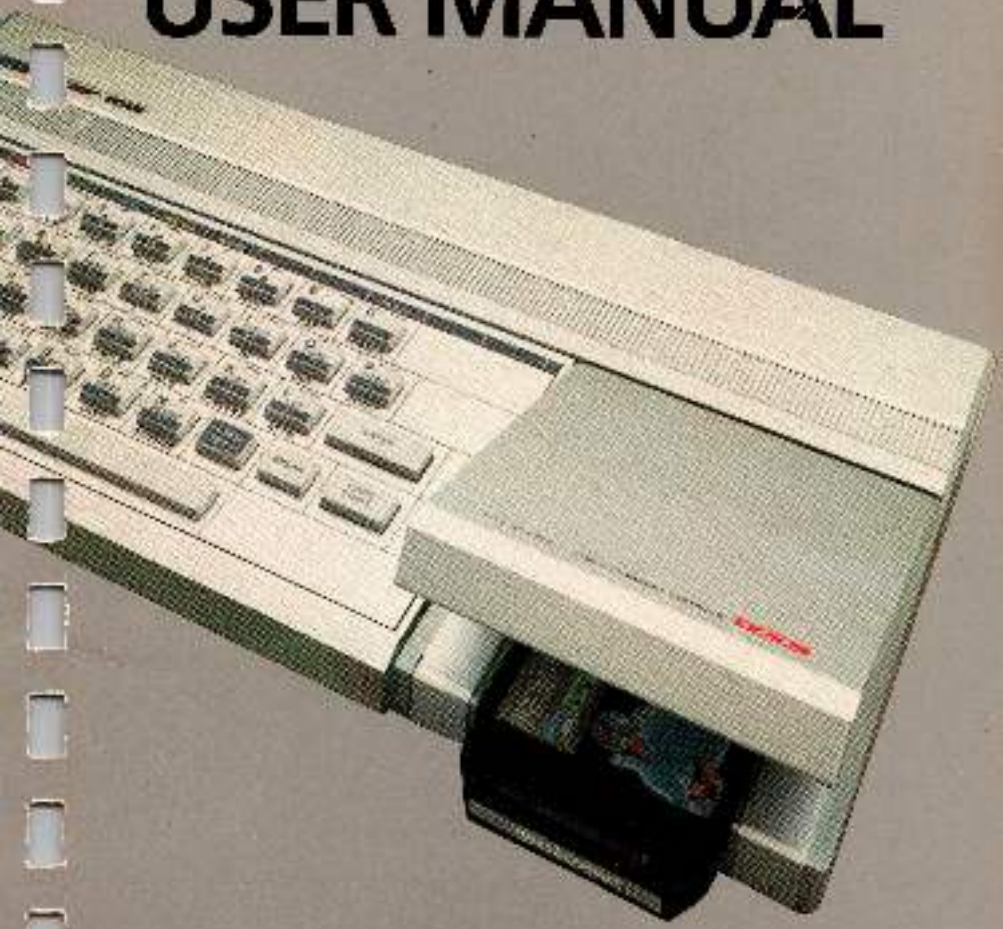
TIMEX sinclair

2068 PERSONAL COLOR COMPUTER

USER MANUAL



TIMEX sinclair

2068 PERSONAL COLOR COMPUTER

USER MANUAL

Charles F. Durang
Author

Judith Richland
Graphic Design

Special Acknowledgements

Charles F. Durang
Author
N. Attleboro, MA

Judith Richland
Graphic Design
Richland Design Associates
Cambridge, MA

We would like to acknowledge the many people who gave us their feedback, comments, and contributions:

Bruce Brown
Connecticut Computer Society
West Hartford, CT

Barbara Cobbol
Naugatuck, CT

Gregory Collin, Ph.D. and Staff
Urban Schools Collaborative
Northeastern University
Boston, MA

Nancy Gardner
Cambridge, MA

Jack Hodgson
Sinclair/Timex User Group
Boston Computer Society
Boston, MA

Larry Johnson
Boston Museum School
Boston, MA

Monique and Marcella Paris
Waterbury, CT

Bill Russell and Central Pennsylvania Users Group
Centre Hall, PA

Serif & Sans, Inc.
Typography
Boston, MA

Jackie Strassberger
Belmont, MA

Russ Walter
The Secret Guide to Computers
Boston, MA

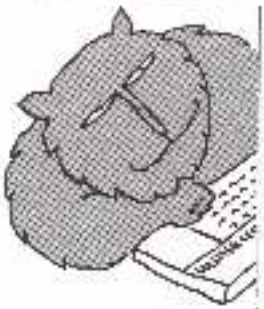
Ellen Weinberger
Medford, MA

and especially
Steven Vickers and Robin Bradbeer,
from whom we learned much.

plus the letters from the consumers and a very special thanks for all of the time and effort contributed by Timex personnel.

Susan C.T. Mahoney
Project Manager
Timex Computer Corporation

© 1983 by Timex Computer Corporation
© 1982 by Sinclair Research Limited



This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a TV Interface Device in accordance with the specifications in Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- reorient the receiving antenna
- relocate the computer with respect to the receiver
- move the computer away from the receiver
- plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful: "How to Identify and Resolve Radio-TV Interference Problems". This booklet is available from the US Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

WARNING: This equipment has been certified to comply with the limits for a TV Interface Device pursuant to Subpart J of Part B of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the TV Interface limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

90-Day Limited Warranty

Congratulations on the Purchase of Your Timex Sinclair Computer!

We hope you'll take the time to read the owner's manual. This will help you to use your Timex Sinclair Computer most effectively and with the greatest of pleasure.

Your new Timex Sinclair Computer, incorporating the latest electronic technology, has been manufactured under stringent quality control standards. Yet, no matter how well designed and constructed, your computer may at some time require service.

To assure that you enjoy the traditional satisfaction of owning a Timex product, Timex computer repair service offers:

- 90-DAY LIMITED WARRANTY
- LOW COST 12-MONTH SERVICE CONTRACT
- FACTORY REPLACEMENT PARTS
- RELIABLE REPAIRS
- PROMPT RETURN OF YOUR COMPUTER

The Timex Computer Club

The Timex Computer Club is an exclusive group of Timex Sinclair Computer Owners. Membership in the Timex Computer Club will allow you to increase your enjoyment of your Timex Sinclair computer. As a member, you will receive regular easy notice of Timex Computer Corporation technological advances, new hardware and software products, creative programming ideas and special services and software offers. You will also be able to share computer ideas and achievements with other club members all over the country! **For enrollment see card in back of book.**

NOTE: The 90-Day Limited Warranty on your Timex Sinclair Computer is in no way affected if you choose not to send us the Purchase Information Card. However, we must have the information to enroll you in the Timex Computer Club.

90-Day Limited Warranty

Basic Coverage. This Timex Scholar Computer is warranted to the owner for a period of 90 days from date of original purchase against defects in manufacture. This Limited Warranty is given by Timex Computer Corporation—not by the dealer from whom it was purchased.

What Timex Will Do: If a defect in manufacture of the Computer is discovered within 90 days from date of original purchase, Timex Computer Corporation will, at its option, repair or replace the defective unit.

What You Must Do: You must return the Computer, with sales receipt, indicating date of purchase, to Timex Product Service Center with a written explanation of the reason for the return. It is recommended that you enclose both cables, TV/Computer switch, and Power Plug with your shipment.

Return your unit, postage pre-paid to:

Timex Product Service Center
P.O. Box X
7004 Murray Street
Little Rock, AR 72203

*To protect against in-transit loss, we recommend you insure your Computer.

Limitations:

THE ABOVE REMEDY IS EXCLUSIVE. TIMEX COMPUTER CORPORATION LIMITS THE DURATION OF ANY WARRANTY IMPLIED BY STATE LAW INCLUDING THE IMPLIED WARRANTY OF MERCHANTABILITY, TO 90 DAYS FROM THE DATE OF ORIGINAL PURCHASE. TIMEX COMPUTER CORPORATION IS NOT LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGE. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state. Some states do not allow limitations on how long an implied warranty lasts, or the exclusion of limitation of incidental or consequential damages, so the above limitations or exclusions may not apply to you.

*This warranty is void if the Computer has been tampered or ill-treated or if the defect is related to servicing not performed by us.

Join The Club!

Get the Most out of Your Timex Sinclair Personal Computer

Join the Timex Computer Club!

The Timex Computer Club is an exclusive group of Timex Sinclair Computer Owners. As a member you will receive early notices and up-dates of Timex Computer Corporation technological advances, new hardware and software products, creative programming ideas and special offers. You will also be able to share computer ideas and achievements with other club members all over the country!

To enroll in the Timex Computer Club, simply fill out the card in the back of this manual and mail it to the address printed on the card. We welcome you and are looking forward to hearing from you.

Also, if you need to know:

- The location of the closest Timex Computer retailer
- How to get in touch with a local Timex Computer User Group or how to start one
- More technical information about Timex Computer Corporation (TCC) Products and Services

Contact our Toll Free "Hot-Line" 1-800 24 Timex
8:00 A.M. to 8:30 P.M. Monday-Friday Eastern Time
(subject to change)

Table of Contents

I Getting Started

Introduction: **You and the Timex Sinclair 2000** 1

Chapter 1: **How To Set Up the Computer** 5

Chapter 2: **Finding Your Way Around the Keyboard** 11
- MODE DELETE
- CAPS SHIFT CAPS LOCK
- SYMBOL SHIFT
- TRUE VIDEO
- INVERSE VIDEO

Chapter 3: **Telling the Computer What To Do** 27
- PRINT ENTER ?
- " " Strings SQR

Chapter 4: **Using Ready-To-Run Programs** 37
- Timex Command Cartridges
- LOAD RUN SAVE VERIFY
- LINE MERGE
- REM Report Codes

Chapter 5: **Using Colors** 53
- INK PAPER BORDER

Chapter 6: **Drawing Lines and Circles** 58
- PLOT DRAW CIRCLE

Chapter 7: **Sound** 66
- BEEP

II Programming for Beginners

Chapter 8: **Writing a Program** 71
- NEW Line #s GOTO
- scroll? BREAK CONT

Chapter 9: **Arranging Output on the Screen** 79
- ; , AT TAB : ↑↓
- EDIT

Chapter 10:	**Saving Time and Space with Variables** LET = Variables $ String Variables CLS CLEAR	89
Chapter 11:	**Mathematics with the T/S 2000** + - / ^ ( ) RND RAND INT	95
Chapter 12:	**Programs That Ask for Information** INPUT STOP READ DATA RESTORE	103
Chapter 13:	**Programs That Repeat: Looping** FOR...TO NEXT STEP LIST	113
Chapter 14:	**Programs That Decide: Branching** IF...THEN < <= > >= <>	121
Chapter 15:	**Programs within Programs: Subroutines** GOSUB RETURN	133
Chapter 16:	**Arrays** DIM Subscripts String Slicing SAVE DATA	143
III Special Features of the T/S 2000		
Chapter 17:	**Graphics** SCREENS	151
Chapter 18:	**User Defined Graphics** BIN POKE USR	163
Chapter 19:	**Time and Motion** INKEY$ PAUSE STICK	169

Table of Contents

Chapter 20:	**Color** BRIGHT FLASH OVER INVERSE	177
Chapter 21:	**Sound and Music** SOUND	185
Chapter 22:	**Checking Up** POINT ATTR FREE CODE CHR$	197
Chapter 23:	**Using the Printer** LPRINT LLIST COPY	203
Chapter 24:	**Input and Output** PEEK IN OUT OPEN CLOSE FORMAT ERASE MOVE CAT RESET	211
Appendices		
Appendix A:	**Review of T/S 2000 BASIC**	217
Appendix B:	**The Character Set**	239
Appendix C:	**Display Modes and Memory**	247
Appendix D:	**The System Variables**	261
Appendix E:	**Using Machine Code**	267
Appendix F:	**Keyword Table**	271
Appendix G:	**Index**	279
Appendix H:	**Report Codes**	287

Additional commands covered in Appendices:

PEEK ON ERR DEF FN FN STR$ VAL VAL$ LEN EXP SGN SIN COS TAN LN PI ASN ATN

You and the Timex Sinclair 2000

Chapter Preview

Throughout this manual, the term TS 2000 is used to refer to any of the models in the Timex Sinclair 2000 series of computers.

What Is a Computer Good For?

Your new Timex Sinclair 2000 computer is a very special instrument. It is a tool that can increase the power of your mind as a hammer or a wheelbarrow assists your muscles. For the beginner, it is easy to use and easy to understand. For the expert, it is an extremely sophisticated and powerful device.

Let's explore the comparison we have suggested: the Timex Sinclair as a tool, like a hammer or wheelbarrow.

From the beginning of time, humans have invented tools to help protect their reach, supplement their power, and increase their stamina. For almost as long, we have supplemented our minds with tools.

It was hard for us to remember a large number of things, or specific numbers, so we kept track of the sheep in our flock with notches on a stick, or pebbles in a bag. We invented writing to keep records, in words and numbers.

It was hard for us to manipulate many or large numbers, so we devised written mathematical systems, and increasingly complicated machines, leading eventually to the "adding machine" or calculator.

We also found it difficult to perform boring operations over and over again without making mistakes, so again we invented machines and systems (like "accounting") to help us keep track.

The computer is the ultimate machine to assist our minds. It can remember—and find—vast amounts of information. It can manipulate this information in ways beyond what we can do "in our heads." It can perform tedious repetitions of simple tasks over and over and never make a mistake. It can do for us all the things we have trouble doing, and it can do some things for us much more quickly than we could do them alone.

Just as we cannot drive nails with our hands, but can with a hammer, and cannot carry hundreds of pounds of garden soil in our arms, but can in a wheelbarrow . . . so the computer helps us do things we can't do alone.

But humans have to guide the hammer and wheelbarrow in their work. And we must decide what problems need solving with a computer, and how we must go about solving it. The computer is a good laborer, but you are the foreman.

Some of our other tools have more than one use—you can pull nails out as well as drive them in with a claw hammer—but the computer has

thousands of uses. It is as much a "generalist" as a human being, and can do all kinds of startlingly different things . . . as long as you tell it how.

Let's begin by looking at some of the things your Timex Sinclair 2000 can do, and how you guide it in its efforts.

Using Your Timex Sinclair 2000

The Timex Sinclair 2000 is a machine you can use for many different purposes, in many different ways. You can start in just a few minutes with pre-recorded programs to

- keep household records
- play games
- supplement your child's education
- assist in your work
- and learn more about computing.

We said that the computer could do many things, but only if you showed it how. In Part II, we will show you how to write your own programs — lists of instructions for the computer. And we will go on from there to more advanced programming; with this and other books, you can become as skilled with the computer as you choose. The Timex Sinclair 2000 can go with you as far as you decide to go on this journey: unusual for a home computer, it can handle up to 16 megabytes—16 million characters—of information. It can play music you compose through up to four individually controllable sound channels. And it has advanced color and graphic capabilities, including a dual screen mode for animation, a full width mode and Extended Color Mode.

Note: 64 characters, plus two 8 character-wide margins exactly fills a standard 8½ column format page for word processing applications.

But one of the best things about using the TS 2000 is that you don't have to invent your own programs. If you write a program, you can save it

to be used by the computer over and over, whenever you need that particular job done. So it follows that programs written by other people can also be used by the computer. This means you can put the TS 2000 to many uses with programs that have already been written.

Just as you can insert a special wheel into a sewing machine and have it do a particular stitch, you can insert a program into your computer and have it perform a certain task: run through educational drills for the kids, provide the environment for an "adventure" game, save and adjust quantities in recipes, help you with your tax records — all without your having to learn to write programs!

We hope you'll eventually want to learn how to write your own programs—it's good thinking practice even at the beginning level—but first we'll talk about how you can use your Timex Sinclair 2000 immediately, with prerecorded programs that are widely available.

Part I of this book gets you started, setting up the computer and learning the keyboard. It also shows you how to use prerecorded programs. In Part II we study the basics of programming, and in Part III some of the special features of the TS 2000.

The Appendices include useful information you may want to refer to at any time, and they also contain some material that is addressed to experts who know computing well and need details of how the Timex Sinclair 2000 functions internally.

Congratulations on joining the Computer Age! You will surely find it enjoyable, useful and educational.

How To Set Up the Computer

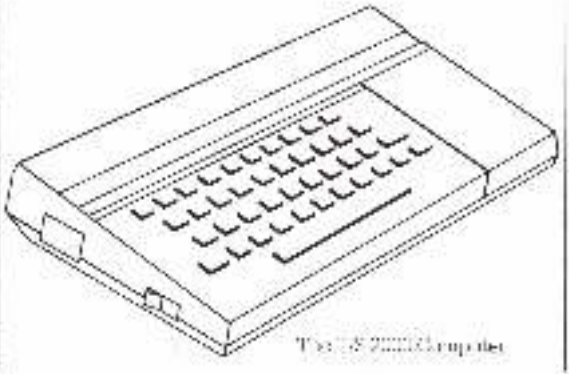
1

Chapter Preview

This chapter shows you how to connect the T/S 2000 to your TV and cassette recorder, and start using it right away.

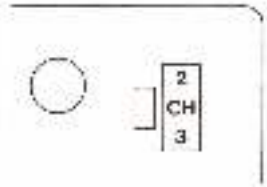
In the box, we've provided everything you need to start using your Timex Sinclair 2000 computer immediately, with your own (color, portable) TV set and an inexpensive cassette recorder.

Here's what you should have:



The T/S 2000 Computer

Chapter 1: How To Set Up the Computer



Bottom of computer

1. **The Timex Sinclair 2000 itself.** Though tiny, it has as powerful as computers that filled a room only ten years ago.

   On the top is the keyboard; on the bottom you'll find a switch labelled Ch. 2/Ch. 3. More about that in a moment.

   On the back you'll see sockets marked TV, POWER, MIC, EAR, and MONITOR, and a long slot where you can attach "peripheral devices" like the Timex 2040 printer.

Rear view of computer

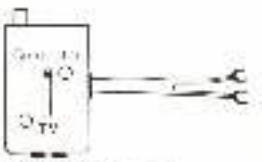
Left side of computer

   On the left side you'll find the ON/OFF switch, and sockets on both left and right sides where you can plug in joysticks.

TV cable

2. **A television cable**—either a long one with two larger plugs at one end or two shorter cords (approximately 4 feet lengths) to connect the computer to your TV set. If you have the two 4 foot cables, use the double female adapter to join them.

TV cables with adapter



Transfer switch box

3. **A transfer switch box**, allowing you to switch between receiving television programs through the antenna and using the TS 2000.

Dual audio cable

4. **A shorter dual audio cable**—the one with two smaller plugs at each end—for connecting the computer to your tape cassette recorder.

Power supply unit

5. **A power supply unit**, with a plug for the wall socket and one for the computer

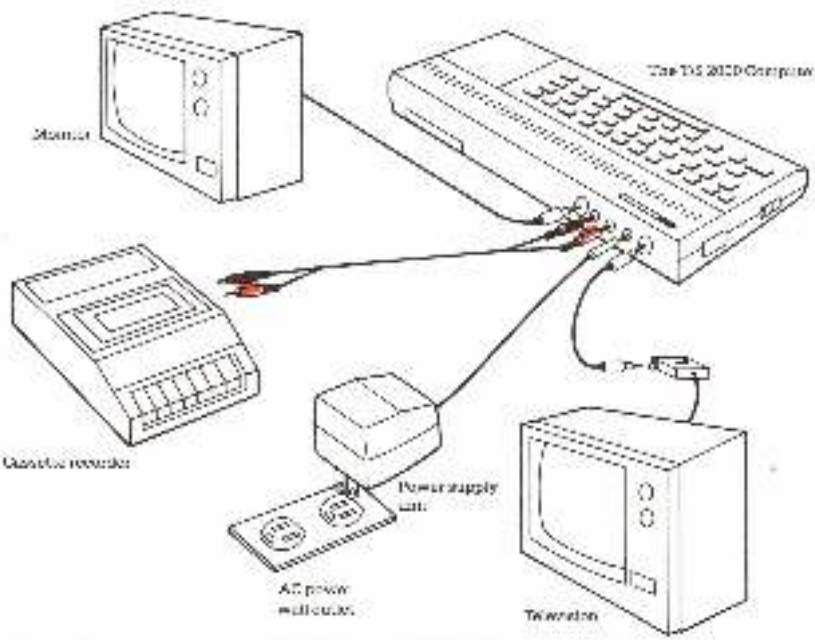
Manual

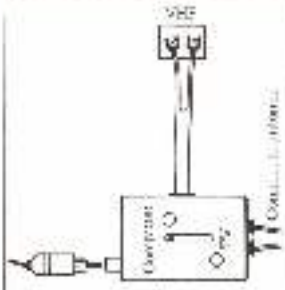
6. **This manual.**

7. **Some free software to help you get started.**

Here's how to quickly connect your Timex Sinclair 2000 (turn page for details):

The illustration shows how you can use a TV or a monitor with the TS 2000. Only a television will be necessary to operate the computer.





Connect transfer switch box to VHF terminals on TV

UHF/VHF matching transformer

First, disconnect the VHF TV antenna wires from your television set (you can leave the UHF wires alone).

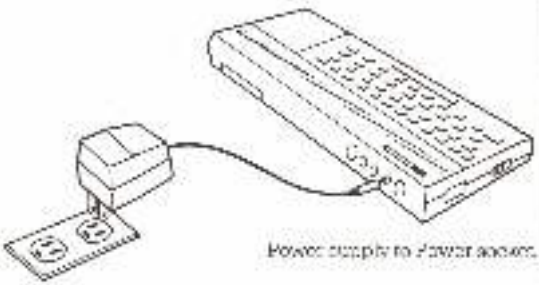
Connect the wires from the transfer switch box to the terminal screws on your TV set instead, and connect the antenna wires to the screws on the transfer switch box.

