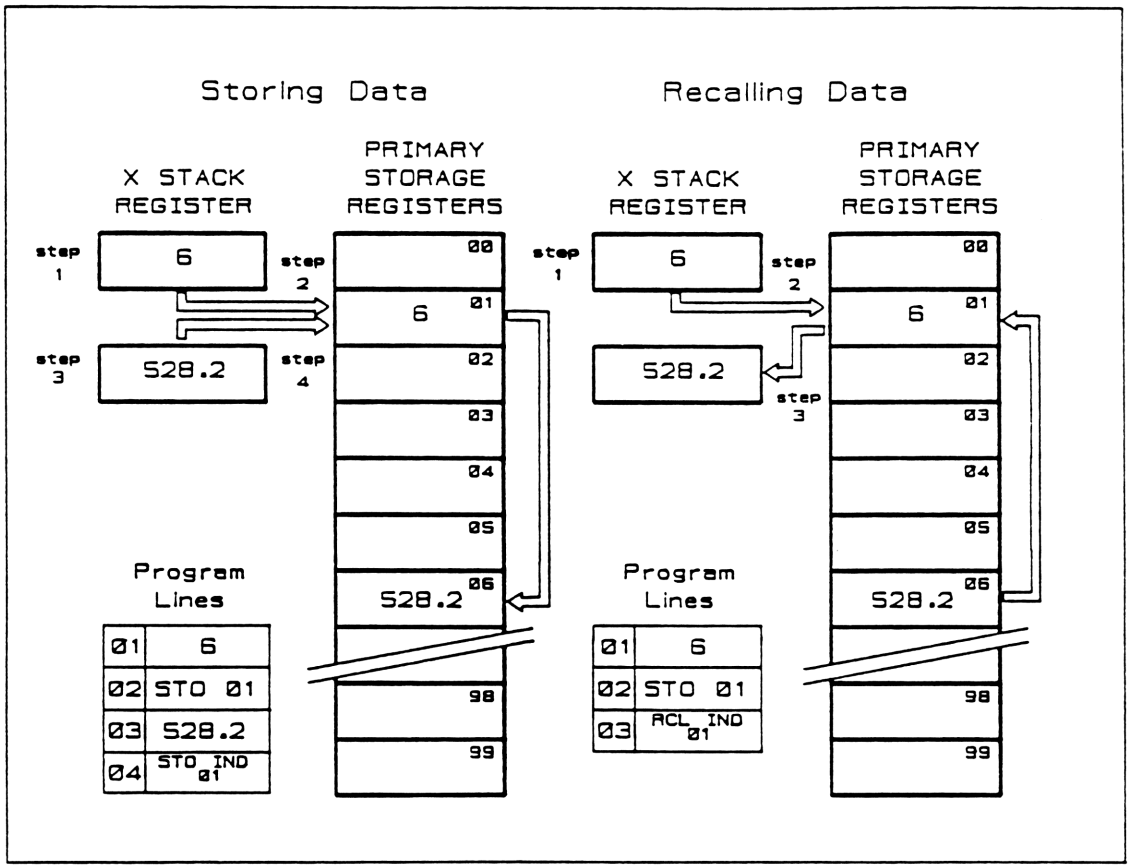
DIRECT ADDRESSING of
PRIMARY STORAGE REGISTERS



The procedures for INDIRECT ADDRESSING using the PRIMARY STORAGE REGISTERS are summarized in Figure 9.2. Examples of program lines which complete them are shown as inserts in the figure. The sequences of data storage and retrieval are a little more complex than DIRECT ADDRESSING, but they follow the same general procedures as sending a letter "in-care-of" someone else. First, a direct address is identified, then an indirect address is designated. To store a number in the PRIMARY STORAGE REGISTERS using INDIRECT ADDRESSING, the direct address needs to be identified first. For the illustration in Figure 9.2., this is done by storing a number (6 in this example) in any of the PRIMARY STORAGE REGISTERS. REG 01 was chosen for this example, although any other would have worked just as well.

Figure 9.2.

INDIRECT ADDRESSING of
PRIMARY STORAGE REGISTERS



The INDIRECTLY ADDRESSED number (528.2, for the example in Figure 9.2.) is first placed into the X STACK REGISTER (either by keying the appropriate numbers, or by constructing a line in a program to do it) and is then sent for storage by INDIRECT ADDRESSING to the designated PRIMARY STORAGE REGISTER. The location of the INDIRECT ADDRESS (REG 06 in this example) is the number previously stored in the PRIMARY STORAGE REGISTER serving as the DIRECT ADDRESS. The 4th program line listed in Figure 9.2. which controls this

process indicates, "Take the number which is in the X STACK REGISTER, and store it in the PRIMARY STORAGE REGISTER whose number is now listed in REG 01." For the example, 528.2 is stored in this way in REG 06, since the number "6" is in REG 01 which is being used as the direct address.

To see the process of INDIRECT ADDRESSING a little more clearly, execute the program lines shown in Figure 9.2. First turn the computer ON, then key "6", then "STO 01". The view window will display "6" at the end of these 2 steps. Next, key "528.2", then key STO SHIFT to see "STO IND _ _". Then key "01" in response to this alpha prompt. You will see "528.2" in the view window at the end of the sequence. The process will have stored, however, the number "528.2" indirectly in REG 06.

It is easy to verify that "528.2" has been stored by INDIRECT ADDRESSING in REG 06. In fact, there are 2 ways to test that the process has been successful. One way is simply to recall the contents of REG 06 by DIRECT ADDRESSING, as summarized in Figure 9.1. To do this, first erase the contents of the X STACK REGISTER as it is now displayed in the view window, then key "RCL 06". You will then see, of course, "528.2" in the view window, as the numerical contents of REG 06 is copied into the X STACK REGISTER.

The second way to verify that "528.2" was successfully stored by INDIRECT ADDRESSING is to follow the procedure summarized in Figure 9.2. Erase the X STACK REGISTER, then recall the contents of REG 06 to the view window using the process of INDIRECT ADDRESSING . You can do this quite easily by keying "RCL IND 01" to tell your computer to "recall the contents of the PRIMARY STORAGE REGISTER whose number is now stored in REG 01". Since "6" is now stored in REG 01, and "528.2" is stored in REG 06, you will see "528.2" copied into the view window.

As shown by the examples in Figures 9.1. and 9.2., numerical data can be placed into and recalled from storage in any PRIMARY STORAGE REGISTER using either the procedure of DIRECT ADDRESSING, or that of INDIRECT ADDRESSING. The basis for making the choice between these two techniques for data storage and recall will become clear in the several examples presented later in this chapter. These examples will also show how ALPHA data can be manipulated in a similar way (using ASTO _ _, ARCL _ _, ASTO IND _ _, and ARCL IND _ _ lines in a program), and how arithmetic operations, flag test and control, the execution of TONES, and a variety of other procedures are dealt with in a similar fashion. First, though, it will be useful to see the relationships between the use of INDIRECT ADDRESSING and the control of data storage and retrieval from the EXTENDED DATA STORAGE REGISTERS.

Section 9.2. Manipulating the Contents of the EXTENDED DATA STORAGE REGISTERS using INDIRECT ADDRESSING

This section reviews how data are stored and recalled from the EXTENDED DATA STORAGE REGISTERS of the HP-41C and HP-41CV computers. Data manipulation in these registers and in the EXTENDED MEMORY of the HP-41CX is quite different. Owners of this model are directed for details to their owner's manual.

Up to this point, the EXTENDED DATA STORAGE REGISTERS have remained empty in executing sample programs. What a waste! These registers provide 218 memory locations in addition to those created as PRIMARY STORAGE REGISTERS. The cost in using the EXTENDED DATA STORAGE REGISTERS is that you will have to become familiar with INDIRECT ADDRESSING techniques, because this is the only way to store data into them, or recall data from them. Even though

INDIRECT ADDRESSING may seem a little awkward at first, it soon becomes as simple to use as the technique of DIRECT ADDRESSING with a little practice. Figure 9.3. illustrates an example of how to store data in the EXTENDED DATA STORAGE REGISTERS, and Figure 9.4. shows how data are recalled from them.