Plug the long connecting cable in at the transfer switch box and into the TV socket on the TS 2000.

If you already have a transfer switch box on the TV, for a TV game or another computer (like the Timex Sinclair 1000), try leaving it on; they all work pretty much the same, and chances are you can use the existing one. Then you can use the one that came with your TS 2000 for a second TV set and connect either of your computers to either set.

Note: If you have cable TV, or a 75-ohm antenna lead (a round wire ending in a screw terminal), you will need a small device to convert this to the flat, two-wire lead that connects to the transfer switch box. There are several versions of this device, which may be called a "UHF/VHF matching transformer," "75-to-300-ohm converter," "cable adaptor," or "VCR adaptor." Someone at your local electronics store will be able to help you; the cost will be from three to ten dollars. You may have to contact your cable company if their wire goes into your set instead of being attached to the back.

Second, plug the power supply into the wall unit and the POWER socket on the computer.



Power supply to power socket

Channel 2/Channel 3

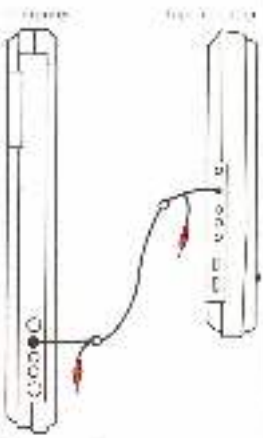
Third, turn on the TV. Set it to channel 2 or channel 3, whichever one is not being used for broadcasting in your area. Turn the sound all the way down. Make sure the switch on the bottom of the TVS 2000 is set to the same channel. Use a ball point pen or similar instrument to set the channel switch to the desired channel.

Now, turn on the computer with the switch on the left side.

You should have a picture like this on the screen.

The copyright notice at the bottom of the screen means the computer is ready for action.

Fourth, connect your recorder to the computer with the dual audio cable. Connect the earphone socket on the recorder to the EAR socket on the computer. In order to load a program from a cassette into the computer, and the microphone socket of the recorder to the computer's MIC socket, to save programs you've added information to, or write on yourself. Make sure you use the same-coloured plugs for the EAR to EAR connection and for the MIC to MIC connection. More about this in Chapter 4.



Connect EAR to EAR.
Connect MIC to MIC.

Note: The picture on your TV screen should be clear. If you are getting interference, try the following steps in order:

1. Adjust the tuning control on the set (be sure the Automatic Fine Tuning is off), then try the brightness, contrast, and horizontal hold (horizontal is usually on the back of the set);
2. Make sure the computer is set to the same channel as the TV set, and is turned on.

3. Move the computer away from the TV set or, if possible, place it lower than the set.
4. Plug the computer into a different outlet from the one being used for the television set. Often outlets on opposite walls of a room are on different branch circuits.
5. You may wish to try a longer (shielded) cable between the switch box and the computer to move the TS 2000 still further away from the TV.
6. Consult an experienced radio/TV repairman; your set may need adjusting.

Now you are ready to use your Timex Sinclair 2000.

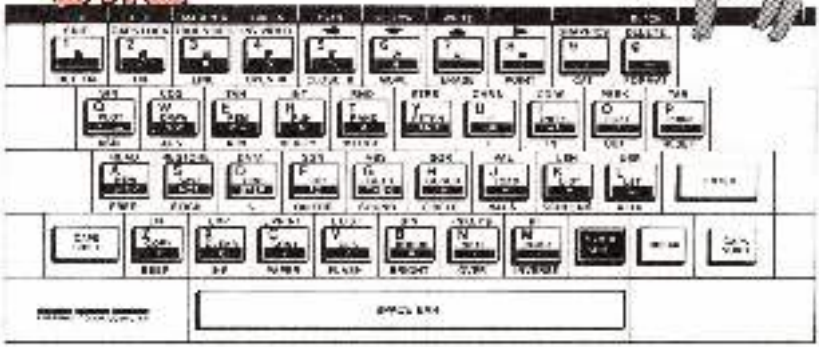
Finding Your Way Around the Keyboard

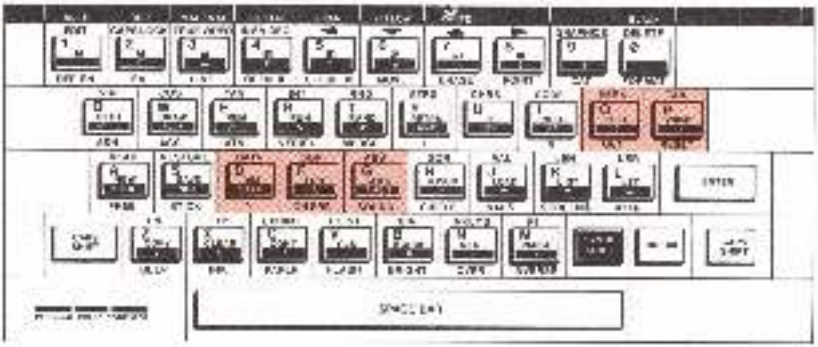
2

Chapter Preview

How the cursors—5, 6, 7, 8, 0—and CAPS SHIFT and SYMBOL SHIFT keys help you use all the functions on all the keys. We investigate DELETE, CAPS LOCK, TRUE VIDEO and INVERSE VIDEO, and learn how the left and right arrows work.

You make the Timex Sinclair 2068 perform by pressing the keys on its keyboard (notice we didn't say "typing," because it's easier than that, as we





shall see). At first glance, the keyboard looks impossibly complicated—each key has five or six labels—but you'll quickly learn how to use it. By the end of this chapter, in fact.

If you do know how to type, you'll notice that the largest letters on the keys—the letters and numbers, in most cases—are arranged just like a typewriter's keyboard.

The good news, for typists and non-typists alike, is that you won't have to type in your commands to the computer. Instead, you'll find they are indicated by complete words on and around the keys. In many cases, these "keywords" are written above or near the key for the letter that the word begins with. For example, notice the word PRINT on the P key, and POKE and PEEK on and above the key just to the left of it. Look at the keywords on or above the D, F and G keys, too.

All of those words and symbols on and around each key mean, of course, that each key can perform many different functions as you give instructions to the computer.

What a key "means" to the computer when you press it depends on two things:

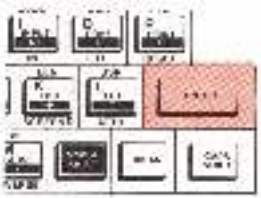
1. Which "cursor" is on the TV screen (we'll see a cursor in a moment) and

2 Whatever you press either CAPS SHIFT or the SYMBOL SHIFT key while you press another key.

Let's do a few things with the keyboard to see how to get all the different meanings from the keys. First, we'll ask you to do EXACTLY as we tell you. Then, we'll have a chance to experiment.

The K Cursor . . . Words on the Keys

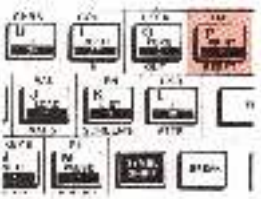
Set up and turn on the computer and television set as we did in the last chapter. You should now have the copyright notice in the lower left-hand corner of the screen.



Press the key marked ENTER.

Now, instead of the copyright notice, you have a flashing K in the lower left-hand corner of the screen. (It's actually alternating between a black-on-white K and a white K in a black square. Adjust the tuning on your TV if you need to, to get it clear.)

We'll use, in our examples, the keys you'll use most often on your Timex Sinclair 2000. Start with the P key, near the upper right-hand corner of the keyboard.

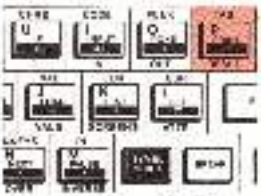


With the K cursor on the screen, press and release the P key.

Two things have happened: the word PRINT has appeared at the bottom of the screen, and the cursor has changed to a flashing L.

K means *KEYWORD. Whenever the* K *is on the screen, pressing a key will cause the "keyword" on the key (like PRINT, POKE, INPUT, etc.) to appear on the screen.*

The cursor has also moved to a point after the word PRINT. The cursor marks the spot on the screen where the next item will be printed, in a while we'll see how to move the cursor so as to put items where we want them, or to get to items we want to change.



Press and release the P key.

The L Cursor . . . the Main Characters on the Keys

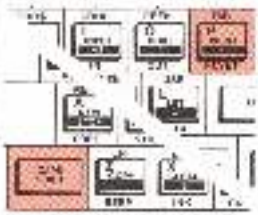
Now press and release the P key again.

You will get a lower case p, and the screen will look like this.

Incidentally, we keep saying "press and release" because the TS 2000 keyboard has an "auto-repeat" feature. If you hold down a key for more than a second or so, it will repeat the character for as long as you press the key. Keep this in mind, and just press the key briefly if you need a single character or keyword.

Press a few more letter keys, and a few number keys.

L means LETTER. When the L is on the screen, pressing a key will produce the main symbol on the key — the letter or number just above the keyword.

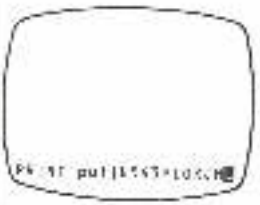


While holding CAPS SHIFT, press the P key

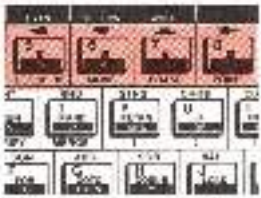
The CAPS SHIFT. . . Capital Letters

Notice, at either end of the bottom row of keys, a key marked CAPS SHIFT. While holding it down, press the P key again. Aha, a capital P.

Try this with a few other letter keys. But not the number keys, yet.

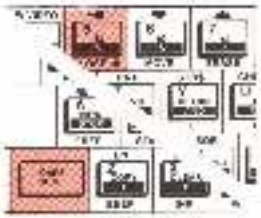


Let's look at the number keys. The words and symbols just above the keys (but below the names of colors) are obtained by holding CAPS SHIFT and pressing the keys. For instance:

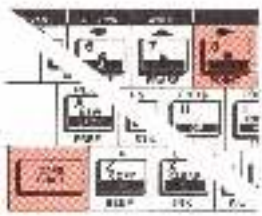


While holding CAPS SHIFT, press 5, 6, 7 or 8 for arrows

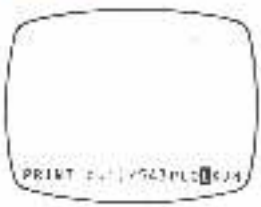
Start with the "arrow" keys: 5, 6, 7 and 8. Holding CAPS SHIFT, press 5. Notice the cursor moving to the left, in among the letters. Try it a few more times. Try holding it down (while still holding down the CAPS SHIFT).

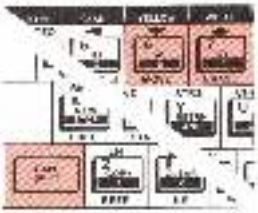


Try this: hold CAPS SHIFT, press 8 and then — while holding 5—let go on CAPS SHIFT.



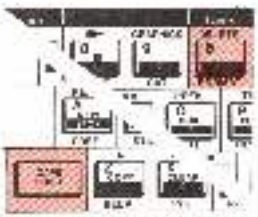
Do the same with the 8 key and move the cursor the other way. Move it back and forth a few times.





Try the 6 and 7 keys.

The screen blinks, but the cursor doesn't move. These keys are used to move between lines in BASIC programming, and we'll discuss that much later.



While holding CAPS SHIFT, press the 0 key.

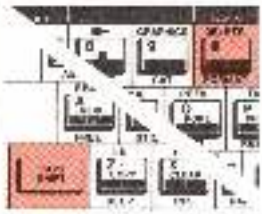
The DELETE Key

Move the cursor back to the end of the line, using CAPS SHIFT and the 8 key. Then hold CAPS SHIFT, and press the 0 key. You can DELETE a character at a time, in reverse! If you hold it, you can erase many characters in a row. (Keywords are entered with one stroke, as they are printed.)

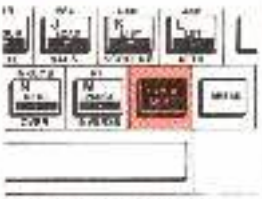
Note: DELETE works somewhat differently when the K cursor is on the screen. This will be explained in detail in Chapter 15. For now, note that if you use the auto-repeat feature (holding down the key) to erase an entire line of text, you'll eventually get back to the K cursor. Then DELETE will appear on the screen as a keyword ... which you can then delete!

This is very important. Now you know how to "erase" if you have something on the screen you don't want.

Try using the 5 and 8 keys to move the cursor to a particular letter in the middle of the line, and then DELETE it. Remember, it deletes the character to the left of the cursor.



Now is a good time to practice the "auto-repeat" feature. Hold down a key until its character is printed several times on the screen. Then hold down CAPS SHIFT and DELETE until the characters are erased.

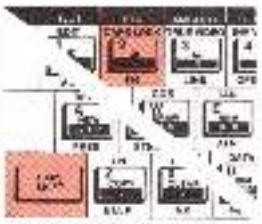


The SYMBOL SHIFT key

SYMBOL SHIFT . . . Words & Symbols in Black Bands on Keys

Near the right hand CAPS SHIFT key is a key marked, in a black box, SYMBOL SHIFT (actually, it's abbreviated SYMBL SHIFT to fit on the key). Can you guess what you get by holding SYMBOL SHIFT down and then pressing a key?

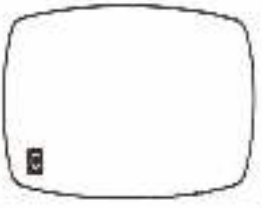
The black band is the clue: you get the word or symbol in the band on the key. Try a few, and erase them using DELETE.

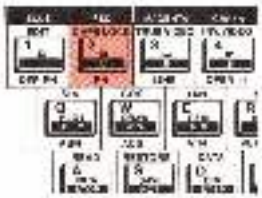


Press CAPS SHIFT while pressing 2.

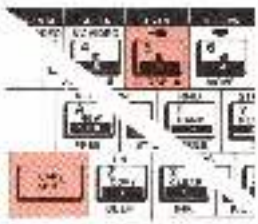
CAPS LOCK . . . the C Cursor

Try holding CAPS SHIFT and pressing 2. Notice that the cursor changes to a flashing C. Now try typing a few letters. And a few numbers. The CAPS LOCK feature locks you into capital letter mode, just as on a typewriter, but it lets you use the numbers, too.



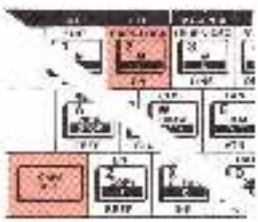


The CAPS LOCK feature keeps you in capital letter mode, but always has a number.



Press CAPS SHIFT while pressing the 5 and 6 keys

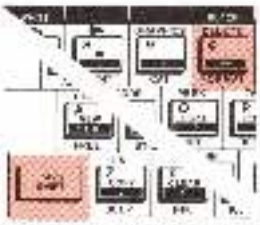
You can still get the words and symbols above the numbers with CAPS SHIFT. Try holding CAPS SHIFT and pressing the 5 and 6 keys. (For now, don't use the 1, 3, 4, or 8 keys with CAPS SHIFT. We'll get to them.)



Try pressing CAPS SHIFT and the 2 key again. It changes back to the L cursor. Press it again (holding CAPS SHIFT). It turns the CAPS LOCK on and off, alternately. (Unlike using a typewriter, you can't release the CAPS LOCK by just pressing CAPS SHIFT.)

All the letters you type when the C *cursor is on the screen will be capitals.*

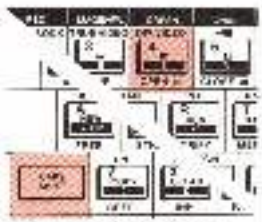
Probably by now you have a long line of miscellaneous characters on your screen, maybe even two lines' worth. Better DELETE them all.



Remember: the 0 key deletes the character to the left of the cursor

Remember: CAPS SHIFT and the 0 key DELETE the character or keyword to the left of the cursor; CAPS SHIFT with the 5 and 8 keys move the cursor to the left and the right respectively.

Now you're back to the K cursor. Experiment some more if you like. You can always use DELETE to get back to the beginning.

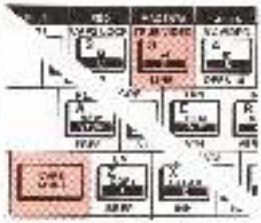


Hold CAPS SHIFT while pressing 4

INVERSE VIDEO and TRUE VIDEO

There are a few other letters above the top row of keys we want to investigate. Hold CAPS SHIFT and press 4 to get INV VIDEO (the cursor will not change). Then type a few characters.

This is the INVERSE VIDEO mode. Characters are printed in white on a black background. Unfortunately, no cursor tells you that you are in this mode if you forget. Of course, you can always DELETE unwanted characters. See what you get with CAPS LOCK on and off.



Press CAPS SHIFT while pressing 3 for TRUE VIDEO

To get out of INVERSE VIDEO mode and back to "normal" press CAPS SHIFT and 3: TRUE VIDEO.

Something funny happens if you go back, with the cursor, to the middle of a line of INVERSE characters and press TRUE VIDEO: the rest of the line changes.

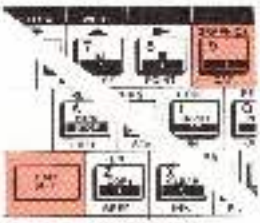
If you then press INVERSE VIDEO, it all changes back again.

To insert characters in TRUE VIDEO in the middle of a line of INVERSE characters, move the cursor to the spot where you wish the insertion, press TRUE VIDEO.

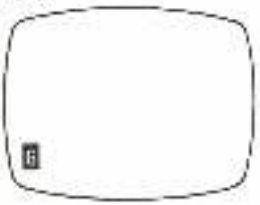
type the characters,

then press INVERSE VIDEO again to return the rest of the line to white-on-black.

You'd to follow the same steps to insert INVERSE characters into the middle of a TRUE sequence.



Press CAPS SHIFT while pressing 9 for the graphic mode



The G Cursor . . . Graphics Mode

On the number keys in the top row of the keyboard, you'll see small graphic symbols. To print them on the screen, you must switch to graphics mode. Holding CAPS SHIFT, press the 9 key.

Notice that the cursor has changed to a flashing G.

Now if you type any of the keys with a graphic symbol on them (the number 1-8 keys), you'll get that symbol. Try some.

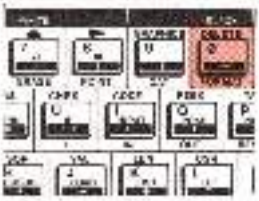
Note: The symbol you will obtain is shown by the gray or any colored portion of the square on the key—not the black portion. At this point it may seem odd that the black part of the design on the key does not correspond to the black figure on the screen. Later, however, we will see that the symbol on the screen is not always black, but "INK colored" and the part of the square that is key-colored is also INK-colored.

Try a few.

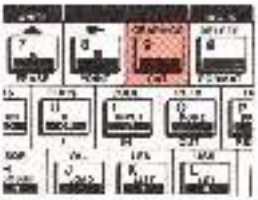
You can obtain 16 different graphics from the eight keys. The inverse of each symbol—the portion of the square in black on the key—is obtained by holding either CAPS SHIFT or SYMBOL SHIFT while you press the key. You can see the difference most easily by pressing the 3 key first unshifted and then SHIFTed.

(If you are in INVERSE VIDEO mode, an unSHIFTed key will give you the portion of the square in black on the key.)

The lower three rows of keys—without graphic symbols—will give you capital letters for A through U, and a curious mix of symbols for V-X. None of this is very useful just now, but later on, in Chapter 18, we will use the A-U keys to design our own graphic symbols!



Try to DELETE. Notice you don't need to hold the SHIFT down in order to DELETE in graphics mode.



Press 9 to leave the graphics mode

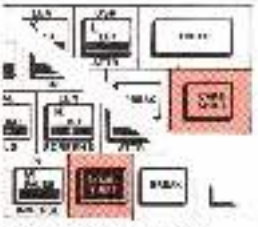
To leave graphics mode and return to the L cursor, press 9. (To re-enter graphics mode, you must press 9 while holding CAPS SHIFT.)

By the way, you cannot change from TRUE VIDEO to INVERSE VIDEO (or vice versa) while in the graphics mode. You must leave the graphics mode to do it.

When the G cursor is on the screen, you can get the graphic symbols on keys 1 through 8.

The E Cursor . . . the Words above the Keys

Press CAPS SHIFT and SYMBOL SHIFT at the same time. The cursor changes to a flashing E. This is called the *extended mode*. When the E cursor is on the screen, you can obtain the words written on the keyboard above each key.

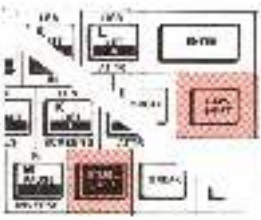


Press CAPS SHIFT and SYMBOL SHIFT to activate extended mode

Extended mode

Since these are commands or mathematical functions, the E changes back to L after you press one key. It works like the keyword X cursor. Try a few.

The top row of keys is a special case. As we saw, the words just above the keys were reached from the L cursor, using CAPS SHIFT. The extended mode is used to select colors. This works well in programs, as we will see. But if you play with TRUE VIDEO, INVERSE VIDEO, CAPS and SYMBOL SHIFTS, and the various color keys, you'll find the results in the immediate mode are largely unsatisfactory and often unreadable. Don't worry about it for now.