As shown in Figures 9.3. and 9.4., the procedure for using INDIRECT ADDRESSING to store and retrieve data using the EXTENDED DATA STORAGE REGISTERS is identical to that used for the PRIMARY DATA STORAGE REGISTERS (Figure 9.2.) with one exception. Since the EXTENDED DATA STORAGE REGISTERS are numbered 100 to 318, a three digit number within this range must be stored in the PRIMARY STORAGE REGISTER which will be used for the direct address. In the example, the EXTENDED DATA STORAGE REGISTER numbered 103 was selected for data storage, and for that reason "103" was stored (by DIRECT ADDRESSING) in REG 01.

Were you to try storing a number from the X STACK REGISTER to REG 10, for example, without having created 10 PRIMARY STORAGE REGISTERS, you would correctly expect to see the ERROR MESSAGE "NON-EXISTENT". Similarly, were you to try storing a number to the EXTENDED DATA STORAGE REGISTER 103, for example, without having created the storage space using the SIZE function,

Figure 9.3.

INDIRECT ADDRESSING of
EXTENDED DATA STORAGE
REGISTERS: Data Storage

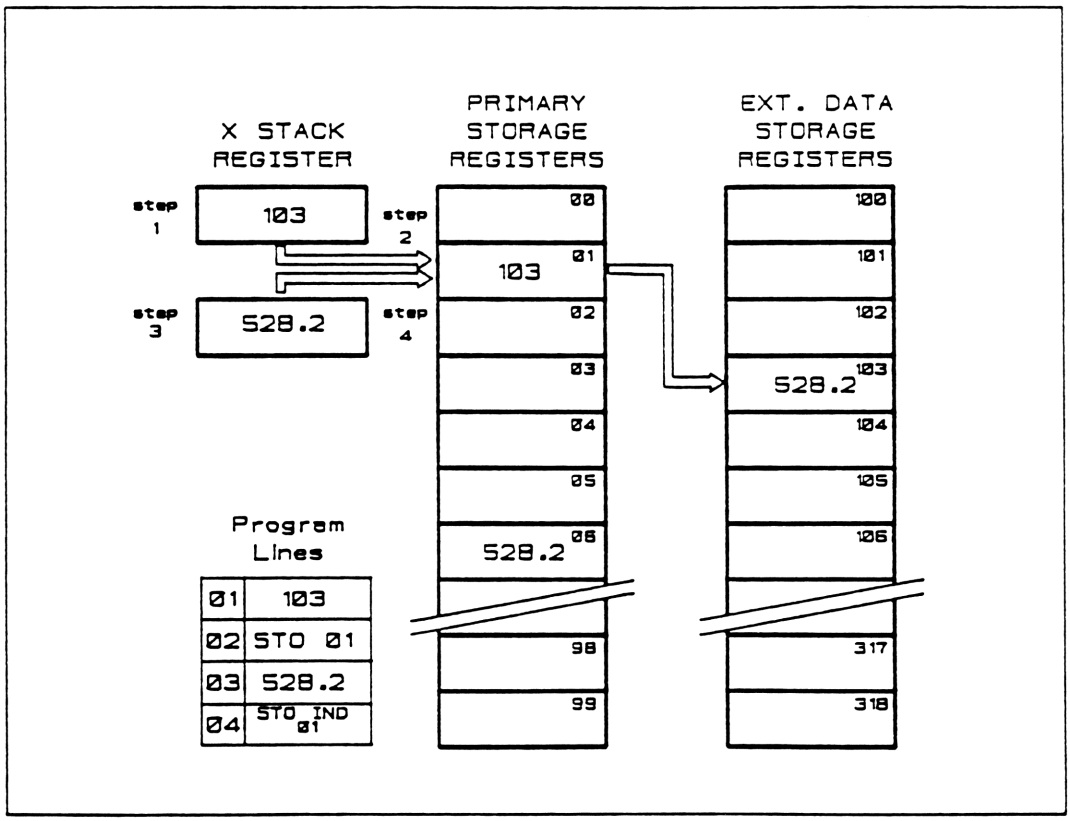
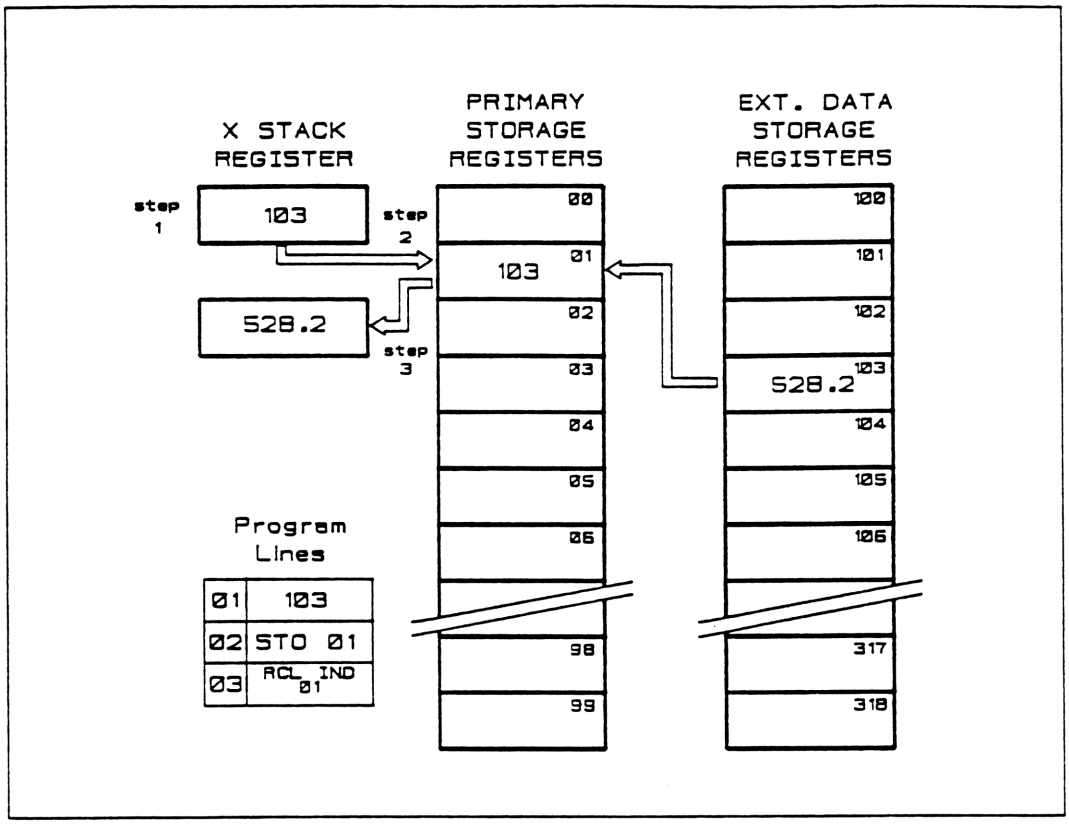


Figure 9.4.

INDIRECT ADDRESSING of
EXTENDED DATA STORAGE
REGISTERS: Data Recall



you would correctly expect to see "NON-EXISTENT" again. A common requirement for data storage by DIRECT ADDRESSING or by INDIRECT ADDRESSING techniques is that the target data storage location much have been constructed by the SIZE operation (see Figure 3.1. and Section 3.3.).

The next section presents sample programs in which INDIRECT ADDRESSING is used to perform not only data storage and recall functions, but for other operations also.

Section 9.3. Tones

The programs TONESDN and TONESUP listed in Figure 9.5. are designed to illustrate several applications of INDIRECT ADDRESSING. First, each program is described line-by-line to reveal the rationale for constructing each step, then some hints are given for entering the program into the computer, trying it and modifying it. TONESDN is designed to sound in descending order each of the 10 TONES (numbered 0 to 9) you control with your computer. TONESUP is designed to sound these tones in their increasing numerical order. Each program stops once the last tone has been sounded.

Figure 9.5.