**SHIFT Keys With E Cursor . . .
Words & Symbols under Keys**

If you enter the extended mode by pressing CAPS SHIFT and SYMBOL SHIFT, then hold SYMBOL SHIFT while pressing another key, you'll obtain the function or command written under the key. Once again, the cursor returns to L.

(Most of the time you can use CAPS SHIFT instead of SYMBOL SHIFT to obtain functions written under keys. In this manual, however, we'll refer only to SYMBOL SHIFT.)

By the way, if you enter extended mode by mistake and decide you want to get out of it without typing anything, just press both CAPS SHIFT and SYMBOL SHIFT at the same time again.

When the E *cursor is on the screen, you get the keywords shown two keys up, by pressing* SYMBOL SHIFT, *the keywords under the keys.*

By now, you ought to feel reasonably comfortable with the keyboard. There are a lot of commands and characters available on it, but most of the

time you will be in the **L** mode. Here is a reminder diagram showing the cursors and SHIFT keys needed to obtain the different symbols on the keys:

There are three symbols that do not even show on the keyboard, but are available for use! They are:

© — the copyright symbol. Under the P key, when you are in extended mode (the **E** cursor) from *keyword* mode (the **K** cursor) you obtain the command RESET as shown on the keyboard by pressing the P key while holding either SHIFT key. When you enter extended mode from *letter* mode (the **L** cursor), the © symbol appears when you press P with a SHIFT key.

{ — the left bracket. Obtained the same way, instead of ON ERROR, from the F key.

} — the right bracket. The same, instead of SOUND, from the G key.

We have not covered all the commands and symbols on the keyboard. Some you will know from mathematics, some you will know if you have programmed other computers in BASIC, and some are for use with the special features of the Timex Sinclair 2000 and the peripheral devices that can be connected to it.

Repeat this chapter any time you like, or use the Timex Sinclair 2000 Keyboard Tutorial program supplied.

You are well on your way!

Note: Appendix F, the Keyword Table, can be very helpful to you in your programming. (You may want to glance at it now.) It tells you how to obtain any of the available keywords and single-key functions on the TI-99/4A.

The Keyword Table can be useful if:

1. You want to use a certain keyword and can't locate it on the keyboard.
2. You aren't sure if a certain word in a program is a keyword.
3. You are having trouble entering a program line (you receive a syntax error marker) or running a program (a report code stops the program at a particular line)—it may be that you have typed in a word, letter by letter, that you should have entered as a keyword.

It is especially easy to make this mistake with the keywords **AT** and **TO**, and the Keyword Table can remind you to look for the keyword.

Telling the Computer What To Do 3

Chapter Preview

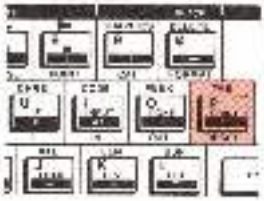
PRINT and ENTER help you start giving orders to the T/S 2000. Learn how to put words and numbers on the screen. We look at the quotation marks, "strings," and how to use functions like SQR.

As we have said, the computer is a very versatile machine. It can do many things as long as it is told what to do. But it has to be told in words it can understand and in small steps it can execute.

Now you can use your knowledge of the keyboard to start giving directions to your computer.

The Timex Sinclair 2000 is built to understand orders given to it in BASIC (which stands for "Beginner's All-Purpose Symbolic Instruction Code"). Invented at Dartmouth College, BASIC looks more like English than other computer languages and is very easy to use.

The keywords above the keys, which are commands to the computer, are in English. They are also in BASIC, which means each word always means exactly the same thing.



For example, the keyword we will probably use the most is **PRINT**. In English, this can mean:

1. Make letters on paper with a writing instrument held in your hand,
2. Transfer letters from a printing press to paper, or
3. Publish.

There are probably several other meanings you could think of slightly different from these.

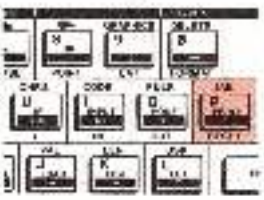
In BASIC, **PRINT** means only one thing:

1. Print on the screen whatever follows the word **PRINT**.

Let's have some more practice at giving the computer commands and see how it performs.

Start by setting up the computer so the screen looks like this:

Now we'll have the computer figure out the sum of 2 and 2, and show the result on the screen.



First, type **PRINT**. Press the P key, and the word **PRINT** appears. Remember, although you can spell out the word P,R,I,N,T, you give the TS 2000 its orders using the keywords above the keys. These appear if you press a key while the **K** cursor is on the screen.

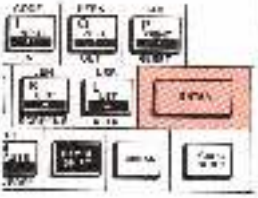
In case you haven't noticed, we are indicating keywords in this manual by printing them in **BOLD FACE TYPE**.

Besides the word **PRINT** appearing on the screen, you see the cursor has changed to **L**.

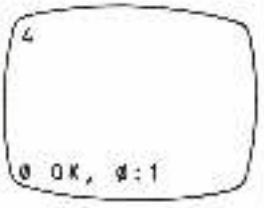
Next, type 2. The 2 appears on the screen, and the L moves to the right. Notice, by the way, that there is a space between the PRINT and the 2 on the screen. Timex/Sinclair BASIC keywords come with their own spaces, and you don't have to put them in (it won't do any harm if you do put in extra ones).

Third, type a plus symbol (+). Remember how we obtained symbols that are in the black bands on the keys — using SYMBOL SHIFT. Hold down SYMBOL SHIFT and press the K key.

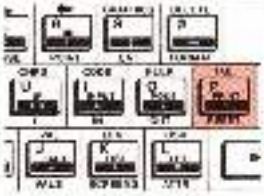
Next, type another 2. The screen will look like this:



Finally, press ENTER. Whenever you are finished with a line of a command, and want the computer to do something, you signal this by pressing ENTER.



The computer will compute and display the answer. Of course, you don't need a computer to figure that one out, but you can use it for more difficult math (see Chapter 11).

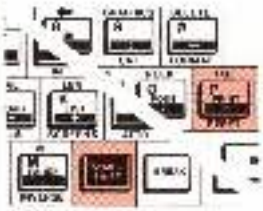


Let's try something else. The 0 OK report code is hiding a K cursor—if you press a key while a report code is on the screen, you'll get a keyword just as if a K was showing. So press P again and get PRINT.

It's okay that the answer to the previous calculation is still on the screen.

Now, using SYMBOL SHIFT, get quotation marks by pressing P again.

And press 2. And – (using SYMBOL SHIFT again). And another 2.



Press SYMBOL SHIFT while pressing P

Finally, close the quotation (SYMBOL SHIFT P). The screen should look like this:

Now press ENTER.

Aha. Let us examine why the difference. The first time, when we typed

PRINT 2 + 2

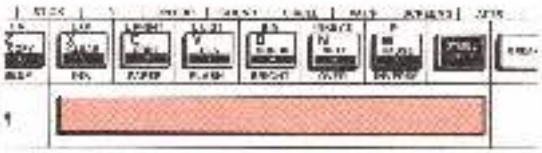
we were telling the computer to evaluate a mathematical expression and print the answer.

The second time, when we typed

PRINT "2 + 2"

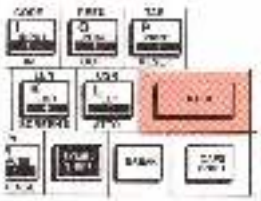
we were telling the computer, "Don't do any calculation, just print whatever is in quotes."

Try this (remember that, inside the quotes, you will have to put in spaces when you want them, using the space bar at the bottom of the keyboard):



PRINT " ANYTHING THAT APPEARS IN QUOTES
NO MATTER HOW LONG IT IS "

Don't forget to press ENTER when you are done.



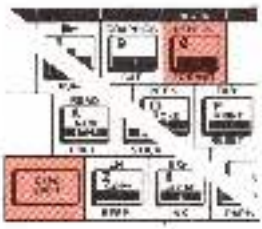
That's some computer jargon for you. Anything we put in quotes is called a string. This sounds odd at first, but it essentially means that the entire "string" of characters within the quotes is treated by the computer as a single item.

Now, press:

NEW ENTER

This erases the sentence from the screen. You can use NEW anytime you want to clear everything from the computer and start over as if you had just turned it on.

You can also pretend the copyright notice is a K cursor, just as you can with a report mode.



What To Do If You Make a Mistake

If you make a mistake, you can erase it with the DELETE key (CAPS SHIFT 0). Pressing DELETE erases the character or keyword just to the left of the cursor.

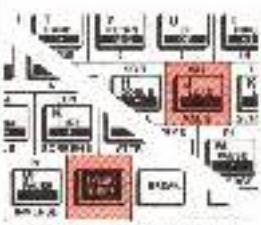
And, you can move the cursor to whatever you want to make a deletion by using the left and right arrow keys (CAPS SHIFT 5 and CAPS SHIFT 8).

Type this in:

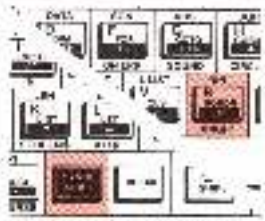
PRINT 5-*2

(Use SYMBOL SHIFT J for the minus sign, SYMBOL SHIFT B for the * — the TS 2000's multiplication sign)

and press ENTER.



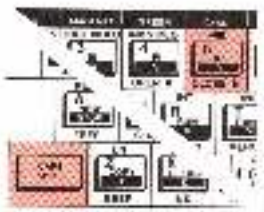
Press SYMBOL SHIFT while pressing J



Press SYMBOL SHIFT while pressing 2

The screen will look like this:

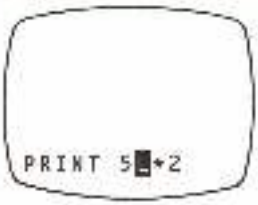
The **?** is the *syntax* error marker. It means that the TS 2000 cannot execute that command (it can't tell if you want to subtract or multiply). Suppose we meant PRINT 5+2. Let's make the correction, the way we did in Chapter Two.



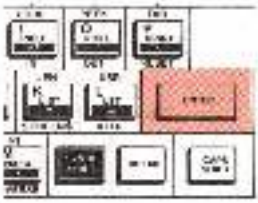
Press CAPS SHIFT while pressing 5

First, we need to move the **L** cursor to the scene of the crime. Press the left arrow (CAPS SHIFT 5, holding down the CAPS SHIFT key while pressing the 5 key). The cursor moves one place to the left.

Still holding down CAPS SHIFT, press the left arrow again, which puts the cursor in the proper position.



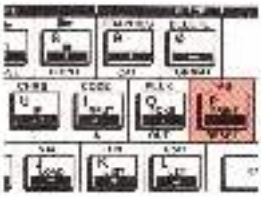
Now press DELETE (CAPS SHIFT 0) and we have this on the screen:



Now, we can press ENTER. We don't need to move the L cursor back to the end of the line; it doesn't matter where it is, as long as the rest of the line is correct. The cursor is just a place marker, for your eyes only.

You can also make insertions, as we did in Chapter Two, by moving the cursor to the desired location and then typing the character(s) you want to insert.

Time for practice:

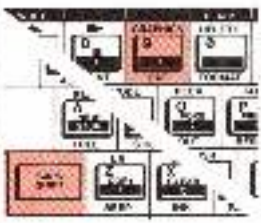


Try typing in PRINT " and then any sentence you like. Don't forget to close the quotes at the end. But don't press ENTER yet.

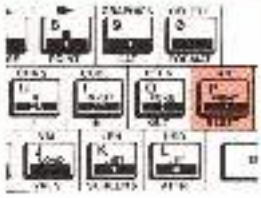
Practice moving the cursor backwards and forwards through the line with the left and right arrow keys.

Make any corrections you like, before pressing ENTER. Besides correcting errors you may notice, try adding *new* words by moving the cursor to the spot where you want to insert them and then typing the words.

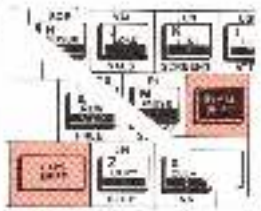
Now, press ENTER.



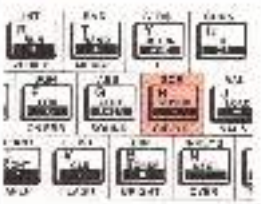
Practice making up a line of graphics, by pressing PRINT " and then getting into graphics mode with CAPS SHIFT 9. Remember you have to use CAPS SHIFT 9 (GRAPHICS) again to get out of the graphics mode in order to close your quotes and ENTER the line.



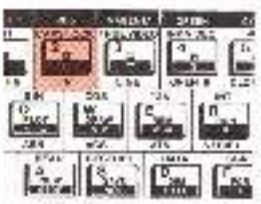
In case you are mathematically inclined, here's a quick early hint on the use of functions. Press PRINT, then enter the extended mode by pressing both CAPS SHIFT and SYMBOL SHIFT simultaneously.

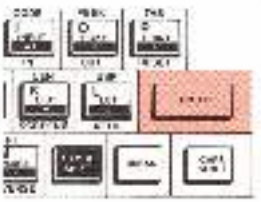


With the E cursor on the screen, press the H key giving you SQR (for *square root*).



Then press 2 and ENTER. (You don't need quotation marks.)





The computer evaluates the mathematical expression and prints the square root of 2 — 1.4142136 — on the screen.

We won't spend much time on the other functions in this manual, except to mention them in Chapter 11 and list them in Appendix A. If you are a mathematician, you will recognize the abbreviations over and under the keys. If not, don't worry about them.

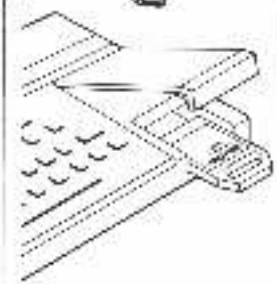
Summary

1. PRINT (keyword command located on the P key) tells the computer to print something on the screen.
2. If a number follows PRINT, the number will be printed on the screen. If a mathematical expression (like 2 – 2) follows PRINT, the expression will be evaluated and the result printed (for example, 4).
3. Anything in quotation marks after PRINT will be printed on the screen exactly as it appears. The material in quotes is called a *string* because it is a string of characters.
4. When you press ENTER, you signal the computer that you are finished writing your command and would like it carried out.
5. NEW clears the computer to start over.

Using Ready-To-Run Programs 4

Chapter Preview

How to use ready-to-run programs on Timex Command Cartridges or cassette tape, and store your own programs with LOAD and SAVE. We look at RUN, REM, LINE, MERGE, and VERIFY.



Inserting cartridge with the label side up

As we mentioned earlier, you can use programs which have been written by other people to operate your TS 2068. These publishers make programs, games, household applications, business subjects and home education programs—and others are available from many software publishers.

Timex Command Cartridges

When you buy programs on Timex Command Cartridges, all you have to do is insert them, with the label side up, in the cartridge port to the right of the keyboard. These programs are self-starting, and no programming knowledge is required. You just follow the directions to use the program.

Chapter 4: Using Ready-To-Run Programs

To insert a Timex Command Cartridge, follow this procedure:

1. Turn off the T/S 2000.
2. Lift the cartridge door.
3. Insert the Command Cartridge with the label side up.
4. Close the cartridge door.
5. Turn on the T/S 2000.
6. The program will begin.

If the program does not begin, repeat all steps in order. *Always turn your T/S 2000 off before inserting or removing a cartridge.*

Programs on Tape Cassettes

You can also purchase programs recorded on tape cassettes, and "load" them into the computer using a suitable cassette recorder.

Sometimes you will need to load a program into the computer from a tape. When you are done and turn off the computer, it will disappear from the T/S 2000's internal memory but, of course, you will still have it on tape to load and use again.

Often you will type in a program from a listing in a book in order to use it, and then will save it on to a tape cassette. The next time you want to use it, you won't have to type it in; instead you can simply load it from the cassette.

And sometimes there will be programs you will load from a tape, add data to, and then save the program with the new data on another part of the tape, separate from the original program.

Let's look at how each of these tasks is done.

Loading a Program from Tape

Important: Before using a tape recorder with your computer, please read the enclosed sheet entitled "Timex Sinclair 2000 Compatible Cassette Recorders." This sheet contains the latest recorder recommendations.

Every program should have a name, and any cassette that has more than one program on it should provide you with an index listing the names of all the programs on the tape.

With all the components of your system connected and turned on, as discussed in Chapter 1, make sure that your tape is rewound to the beginning and that the K cursor is on your TV screen.

Connect the EAR socket on the computer to the earphone socket on the tape recorder. Turn the volume control on the tape recorder to about three quarters of the maximum volume. If it has tone controls, adjust them so that treble is high and bass is low. (With a single control, set it at *high*, which will give maximum treble and minimum bass.)

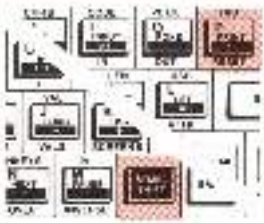
Then press

LOAD

which is what you get when you press the J key while the K cursor is showing (remember, whenever the K cursor is on the screen, pressing any key will give you the keyword command on that key).

You'll notice the cursor has changed to L. This means the computer will now give you the main symbol on any key you press, or — if you hold **SYMBOL SHIFT** down while you press another key—the shifted symbol, which is in the black band (the same color as SHIFT) on the key.

You need to tell the computer the name of the program you want to use, so you must put the name in quotes. Suppose you want to run a program for a game, called STAR ZAP.

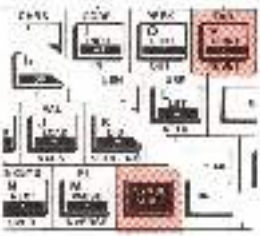


Hold down the SYMBOL SHIFT key and press the P key, and you'll get quotation marks.

Then type in the name of the program, making sure you have it exactly right.

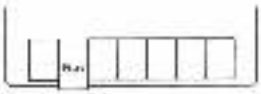
A program name can have up to ten characters including spaces. *If there are spaces in the name, you must include them.*

The computer makes a distinction between capital letters and lower-case letters. You must have the name in all capital letters if that's how the index shows it, or in lower case letters, or capitals and lower case if it is listed that way.

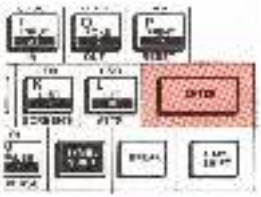


Then type SYMBOL SHIFT P again for quotation marks.

Your screen will look like this.

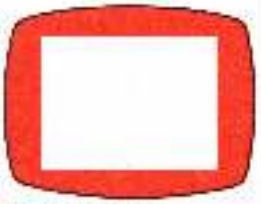


Now press the PLAY button on your cassette recorder, and then press



ENTER

on the computer keyboard. (LOAD is a command that tells the computer what you want it to do. "STAR ZAP" tells it what to do it with, and ENTER is the signal that the instructions are finished, and the TS 2000 should start the job.)



Searching Pattern

The border of the TV screen will alternate between pale blue (cyan) and red, during the time the computer is searching for the program on the tape.

Program Found

When the program has been found, the screen border will show a pattern of lines in the same shades of blue and red, and shortly the name of the program will appear on the screen.

Loading Picture

Next, as the program itself is being loaded into the computer, the border pattern will become thinner, faster moving yellow and dark blue lines.

When the computer has finished loading the program, one of two things will happen:

1. Most commercial programs will begin running automatically, usually with a "title screen" or instructions to the user. (A title screen may indicate that the tape is still loading; if not, you should stop the tape immediately at this point so as to be in the proper position to load the next program if you wish.)

or

2. The screen will be blank, except for a **0 OK, 0:1** in the lower left hand corner. This is a report code and means that the computer has successfully loaded the program. (There is a list of report codes at the very back of this book, they are the TS 2000's way of telling you that it has finished a job, or that it has encountered some problem.)

Report Code

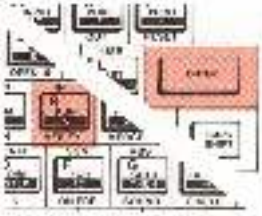
Again, stop the tape immediately. To execute the program, then, you press

RUN

and

ENTER

This will start the program.



Press RUN and ENTER

If you are using a cassette with more than one program on it, and wish to load a program that is not the first one, you will see the searching and loading patterns more than once. Each program that goes by will cause a "loading" pattern on the screen—and the name of the program will be written on the screen—even though the computer is not loading it. The TS 2000 will only start actually loading when it comes to the program you have named.

If you want to find out what programs are on a tape, you can type

LOAD "GEORGE"

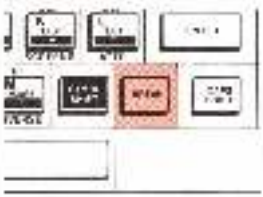
if you know there is no program named GEORGE (or any other name you make up) on the tape. Then, as it searches for GEORGE, the TS 2000 will print on the screen a list of the programs which are actually on the tape.

You can also make up your own index to a tape by setting the tape counter to 0000 before starting this process, and noting the number showing when each program name is entered on the screen.

Occasionally a program will fail to load properly, and you'll have to investigate the reason and/or try again. You'll know a LOAD has misfired if

1. The K cursor comes back on the screen.
2. The "searching" pattern comes back on the screen after the "loading" pattern (if you are certain that loading pattern was for the program you wish to load, and not just another program going by).
3. You see the report code

   R, Tape loading error



The "tape loading error" report means that the TS 2000 found the program (was able to read its title), but couldn't load it because of errors within the program (for instance, interference might have added or deleted just one bit of information, thus throwing the entire "reading" off).

If the **K** is on the screen, you don't have to do anything except rewind the tape in order to try again.

If the searching and loading process are still going on the screen, you need to press the **BREAK** key to stop the process. Then you'll be ready to check into the problem.

The most likely problem is that the volume level is too high or too low.

The best adjustment is to turn the volume up as loud as it will go without causing the silent spaces on the tape to be noisy. You can check this by disconnecting the plug in the recorder's earphone socket and listening to the tape on the speaker. If the silence is very noisy, you may have other problems.

Some tape recorders can record a 60 cycle AC hum. This can be avoided by operating them on batteries.

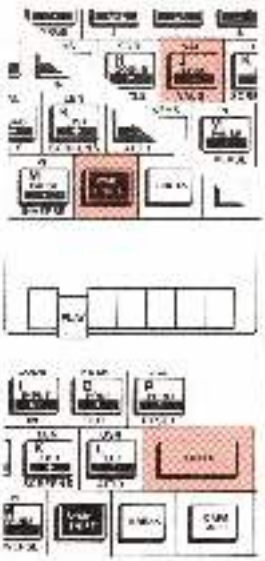
Some tape recorders — especially old, worn ones — are intrinsically noisy and produce a lot of extraneous noise on their tapes. You may have to invest in another recorder.

You may have to wiggle the plug in the earphone socket on some recorders contact is lost if the plug is pushed in too far. If you pull it out just a bit, you may find it settling in to a more secure position.

It is possible that you have a tape that was recorded (SAVEd) on another recorder, and the recording heads on that machine are out of alignment. This is likely if the program can be loaded from that recorder, but not from yours. If you have trouble with a lot of tapes, including commercial ones, your recorder's heads may need adjusting. It is even possible that both recorders are slightly out of adjustment—not enough to keep them from saving and loading their own or commercial tapes, but enough that they cannot use each other's tapes.

It is possible to load a program without using its name. If you type

LOAD ""

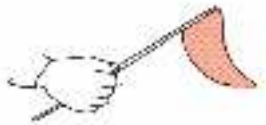


(that is, press the J key and then SYMBOL SHIFT P twice—do not put a space between the two quotation marks), then start your recorder on PLAY and press ENTER, the ZX Spectrum will load the first program it comes to. This is useful if you have only one program on a tape, if you know you want the first one, or if you know you want the next one but have forgotten its name.

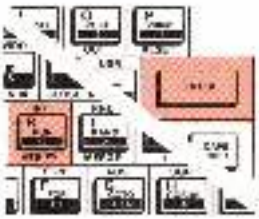
Typing In a Printed Program and Saving It on Tape

Many shorter programs are available in books and magazines. You can use them by simply typing them in. Type them exactly as they appear in the publication, making sure your spelling is corrected, and all punctuation and spaces as well.

You can check your listing by comparing what you have on the screen with the printed version. See Chapter 2 for how to easily make corrections.

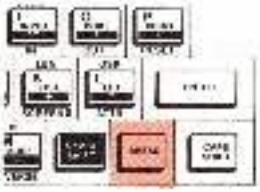


Beware: It is possible for the original listing in the book or magazine to have errors. You may type it correctly and still have trouble.

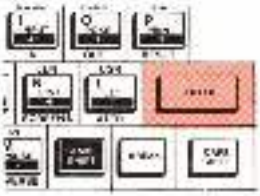


When you've finished typing the program in, execute it by pressing

RUN and ENTER



as above. When you've finished using it—either you reach the end, or you interrupt the program by pressing the key marked BREAK together with the CAPS SHIFT key—you can get the listing back on the screen by pressing



ENTER

again. Then, after verifying (by using it) that the program works and that you've typed it in correctly, you can save it for future use on tape. (You can save a program even if it doesn't work, in order to come back to it and fix it, or "debug" it.)

Saving a Program on Tape

As we said earlier, every program should have a name. The TS 2000, in fact, won't save a program on tape without a name. You can make up a name for a program you invent, use the name of a program you have typed in as above, or even change that name to something you like better. Whatever you call the program when you save it will be the name you have to ask for to load it later.

Remember, the program name can be up to ten characters long. It can be more than one word, but any spaces count toward the ten-character limit.

Note: It is a good idea to put the name of a program into the listing of that program, so you can double-check that you have the right one. The easiest way is to use a REM line at the beginning.

You'll notice that most programs are numbered in multiples of 10, so if a program doesn't already have a line giving its name in the listing, you can just type

5 REM PROGRAM—SAMZAP ENTER

using a line number lower than the lowest in the listing (the computer will then automatically put it at the beginning) and, of course, the actual name of your program.

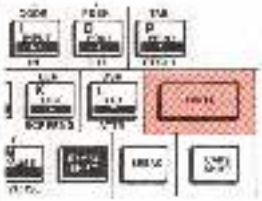
A program line that begins with REM (for REMark or REMinder) is disregarded by the computer when executing the program. It appears in the listing as an aid for the user.

Connect the MIC socket of the computer to the microphone socket of the recorder. Position the tape in a part that is blank, or a part that you are prepared to overwrite. Type:

SAVE "STARZAP"

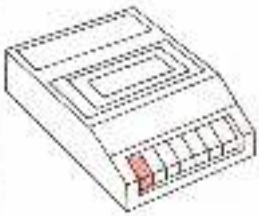
using the keyword SAVE on the S key. Then press

ENTER



The computer will then print on the screen:

Start tape then press any key

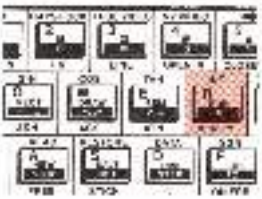


Start the cassette recorder, in RECORD position, and touch any key on the TS 2000's keyboard

RECORD button

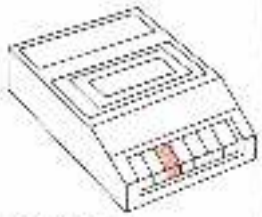
Saving Starzap

Watch the TV screen. You'll see a pattern of lines similar to the "loading" patterns—and eventually the screen will show **0 OK, 0:1** which in this case means "the SAVE is complete."

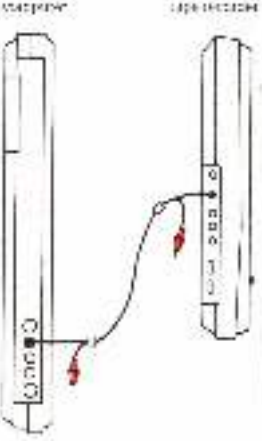


VERIFYing a SAVE

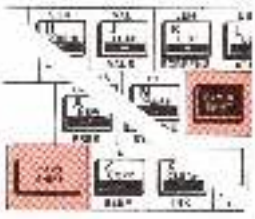
As a check on whether the recorder has received the program correctly, you can use the VERIFY function, located under the R key.



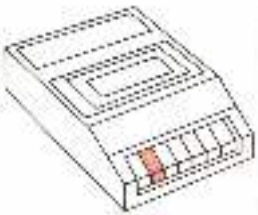
First, rewind the recorder to the place where you began the SAVE.



Second, be sure the EAR sockets — on the computer and on the recorder — are connected.



Third, type VERIFY "DATA ZAP" (to get VERIFY press CAPS SHIFT and SYMBOL SHIFT simultaneously, producing the E cursor, then press SYMBOL SHIFT R).



PLAY button

Fourth, start the cassette player—in PLAY mode—and press

ENTER

The computer then compares the program on the tape with the program still in its memory. If it finds the title, it will print on the screen

Program: STAR ZAP

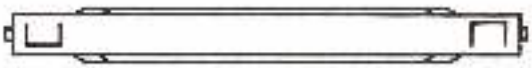
and then go on to display the same kind of border patterns as a LOAD. If the program is verified, the report code 0 OK will appear at the lower left corner of the screen.

If there is a mistake on the tape, the report will be R Tape loading error, you should try the SAVE again.

If the program name does not appear on the screen, the SAVE did not take place at all. You need to check:

1. That the plugs were correctly connected.
2. That the volume setting on the recorder was high enough.
3. That you were not attempting to record on the "leader" at the beginning or end of the tape.

4. That the RECORD tabs are in place on the cassette.

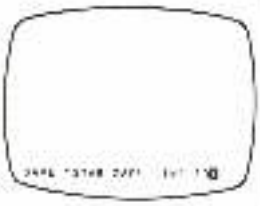


Finally, if the program name appeared, but the computer did not stop with 0 OK after the LOAD pattern and instead continued to search, it is likely that you have made a spelling error in the program name either in the SAVE or the VERIFY command. (If they do not match exactly, the computer will not recognize them as the same name.)

Saving for Automatic Start

You can save your own programs in such a way that they "self start." All you need to do is add the command LINE and the line number where you want the program to start (usually, but not always, the first line of the program). For example,

SAVE "STARZAP" LINE 10



LINE is the function located under the 3 key; you need to press CAPS SHIFT and SYMBOL SHIFT to obtain the E cursor, then press SYMBOL SHIFT 3.

When you LOAD any program you save this way it automatically starts running at the line number you've entered.

Saving Programs with Your Own Data Entered

Some programs are meant for you to enter your own data into—saving lists, figures, etc. These are easily saved by following the same procedures we've just discussed.

1. LOAD the program as we've described.
2. RUN the program, entering your own data as it is called for.

3. SAVE the program with the data in it, using a new name to distinguish it from the original program. If, for example, you load a program called "Calculator" and then fill in your personal financial records, you may want to save the filled-in version under the name "finances."

4. VERIFY the saved program.

Loading Programs with MERGE

The MERGE command can be used instead of LOAD if you wish to combine two programs. Where LOAD clears all previous program data out of the computer before loading in the new one, MERGE leaves the old one in while loading the new.

However, if any of the same line numbers appear in both programs, the new lines will take precedence over—and erase—the old ones. This means that to MERGE programs requires careful planning. (If you save a program using LINE, and load it with MERGE rather than LOAD, it might not jump to the appropriate line number and start automatically.)

Searching Pattern

Searching Pattern

Finding Pattern

Loading Pattern

Saving Pattern

As you can see, there are many ways to use your Timex Sinclair 2000 without learning computer programming. But if you'd like to look into it, try the next chapters and see how you like it.

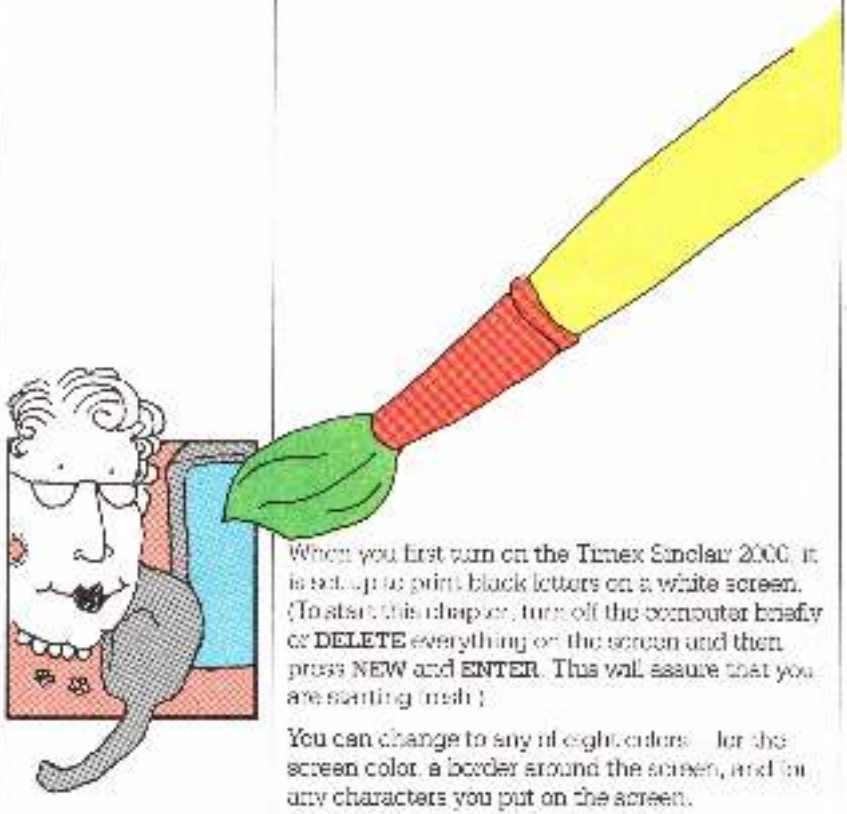
Summary

1. Many programs are available for use with the Timex Sinclair 2000 computer, and you don't have to know how to program to use them.
2. The LOAD command followed by the name of a program in quotes causes the computer to load that program from a tape cassette.
3. LOAD followed by two quotation marks with nothing between them causes the TS 2000 to load the next program on the cassette tape.
4. The SAVE command followed by the name of a program in quotes tells the computer to send that program to a cassette recorder running in recording mode, saving the program on the tape.
5. You must have a name for the program when using SAVE, but you can make up any name you like—up to ten characters long (including spaces)—and change the names of alternative versions of the same program (like a program into which you put information that is updated periodically).
6. If you add LINE and a line number to a SAVE command, the program will automatically start at that line number when you LOAD it.
7. The VERIFY command checks the program saved on tape against the program in the computer's memory, so you are sure the program is safely on tape before you clear it from the TS 2000.
8. When you load a program with MERGE instead of LOAD, a program already in the computer is not erased. Line numbers that appear in both programs, however, are eliminated from the old program, so you must be careful that programs are designed to be MERGED.

Using Colors 5

Chapter Preview

Learn how to use the INK, PAPER, and BORDER commands to do some colorful computing.



When you first turn on the Timex Sinclair 2000, it is set up to print black letters on a white screen. (To start this chapter, turn off the computer briefly or DELETE everything on the screen and then press NEW and ENTER. This will assure that you are starting fresh.)

You can change to any of eight colors for the screen color, a border around the screen, and for any characters you put on the screen.

Press BORDER—the B key, with the E cursor (for the copyright notice if you've just turned on the

TS 2068) on the screen. Then press the 1 key. You will have this on the screen:

BORDER 1

(It's spelled BORDR on it on the key, but BORDER on the screen.)

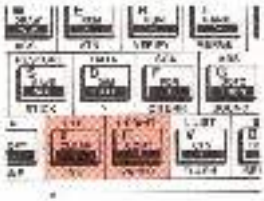
Now press the ENTER key. Isn't that a welcome change from black and white? You should have a dark blue on the screen—the same color as is written above the key you pressed.

By the way, it goes without saying that you'll only see the colors we are describing on a color TV set. But if you go through this chapter on a black and white set, you'll find that the colors from 0 to 7 are arranged in order from black through shades of gray to white.

Let's look at some other colors. Press

BORDER 2 ENTER

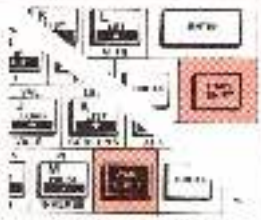
and so on up through BORDER 7. When you hit BORDER 7 (white), the border will again match the screen. Don't forget black, though: BORDER 0.



Now, suppose we want to change the color of the business area of the screen. Look below the X and C keys in the bottom row of the keyboard: the keywords INK and PAPER are there.

We call the screen area PAPER because you print on it, and the color you print with is INK. Let's try to change the PAPER color first.

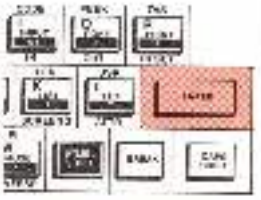
Remember how to get the keywords under the keys?



Press CAPS SHIFT while pressing SYMBOL SHIFT

Press both CAPS SHIFT and SYMBOL SHIFT at the same time, producing the E cursor on the screen. Then hold SYMBOL SHIFT and press the O key: you'll get PAPER. Press 1, so the screen shows:

PAPER 1



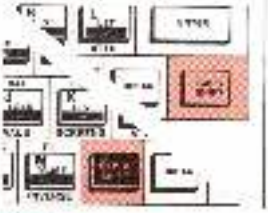
then press ENTER. Hmmm. Press ENTER again.

You have to press ENTER twice to see your choice of paper color. We'll come back to this in Chapter Twenty, and explain why.

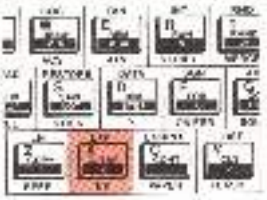
Go through the colors as you did for BORDER. When you get to 0, you'll have an all-black screen.

Try out whatever combinations you like: you can select a BORDER color and then a PAPER color, or change one but not the other.

Now, let's get back to a white screen, with PAPER 7 (it doesn't matter what color you want to leave the border).



Press both SHIFTs to get the E cursor, then hold SYMBOL SHIFT and press the X key:



INK

try the 1 key, so the screen shows:

INK 1

then press ENTER. Not much happening, eh? Well, we'll have to come back to INK, in the next chapter. You have to put something on the screen to make the INK color show.

By the way, if you select the same color for PAPER and INK, you won't be able to see anything!

Before going on to the next chapter, you may wish to return the screen to its original black-on-white. You can again briefly turn off the computer, or you can press:

BORDER 7	ENTER
PAPER 7	ENTER (twice)
INK 0	ENTER .

Summary

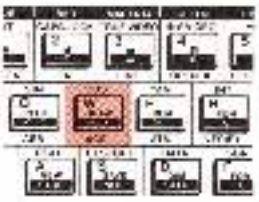
1. BORDER allows you to set the color of the border around the screen, using the colors above the top row of keys on the keyboard.
2. PAPER allows you to specify, using the same keys, the color of the area of the screen on which things will be printed.
3. INK specifies the color of the symbols you will place on the screen.

Drawing Lines and Circles

6

Chapter Preview

This chapter shows you how to do easy artwork anywhere on the screen with PLOT, DRAW, and CIRCLE.

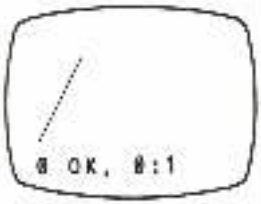


Another of the outstanding features of the Timex Sinclair 2000 is its high resolution graphics capability. Later in the manual, we will address a number of topics in graphics, but just for fun, let's go through two of the simplest and most powerful commands, DRAW and CIRCLE.

First, to get oriented, you need to know that the TV screen is 256 positions across and 176 positions high—there are 45,056 positions through which you can draw graphics. Each of these positions is called a *pixel* for *picture element*. (See the chart in Chapter 17.)

Let's illustrate by starting with the DRAW command. With either the copyright notice or a K cursor on the screen (we consider the copyright notice to "hide" a K cursor), press the W key.

You'll see the keyword DRAW appear on the screen. Type 50, then a comma (SYMBOL SHIFT N), and then 100. The screen should have this on it:




```
DRAW 50, 100
```

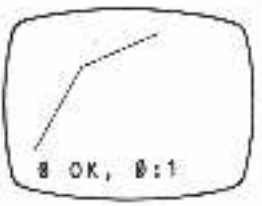
Press ENTER. That line was drawn from the last *plot position* (which when we start out is in the lower left corner of the screen), to a point 50 positions to the right and 100 positions up.

Leave this on the screen and let's try something else. Remember from the last chapter how to get the INK command? Press

```
INK 2      ENTER
```

Now press

```
DRAW 120,50      ENTER
```

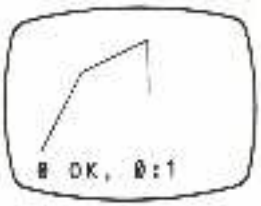


Aha! This time, the line went from the end of the previous line to a point 100 positions over and 50 positions up—and it was drawn in INK color 2, red.

Again, leave what you've drawn on the screen. In fact, during this chapter **don't clear the screen** or type NEW unless we ask you to. (Some of the commands we use as examples will run off the screen if they don't begin at the proper location on the screen.)

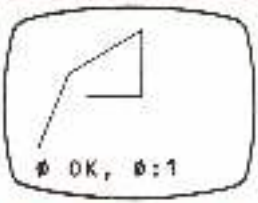
Let's try another:

```
INK 1      ENTER
DRAW 0, -75      ENTER
```



The line is blue, the 0 meant it stayed in the same position horizontally on the screen, and the minus 75 drew the line 75 positions down.

Chapter 6: Drawing Lines and Circles



How would you draw a line leftwards . . . horizontally . . . in green?

INK 4 ENTER
DRAW −75,0 ENTER

Let's see if we can change the color of the screen. First, press

BORDER 2 ENTER

Okay, a red border. Now

PAPER 5 ENTER ENTER

Oops! We have a yellow screen, all right, but we've erased all our drawings. We will find out more about this, as we've said, in Chapter Eight. For now, let's remember that *you can't change the screen color without clearing the screen* of anything on it.

Now let's try

DRAW 120,100 ENTER

The line is green, because the INK color is still the last color we chose. Press ENTER again. Another way to erase! There are some things to beware of.

Let's change the ink color again:

INK 0 ENTER

Now let's draw a circle. Press CIRCLE, the key word under the H key (SYMBOL SHIFT with the E cursor on the screen). Then you need three numbers—the first two locate the center of the circle across the screen from the left and up from

the bottom, and the third for the radius of the circle. Position 125 is about halfway across the screen, position 90 is about halfway up, and 60 seems like a sensible size for a circle. So

CIRCLE 125,90,60 ENTER

How about a smaller different colored circle around the same center?