TONESDN and TONESUP

Program Listing:

```
01 LBL "TONESDN"
02 10
03 STO 01
04 9
05 STO IND 01
06 LBL 00
07 TONE IND 10
08 1
09 ST- IND 01
10 RCL IND 01
11 -1
12 X<Y?
13 GTO 00
14 END
```

Program Listing:

```
01 LBL "TONESUP"
02 10
03 STO 01
04 0
05 STO IND 01
06 LBL 00
07 TONE IND 10
08 1
09 ST+ IND 01
10 10
11 RCL IND 01
12 X<Y?
13 GTO 00
14 END
```

Lines 2 and 3 of the TONESDN program place (using DIRECT ADDRESSING) the number "10" into REG 01 which will serve as a direct address. Lines 4 and 5 place (by INDIRECT ADDRESSING) the number "9" into REG 10. At line 7, the first tone is chosen and then sounded (by INDIRECT ADDRESSING) using the number stored in REG 10. When the program is initially activated, the number stored in REG 10 is a "9", having been stored there by INDIRECT ADDRESSING at program line 5. Lines 8 and 9 decrease the number stored in REG 10 by 1, then at lines 10 to 12, the number in REG 10 is tested to see if the last tone in the sequence (TONE 0) has been sounded. If the number in REG 10 is greater than zero, program flow is directed to the LOCAL LABEL 00, from which the next lower tone is sounded (line 07) and the number in REG 10 is decremented once again by 1. Program flow continues to loop through LBL 00 until the conditional test at line 12 is determined not to be true (that is, the number stored in REG 10 is no longer equal to or greater than zero), and the program is halted at line 14. Pressing R/S (or performing XEQ "TONESDN") starts the TONESDN program again.

There are several features of the TONESDN program which, although perhaps intuitively obvious, are worth review. The number listed in line 2 could have designated any PRIMARY STORAGE REGISTER, not just REG 10, as long as the MAIN MEMORY space had been allocated (using SIZE) for it. Also, it could have been any designated memory location in the EXTENDED DATA STORAGE REGISTERS. Further, not all tones needed to be sounded. The first tone in the descending sequence of tones controlled by this program could have been selected by placing any number between 1 and 9 in line 4. Using the number "0" would have allowed only TONE 0 to be heard. Also, each of the tones 9 to 0 did not have to be sounded. Only the odd numbered tones would have been heard had the number "2" been placed in line 8. Were the number "8" located in line 4 and the number "2" in line 8, only even numbered tones would have been sounded in decreasing numerical order.

It may be apparent that since TONESDN uses the PRIMARY STORAGE REGISTER 10 as an indirect address for data storage, line 9 could have been listed as "ST- 10", instead of as "ST- IND 10". For the same reason, line 10 could have been constructed to be "RCL 10" to bring the number in REG 10 to the X STACK REGISTER by DIRECT ADDRESSING, just as well as copying it there by INDIRECT ADDRESSING with the statement "RCL IND 10". A few hints may help for keying the TONESDN program into your computer. Line 7 is keyed by XEQ "TONE", then SHIFT, then keying in response to the alpha prompt, "TONE IND _ " the number of

the PRIMARY STORAGE REGISTER which contains the number designating which tone to sound.

As shown by the TONESUP program in Figure 9.5., TONESDN needs only minor editing to reverse the sequence in which tones are sounded. The tone to begin the sequence is designated by the number "0" at line 4, and incremental unit is defined at line 8. Since TONESUP requires testing for the highest numbered tone (TONE 9) in the sequence, not the lowest (TONE 0), as TONESDN operated, the number "9" needs to provide the reference for the conditional test at line 12 against whatever number is stored in REG 10. That number is recalled to the X STACK REGISTER by INDIRECT ADDRESSING with the statement at line 11.

A useful exercise would be to merge the programs TONESDN and TONESUP so that tones are first sounded in descending, then in ascending order in the same program. It would also provide good practice if you could write a program which sounded 3 (or more) cycles of ascending and then descending tones before the program stopped. Then, explore how the program can constructed to be as short as possible using INDIRECT ADDRESSING techniques.

Section 9.4. CALCX

The CALCX program listed in Figure 9.6. is designed to illustrate another useful application of INDIRECT ADDRESSING. It shows how sequentially entered data can be requested and placed into order-numbered PRIMARY STORAGE REGISTERS using a program loop, and then recalled from them for a specific calculation.

In CALCX, REG 00 serves both as a counting register and as a source for the display of a sequentially increasing number used as a suffix to the alpha prompt "ENTER N". Since the number "1" is placed in REG 00 as soon as the program is started (lines 4 and 5), the first time "ENTER N" (line 7) is displayed, it will appear as "ENTER N1". Whatever piece of data is keyed into the X STACK REGISTER in response to this alpha prompt, when R/S is pressed, the entered number will be stored by INDIRECT ADDRESSING (line 10) into the PRIMARY STORAGE REGISTER designated in REG 00. Since REG 00 contains the number "1" when the program is initialized, not only will the alpha prompt "ENTER N1" be displayed, entered data will be stored in REG 01.

Lines 11 to 13 determine whether all data (N1 to N5) have been entered. If not, program flow is directed to LBL 00 from which the number in REG 00 is increased by 1, the next sequential data entry is requested, and then stored in a correspondingly numbered PRIMARY STORAGE REGISTER. After N5 has been entered and stored in REG 05, X is calculated and displayed, and the program ends. A new calculation of X begins with the entry of a new set of data for N1 to N5 after R/S is pressed.

Even a superficial comparison of how data were requested and entered in the SOLVX (Figure 7.2.) and CALCX (Figure 9.6.) programs shows how the technique of INDIRECT ADDRESSING conserves program lines and allows program flow to be facilitated. Figure 9.7. lists those functions which can be used in INDIRECT ADDRESSING statements.

Section 9.5. FLTEST

INDIRECT ADDRESSING has many more powerful applications than just entering data sequentially. Any of the operations shown in Figure 9.7. can be used to activate functions in an ordered fashion, the steps of which need not start at 1 and need not progress in units of 1. The style of the CALCX program which asked for N1 first, then N2, etc. was selected just for the purposes of that program. Numbered data requests and the correspondingly numbered data storage locations could have

Figure 9.6.

CALCX

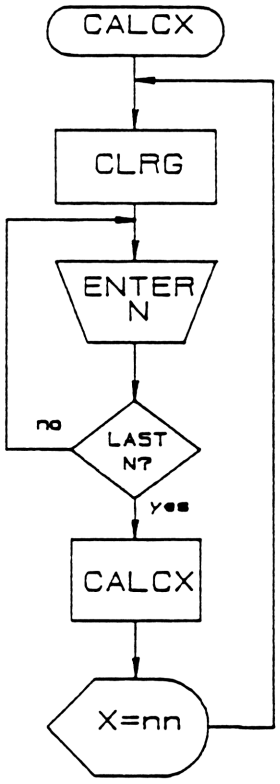
Problem Statement:

This program is designed to solve the following equation.

Equation:

$$X = \left(\frac{N_2 + N_3}{\text{LOG } N_4}\right)^{N5} - \left(\frac{N_4}{N_1}\right)$$

Flow Diagram:



Program Listing:

```
01 LBL "CALCX"
02 BEEP
03 CLRG
04 LBL 00
05 1
06 ST+ 00
07 FIX 0
08 TONE 5
09 "ENTER N"
10 ARCL 00
11 PROMPT
12 STO IND 00
13 4
14 RCL 00
15 X≤Y
16 GTO 00
17 FIX 2
18 RCL 02
19 RCL 03
20 +
21 RCL 04
22 LOG
23 /
24 RCL 05
25 Y^X
26 RCL 04
27 RCL 01
28 /
29 -
30 TONE 9
31 "X="
32 ARCL X
33 AVIEW
34 STOP
35 GTO "CALCX"
36 END
```

Figure 9.7.