INK 2 ENTER
CIRCLE 125,90,27 ENTER

And maybe a great big one:

CIRCLE 125,90,100 ENTER

If you try to draw a circle that will go off the screen, you get this report:

B Integer out of range

This would be a good time to clear everything from the computer, using:

NEW ENTER

and practice any kind of drawing you'd like, with DRAW, CIRCLE, INK, PAPER, and BORDER.

Two more things you may wish to try:

1. You can choose the position to start to DRAW a line. Use the PLOT command.

   PLOT 50,100 places a small dot on the screen fifty positions over and 100 positions up from the original lower-left corner position, or from the end of any previous DRAW line or the right-

most point on any previous circle. The next DRAW command would start from there.

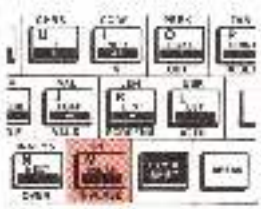
PLOT 127,87 ENTER

puts you in the center of the screen. Start from there to do the next exercise.

2. You can draw an arc by adding a third number to the DRAW command. Once you have your starting point, the first two numbers after DRAW still select the ending point, but a third number selects a portion of a circle (by describing the angle covered by that arc, in *radians*).

DRAW 57,7,PI ENTER

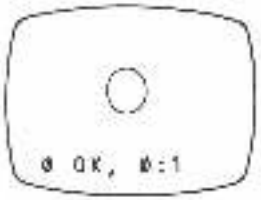
(PI is the keyword over the M key, obtained with the F cursor) draws a half circle to a spot 57 positions to the right on the screen.



Why is it the bottom half of a circle instead of the top half? Because all circles are drawn counterclockwise on the TIS 2000.

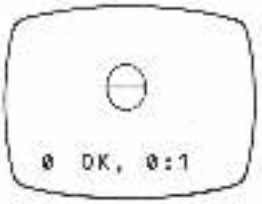
What would happen if you then entered

DRAW -50,0,PI ENTER

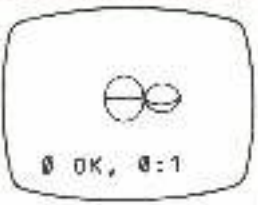


Try it and see. Then try

DRAW 0,0 ENTER

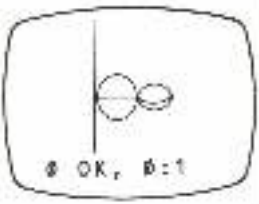


A complete circle would be drawn by having the third number be 2·PI. A quarter circle would be .5·PI. Since the starting and ending positions of the arc are specified, a smaller portion of a circle



would produce an arc from a larger circle. Try a few to prove it to yourself:

```
DRAW 50,0,.8*PI      ENTER
DRAW -50,0,.5*PI     ENTER
DRAW 50,0,.3*PI      ENTER
```



What do you get with

```
INK 1      ENTER
PLOT 125,0      ENTER
DRAW 0,176      ENTER
```

Try it and find out. Keep trying things as you go through this manual. We will, as we get in to the BASIC programming material, spend little time considering colors. You should experiment with adding INK, PAPER and BORDER commands to the programs we cover.

(What you should get is a blue line in the center of the screen.)

Summary

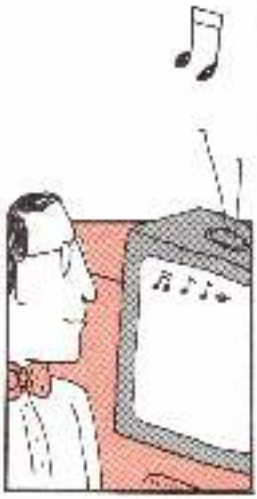
1. PLOT places a dot on the screen at a location you specify with two numbers: the first choosing a position across the screen from the left, and the second placing the dot up from the bottom. The plot position (0,0 when the computer is turned on) is moved to the location of the dot.
2. DRAW draws a line from the current plot position to a location specified by two numbers: across from, and up from, the plot position. Adding a third number allows you to draw an arc. (The plot position moves to the new location.)
3. CIRCLE draws a circle at a location specified by two numbers (as in PLOT), with a radius specified by a third number.

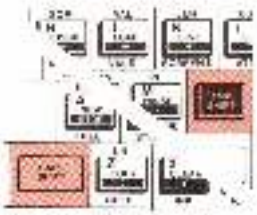
Sound

7

Chapter Preview

In this chapter, we learn to compose and play music with the BEEP command.

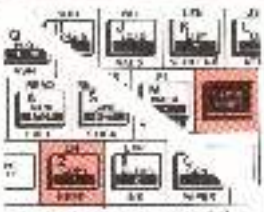




Press CAPS SHIFT while pressing SYMBOL SHIFT

Your Timex Sinclair 2000 has both simple and complex ways of making sounds. With the SOUND command, located under the G key, it can play music through three different channels—can harmonize with itself! We won't get into that until late in this book, as it is quite complicated. But for now, let's use a simpler command: BEEP.

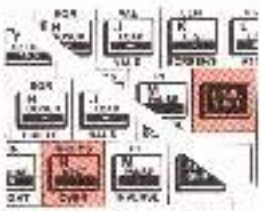
Press both the CAPS SHIFT and the SYMBOL SHIFT keys to obtain the **E** cursor.



Press SYMBOL SHIFT while pressing Z

Then press SYMBOL SHIFT and the Z key and you'll get the keyword under that key:

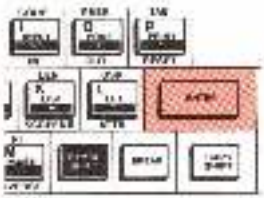
```
BEEP
```



Press SYMBOL SHIFT while pressing N

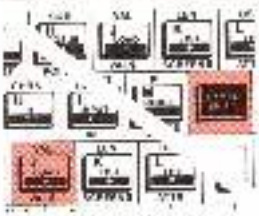
Then press 1. Holding SYMBOL SHIFT press the N key for a comma. And press 0. The screen should look like this:

```
BEEP 1,0
```



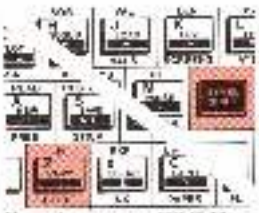
BEEP needs two numbers after it, separated by a comma. The first number is the duration of the note in seconds, and the second is the pitch. 0 is the middle C. Press ENTER and get middle C for one second.

Brief Music Lesson: Our eight-note scale is constructed of twelve half tones (must me) with two halftone steps between each note except for one step between 4 and 5 (FA and SOL) and between 7 and 8 (or 1 and DO).



For notes below middle C, press SYMBOL SHIFT while pressing J

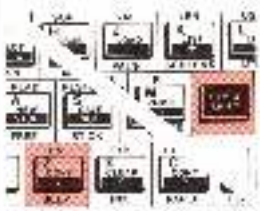
If you want notes above middle C, sound a half-tone for each number. Notes below middle C are counted as negative numbers (using the minus sign, SYMBOL SHIFT J).



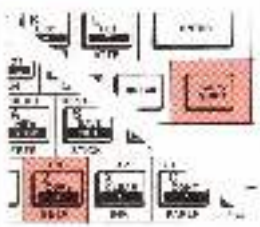
For colons, press SYMBOL SHIFT while pressing Z

You can use colons (SYMBOL SHIFT Z) to string a number of commands together. So you can make a scale (DO, RE, MI and so on) by typing:

```
BEEP 1,0:BEEP 1,2:BEEP 1,4:BEEP 1,5:BEEP
1,7:BEEP 1,9:BEEP 1,11:BEEP 1,12
```



(You can speed this process up if you realize that, after the second number in each pair, you can hold SYMBOL SHIFT and rapidly press Z, CAPS SHIFT and Z again to get the colon and the next BEEP.)



Then you can "play" the TS 2068 by pressing ENTER.

If you like, you can try out various tunes at various durations; you can use decimal points to play fractions of seconds and fractions of tones (if you like Indian or Oriental music). Try

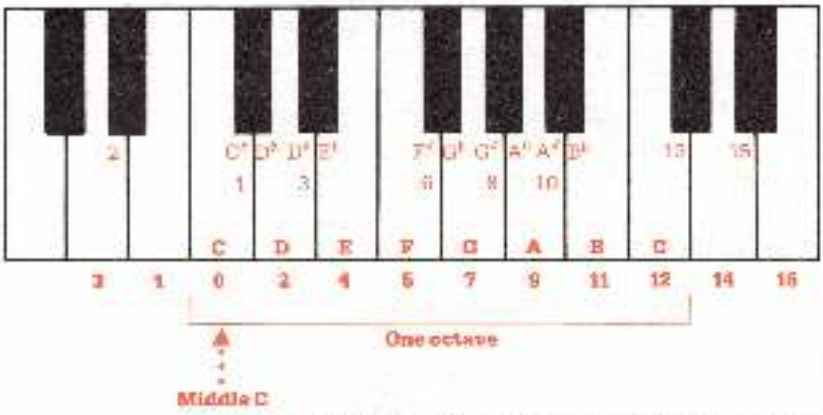
```
BEEP 0.9,77
BEEP 1.09,14
```

You'll find that 10 seconds is the longest note you can play with BEEP, 69 is the highest and -60 the lowest. If you enter any other figures, you'll see the report code

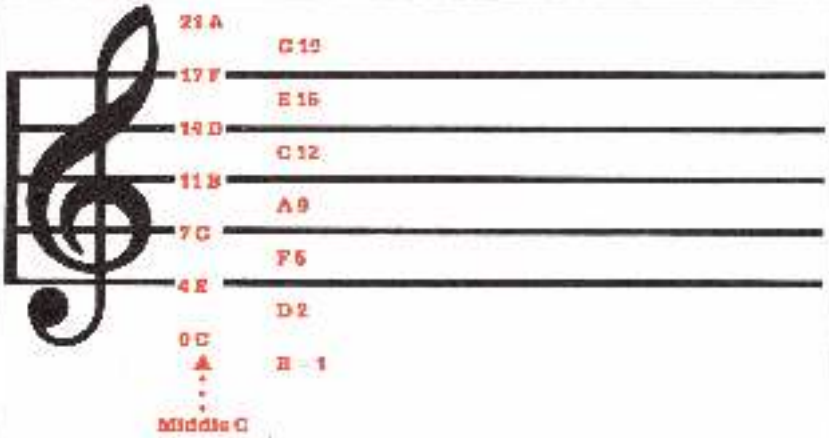
```
B Integer out of range, 0:1
```

The tones played by BEEP can be heard not only through the TS 2000's internal speaker, but also through the MIC socket. You can connect an amplifier to MIC and play the sound through large external speakers.

Here are two diagrams relating the pitch numbers to the piano keyboard and the treble clef on a music staff. Let there be music!



Add 12 to each value to repeat one octave higher. Subtract 12 to repeat one octave lower.



Add 1 for a sharp (#) on a line or space in the key signature or in front of a note.
Subtract 1 for a flat (b).

Summary

1. BEEP followed by two numbers separated by a comma sounds a musical tone. The first number is the duration of the note in seconds (decimals are allowed). The second identifies the pitch. 0 is middle C, positive numbers select half tones shown C, negative numbers go below C (again, decimals are allowed).

Writing a Program 8

Chapter Preview

How to start with NEW, repeat with GOTO, stop with BREAK, and continue with CONT.

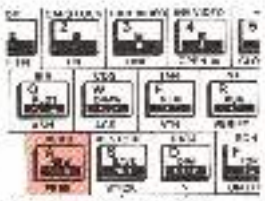
We are now ready to write our first computer program.

In the last chapter, we were operating in what is called the *immediate mode*, which means that the computer executed each command immediately (after you pressed ENTER).

When we write programs, we give a number of commands and the computer executes them, in order, when we tell it to. Until we tell it to execute and afterwards, for that matter, the TS 2000 remembers all the commands.

Although we will start small, there are programs of many thousands of lines, which direct computers to carry out lengthy and complicated procedures. The power of computing is in the ability of the machine to receive, store and carry out many different complex programs.

Let's get started. If you are just plugging in the computer to start this chapter, you should have the K cursor in the corner of the screen—or the copyright notice, which "hides" a K cursor.



Press NEW to clear the screen and the computer's memory.

If you are continuing from the last chapter, press

NEW

(the A key, while the K cursor or any report such as "hides" a K cursor—is showing), and

ENTER

This clears both the screen and the computer's memory, so it is ready to start a NEW program.

Type in:

PRINT "HELLO, SUE"

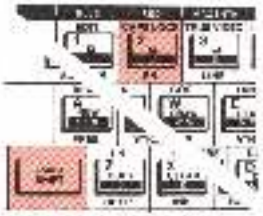
and press ENTER. Just as in the last chapter, the computer executes the command immediately. (By the way, you can use your own name, if it isn't Sue. . .)

Now type in

10 PRINT "HELLO SUE"

Notice that the K cursor doesn't change to L as you type the 1, and then the 0. This is why there are no keywords over the number keys: so you can put in program line numbers. It changes to L after the keyword PRINT.

If you are a typist, be careful to use the numeral 1 and not the lower case L for a 1. Each character, like each keyword, can only mean one thing to the ZX SPECTRUM.



Press CAPS SHIFT while pressing 2 for CAPS LOCK

In fact, let's write all our programs in capital letters. Press CAPS SHIFT 2 to obtain CAPS LOCK and leave it there when writing programs. You'll still have access to the numbers and will be able to use CAPS SHIFT and SYMBOL SHIFT to obtain other symbols and punctuation. As we said in Chapter Two, in CAPS LOCK mode, all your letters will be capitals.

Also, be careful to use the numeral 0 (zero) and not the letter O. We will show zeros with the slash mark through them to distinguish them from letter O's (this is common in computing).

Now press ENTER. Notice that the entire program line has appeared at the top of the screen . . . instead of just the words HELLO, SUE.

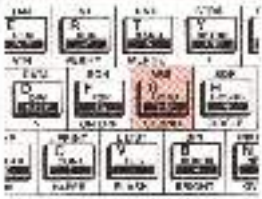
You will also notice a symbol between the 10 and the PRINT. This is the program cursor. It is placed at the line most recently entered into the program.

The only difference between the first line we typed and the second was the *line number*. When we put a number in front of a command, it becomes a *program line* and is not immediately executed.

The cursor is only often at the bottom of the screen.

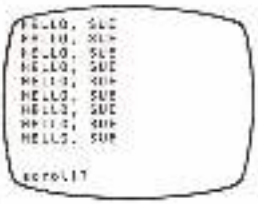
Type this:

20 GOTO 10 ENTER



Note that GOTO is a keyword (on the G key) and should not be spelled out (in fact, it appeared as soon as you pressed the G). Notice, also, that the program cursor now shows at Line 20.

Now you have a complete, if brief, program. The command in line 20 simply tells the computer to go back to line 10 and start over. Can you guess what will happen when you execute the program?



Let's try it and see. To execute a program, you simply press

RUN
(the keyword on the R key), and
ENTER

How about that? That's a lot of stuff on the screen for such a short program. As we said, the computer is not very smart. But it is fast, accurate, and tireless when doing repetitive work.

One of the most powerful commands in BASIC is GOTO, which directs the computer to a particular program line, rather than the next line in numerical order. GOTO is sometimes used to tell the computer to go back to an earlier line and repeat a process over and over again.

The question "scroll?" at the bottom of the screen informs you that the screen is full and asks you if you want to continue printing. If you press either N (for "no") or BREAK, the program will stop with the report code

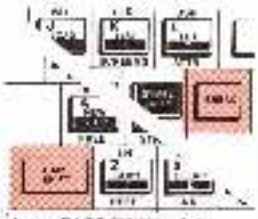
D BREAK—CONT repeats, 10:1

If you then press CONT (for CONTinue)—the C key — you'll see the bottom entry flicker for a while and then it will stop with the "scroll?" question again.

What is actually happening is that 22 more lines of "HELLO SUE" are being printed, and the screen is being scrolled upwards.

This is also what happens if you press any key other than N or BREAK in answer to "scroll?"

(Incidentally, this is much more useful in a program where the information is changing—such as a counting program where each 22 lines are of higher totals—than it is where the information is simply being repeated.)



Press CAPS SHIFT while pressing BREAK

Here's something else. Stop the program with BREAK, then restart it with RUN and ENTER. While the computer is running "HELLO SUE" down the side of the screen, press the BREAK key (CAPS SHIFT must be pressed simultaneously) and notice that the list stops at whichever point it was when you pressed BREAK.

You have to be fast. Try pressing RUN, then holding CAPS SHIFT while you touch ENTER and then, quickly, BREAK.

The computer checks after doing every program line to see if anyone has pressed BREAK; if so, it stops the program.

(You can also use BREAK to stop a runaway program—like an "endless loop," about which we'll say more later—or a misfired LOAD from a tape cassette. If BREAK doesn't work, you may have to resort to turning the power switch off and then on again, which, of course, means you lose any information in the computer.)

Now that when you want BREAK you have to press CAPS SHIFT, too.

CONTinue lets you continue when the screen is full, when you interrupt the program with BREAK or STOP, or when the program itself interrupts with the STOP command. (STOP—SYMBOL SHIFT A—is used rather than BREAK within a program, or when the program is waiting for input—yes, we'll be talking about STOP later, too—and BREAK is used while a program is in full cry.)

Okay. After you've had your fill of fooling around with BREAK and CONT, let the program stop with a full screen, press BREAK and press ENTER again.

Your program is back on the screen again. You can RUN it again (go ahead, press RUN and ENTER).

You can also leave it in the memory and go on to something else. It will stay there until you erase it or unplug the machine.

But there is one caution I want to look into it. Get the program listing back on the screen by pressing BREAK, then ENTER when it stops.

Now, type in

10 PRINT "GOODBYE, SUE!"

and press ENTER.

This new line 10 replaces the old line 10 in the program. (And the program cursor shows that line 10 was the last one you entered, even though it is not the last one in numerical order in the listing on the screen.)

There can only be one line 10 and any time you enter a new one, you lose the old one. This means that, if you leave a program in the TS 2000 (instead of restarting with NEW), you run the risk of having the new program erase the old one. Or, worse, of having some lines from the old program interwoven with lines from the new one.

Try this:

5 REM PROGRAM—GOODBYE

(and press ENTER)

Program line 5 has been inserted into the program where it belongs in numerical order. This is why we usually number the lines in multiples of 10: it gives us room to insert new lines if we find we need them.

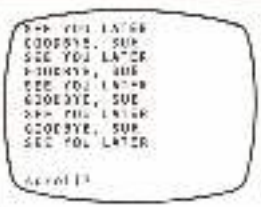
Now press RUN and ENTER

The program line (5) beginning with the keyword REM doesn't "do" anything in the program. Any line beginning with REM — for REMark or REMinder—appears in the listing to help the user understand the program—but is disregarded by the computer when running the program. Remember that capital and lower case letters are different to the CPS 2000, it is wise to use the same kind of letters in the REM statement as you use in the actual program name for LOADing and SAVEing, as a REMinder. . . .

As another example, try typing

15 PRINT "SEE YOU LATER"

(and ENTER)



and RUN the program. Then press BREAK to stop it and ENTER again to show the listing. Can you see why the program does what it does?

Now you might want to try to write a few simple programs of your own, using lines of strings to print or of mathematical calculations. Do not worry about making mistakes; if you make mistakes, the computer will do exactly what you tell it to do, even if the result doesn't make any sense!

Summary

1. NEW erases everything that might have been typed into the computer, to make room for a new program.
2. When you put line numbers in front of commands, they become program statements and are not executed immediately. Instead, they are carried out in numerical order in response to the RUN command.
3. GOTO is a very powerful BASIC command. It directs the computer to a line in the program other than the next one in numerical order, and allows the computer to repeat sequences of program lines over and over.
4. BREAK stops a program while it is executing.
5. CONTinue restarts a program that has stopped for certain reasons, most often because BREAK has been pressed or the screen is full.
6. "Scroll?" is a question the computer asks when the screen is full; you decide whether to stop or to continue the execution of the program.

Arranging Output on the Screen

9

Chapter Preview

This chapter shows you how to use EDIT, AT, TAB, commas, and semicolons to move things around. We use the up and down arrows.

You can do wonders with punctuation marks in a TIS 2000 BASIC program. For instance, let's play with the program we wrote in the last chapter.

```
 5 REM PROGRAM—GOODBYE
10 PRINT "GOODBYE, SUE
15 PRINT "SEE YOU LATER
20 GOTO 10
```

The Program Cursor and the EDIT Command

We'll add some punctuation. If you retyped that program for this lesson, you'll see the *program cursor*, which looks like this—>—at line 20, which was the last line you typed in. Press the up arrow (CAPS SHIFT 7) and it will move to line number 15. Press the up arrow again and the cursor moves to line 10.

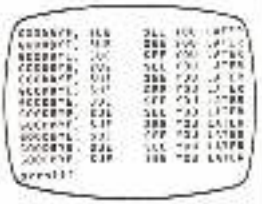
(If you are starting this chapter right after finishing the last one, the program cursor is at Line 15. Press the up arrow just once to move it to Line 10.)

Then press EDIT (CAPS SHIFT 1) and line 10 will appear in the workspace at the bottom of the screen.

The Comma

Now move the cursor to the end of the line by repeatedly pressing the right arrow (CAPS SHIFT 8). Then type in a comma (SYMBOL SHIFT N) and ENTER.