INDIRECT ADDRESSING
Statements

A. Display:

Statement	Function
CATALOG	Catalog review
ARCL	Recall an ALPHA statement
VIEW	View contents of a register
FIX	Standard display
SCI	Scientific display (10^3)
ENG	Engineering display (10^3 in mult. of 3)

B. Program/Keyboard Control:

Statement	Function
STO	Store a number
RCL	Recall a number
STO +	Store additively
STO -	Store with subtraction
STO *	Store with multiplication
STO /	Store with division
ASTO	Store an ALPHA statement
GTO	Go To (line or LBL)
XEQ	Execute (GLOBAL or LOCAL LABEL)
DSE	"Decrement and skip if equal to"
ISG	"Increment and skip if greater than"
TONE	Tones
ΣREG	Register accumulation
X<>	Exchange contents of X STACK REG that of another REG

C. Conditional Tests:

Statement	Function
SF	Set FLAG
CF	Clear FLAG
FS?	FLAG set conditional test
FC?	FLAG clear conditional test
FS?C	FLAG set test/clear
FC?C	FLAG clear test/clear

progressed by any increment, as controlled by the number listed at line 05. Also, the request for numbered data could have started anywhere, as demonstrated in the program FLTEST shown in Figure 9.8.

The FLTEST program is designed to display the status of FLAGS (whether each is SET or CLEAR), beginning and ending at any numbered FLAG designated by the program's user. Data requests to define the range of FLAGS to test are structured in lines 04 to 09. Because FLAG designations are whole numbers (see Figure 8.1.), the digit display is controlled at line 02 to show no numbers to the right of the decimal.

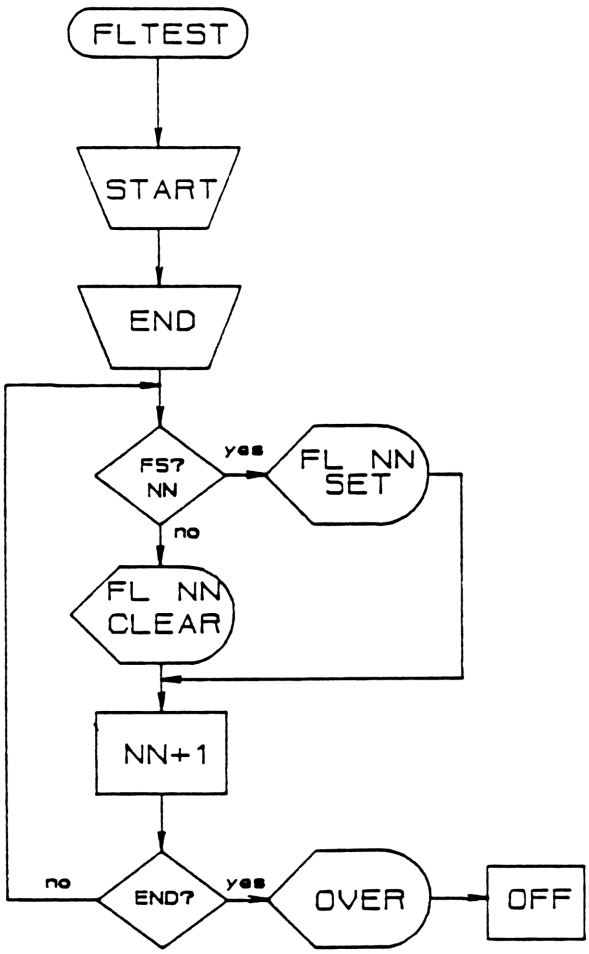
Figure 9.8.

FLTEST

Problem Statement:

This program is designed to test and display the status (SET or CLEAR) of all FLAGS over a user-defined range.

Flow Diagram:



Program Listing:

```
01 LBL "FLTEST"
02 FIX 0
03 BEEP
04 "START?"
05 PROMPT
06 STO 00
07 TONE 5
08 "END?"
09 PROMPT
10 STO 01
11 LBL 00
12 FS? IND 00
13 GTO 01
14 "FL"
15 ARCL 00
16 "⊢CLEAR"
17 GTO 02
18 LBL 01
19 "FL"
20 ARCL 00
21 "⊢SET"
22 LBL 02
23 AVIEW
24 1
25 ST+ 00
26 RCL 00
27 RCL 01
28 1
29 +
30 X≠Y?
31 GTO 00
32 9
33 STO 00
34 LBL 03
35 TONE IND 00
36 TONE IND 00
37 1
38 ST- 00
39 5
40 RCL 00
41 X≠Y ?
42 GTO 03
43 "FL TEST OVER"
44 AVIEW
45 PSE
46 TONE 9
47 OFF
48 END
```

The determination of whether the first flag in the user-designated range is SET or not is made by the INDIRECT ADDRESSING statement in line 12. If this first FLAG is SET, the answer to the conditional statement "FS? nn" is true and program flow continues at line 13 where the program branches unconditionally to LOCAL LABEL 01 at line 18. The ALPHA statement "FL nn SET" is constructed using INDIRECT ADDRESSING in lines 19 to 21. If the tested flag is CLEAR, program flow continues from the conditional test at line 12 to lines 13 through 16 where the ALPHA statement "FL nn CLEAR" is constructed. Line 23 displays the appropriately structured answer to the FLAG test at line 12.

Lines 24 to 30 increase by one the number of the next FLAG to be tested and determine if the FLAG test sequence has been completed. If it isn't complete, program flow is directed by to the LOCAL LABEL 00. If the FLAG test sequence is complete, a series of TONES is sounded using INDIRECT ADDRESSING (lines 32 to 42, an ALPHA statement is displayed to indicate this condition (lines 43 and 44) and the computer is automatically turned off by the instruction at line 47.

Section 9.6. EXAM

The program GRADES introduced early in the book (Figure 6.5) illustrated how the HP-41 computer is useful for making decisions as well as for just making numerical calculations. Although the program worked well for that, it was not constructed to be practical, nor was it efficiently written. It is unlikely, for example, that an instructor using the GRADES program would write into the program lines 06, 10, 14, or 18 to designate the cut-off points for different grades. More likely, these criteria would be different from exam to exam. The GRADES program would be more useful if it asked the user for this information when the program was initialized. Also, the 5 separate statements required in the GRADES program to announce each grade decision (lines 22 to 38) are inefficiently written.

The program EXAM (Figure 9.9) functions to provide similar information as that given by the GRADES program, but it exploits many of the advantages of INDIRECT ADDRESSING to make the determination of an exam grade more useful and more efficient. For example, each time the program is run, the user customizes it for how the class curve mandates grade distribution. Also, the lines which control grade display are more compact.

The first 20 lines of the EXAM PROGRAM are instructions for entering the cut-off grades for whatever examination the user is working with. Since the highest possible grade is a 4.0, this number is initially stored in REG 00 by lines 2 and 3. Minimum percent scores for each grade (0.0 to 4.0) are entered sequentially into REGs 02 to 08 using a counter number stored in REG 01 (by lines 5 and 6) and employing an INDIRECT ADDRESSING statement at line 13. Multiple operations of the loop constructed by lines 08 to 20 step through the displayed requests for grading criteria, storing each piece of new information in a separate PRIMARY STORAGE REGISTER.

There are other ways to have this information requested and stored in the execution of this first stage of program operation. The reader is encouraged to rewrite the EXAM program using the style of the GRADES program and not employing INDIRECT ADDRESSING techniques. It will soon become clear how much more laborious and time consuming any process of sequential data entries would be without using INDIRECT ADDRESSING.

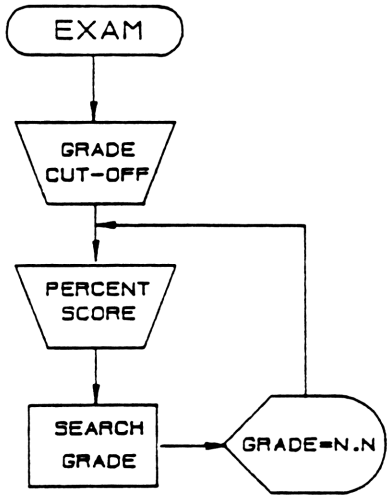
The selection of the correct grade corresponding to a specific percent score for any student is accomplished by program lines 32 to 43. Analogous to the entering of sequential data at the beginning of the program, the determination of a student's grade is made by an INDIRECT ADDRESSING statement at line 34 and a subsequent conditional test (line 35) either to direct program flow to another pass at finding the correct

Figure 9.9.

EXAM

Theodore Tutor teaches a large enrollment class for which there are many examinations during the term. He wrote the EXAM program to indicate a student's grade for an exam based on its percent score (% scores range from 0 to 100). Exams are graded on a 4.0 scale (highest = 4.0; next = 3.5, next = 3.0, etc.) with no grade of 0.5. The cut-off percent score for each grade is different for each exam. Also, percent scores are rounded up from 0.5, for example, a score of 89.5% would be graded as a 90%, but a score of 89.4% would be graded as an 89.0, etc.

Flow Diagram:



REG	data
00	grade display
01	counter
02	min. for 4.0
03	min. for 3.5
04	min. for 3.0
05	min. for 2.5
06	min. for 2.0
07	min. for 1.5
08	min. for 1.0
09	% score
10	earned grade

Program Listing:

```
01 LBL "EXAM"
02 BEEP
03 4
04 STO 00
05 2
06 STO 01
07 FIX 1
08 LBL 00
09 TONE 5
10 "MIN FOR"
11 ARCL 00
12 PROMPT
13 STO IND 01
14 1
15 ST+ 01
16 .5
17 ST- 00
18 RCL 00
19 X>Y?
20 GTO 00
21 LBL 03
22 2
23 STO 01
24 TONE 5
25 "%SCORE?"
26 PROMPT
27 FIX 0
28 RND
29 STO 09
30 4
31 STO 10
32 LBL 01
33 RCL 09
34 RCL IND 01
35 X≤Y?
36 GTO 02
37 1
38 ST+ 01
39 .5
40 ST- 10
41 RCL 10
42 X≠Y?
43 GTO 01
44 0
45 STO 10
46 LBL 02
47 FIX 1
48 TONE 9
49 "GRADE="
50 ARCL 10
51 AVIEW
52 PSE
53 GTO 03
54 END
```

grade, or to terminate the grade selection process to announce the grade by the steps listed under LBL 02. This selection process could be completed as it was in the GRADES program, but not as efficiently nor executed as quickly as it can be using INDIRECT ADDRESSING techniques as in the EXAM program.

I hope the reader will notice the challenge in the problem statement of the EXAM program (Figure 9.9) that this grading scheme does not allow for a grade of 0.5. Were it to, the grade display sequence controlled under LOCAL LABEL 01 could simply be continued to its natural end by the successive subtraction of a constant (0.5; line 39). Notice also how simply the problem of not reporting a 0.5 grade is solved by lines 44 and 45 which interrupt the last steps of the decrementing sequence begun in LBL 01.

There is another subtle challenge in writing the EXAM program. It was decided that a student's percent score on an exam would be rounded upward from .5 in the decision of assigning a grade. A first guess as how to accomplish this might be to include the instruction FIX 0 at program line 27 to truncate any data entry requested at line 25 with a digit to the right of the decimal to a whole number. Such a maneuver would control the display, of course, but the number would be stored as entered at line 29, fractional component and all. The difficulty is easily solved, though, by using the

Figure 9.10.

Sample Problem: EXAM

Mr. Tutor used the EXAM program to calculate grades for the following 3 exams, each of which was curved differently.

Grading Criteria

	Minimum % Score for a Grade of:						
EXAM	4.0	3.5	3.0	2.5	2.0	1.5	1.0
1	90	85	80	75	70	65	60
2	95	88	85	78	65	53	40
3	85	79	75	70	63	55	49

Grade Assignments

	Exam 1		Exam 2		Exam 3	
Name	% Sc.	Gr.	% Sc.	Gr.	% Sc.	Gr.
Tom	92.3	4.0	90.3	3.5	84.5	4.0
Dick	79.6	3.0	63.7	1.5	71.2	2.5
Harry	68.1	1.5	39.6	1.0	76.3	3.0
Lewis	83.4	3.0	78.1	2.5	78.8	3.5
Clark	62.0	1.0	61.8	1.5	64.0	2.0

instruction RND at line 28. The RND operation rounds the number to the control level established by the FIX 0 statement in the preceding line. The application of the RND function in the EXAM program serves to provide the required rounding up from 0.5 of the calculated course percent score.

Figure 9.10 shows sample problems for using the EXAM program. Were there only one exam to be graded in this example, perhaps program construction similar to the GRADES program would work well enough. But there are several exams to be scored, each of which has different criteria for assigning grades, as shown in the upper part of Figure 9.10. Using the EXAM program, percent scores and grades for each student are readily assigned despite the complexity of there being different academic curves for the course's exams.

The purpose of the FLTEST and EXAM programs is to show how INDIRECT ADDRESSING statements are used in association with the functions of conditional testing (FS? nn), TONES, data storage and recall. Any of the functions listed in Figure 9.7. could be used in a similar way by INDIRECT ADDRESSING. If there is any question as to the utility of using INDIRECT ADDRESSING, the reader is encouraged to rewrite any of the programs in this chapter in some other way and then compare the different versions for efficiency and speed of program execution.

The next chapter presents a simple program for balancing a checkbook. It contains only a couple of new programming techniques, and is offered more as a review and as a practical example for using the HP-41 computer.

Chapter 10

Wrapping It Up

Section 10.1. The CHECKS Program

Even though many use their HP-41 computers to execute sophisticated programs and solve complex equations, most of us depend on these marvelous machines to make calculations just for everyday problems. This chapter presents an uncomplicated program for helping with one such task, that of balancing a checkbook. There is nothing particularly clever about this program, but even so, it is a reliable tool for reducing the drudgery of checkbook record keeping. Also, working through it will provide a good review of many of the programming principles presented so far.

The CHECKS program is organized to provide several different kinds of information about the checkbook's balance. For example, it calculates a "new balance" for the account. This is perhaps the most important piece of financial information, since it shows the amount of money one has for the rest of the pay period. Were everyone's checking account always in agreement with the bank's records, no more data than "new balance" would be needed. But with mis-entered checks, incorrectly tabulated records, forgotten entries and poor arithmetic skills in annotating the check register, additional information may be needed. In order to track down errors which have crept into our checkbook records we may need to know, for example, how much money is represented by past and present outstanding checks, the sum of checks written this month, what one's total credit is at any time, and how much the account is out of balance. The CHECKS program provides these data too.

As an introduction to how this section is organized, Figure 10.1. describes the CHECKS program, lists the equations for it, and defines the symbols it will use. Figure 10.2. shows the flow diagram for the program. Figure 10.3. is a list of the program's steps, and Figure 10.4. presents a sample problem. Figure 10.5. shows a record keeping form organized to receive data in the order the CHECKS program presents it. "User Instructions" for the CHECKS program listed in Figure 10.6. will be helpful for working through the sample program and also for using the program on a month-to-month basis for balancing one's own checking account. For the most part, however, ALPHA prompts in the program are self-explanatory, and the "User Instructions" will be of most help only for the first couple of times CHECKS is used.

Section 10.1.1. General Description of the CHECKS Program

The CHECKS program is designed so that information is entered directly from the check register and the bank's account statement a step at a time in response to ALPHA prompts. The program's job is to assemble this information into appropriate categories, do the associated arithmetic, and then present summary data in an ordered way for record keeping. The program does automatically what a person must do on one's own with paper and pencil to reconcile personal checking account records with those of the bank, but it does it more quickly and easily.

There are 4 categories of information required to run the CHECKS program. One comes from the bank's records, and the other 3 come from one's own records.

Figure 10.1.

CHECKS

Problem Statement:

This program is designed to balance a checking account.