The program doesn't look much different. But press RUN and ENTER.



The TS 2068 screen is 32 characters wide (they are numbered 0-31 instead of 1-32, by the way), and the comma moves the *print position* to the beginning of the next half screen.

If you then add a comma to the end of line 15 in the same way, the output on the screen will not change. A PRINT statement with no punctuation moves the PRINT position to the beginning of the next line, and a comma after a PRINT statement that *finishes in the second half of the screen* also moves the PRINT position to the beginning of the next line.

A comma at the end of the print statement moves the print position to either column #16 (the right half of the screen) or position #0 (the beginning of the next line), whichever is next.

Reminder: A comma inside quotation marks is printed as a comma. A comma outside quotation marks moves the print position to the beginning of the next half screen.

The Semicolon

Using the same technique as before (arrows, cursors, EDIT key) and the DELETE key, replace the comma at the end of line 10 with a semicolon.

Move the program cursor to Line 10 with the up and/or down arrows (CAPS SHIFT 6, CAPS SHIFT 7).

Bring that line down into the work area with EDIT (CAPS SHIFT 1).

Move the cursor to the end of the line with the right arrow (CAPS SHIFT 8).

Erase the comma with DELETE (CAPS SHIFT 0).

Add a semicolon (SYMBOL SHIFT O — the letter O, not a zero).

Press ENTER.

Now RUN the program.

Hmmm. What have we here?

A semicolon moves the print position to the next space after the end of the previous statement.

We have to figure out a way to leave a space between the two phrases.

The way you do it is to add a space inside the quotation marks. Same way as before.

Press ENTER to get the program listing.

Move the program cursor to line 10.

Press EDIT to bring it down.

Move the cursor with the right arrow to a position just before the last quotation marks.

Type a SPACE. It is inserted at the point of the cursor.

Press ENTER.

Press RUN and ENTER.

Note If, for any reason, you want to separate two numbers, you have to add spaces *in quotes* which means you have to add the quotes, too.

PRINT 1234	prints	1234
PRINT 12 34	prints	1234
PRINT 12" "34	prints	12 34

Don't forget the semicolons . . .

The Apostrophe

There is one more punctuation mark that doubles as a "control character" to move the PRINT position. The apostrophe (SYMBOL SHIFT 7) moves the PRINT position to the beginning of the next line. So, instead of typing

```
10 PRINT "Hello "
20 PRINT "there"
```

you could type

```
10 PRINT "Hello" '"there" and get the same effect.
```

There are other ways to position printing on the screen. Let's try them.

TAB and AT

First, let's clear the computer by pressing NEW and ENTER.

Then, type in this program:

```
 5 REM PROGRAM—CHAPNINE
10 PRINT INK 1; TAB 6; "CHAPTER
   NINE"
20 PRINT AT 5,3; "PUNCTUATION AND
   THE SCREEN"
30 PRINT
40 PRINT
50 PRINT
60 PRINT
70 PRINT INK 2; TAB 5; "INFORMATION"
80 GOTO 70
```

Press RUN and ENTER. Nice page, eh?

Reminder: If you press the N key (for "no") or the BREAK key in response to the question "scroll?" at the bottom of the screen, the screen will not scroll, and you can do something else.

If you press any other key, the program will scroll to print the next 22 lines. You must stop the scrolling with N or BREAK before you can do anything else.

Let's examine the program, line by line. Press ENTER to get it back on the screen.

Line 5 is just our standard REM statement, containing the name of the program (the same name we use to LOAD or SAVE it). This is not required, but is probably a good habit to get into.

Line 10 prints beginning at column 10 across the screen—just like the TAB key of a typewriter, except that you specify the column number in the program line. Remember to obtain TAB from the extended mode (press both SHIFT keys, then the P key), rather than spelling it out.

Notice the semicolon after TAB 12. TAB 10 puts the "print position" at the 10th column, but a comma would move it to column 16 and defeat the purpose of the TAB statement. And no punctuation at all would result in a syntax error marker. For the same reason, the INK command is followed by a semicolon.

PRINT, "JOE" (with two commas before "JOE") is the same as PRINT TAB 16; "JOE"—you can put the commas ahead of the PRINT item.

Line 20 prints AT a location defined by the two numbers—the first is the line number, the second is the column number (five lines down, three columns across). AT is also a keyword, SYMBOL SHIFT on the I key. Notice, again, the semicolon.

Since INK 1 in line 10 was in a print statement, it specified the ink color for only that statement. Line 20 returns INK to the normal, or "default" value of 0, black.

Lines 30-60 each print a blank line, effectively moving the print position down by four lines, before line 70.

Line 70 prints on the next line and at the specified TAB location, its information, in INK color 2, red.

Line 80 causes line 70 to repeat until the screen is full.

Could you change line 70 to read

PRINT AT 10,5; "INFORMATION"

Try it and RUN the program.

Hint: You'll have to use BREAK to stop the program. This is because lines 70 and 80 form an "endless loop" that is never stopped by a full screen, since line 70 keeps returning the PRINT position to line 10.

Does it make any difference if you add a comma at the end of line 70? Why or why not?

How about if you add a line 75 that reads like the original line 70, and change the GOTO in line 80, like this:

70 PRINT AT 10,5; "INFORMATION
75 PRINT INK 2; TAB 5; "INFORMATION
80 GOTO 75

Then RUN and ENTER.

How To Print Quotation Marks

You may be wondering, if quotation marks tell the computer where to start and stop a string, how can you print quotation marks themselves on the screen?

If you type *two* quotation marks together after you have started a string with a single quotation mark, the computer will print a quotation mark.

Let's eliminate line 75 by typing 75 ENTER, change line 80 back to

80 GOTO 70

(just type in the new line 80 and it replaces the old one), and then change line 70 to look like this:

70 PRINT TAB 5;" "" "INFORMATION" "" "

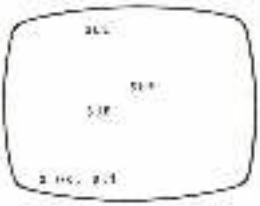
(after TAB 5 you type SYMBOL SHIFT P three times and, after typing INFORMATION, type three more SYMBOL SHIFT P). You will get a curious looking arrangement in your program listing but the right kind of quotation marks when the program prints it.

Here's something else you may want to try—you can have multiple PRINT items in one PRINT statement.

First, we need to remove the CHAPNINE program from the computer. Type NEW and ENTER (this is the last time we'll remind you to do this ...).

Then type in the following one-line program—very carefully!

10 PRINT TAB 5; "SUE"; AT 5,10; "SUE"; TAB 5; "SUE"



RUN the program.

It will print the screen shown at the right. TAB 5 prints in the first line at column 5. AT 5,10 prints at line 5, column 10, the four commas then move the print position first to line 6, column 15, then line 6, column 0, then line 8, column 15, then line 7, column 0, and then the TAB 5 moves it to column 5, still on line 7.

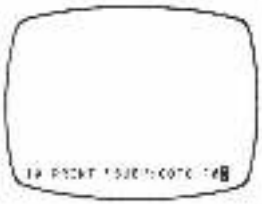
Try a few arrangements yourself.

Multiple Statement Lines: The Colon

You can also have more than one program statement on a line, separated by a colon. We did this with commands (without line numbers) earlier in the manual.

Generally, keeping each statement on a separate line makes it easier to understand and to edit (make changes in—a program.

But you can sometimes save memory space without sacrificing understanding if you combine closely related statements on a line. Press NEW and ENTER, then try this:



10 PRINT "SUE" : GOTO 10
ENTER
RUN and ENTER

Summary

1. After a PRINT command with no punctuation following, the print position for the next PRINT command moves to the beginning of the next line on the screen.

   PRINT "SUE"

2. A comma after a PRINT command moves the print position to the middle of the screen, or to the beginning of the next line, depending on whether the end of the item that has just been printed is in the left- or right-hand half of the screen. You can use more than one comma to move the print position as far as you like by half-lines.

   PRINT "SUE",,

3. The semicolon moves the print position just one character space to the right.

```
PRINT "SUE";
```

4. The quotation mark moves the print position to the beginning of the next line.

```
PRINT "SUE"
```

5. TAB sets the print position at the column called for by the number following TAB. Remember, column 0 is really the first column, column 9 is really the tenth, and so on.

```
PRINT TAB 10, "SUE"
```

6. AT sets the print position according to two numbers or round brackets, separated by a comma: the first is the line number (counting down from the top of the screen) and the second is the column number (counting across from the left edge).

```
PRINT AT 5,15; "SUE"
```

7. TAB and AT move the print position *before* printing what is on their program line, the comma and semicolon move the print position *after*, in preparation for the next line, so you could have both.

```
PRINT AT 5,15; "SUE",
```

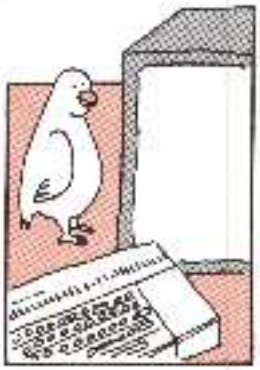
8. You can print quotation marks inside a string by typing two quotation marks in order to print one.

9. The colon allows you to put multiple statements on one line; after a colon the E cursor returns, allowing you to start a new command with a keyword.

Saving Time and Space with Variables 10

Chapter Preview

You learn to use the LET command to name numbers, words or sentences. We clear the screen with CLS and the memory with CLEAR, and introduce "variables."



```
10 LET A = 6328
20 LET A$ = "FRED"
30 PRINT A
40 PRINT A$
```

Type in the above program. Remember the SYMBOL SHIFT key for $ = and " . Then press RUN and ENTER. Can you see what has appeared? Press ENTER again and your *program listing* will be back on the screen.

```
10 LET A=5328
20 LET A$="FRED"
30 PRINT A
40 PRINT A$

Ok
```

Lines 10 and 30 have the same effect as if you had typed PRINT 5328. The letter A, when preceded by LET and followed by =, becomes a *variable*. In the program above, of course, it takes more time and more memory space in the computer to do it this way. But if you had a program where the number was printed several times, using the variable A instead of the entire number 5328 each time would be easier to type, and would save precious memory space in the computer.

In some programs, as we will see later, the number A stands for changes during the program; that's why A is called a variable.

The program statement

LET A = 5328

is called an *assignment statement* because it *assigns* a value (5328) to a variable. The letter A is called a variable name.

A variable name does not have to be a single letter. It can be any length and contain letters or numbers *but the first character must be a letter.* The statement could be written

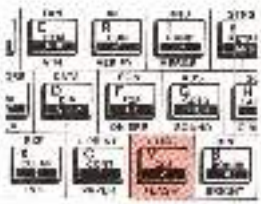
LET NUMBER = 5328 or
LET THIS NUMBER = 5328 or
LET A5328 = 5328

and so forth.

Lines 20 and 40 do the same thing as lines 10 and 30, except that A$ is a *string variable*. (A$ is pronounced "A string" rather than "A dollar sign.")

Strings can be any length — whatever is between quotation marks — but the name of a string variable must be one letter followed by $. You could write

```
LET A$ = "HELLO"
LET B$ = "WAY DOWN UPON THE SWANNEE
RIVER"
LET C$ = "6525 "
```



Now we have the program on the screen. Press the V key, which in keyword mode will give you CLS, and press ENTER.

CLS stands for CLEAR SCREEN. The program is gone. But if you press ENTER again, it comes back. It was taken off the screen, but it stayed in memory.

Now press the A key, for NEW, and ENTER. Then press ENTER again. The program, this time, is gone. NEW erases everything in the computer and on the screen, and readies the TS 2000 for a NEW program.

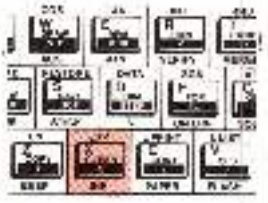
Let's go back to the immediate mode and try something else. Type

LET A = 5 ENTER

then press ENTER again. No program. Nothing in the computer? Try

PRINT A ENTER

Well, what do you know? The computer saves variables! Now this can get cluttered after a while, so there is a command to clear variables out of the memory. Press



CLEAR ENTER

then

PRINT A ENTER

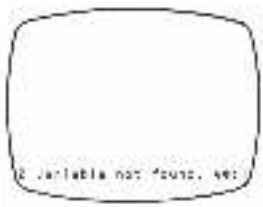
We see the screen report code 2, *Variable not found*. That's because we took the variable A and its assigned value out of the memory with CLEAR.

Let's try something else. Type this program:

10 PRINT A$
20 GOTO 10

remembering, of course, to press ENTER at the end of each line.

From now on, we're not going to talk about ENTER, but you must remember to press ENTER after every command or program line.



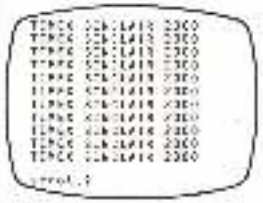
Can you guess what will happen when we RUN this program? Try it.

(Did you remember ENTER?)

A "2 Variable not found" report code again — because we didn't define the variable A$. Press ENTER to bring back the program. Now type, without a line number,

LET A$ = "TIMEX SINCLAIR 2000"

(or any other words if you prefer). Press ENTER again to get the program back on the screen and notice that the assignment statement isn't in it (because we didn't give it a line number).



Now type

GOTO 10

GOTO 10 starts the program running, and it uses the variable that was stored in the computer's memory. Why did we not GOTO 10 instead of RUN? Try to execute the program using RUN.

When you use RUN to execute a program, in effect you are saying CLEAR the variables and then GOTO line 1—the beginning of the program. This is so that leftover variables do not gum up the program. But if you want to use previously entered variables, just start a program with GOTO.

Enter a new string value for A$, using the LET statement. Run the program again, using GOTO. Then press CLEAR. Get the program back again with ENTER. Run it with GOTO again.

CLEAR eliminates variables, but leaves the program in the TS 2000.

Summary:

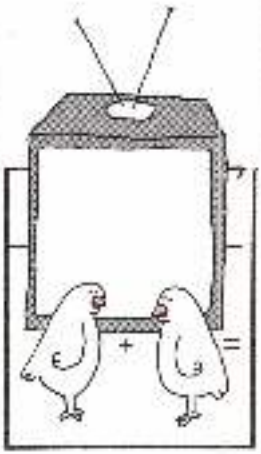
1. LET assigns values to variables.
2. Numeric variable names begin with a letter and can be any length.
3. String variable names are a single letter and $.
4. String variables can be any length, enclosed by quotes.
5. CLS clears the screen.
6. CLEAR erases variables from the memory.
7. RUN starts a program after clearing variables.
8. GOTO starts a program (at any line number you choose) without clearing the variables.
9. NEW clears everything from the computer.

Mathematics with the T/S 2000

11

Chapter Preview

You can add, subtract, multiply, divide, and use built-in functions like RND and INT.

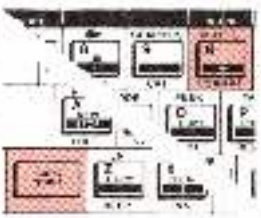


If you have used a calculator, you are used to typing in something like this

2 + 2 =

and getting the answer. Try this on your T/S 2000. Nothing happens. Press ENTER. A syntax error marker appears.

This doesn't look promising. Press CAPS SHIFT 0 (DELETE) until you get rid of it all.



Press CAPS SHIFT while pressing DELETE

Chapter 11: Mathematics with the T/S 2000

You can use your computer as a calculator, but you have to ask the right questions. Try this:

PRINT 2 + 2 ENTER

Okay, that's more like it. You can use any mathematical operation in the same way. These are:

+	SYMBOL SHIFT K	addition
−	SYMBOL SHIFT J	subtraction
*	SYMBOL SHIFT B	multiplication
/	SYMBOL SHIFT V	division
↑	SYMBOL SHIFT H	raising to a power

The addition and subtraction signs are the ones you are used to. Division uses a sign you have probably seen, because the computer does not have the ÷ sign in its character set. And an asterisk is used to stand for multiplication because the X is being used as a letter.

Raising a number to a power is a special case. The T/S 2000 cannot insert or understand *superscripts*, which is what we call our usual notation, so we use the ↑ symbol, which is common mathematical notation.

3^2	= 3 squared	=	3↑2
3^3	= 3 cubed	=	3↑3
3^4	= 3 to the fourth power	=	3↑4
3^{10}	= 3 to the tenth power	=	3↑10

All of these mathematical operations can take place in programs, of course.

Priorities and Parentheses

If you have a program line containing a number of mathematical operations, the Timex Sinclair 2000 will perform them in this order:

First, it will work out any powers, starting with the left end of the line and working to the right.

Second, it will do all multiplication and division, again working from left to right.

Finally, it will do all addition and subtraction, once again from left to right.

These are called the *priorities*, and are part of the way the computer is designed. Just as the character set includes more than just the letters of the alphabet, in fact, the priority rankings go well beyond the basic mathematical operations—a complete table can be found in Appendix A.

You may want to have operations performed in an order different from the computer's way, and you can arrange this by using parentheses. Anything in parentheses is done first (left to right) and the result is treated as a single number.

For example:

3+4+3 = 15

because 3*4 = 12 (multiplication before addition) and then 12 + 3 = 15. But

3*(4+3) = 21

because 4 + 3 = 7 (parentheses first) and then 3*7 = 21.

You can go for *nested* parentheses inside parentheses. The innermost parentheses will be done first, and then the computer will work its way out from there.

Scientific Notation

Sometimes, when a number is going to be more than 14 spaces long, the T/S 2000 will print it in *scientific* notation instead. This is a number with one digit to the left of a decimal point, some digits to the right of it, and then an E (for exponent), a + (or sometimes a minus) and a number which multiplies the rest of the expression by powers of 10.

For example,

2.34E-14

is 2.34 times 10 to the 14th power.

Try typing in any number more than 14 digits long (after a PRINT statement), like

PRINT 23456789023456789

and see what happens.

You can also use scientific notation in entering numbers, and the computer will convert the number to an ordinary expression—until it gets to be more than 14 digits long, and then it will go into scientific notation itself. Try

PRINT 2.34E0 ENTER
PRINT 2.34E1 ENTER
PRINT 2.34E2 ENTER

and so on. (You don't have to use the + sign, although the computer does.) At what point does the computer start returning scientific notation?

Rounding Errors

All computers have a problem with *rounding errors*—answers which are sometimes slightly incorrect due to the "rounding off" process. This is inherent in the binary-to-decimal conversion operation. (Most large computers have editing routines to correct for this built into their math handling procedures.)

If you were to write

```
10% LET A = 1.21 - 1
11% PRINT A
```

you would get an answer from the computer of 0.009999999998, when the correct answer is, of course, .01.

You can correct for this by adding

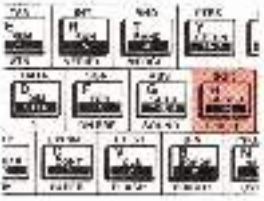
135 LET A = INT(A*100 + .5)/100

(INT is the function located above the R key, and obtained with the E cursor.)

Functions

Many mathematical functions are built into Sinclair BASIC. To take the square root of 9, for example, you would type

PRINT SQR 9

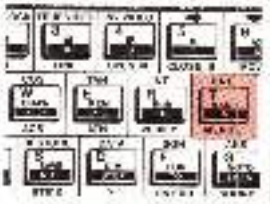


using the function SQR above the H key.

We will not spend any more time on functions at this time; you'll find them all delineated in Appendix A. As we said back in Chapter Three, mathematicians probably know them anyway and the rest of us probably don't need them.

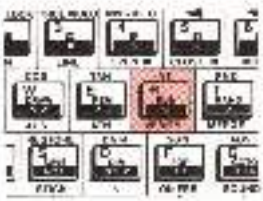
The Random Number Generator

There is one function we want to discuss a bit further, because it is very useful in programming conversational exercises and games. That is the random number generator.



The function RND, located above the T key, gives you a number between 0 and 1. Try a few of them.

PRINT RND



Now, this may not seem terribly useful, but you can use it to obtain whole numbers in any range you want. For example, if you want the computer to "pick a number between 1 and 6," you type

```
PRINT INT (RND*6) + 1
```

INT (for integer) is the function above E, conveniently close to RND. You have to dip in to extended mode twice to do this. Here's what is happening:

1. RND*6 is generating a decimal fraction between 0 and 1, and multiplying it by 6.
2. INT is rounding that number down to a whole number. If it's 3.09345622, it becomes 3. If it's 0.07888545, it becomes 0. If it's 5.7769, it becomes 5.
3. That gives you a whole number between 0 and 5, and you wanted one between 1 and 6, so we add one (+1). This step is necessary because the INT function rounds down, not up.

You may want to write that formula down somewhere, because you'll use it a lot.

"Picking a number between 1 and 6" simulates the roll of a die. Do it twice and you have a dice roll.

The value returned by a RND function can be assigned to a variable—

```
LET A = INT (RND*32)
```

—and used for all kinds of things, including deciding where on the screen some symbol is to be placed!

One last thing: RND is actually not a true random number generator, but only a "pseudo-random" function. It actually gives you, one at a time, numbers from a long table that has been randomly generated. The table is so long that you won't be able to memorize it, but you can memorize

the first few numbers which will always be the same the first time you turn on the computer.

Prove it. A couple of times, turn it off and then on again and try

```
PRINT RND
```

a few times. It's the same sequence!

To get away from the early part of the list, you use the RAND keyword, on the T key. When you first plug in the computer, or within a program before any RND functions, type

```
RAND 0
```

When you use the 0, RAND (which stands for *randomize*) finds a place to start in the table based on how long your computer has been on (how many frames have been sent to the TV), which is about as random as you'll need.