Equations:

CREDIT = (ΣDEP) - (PRIOR BAL)
NEW BAL = (CREDIT) - (RCK) - (OCK) - (SERV CHARGE)
ΣCKS = (RCK) + (OCK) + (SERV CHARGE)
ΣOCKS = (OCK) + (PAST OCKS)
CAL BAL = (PRIOR BAL) + (ΣDEP) - (RCK) + (PAST OCKS) - (SERV CHARGE)
BAL ERROR = (CURR BAL) - (CAL BAL)

where:

CURR BAL = current balance (shown on bank statement)
PRIOR BAL = account balance at beginning of period
RCK = returned checks in this period
OCK = outstanding checks in this period
PAST OCKS = outstanding checks in all past periods
SERV CHARGE = service charges in this period
ΣDEP = sum of deposits in this period
ΣCKS = sum of checks in this period
NEWBAL = new balance for this period
ΣOCKS = sum of all outstanding checks
CALBAL = clculated balance for this period
BAL ERROR = balance error; if zero, account balances

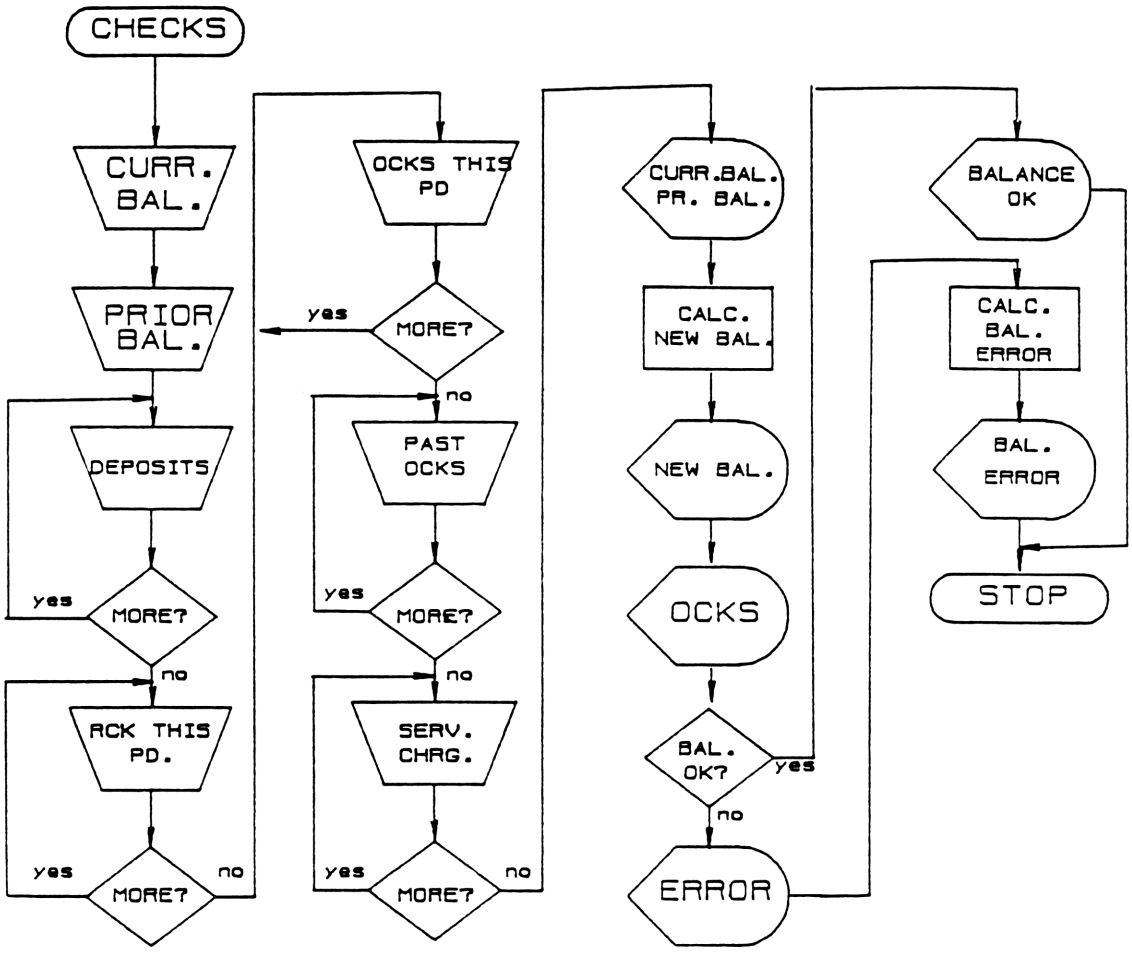
The first is the "Current Balance" which is listed on the bank's statement of the account. It is the receipt of this statement in the mail along with either a list of returned checks, or the returned checks themselves which triggers most people into grudgingly tackling the dreaded job of getting their checking account balanced for the month. The "Current (or indicated) Balance" on the statement shows what the bank's records report to be the status of the account at the time the statement was issued. It is against this standard that one must reconcile one's own records. If calculations from one's own records eventually yield a "Calculated Balance" equal to the bank's "Current Balance", then all is well and the account is in order.

The second kind of information required for the CHECKS program is provided by indicating either one-at-a-time or as a sum the deposits made since the last time the account was balanced. This information comes either from an accurately maintained check register or from deposit slips. The third piece of information, "Prior Balance", is what the account balance was at the beginning of this balance period. This information comes from the check register one maintains, similar to that shown in Figure 10.4.

The last category of information is related to the individual checks one has written. Some of these checks, the "Returned Checks", will have been written, cashed by their recipients and returned to the payee along with the monthly bank statement. Others, the outstanding checks, will have been written either prior to this balance period, the "Past Outstanding Checks", or written during the present period, the "Outstanding Checks in This Period", but neither type will have been returned yet. Along with these debitures must be included also any "Service Charges" enforced by the bank itself.

Figure 10.2.

Flow Diagram: CHECKS



Most of the problem one has in getting the checkbook balanced is because of keeping all of this information straight, not just in doing the simple arithmetic. The value of the CHECKS program is that it asks you for specific items of information one at a time, keeps them in the approp riate categories, and does the arithmetic for you too. It also makes these calculations much faster than one can do with paper and pencil. From most people's point of view, the faster the checkbook can be balanced with accuracy and with the least hassle, the better - another bonus for knowing how to use the HP-41 computer.

Section 10.1.2. Using the CHECKS Program

The sample program shown in Figure 10.4. shows how John and Jane Dough use the CHECKS program to balance their account this month. They've just received their bank statement (top of Figure 10.4.) and they have their check register at hand (bottom of Figure 10.4.). As soon as they start the program with XEQ "CHECKS" and hear the BEEP tones, they are asked for the "Current Balance". Referring to this

month's checking account statement, the Dough's enter $328.12, first by keying this number into the X STACK REGISTER, and then pressing R/S. After hearing a tone sequence, they are asked for the "Prior Balance" of their account. Looking at their check register they first key $150.37, then press R/S.

The next ALPHA prompt asks them to enter the first deposit they made into their account since its last balance. As recorded in their check register, the first deposit in this balance period was $12.16 made on June 13th. Keying this amount then pressing R/S, they are asked "MORE?", implying were there any additional deposits in this period. They key next the amount of the second deposit they made on June 15th of $29.34, (then press R/S). Once again they are asked "MORE?" and given an opportunity to enter the amount of the third deposit ($126.95), then in response to "MORE?" again, they key then enter the amount of their last deposit in this period ($714.00 on July 1).

When asked "MORE?" again, they now key a zero (or just press the decimal key) and press R/S to indicate in the code of the CHECKS program that they had entered

Figure 10.3.

Program Listing: CHECKS

```
01 LBL "CHECKS"
02 CLRG
03 FIX 2
04 .00301
05 STO 07
06 BEEP
07 "CURR BAL?"
08 PROMPT
09 STO 00
10 XEQ 01
11 "PRIOR BAL?"
12 PROMPT
13 STO 01
14 XEQ 01
15 "DEPOSITS?"
16 PROMPT
17 LBL 02
18 ST+ 02
19 XEQ 03
20 X=Y?
21 GTO 04
22 GTO 02
23 LBL 04
24 XEQ 01
25 "RCK THIS PD?"
26 PROMPT
27 LBL 05
28 ST+ 03
29 XEQ 03
30 X=Y?
31 GTO 06
32 GTO 05
33 LBL 06
34 XEQ 01
35 "OCK THIS PD?"
36 PROMPT
37 LBL 07
38 ST+ 04
39 XEQ 03
40 X=Y?
41 GTO 08
42 GTO 07
43 LBL 08
44 XEQ 01
45 "PAST OCKS?"
46 PROMPT
47 LBL 09
48 ST+ 05
49 XEQ 03
50 X=Y?
51 GTO 10
52 GTO 09
53 LBL 10
54 XEQ 01
55 "SERV CHARGE?
56 PROMPT
57 LBL 11
58 ST+ 03
59 XEQ 03
60 X=Y?
61 GTO 12
62 GTO 11
63 LBL 12
64 TONE 9
65 TONE 9
66 RCL 00
67 "CURR BAL="
68 XEQ 18
69 TONE 9
70 RCL 01
71 "PRIOR BAL="
72 XEQ 18
73 TONE 9
74 RCL 02
75 "ΣDEP="
76 XEQ 18
77 RCL 01
78 +
79 TONE 9
80 "CREDIT="
81 XEQ 18
82 RCL 03
83 RCL 04
84 +
85 TONE 9
86 "ΣCKS="
87 XEQ 18
88 RCL 02
89 RCL 01
90 +
91 RCL 03
92 -
93 RCL 04
94 -
95 TONE 9
96 "NEW BAL="
97 XEQ 18
98 RCL 04
99 RCL 05
100 +
101 TONE 9
102 "ΣOCKS="
103 XEQ 18
104 RCL 01
105 RCL 02
106 +
107 RCL 03
108 -
109 RCL 05
110 +
111 STO 06
112 RCL 00
113 X=Y?
114 GTO 13
115 GTO 14
116 LBL 01
117 TONE 8
118 TONE 9
119 TONE 8
120 TONE 1
121 RTN
122 LBL 03
123 TONE 3
124 "MORE?"
125 PROMPT
126 .
127 STO X
128 X<>Y
129 RTN
130 LBL 13
131 "BALANCE OK
132 AVIEW
133 TONE 8
134 ISG 07
135 GTO 13
136 GTO 17
137 LBL 14
138 "***ERROR***"
139 AVIEW
140 TONE 8
141 ISG 07
142 GTO 14
143 GTO 17
144 LBL 15
145 "PROGRAM END"
146 AVIEW
147 STOP
148 GTO "CHECKS"
149 LBL 17
150 RCL 06
151 "CALBAL="
152 XEQ 18
153 RCL 00
154 RCL 06
155 -
156 X=0?
157 GTO 15
158 TONE 9
159 "BAL ERROR="
160 XEQ 18
161 GTO 15
162 LBL 18
163 ARCL X
164 AVIEW
165 STOP
166 RTN
167 END
```

Figure 10.4.

Sample Program: CHECKS

BNB BUDGET NATIONAL BANK

CHECKING ACCOUNT STATEMENT

FROM 6/12 TO 7/13

INCLUDED IN STATEMENT	
NO. DEPOSITS	NO. CHECKS
4	9

ACCOUNT NO. 123456789

BEGINNING BALANCE		TOTAL ADDED FOR DEPOSITS		TOTAL DEDUCTED FOR CHECKS		SERVICE CHARGE*		CURRENT BALANCE
168.69	+	882.45	−	714.02	−	9.00	=	328.12

*AMOUNT OF SERVICE CHARGE SHOULD BE DEDUCTED FROM YOUR CHECKBOOK BALANCE.

John and Jane Dough
123 Fourth Street
Anywhere, USA

LOW BALANCE	
DATE	AMOUNT
6/15	1967.54

AVERAGE BALANCE 2312.83

EPCC 308-00 REV. 6-76

PLEASE RECONCILE YOUR STATEMENT PROMPTLY – SEE REVERSE SIDE

RECORD ALL CHARGES OR CREDITS THAT AFFECT YOUR ACCOUNT

NUMBER	DATE	DESCRIPTION OF TRANSACTION	PAYMENT/DEBIT (−)		√ T	FEE (IF ANY) (−)	DEPOSIT/CREDIT (+)		BALANCE $	
482	5/28	Tim's Party Store	$ 10	08		$	$			
483	5/29	Central Grocery	17	09	#					
484	5/30	Book Store	5	92						
485	5/30	City News	2	32						
486	5/31	Electric Power Co.	79	13	#					
-	- -	- balance (6/13) - - - -	- -	-	-	- -	- - -	-	150	37
487	6/13	ABC Grocery	10	23	#		12	16		
488	6/13	State Police	14	78						
489	6/14	I. Gotoff (law firm)	54	16	#					
490	6/14	National Gasoline Co.	87	56	#					
491	6/15	GOTCHA Realty	478	10	#		29	34		
492	6/15	City Telephone Co.	19	12						
493	6/17	Joe's Grill	5	85	#		126	95		
494	6/18	Sam's Tire Repair	13	13	#					
495	6/25	Purity Cleaners	2	45						
-	6/30	(service charge)	5	00						
496	7/1	Bill's Service Station	2	45	#		714	00		
497	7/3	Travel Agency	50	00						
-	7/5	(service charge)	4	00						
498	7/8	Discount Center	12	09	#					
499	7/10	Hewlett-Packard Co.	60	17						
500	7/14	Downtown Grocery	50	45	#					
-	- -	- balance (7/14) - - - -	- -	- -	-	- -	- - -	-	163	28

REMEMBER TO RECORD AUTOMATIC PAYMENTS / DEPOSITS ON DATE AUTHORIZED.

the last amount in this category of information, that is the listing of deposits. Pressing R/S takes the program to the display an ALPHA prompt to request data for the next category of information, that of listing their returned checks for this period.

In response to the prompt "RCK THIS PD?" the Doughs examine their check register to see that the first returned check in this period was for $10.23 written to ABC Grocery. Entering this amount, they are then asked if there are more returned checks to enter ("MORE?"). They enter $54.16 and continue entering each of the next returned checks in this period. After the amount of check number 500 has been entered, a keyed zero followed by a press of the R/S toggle terminates data entry for this category of expenditures.

Figure 10.5.

CHECKS Records Form

Checking Account Balance

Date: 7/14

Current Balance	$ 328.12
Prior Balance	$ 150.37
Total Deposits	$ 882.45
Credit	$ 1,032.82
Total Checks this period	$ 869.54
New Balance	$ 163.28
Total Outstanding Checks	$ 164.84
Calculated Balance	$ 328.12

The CHECKS program then displays "OCK THIS PERIOD?", asking for data about checks numbered 488, 492, 495, 497 and 499. The Doughs need to be careful not to make an error by entering the amounts for the 2 service charges incurred on June 30 and July 5. The amount of each of the "outstanding checks in this period" is entered using R/S and then keying a zero and pressing R/S after the last entry has been made. Following the instructions in Section B of Figure 10.6., additional data about "Past Outstanding Checks" and then "Service Charges" are entered in a similar way.

Once the amount of the last service charge (that charged against the account on July 5) has been keyed and entered with R/S, the computer presents the first of several pieces of data display, that of the account's indicated balance. This is not the result of a

Figure 10.6.

User Instructions: CHECKS

A. Preparation:

1. Arrange all returned checks in numerical order.
2. Mark the checkbook register to indicate returned checks.
3. SIZE HP-41 computer for 010.
4. Initialize program by XEQ "CHECKS".

B. Data Entry:

(Hint: key amount, then press R/S. Enter a zero in response to "MORE?" after last entry has been made in a category)

1. Respond to "CURR BAL?" by entering Current Balance from bank statement.
2. Respond to "PRIOR BAL?" by entering account balance at end of last period.
3. Respond to "DEPOSITS?" by entering each deposit since last balance.
4. Respond to "RCK THIS PD?" by entering each returned check since last balance.
5. Respond to "OCKS THIS PD?" by entering each outstanding check since the last balance.
6. Respond to "PAST OCKS?" by entering each outstanding check in previous balance periods.
7. Respond to "SERV CHARGE?" by entering each service charge since last balance. (Hint: Use form similar to that in Figure 10.5) to record data output)

C. Data Output:

1. "CURR BAL" indicates bank's record for your current balance.
2. "PRIOR BAL" indicates account balance at beginning of this period.
3. "ΣDEP" indicates sum of deposits in this period.
4. "CREDIT" indicates account's credit for this balance period
5. "ΣCKS" indicates sum of all checks written in this period.
6. "NEW BAL" indicates new balance for this period. Record in check register and on record keeping form.
7. "ΣOCKS" shows sum of all outstanding checks for both present and past periods.
8. If read "BALANCE OK", your account agrees with that of the bank. If read "ERROR", account does not balance.
9. "BAL ERROR" indicates discrepancy between your records and those of the bank. If the number is positive, the bank shows you have more money in the account than do your records. If the number is negative, the reverse is true. Calculations listed in the record form (Figure 10.5) will help locate these errors.
10. "PROGRAM END" signals no more calculations will be made on the data you have entered. Press R/S to start the program over.

calculation (this amount was entered at the beginning of the program), but it's useful to have it displayed to be recorded on the summary of the checking account balance (see Figure 10.5.). They then press R/S to have the program flow present the next entry for the account summary, its "Prior Balance".