On the other hand, if you use any number other than 0, RAND starts using the table at a certain point based on that number, so that

```
RAND 58
```

will always start RND with the same number. Try it out.

Summary

1. The T/S 2000 will carry out the mathematical operations of + (addition), − (subtraction), * (multiplication), / (division) and ↑ (raising to a power).
2. This can be done anytime within a program. In the immediate mode, you need to use a **PRINT** command to see the result:

   ```
   PRINT 2 + 2
   ```

3. Mathematical operations are carried out in a particular order of priority; you can circumvent the priority ranking by using parentheses. Operations in parentheses are executed first.
4. The Timex Sinclair 2000 can understand scientific notation, and will use it with large numbers.
5. The computer has a number of built-in mathematical functions, which can be accessed with single keys in function mode.
6. The function **RND** generates pseudo-random numbers.

Programs That Ask for Information

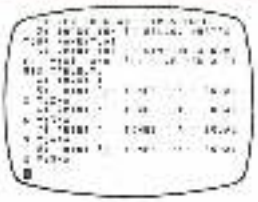
12

Chapter Preview

This chapter covers the use of the INPUT command to enter information into a program, and how the READ, DATA, and RESTORE commands let the computer look information up.

```
 10 REM PROGRAM — TIMES TABLE
 20 INPUT INK 1;" HELLO. WHAT'S
YOUR NAME? ";A$
 30 PRINT INK 2;" GIVE ME A NUMBER, ";A$;", AND
I'LL GIVE YOU A TIMES TABLE. "
 40 INPUT A
 50 PRINT 2;"TIMES ";A;" EQUALS ";2*A
 60 PRINT 3;"TIMES ";A;" EQUALS ";3*A
 70 PRINT 4;"TIMES ";A;" EQUALS ";4*A
 80 PRINT 5;"TIMES ";A;" EQUALS ";5*A
```

Programs can be written so that they will stop and ask for information to work on. One way they do this is with the **INPUT** statement.



Type in the program shown and, before you RUN it, let's walk through it, line by line:

Line 10 is our standard REM statement, telling us what the program is.

Line 20 is an INPUT statement. If you simply enter

INPUT A

the program will, when it reaches that line, stop and wait for the user to enter a number. It will indicate that it is waiting for a number by placing a flashing cursor [L] or [C], depending on whether CAPS LOCK is engaged, at the bottom of the screen.

If you enter, for a program line,

INPUT A$

the computer will indicate that it is waiting for a string (it could be a single letter) by showing the [L] or [C] cursor in quotation marks.

And, you can insert a "prompt" to explain the return of the input desired by putting a sentence in quotation marks between INPUT and the variable to which the input will be assigned:

INPUT "PROMPT";A

Finally, you can specify an ink color (and a paper color, if you like) for the prompt, as we have done. Just be sure each portion of the statement is followed by a semicolon.

20 INPUT INK 1;"PROMPT";A$

Line 30 is an alternate way of inserting a "prompt"; this one will show at the top of the screen. Notice that, since your name — the variable A$ — will be printed in the second half of the screen, we put a comma instead of a semicolon at the end of the line, starting the next PRINT item on the next line. (By the way, it is because that PRINT item is exactly 32 characters wide — a full screen — that we use two commas at the end of the line to position the times table.)

Line 40 is a simple INPUT statement, calling for a number.

Lines 50 through 80 print out a multiplication table. Notice the spaces inserted inside the quotes, before and after the words.

You can use EDIT to easily duplicate repetitive lines like 60-80. After entering line 50, press CAPS SHIFT 1 (EDIT) and line 50 will appear at the bottom of the screen. DELETE the line number and type in 60 instead, then use the cursor arrows and DELETE to make the other changes needed, and press ENTER.

RUN the program. Notice how the prompts helped you to respond to the INPUT statements.

The PRINT statements in lines 50-80 print in black because there was no INK color chosen in that statement, as there was in lines 20 and 30. But try this: DELETE the INK 1 from line 20, and insert it as a separate line.

```
25 INK 1
```

RUN the program again and see what happens in lines 50-80.

INK as a separate statement changes the "default" INK color — the one that every PRINT statement uses unless it has its own INK command.

(One exception is the " prompt" within an INPUT statement, which always uses black or white: whichever provides maximum contrast — unless an INK color is specified elsewhere within the INPUT statement.)

Here's another use of INPUT, asking you to enter a number each time the loop repeats:

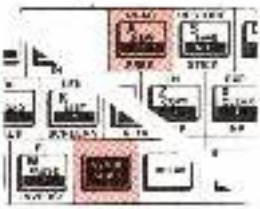
```
10 REM PROGRAM — THIRTEENS
20 PRINT "I'LL MULTIPLY EACH NUMBER YOU"
"GIVE ME BY 13 "
30 INPUT A
40 CLS
50 PRINT A " TIMES 13 IS " A*13
60 GOTO 30
```

With this program, you input a number and the computer shows you the answer.

If you should enter a letter when the computer wants a number, the program will stop with report code 2, **variable not found** (unless, by some chance, you have a variable in memory named by the letter you input . . .).

How do you stop the program when you are tired of it?

When the cursor is on the bottom of the screen, waiting for INPUT, enter **STOP** (SYMBOL SHIFT A) instead of a number.



To stop a program press SYMBOL SHIFT while pressing A.

Why didn't we put the CLS command before INPUT? Logically, it would seem that we ought to keep the steps in the process of showing the question together, and the steps in the process of getting and dealing with the output together. But change line 40 to line 25 and see what happens.

We want the prompt to stay on the screen until you can read it, so we don't clear the screen until after you enter your input.

Try taking out the CLS statement completely. Type:

25 ENTER

and then see what happens when you RUN the program.

You can use an INPUT statement to control the speed at which things happen. In that case, what you input is immaterial and is meaningless:

202 PRINT "PRESS ENTER TO CONTINUE"
210 INPUT AS
220 the next line

would serve the purpose; the program would wait for you to press ENTER before proceeding.

READ, DATA, and RESTORE

There is another way for a program to get information without having it in the body of the program

The DATA statement is a place to store values—either numbers or strings, or even both (numeric and alphabetic)—with each value separated by commas.

The READ statement inputs those values into the program, one at a time, until they are gone.

```
10 REM PROGRAM - CAPITALS
20 READ A$
30 PRINT "WHAT IS THE CAPITAL OF ";A$;"?",,
40 READ B$
50 INPUT C$
60 PRINT "YOUR ANSWER WAS ";C$,,
70 PRINT "THE CORRECT ANSWER IS ";B$,
80 GOTO 20
90 DATA "NORTH DAKOTA","BISMARCK",
"WISCONSIN","MADISON","MINNESOTA","ST.
PAUL"
```

Again, the multiple commas at the ends of the PRINT statements are for spacing on the screen.

RUN the program. It will stop when it has read all the data.

What is most useful about this program is that you can change or add data to it and make a real quiz. (You could even change the questions!)

There can be many DATA statements in a program, but they are treated as a single data list. So you could add to the program:

```
102 DATA "NEW YORK", "ALBANY", "RHODE ISLAND", "PROVIDENCE", "OHIO", "COLUMBUS"
110 DATA "SOUTH DAKOTA", "PIERRE", "CALIFORNIA", "SACRAMENTO", "FLORIDA", "TALLAHASSEE"
```

or even

```
120 DATA "ENGLAND", "LONDON", "FRANCE", "PARIS", "WEST GERMANY", "BONN", "NORWAY", "OSLO"
```

You can have any number of items in a DATA statement. It is simply easier to edit a program if you don't put them all in one long statement. They can be numbers, which the program reads by

```
READ A
```

or strings (one or more letters in quotes), read by

```
READ A$
```

And, of course, the data items don't have to be in pairs; they can be read as individual numbers or strings, or in any groupings that fit your program.

You may want to use a body of data for a number of operations and periodically "reset" the data list to the beginning (that is, have the next READ statement start over and use the first item again). At the proper place in the program, then, you would insert a line like

```
20 RESTORE
```

You can also reset the data list to a specific line (not necessarily all of the data) by using RESTORE with a line number.

For example, if we had added lines 100, 110 and 120 as above, you could use

```
RESTORE 100
```

to have the next READ statement start with "New York" rather than "North Dakota". (This is another reason to use more than one DATA line in a program.)

If you should want to STOP a program that is taking string inputs — the cursor at the bottom of the screen is in quotes — you have to DELETE the left quote before you press STOP. Otherwise the program will treat the word STOP as a string input.

You could also insert this program line

```
65 IF C$ = "STOP" THEN STOP
```

and, if you input the letters STOP in response to the cursor in quotes, the program will stop.

Summary

1. INPUT stops the program to wait for information to be entered by the user.

   INPUT A assigns a number to the variable A; a cursor at the bottom of the screen signals that the computer is waiting for the input.

   INPUT A$ assigns a string to the variable A$; the cursor is enclosed in quotes to signal that a string is needed.

2. STOP, in response to the INPUT cursor prompt, stops the program. If the cursor is in quotes, the left quote must be DELETEd before STOP is entered.

3. DATA statements hold numbers and/or strings separated by commas, for use in programs.

   There can be many separate program lines beginning with DATA, but the computer treats them as a single "data list."

4. READ statements input items from the data list, one at a time in order, for use in the program.

   READ A (or READ A$) assigns the next item in the data list to the variable A (or A$).

   (If the data doesn't match the READ statement, an error will result — for example, if READ A encounters a string.)

   READ A,B,C reads the next 3 items on the DATA list and assigns them to variables A, B and C.

5. The RESTORE statement directs the next READ statement to the first item in the DATA list.

Programs That Repeat: Looping

13

Chapter Preview

Repetitious work is easy with FOR, TO, NEXT, and STEP. We also use LIST to print the program on the screen.

```
10 FOR I = 1 TO 10
20 PRINT "NUMBER" I
30 NEXT I
```

Type this program into your Timex Sinclair 2000. In line 10, do not spell out TO, but use SYMBOL SHIFT F. RUN the program.

We said earlier that GOTO was a very powerful BASIC statement. We used it to make a program repeat, by telling the computer to GOTO an earlier line number and execute the same operations again.

The process of repeating the same operations a number of times is called *looping*, and the part of the program that repeats is called a *loop*. When we simply use GOTO, we have no control over how many times the program repeats; in fact it does not stop (until the screen fills, or we press BREAK).

This is called an *endless loop*, and is not too useful.

The FOR/NEXT Loop

We need loops that are under our control. In Sinclair BASIC the best way to do this is with what is called a FOR/NEXT loop. The program above shows the form of such a loop:

Line 10 contains the keywords FOR and TO, two numbers indicating how many repetitions are desired, and a counter or control variable. A control variable in a FOR/NEXT loop must be one single letter. Computer experts usually use I, but any letter will do.

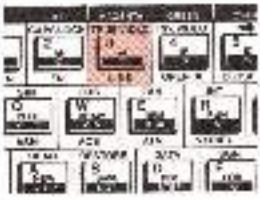
Line 20 contains the action that is to be repeated within the loop. There can be many lines of activity here, not just one.

Line 30 is necessary to close the loop. It tells the computer where to stop and go back to the FOR line.

Try this: get the program back on the screen with ENTER, then add

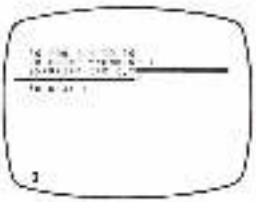
25 PRINT "

Between the quotes is the graphic symbol on the 3 key. Remember how to get it?



After you've pressed SYMBOL SHIFT P for the ", press CAPS SHIFT 9 to enter the *graphics mode*, with the G cursor on the screen. Then press the 3 key 32 times, because the screen is 32 characters wide. Then press CAPS SHIFT 9 to exit the graphics mode and, with the L cursor showing, close the quotes with SYMBOL SHIFT P. (Then ENTER all that.) RUN the program and you'll see a chunky bunch of nuthatching.

Incidentally, another way to determine if you've got your full 32 characters — a full line on the screen — between the quotes is to see that the close quote is one space past the open quote (put on the next line). Put another way, see that the material between the quotes would meet if it were on the same line.



Want some color in the display? Try adding INK 2, to line 25:

```
25 PRINT INK 2 "
```

The LIST Command

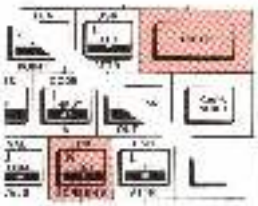
Bring back the program with ENTER. You see the program cursor at line 25—the last line you entered. We want to EDIT line 10. But instead of using the cursor arrows, let's look at another technique.

In most BASICs, pressing ENTER does not bring back a program listing as it does on the TS 2000. The command LIST is required, and you can add a line number for where to begin the listing.

You can also use LIST on the Timex computer. As you noticed, if you simply press

ENTER

after a program has stopped, the listing appears on the screen, *from the beginning to as many lines as will fit on the screen*. The cursor is at the last program line entered.



If you press

LIST ENTER

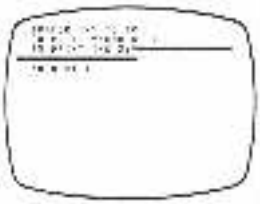
you will see the listing from the beginning, but two things will be different:

1. The program cursor will be at the first line, and
2. The query "scroll?" will appear at the bottom of the screen. If you press N (for No), BREAK, or STOP (SYMBOL SHIFT A), the listing will not scroll. If you press any other key, the next 22 lines will appear. This can be done until the entire listing has been shown.

If you type

LIST 90 ENTER

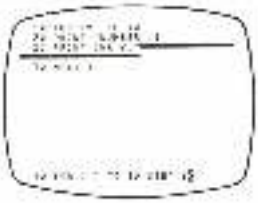
you'll see the program listing beginning with line 90, and the cursor will be at line 90. This is useful for editing a line in the middle of a long program.



So, in order to edit line 10, instead of using the cursor arrows, type

LIST 10 ENTER

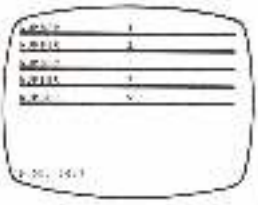
(in this case, you could just press LIST, of course), then respond to the "scroll?" prompt by pressing the N key.



Adding STEP to the FOR/NEXT Loop

Now we have the program cursor where we want it. Press EDIT and bring Line 10 down, then use the right cursor arrow to move to the end of the line and add STEP 2 (using SYMBOL SHIFT D, not spelling out STEP):

```
10 FOR I = 1 TO 10 STEP 2
```



Enter the line back into the program, then RUN.

STEP does just what you'd expect it: 'count by' the number following the command STEP. How would you get the program to count 2, 4, 6, 8, 10 rather than 1, 3, 5, 7, 9? Try it.

You can 'count down' with STEP, too. Try changing line 10 to read:

```
10 FOR I = 10 TO 1 STEP -2
```

The computer will not count down by ones if you say:

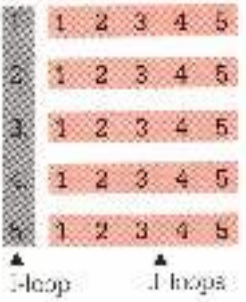
```
10 FOR I = 10 TO 1
```

You have to add STEP -1. Try it and see.

Try re-doing the TIMES TABLE program in the last chapter to replace lines 50-60 with a FOR/NEXT loop. Notice that you can make the table as long as you wish with very little effort.

Hint:

```
FOR I = 1 TO 10
PRINT ... I*A
NEXT I
```



Nested Loops

Can you have a loop inside a loop? Certainly. Many times, you'll want to repeat actions which contain other actions which you also want repeated. You can do this up to 26 times before you run out of letters to use as control variables (remember, a control variable in a FOR/NEXT loop must be one letter long).

That is, you can do it as long as the loops are properly nested. Try this:

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 5
30 PRINT I;":";J;" ";
40 NEXT J
50 PRINT
60 NEXT I
```

Be careful with line 30: get it just right and put a space between the second pair of quotes. And notice we had to pick a second letter besides I to count for the second loop.

RUN the program.

Effectively, the rows are counted by the "I-loop" and the columns by the "J-loop."

The PRINT in line 50 serves to move the print position to the beginning of the next row to start a new I-loop.

For a more color effect—and to perhaps add in understanding the loops—try adding INK I or INK J in line 30, as

```
30 PRINT INK I;I;":";J;" "; or
30 PRINT I;":";INK J;J;" ";
30 PRINT INK I;I;":";INK J;J;" ";
```

And of course, you could always throw in a pause for effect:

```
10 BORDER 5
```

This loop is not correctly *nested*, because the entire J-loop is contained within the I-loop. You will have a problem if the loops overlap. Sometimes you'll get error messages, sometimes you'll just get incomprehensible results. Try exchanging lines 40 and 60, so the program looks like this:

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 5
30 PRINT I;" ";J;" ";
40 NEXT J
50 PRINT
60 NEXT I
```

RUN it and try to figure out what you have!

One other problem you must be careful of when using FOR/NEXT loops: you can't "jump" into the middle of a loop from the outside by using a GOTO. When it hits the NEXT command without having passed a FOR, the machine will abort.

Summary

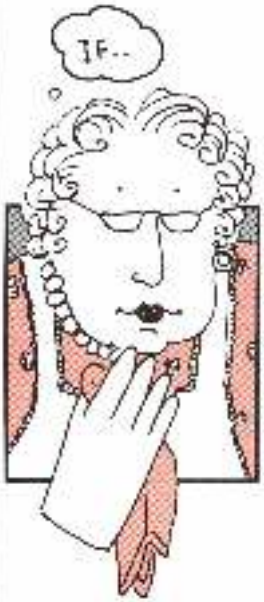
1. FOR/NEXT loops, using the *keywords* FOR, TO and NEXT, and a *control variable*, give you controlled repetitions.
2. Control variables are named by any single letter.
3. STEP is used in a FOR/NEXT loop to count by anything other than ones, and by negative numbers.
4. Multiple loops must be nested, not overlapped.
5. LIST brings a program listing to the screen; LIST with a line number starts the listing with that number and places the program cursor at that line.

Programs That Decide: Branching

14

Chapter Preview

IF and THEN are used, with the mathematical relations =, <, >, <=, >=, and <>, to make decisions. AND, OR, and NOT are used to combine relations.



We've considered three of the four reasons that the computer is such a powerful and valuable tool:

1. It works fast.
2. It can remember a lot of information, including its own instructions (programs).
3. It can repeat operations over and over, tirelessly (looping).

Now let us look into the fourth:

4. The computer can make decisions.

A program can contain a number of instructions, some of which are carried out by the computer under certain conditions and others which are carried out given different conditions. Such a program is said to branch, or to be a *branching program*. The command that makes all this possible is IF.

Type in the program below but before you RUN it, let's talk about it—line by line—to review some earlier points and raise some new ones. THEN, incidentally, is SYMBOL SHIFT G, not spelled out.

```
10 PRINT "NUMBER ", "LARGEST SO FAR"
20 INPUT A
30 LET LARGEST = A
40 PRINT A,LARGEST
50 INPUT A
60 IF LARGEST< A THEN LET LARGEST = A
70 GOTO 40
```

Line 10 is a simple PRINT statement; the comma between the two strings means they will be printed at positions 0 and 16 on the screen—the left edge and the middle.

Line 20 is an INPUT statement. It asks for a number from the user, and assigns the input to a variable named A. When you RUN the program, the L cursor at the bottom of the screen signals you that the TS 2000 is waiting for your input.

Line 30 sets a variable, LARGEST (notice that the variable name LARGEST helps you remember what it is) equal to the variable A.

Line 40 prints those two variables, directly under the labels printed in line 10 (notice the comma again).

Line 50 asks you to input another number and assigns it to A. This number replaces the previous value for A.

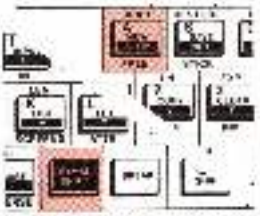
Line 60 is the decision statement. IF the number that is called LARGEST is *less than* (<) the new value for A, the program sets it to be equal to A. . . the latest input is the "largest so far." If LARGEST is already larger than A (or equal to it), it is left alone.

Line 70 creates a loop by going back to line 40. Then only the portion of the program between lines 40 and 70 are repeated, as often as you care to keep entering new numbers.

Okay, now RUN the program and enter some numbers to see how it works.

This program contains the dreaded "endless loop." Fortunately, this is not so serious in a program that stops periodically to wait for input.

You can stop it by responding to the INPUT statement — when the L cursor is at the bottom of the screen as a prompt—with STOP (SYMBOL SHIFT A) instead of a number.



Press SYMBOL SHIFT while pressing STOP.

It is also possible to interrupt this program by pressing the BREAK key, but you have to be fast—this only works when the program is not waiting for input. Try it: you have to press BREAK when the L cursor is not on the screen.

You might want to add a prompt which appears only when input is being called for—but which is not on the bottom line of the screen.

Here's a way to add such a prompt. Add a few lines so the program looks like this (remember, in lines 3h, 4h and 4h, AT is the function on the I key—with SYMBOL SHIFT and is not spelled out):

```
5 LET X = 3
10 PRINT "NUMBER", "LARGEST SO FAR"
15 PRINT AT X + 5,1;"ENTER A NUMBER"
20 INPUT A
25 PRINT AT X + 5,1;"                    "
30 LET LARGEST = A
40 PRINT AT X,8, A, LARGEST
45 PRINT AT X + 5,1;"ENTER A NUMBER"
50 INPUT A
55 PRINT AT X + 5,1;"                    "
60 IF LARGEST< A THEN LET LARGEST = A
65 LET X = X + 1
70 GOTO 40
```

Line 5 sets X equal to 3; line 40 will print A and LARGEST on that line. Line 65 will increase X by one each time through the loop: the numbers will be printed on line 4, then line 5, and so on.