Another press of R/S provides information about the sum of deposits made into the account in this period. Additional information about the credit of the account, the sum of checks written, the new balance and the account's sum of outstanding checks is brought to the view window with successive presses of the R/S toggle. Next, the account's calculated balance is determined and then compared with the bank's current balance. Depending on whether or not these 2 amounts are the same determines, of course, whether the tension-relieving message "BALANCE OK" can be flashed or whether the stomach-wrenching "ERROR" needs to announced. At the end of this message display, the balance error for the account is shown.

If "BALANCE OK" is shown, the home account agrees with the bank's records. The computer can be turned off and records gratefully put away for another month. If "ERROR" is seen, pressing R/S after the message "PROGRAM END" is read takes the program back to its beginning to track down the error(s). Information recorded as shown in Figure 10.6. is helpful for finding the source(s) of these errors.

The CHECKS program is simple and there are several ways in which it can be customized for personal applications. For example, it would be interesting to have the program watch for a user-designated minimum balance and provide a warning signal when it has been exceeded. Also, it would be interesting to modify the program to take advantage of the HP-41 computer's "continuous memory" to accept daily entries of written checks and completed deposits. It would be interesting also to couple it with other programs to help keep track of savings and investment accounts along with checking account transactions. If the reader has access to a printer, the program could be modified to test for FLAG 55 (similar to that described in Figure 8.7.) to help with data recording.

Section 10.2. Where to Go From Here

C O N G R A T U L A T I O N S ! !

Having worked through the exercises of the previous chapters and having gained a general understanding of the principles they tried to illustrate, you can now consider yourself an expert with the HP-41 computer. Now come opportunities for you to apply these abilities to problem solving in your personal and professional life, and to expand your skills beyond the scope of this book. The promise of this book after all was to get you started using your HP-41 computer, not teach you everything about it. You've now just scratched the surface of using this computer, and where you want to go from here depends on you.

As you grow into your computer and write more and more programs for it, you'll soon be faced with having to abandon some of the ones you already have in your machine in order to make room in the MAIN MEMORY for new ones. It's unlikely you'll want to take the time and trouble to key-in the older ones again later, so you need to consider how to expand your computer to interface it with program and data storage devices of one kind or another.

If you decide the primary need to expand the memory of your computer is to store programs, then the least expensive and simplest device to obtain is the HP-82104 card reader. With the inexpensive magnetic cards it uses, you have virtually unlimited ability not only to store programs of your own, but also to "read" magnetic cards written by others for programs you want to use. The card reader plugs into the top of the computer through one of its accessory ports, and is light weight and small enough to remain connected to it while the machine is in general use. The protective vinyl case you received with your computer is made to fit even with the card reader in place. Both encoding a program onto magnetic cards and reading one from them is simple, if the proper steps are followed. Suggestions for these procedures are offered in Figures 5.9. and 5.10.

More complex and sophisticated data and program storage devices, such as HP's digital cassette drive (HP-82161), or optical wand (HP-82153) for reading bar codes are also options, as is a portable 3 1/2 inch disc drive to be used with HP's Interface loop. These and many other valuable peripherals, such as printers, a video interface, as well as an ever-growing library of programs will help you obtain the HP-41 computer system you need for your own special interests and applications.

There are many commercial sources for equipment, supplies and instruction manuals for your computer, the most obvious of which is your regional Hewlett-Packard distributor. Another one, however, which offers not only accessories for the HP-41 made by Hewlett-Packard, but also those manufactured by other companies is EduCALC (27953 Cabot Road, Laguna Niguel, CA 92677). Their frequently updated catalog is a reliable source of what's new for the HP-41 series computers.

Keeping up-to-date on new accessories and software for your computer is part of the fun of owning, operating and growing with this machine. It is suggested that the sooner you become familiar with programs available through Hewlett-Packard's User Library, the better. Information about this service is available from: Solve and Integrate Corp., P.O. Box 1928, 460 SW Madison Avenue, Suite 5, Corvallis, OR 97339-1928. Since you have successfully attained a substantial degree of competency in programming the HP-41 computer by studying the information in this book, you might be closer than you think to publishing programs of your own design in the HP User's Library.

There are many interesting ways to use the HP-41 computer beyond the suggestions in this book and beyond the instructions of its owner's manual. For example, the references listed at the end of this chapter will help you explore coupling your HP-41 computer directly to a tabletop computer (using devices and programs similar to LINK) and examine the flexibility of the HP-IL interface loop. They will also help you become proficient with the HP-41's "Extended Functions", learn the intricacies of "Synthetic Programming" and expand your abilities in many other ways. The pleasure and profit of learning to use the HP-41 computer has just begun!

It would be a mistake not to take advantage of opportunities to learn how to use your computer and its accessories provided by talking and working with others who have an interest in this machine. More and more people with a wide range of business, professional and personal interests are discovering the power of the HP-41 computer. Establishing contact and sharing interests and information with them could very well be one of the most important learning tools available to you.

Section 10.3. References

In testimony to the rapidly expanding interest in portable programmable computers (of which you have now become a part with your developed skills in using the HP-41) is the emergence of many self-help and self-instructional books in this area. These newer books supplement information presented in earlier texts which have proven themselves valuable for a person to expand math skills by self study.

The most useful and reliable references for the HP-41 computer are (to no one's surprise) the many manuals, references and guides published by Hewlett-Packard Co. which are available through the local HP distributor. There is a growing number of other references, though, which are pertinent both to the HP-41 computers and their general use. The following list is by no means comprehensive, but it will provide an entry-level introduction to what is available:

1. Alger, Philip L., Mathematics for Science and Engineering. Mc-Graw Hill Book Co. (1969)

2. Dearing, John, Calculator Tips and Routines (Especially for the HP-41C/41CV. Corvallis Software, Inc., P.O. Box 1412, Corvallis, OR 97339-1412.

3. Garrison, Paul. Programming the TI-59 and the HP-41 Calculators. TAB Books. Inc., PA. (1982)

4. Jarrett, Keith. HP-41 Extended Functions Made Easy. Synthetix Publ., Manhattan Beach, CA. (1983)

5. Keefe, Ed. Computer Science on Your HP-41, Grapevine Publ., Inc., P.O. Box 118, Corvallis, OR 97339-0118. (1987)

6. Kolb, William M. Curve-fitting for Programmable Calculators. Third Edition. Syntex, Inc., P.O. Box 1402, Bowie, MD. (1984)

7. McCarty, George. Calculator Calculus. EduCALC Publ., 27953 Cabot Road, Laguna Niguel, CA 92677. (1980)

8. Mier-Jedrzejowicz, W. Extend Your HP-41. (publ. by author) 40, Heathfield Road, London W38EJ, United Kingdom. (1985)

9. Phillips, William C. Data Processing on the HP-41C/CV. Volume I: Fundamentals of Program Design and File Processing. EduCALC Publ., 27953 Cabot Road, Laguna Niguel, CA 92677. (1983)

10. Simon, William. Mathematical Techniques for Physiology and Medicine. Academic Press, NY. (1972)

11. Smith, Jon. Scientific Analysis on the Pocket Calculator. (Second Edition), Wiley and Sons, NY. (1975)

12. Wadman, T. and C. Coffin. An Easy Course in Programming the HP-41. Grapevine Publ., Inc., Corvallis, OR. (1983).