Lines 15 and 45 print the prompt five lines below X. The movement of the prompt will call attention to it for each new INPUT as it will not be off the screen for very long when it is erased by lines 25 and 55. (To "erase," those lines simply PRINT a line of blank spaces to replace the words in the prompt.)

Line 40 has to be changed to specify the location of the printing of the next pair of numbers because the **PRINT AT** command in lines 15 and 45 moves the print position to line 20. If we did not change line 40, the next pair of numbers would be printed on line 21.

Incidentally, doesn't line 65 look odd? What kind of math is X = X + 1? Well, of course it isn't math at all but another *assignment statement*. It means "let X equal the previous value of X, plus one."

This means you have another way of writing a loop. It is a little clumsier than a **FOR/NEXT** loop, so we seldom use it. The one advantage it *has* is that you could use a more descriptive variable name in place of X—you might say

```
  5 LET LINE = 3
 65 LET LINE = LINE + 1
```

where, you will remember, variable names that count for a FOR/NEXT loop must be only a single letter.

Try erasing line 65 and see what happens. It is because X is not increased and returns to line 3 each time we repeat the loop.

Can you print your prompts in lines 15 and 45 in inverse letters?

The IF Statement

The IF statement checks to see whether a particular condition is true: if it is, the rest of the statement on that line is executed. If it is not, the rest of the line is ignored and the program moves to the next line.

Often the form of the statement is like

```
40 IF A = 5 THEN GOTO 100
```

meaning that if a variable called A is equal to 5 the program goes to line 100 and starts running at that line. If A is not equal to 5, the next line executed would be the one after 40 (probably 50, right?)

In Sinclair BASIC, by the way, we must include the THEN with IF and GOTO (some BASICs let you omit it). Among the things THEN does is to return the ■ cursor to the screen so you can use a keyword like GOTO; otherwise you wouldn't be able to give the computer any commands to execute (A = 5

Sometimes — as in the example at the beginning of the chapter — we have to INPUT the value on which the decision is made. More often, the value results from some calculations within the program. We can even use IF to terminate a loop:

```
10 LET I=1
20 PRINT I*100
30 LET I=I+1
40 IF I=6 THEN STOP
50 GO TO 20
```

```
10 LET I=1
20 PRINT I*100
30 LET I=I+1
40 IF I=6 THEN STOP
50 GO TO 20

0
```

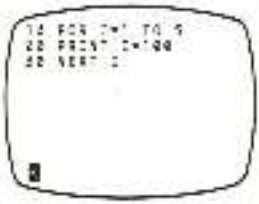
You could save a line by changing line 40 to read

```
40 IF I<6 THEN GOTO 20
```

and eliminating line 50.

```
100
200
300
400
500

0 STOP statement, 40:1
```



```
10 FOR I=1 TO 5
20 PRINT I*100
30 NEXT I

0
```

How would you write that program using a FOR/NEXT loop?

```
10 FOR I = 1 TO 5
20 PRINT I*100
30 NEXT I
```

```
100
200
300
400
500

0 OK, 30:1
```

The IF statement compares values using these mathematical symbols:

=	is equal to	SYMBOL SHIFT L
<	is less than	SYMBOL SHIFT R
>	is greater than	SYMBOL SHIFT T
<=	is less than or equal to	SYMBOL SHIFT O
>=	is greater than or equal to	SYMBOL SHIFT F
<>	is not equal to	SYMBOL SHIFT W

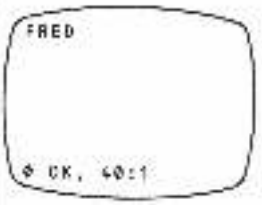
=,<,>,<=,>=,<>

DO NOT assemble a "less than or equal to" sign by typing SYMBOL SHIFT R and L. You must use the Q key for the combination. The same goes for > = and < >.

Comparing Strings

You can also use the symbols to compare strings. Usually you will do this with = or < > to see if an input matches a previously-chosen word. Type in this program, in which Fred wants you to try to guess his name.

```
10 INPUT A$
20 IF A$="FRED" THEN GOTO 40
30 GOTO 10
40 PRINT A$
```



RUN the program. Notice the **C** cursor at the bottom of the screen is in quotes, prompting you to enter a string.

Make a few wrong guesses, then input the right answer.

Now try this: add a new line

```
5 LET B$="FRED"
```

and change line 20 to read

```
20 IF A$<>B$ THEN GOTO 10
```

You can use the cursor arrows, EDIT and DELETE to substitute or you can just type a new line in. You are replacing "FRED" with B$, and the = sign with < > (remember, SYMBOL SHIFT O, not SYMBOL SHIFT R and SYMBOL SHIFT L).

Then guess these names: JIM, HARRY, JOE, and AL. What happened?

The mathematical symbols operate on a string by comparing the first letter of the string. If the first letter of A$ comes earlier in the alphabet than the first letter of B$, then A$ is said to be *less than* B$. If the first letters of two strings are the same, then the second letters are compared, and so on. So with strings

earlier in the alphabet is *less than*
later in the alphabet is *greater than*

In fact, what the computer actually does is compare the code numbers of any characters in the TRS-2000's character set. If you refer to the Appendix titled the character set, you'll find that A>0. The alphabet follows the numerals in the character set, so any letter is greater than any numeral.

Test this with Fred's program. Type in a number—say, 25—in answer to the input request.

How would you add prompts to Fred's program like "Guess my name" and "Wrong. Guess again"?

Hint: you can also change line 40 to read

```
40 PRINT "GUESS MY NAME IS ";A$
```

but only if you have the = sign in line 20, not the <=.

AND, OR, NOT

We call =, <, >, <=, >= and <> *relations*. We can combine them by using the *logical relations* AND, OR and NOT.

```
IF A=B AND C=D THEN GOTO 120
```

means that if both relations are true the program goes to line 100. If A is equal to B but C is not equal to D, though, the GOTO 100 is not executed and the program simply continues to the next line.

```
IF A< B OR C> D THEN GOTO 100
```

sends the program to line 100 if either A< B or C> D is true.

```
IF NOT A = B THEN GOTO 100
```

is the same as

```
IF A< > B THEN GOTO 100
```

In Sinclair BASIC, unlike some other "dialects," you must include both THEN and GOTO in lines like the above, to enable the multiple keyword system to operate.

But in addition, using both makes the program clearer and easier to understand.

Let's add a few lines to the name-guessing program:

```
 10 PRINT "GUESS MY NAME"
 20 LET I=0
 30 INPUT A$
 40 IF A$="FRED" THEN GOTO 110
 50 LET I=I+1
 60 IF A$< > "FRED" AND I=3 THEN GOTO 90
 70 PRINT "WRONG. MY NAME IS NOT ";A$
 80 GOTO 30
 90 PRINT "SORRY YOU LOSE!"
100 GOTO 120
110 PRINT "THAT IS RIGHT!"
120 PRINT "MY NAME IS FRED"
```

RUN the program and try a few guesses. The combination of relations in line 60 stops the game if you've had three guesses. Check the logic in the program and figure out how and why it moves from one line to another.

Try to tidy up the screen a bit by adding

55 PRINT

and

75 PRINT

Now we have a confession to make. Did you discover, when you analyzed the program, that line 60 doesn't need both relations?

Since line 40 transfers the program to line 110 if A$ = "FRED" then, logically, A$ must be not equal to "FRED" if we have reached line 60. So line 60 really only needs to say

60 IF I = 3 THEN GOTO 90

Try it and prove it to yourself. The lesson in this is to always plan a program carefully in advance and work out the logic of it so that you do things in the most direct way. The search for the simplest solution to a programming problem is what makes programming fun!

Summary

1. IF/THEN evaluates a condition; if the condition is true, the program does what is called for after THEN (usually GOTO another location in the program); if not, the next line in the program is executed.

2. IF evaluates mathematical values using the relations

   = equal to
   < less than
   > greater than
   < > not equal to
   < = less than or equal to
   > = greater than or equal to

3. More than one mathematical relation can be combined, using

   AND (both are true)
   OR (either is true)
   NOT (a relation is not true)

Programs within Programs: Subroutines

15

Chapter Preview

Recycle your program lines with self-contained subprograms, using GO SUB and RETURN.

If you hang around with professional programmers a great deal, you'll probably hear the phrase "structured programming." Ask any ten of those pros what that means, and you'll probably get ten different answers.

One answer that never gets mentioned is that "all good programming is structured; that is, it is planned and well organized."

Another answer you'll often hear is that structured programming involves modules within larger programs. A task is analyzed and broken into subtasks, and then each subtask is dealt with in a self-contained sub-program within the main program. This has a number of benefits.

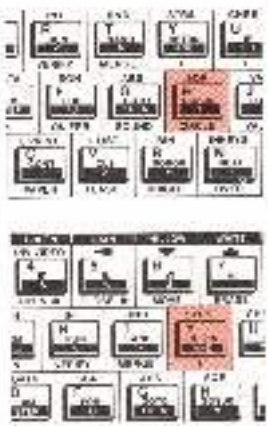
1. The program is easier to understand when looked at later by the person who wrote it or by someone else.

2. The program is easier to change, or "maintain."
3. When it is first being written, a large-scale program can be assigned to a number of programmers, a module to each. This speeds up the development of commercial software.
4. Finally, the process of subdividing a problem into subtasks is an aid in thinking through the process of programming and of problem solving.

All of these benefits, except for number 3, can be useful to us in working with the TS 2000.

Using the first definition we noted above, and the concept of subroutines, we can make our programs "structured."

A subroutine is a self-contained "mini program" which can be "called" by the main program (or by other subroutines, or even by itself, which you'll have to find in other books), as many times as are desired. The subroutine performs its function when called upon, then returns to the main program.



GOSUB and RETURN

There are two very simple commands used for a subroutine, GOSUB and RETURN. GOSUB, with a line number, is inserted in the main program wherever the subroutine is desired; the line number is that of the beginning of the subroutine. RETURN (which is spelled RETRN on the Y key) is inserted at the end of the subroutine itself, and returns the execution of the program to the line following the line containing the GOSUB.

What problem would arise if you tried to use GOTOs instead of GOSUB and RETURN?

Here's an example:

```
10 REM THE MAIN PROGRAM
20
30
40 GOTO 1000
50
60
70

1000 REM THE SUBROUTINE
1010
1020
1030
1040 GOTO 50
```

Suppose you want to call that subroutine from several different places in the program . . . and you don't want to GOTO 50 each time when the subroutine is done.

That's what GOSUB and RETURN are for. GOSUB 1000 sends the computer to line 1000 just as GOTO 1000 does. But it remembers where it came from, and the command RETURN directs the computer to the line after the GOSUB command. For instance:

```
10 REM THE MAIN PROGRAM
20
30
40 GOSUB 1000
50
60
70 GOSUB 1000
80
90

1000 REM THE SUBROUTINE
1010
1020
1030
1040 RETURN
```

In the program model above, the command RETURN, ending the subroutine, directs the computer to line 50 after the first execution (called by line 40) and to line 80 after the second go-around (called by line 70).

GOSUB and RETURN can save work for you, and space in the computer's memory. But perhaps more importantly, they help you organize your programs so that other people trying to use them—and you, coming back to them after a lapse of time—can understand how they work.

```
 10 REM PROGRAM — MATH
 20 LET A = INT (RND*9) + 1
 30 LET B = INT (RND*9) + 1
 40 PRINT " DO YOU WANT TO"
 50 PRINT TAB 10; " ADD — PRESS 1"
 60 PRINT TAB 10; "SUBTRACT — PRESS 2"
 70 PRINT TAB 10; "MULTIPLY — PRESS 3"
 80 PRINT TAB 10; "DIVIDE — PRESS 4"
 90 INPUT D
100 IF D< 1 OR D> 4 THEN GOTO 40
110 CLS
120 GOSUB D*1000
130 INPUT E
140 PRINT AT 10,15; E
150 IF E = C THEN PRINT AT 15,10;" CORRECT"
160 IF E< > C THEN PRINT AT 15, 10;
    "SORRY, WRONG NUMBER"
170 PRINT AT 17,10;" WANT ANOTHER —
    Y OR N?"
180 INPUT AS
190 CLS
200 IF AS< > "Y" THEN STOP
210 GOTO 20
1000 LET C = A + B
1010 PRINT AT 10,10;A;" + ";B;" = ?"
1020 RETURN
2000 LET C = A - B
2010 PRINT AT 10,10;A;" - ";B;" = ?"
2020 RETURN
```

```
3020 LET C = A+B
3030 PRINT AT 10,10;A;"+";B;" = ";C
3040 RETURN
4010 LET C = A/B
4030 PRINT AT 10,10;A;"/";B;" = ";C
4040 RETURN
```

In line 100, use the keyword OR (SYMBOL SHIFT U), but in line 170, spell out the word OR.

This example program includes a number of concepts we have discussed in previous chapters. It also illustrates how the use of subroutines can serve to make a program's structure easy to follow. In fact, this program does not require subroutines to get the job done. Why not? (We'll give you the answer after we call attention to a few of the other features of this program.)

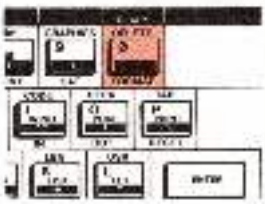
1. Lines 20 and 30 make use of the random number generator.
2. Lines 50-80 use TAB to format an indented column on the screen.
3. Lines 90, 130 and 160 use INPUT.
4. Line 100 is an error trap which repeats the question if you input any number other than one of the choices the program can deal with.
5. Line 120, instead of using 4 different lines with separate GOSUB addresses, uses multiplication to select the subroutine. This technique can also be done with GOTO.
6. Line 140 uses PRINT AT.
7. Lines 150 and 160 are both necessary, unlike the example in Chapter Fourteen. Why?
8. Line 200 STOPs the program if any other key than Y (for yes) is pressed, even though the screen asks for "Y OR N?". This is because, otherwise, inputting anything other than Y or N would be an error, causing the program to "crash." Could you use an error trap like the one in line 100 instead?

9. Each subroutine does a different mathematical operation, but works on whichever random numbers have been generated and provides an answer against which the user's answer is checked.
10. Rounding errors may give you trouble in the division subroutine beginning at line 4000. Can you use the correction routine in Chapter Eleven to remedy this?

The answer to our question is that you could actually use GOTO 2-1000 for each "subroutine" and GOTO 130 instead of RETURN at the end of each one.

Using DELETE To Erase Entire Program Lines

Before leaving this chapter, let's use this long program to illustrate another command. First, though, SAVE the program onto a cassette tape if you want to keep it — our next topic will erase it!



We've seen the use of DELETE (CAPS SHIFT 0) to remove a single character at a time. This can be done when the L or C cursor is on the screen. When the K cursor is showing, something else happens.

With the program listing on the screen, and a K cursor or report at the bottom, press DELETE.

The word DELETE appears on the screen. Type a line number, say 4000.

```
DELETE 4000      ENTER
```

Line 4000 is deleted. Type

```
DELETE 2010,2020      ENTER
```

Lines 2000 through 2020 are deleted. Try

```
DELETE 1000,      ENTER
```

Remaining lines from 1000 to the end are deleted. And

```
DELETE ,70
```

deletes from the beginning through line 70. Now you should have only lines 80-210 left. Delete them — all at once or in chunks — using the DELETE command.

There is a complication in the use of DELETE with the K cursor. Type

```
10 PAUSE 50:
```

but don't ENTER it!

Suppose you've changed your mind — you don't want the colon. Try to delete it. First you press DELETE with the K cursor on the screen, and the word DELETE appears.

Then you press DELETE with the L cursor showing, and you delete DELETE. But now the K cursor is back . . .

It turns out that the auto-repeat key feature is your solution. With the L cursor showing, hold the CAPS SHIFT and 0 keys down: the word DELETE and the colon (and, likely, PAUSE and some of the line number) will be deleted.

You could also, of course, ENTER the line including the colon, then delete the entire line or replace it with a corrected one.

Making Your BASIC Programs Run Faster

Here's a tip that will help you make your BASIC programs run much faster; we present it here because it has a lot to do with subroutines. You won't need it until you start writing long programs, but then it could be extremely useful.

Throughout this manual we discuss writing programs in what seems like a logical order: first you set up (initialize) your variables (LET A = 1, etc.) then you do the main program, then you fill in the subroutines.

This is true for computers, but since the TI 99/4A uses an interpreter, the programming logic is different.

It turns out that the computer searches for a line number it is routed to by a GOTO or GOSUB by checking each line number from the beginning of the program.

This means that if line 16 is LET A = 1 (for instance), and is never used again in the program, it is just an extra item to be sifted through on every GOTO or GOSUB.

Logically, then, it ought to be tucked away at the end of the program — as a subroutine! Your first program line might be something like

```
10 GOSUB 9000
```

and all your initial housekeeping — variables defined, user-defined graphics designed — etc. should be put in that subroutine. Then the program never looks at it again after the first GOSUB, and only one line has to be looked at each time the program goes to search for a new line number.

One exception to this rule is to put all DATA statements at the beginning of the program, since a program will search for them from the beginning each time it needs them.

By the same token, your often-called subroutines might be at the beginning of the program — not at the end. Then comes the main program, then the less-often used subroutines. The program skeleton might look like this:

```
  10 REM TITLE
  70 GOSUB 9010
 120 SUBROUTINE A
 220 SUBROUTINE B
 370 SUBROUTINE C
1010 MAIN PROGRAM
9010 SUBROUTINE I
9110 SUBROUTINE E
9210 INITIALIZATION ROUTINES
9910 ENDING FUNCTIONS
```

You'll have to make sure that each module ends with a direction (usually a GOTO) if you don't want the program to proceed to execute the next line (in example, you'll need a GOTO 2910 at the end of the main program to get to the ending functions).

Summary

1. Subroutines help you use another piece of "structured programming" in BASIC to make programs easier to use and understand.
2. GOSUB directs the program to a specific line, as does GOTO, but stores the location of the program line containing the GOSUB.
3. RETURN, the last logical line in a subroutine, directs the program to the next line after the GOSUB.
4. GOSUB and RETURN must be used together, like FOR and NEXT.
5. DELETE, when the ■ cursor is showing, is used to delete one or more program lines. The range of lines to be deleted is shown by the first and last line number separated by a comma.
6. Long programs will run much faster if seldom-used portions are placed after the most-used ones, with each module treated as a subroutine.

Arrays

16

Chapter Preview

Organize your data and save space with "arrays," using DIM, subscripts, and string slicing. SAVE and DATA work together to store arrays on tape.

An array is a way of structuring a number of values and keeping track of them. Each of the values is called an element of the array. You can think of an array as looking like a calendar:

SUN	MON	TUES	WED	THU	FRI	SAT
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

The Array

An element

You can assign numeric values to *elements* of an *array*, instead of giving each value a separate variable name. This can be used simply to save space, or when there is a relationship between or among the values.

Suppose you have a row of numbers:

12 5 7 22 14

You could assign the values to separate variables:

```
LET A = 12
LET B = 5
LET C = 7
LET D = 22
LET E = 14
```

Or, you can consider the whole row to be an array:

```
LET A(1) = 12
LET A(2) = 5
LET A(3) = 7
LET A(4) = 22
LET A(5) = 14
```

The number in parentheses is called the *subscript*.

Array A

Picture the array this way:

(1)	(2)	(3)	(4)	(5)
12	5	7	22	14

DIM . . . The Dimension Statement

Before you can assign values to each element in the array, you have to reserve space for the array with a dimension statement (DIM). To prepare for the array above, you would have to enter:

```
DIM A(5)
```

There are a few rules regarding array variables:

1. An array variable name must be a single letter (like a control variable in a **FOR/NEXT** loop).
2. An array variable name can be the same as the name of a simple variable (there can be a simple variable named A at the same time as an array named A; you can tell them apart because elements of the array variable are always referred to with the subscript).
3. A **LET** statement creates a previous simple variable by the same name. Similarly, when you create an array with the **DIM** statement, you delete any previous array of the same name. But while you can build a new simple variable from an old one (as in LET A = A + 1), you would simply lose an old array by **DIM**ensioning a new one by the same name. You can, however, use **LET** to change an element of an array, as with

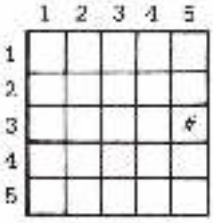
```
LET A(1) = A(1) + 1
```

Arrays in More Than One Dimension

You can have arrays of as many dimensions as you care to try to keep track of, as long as you dimension them properly at the outset.

```
DIM A(5,5)
```

would set up an array you can think of as looking like this:

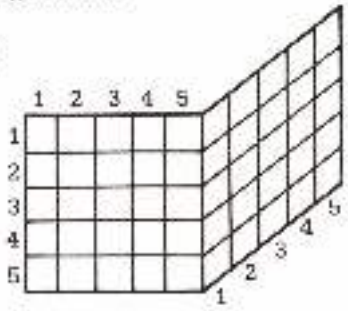


You can think of the first number as identifying the row, and the second number as identifying the column. Therefore, location # in the diagram is A(3,5).

What value multiplies the location (3,5) in our example?

You can have an array in three dimensions; think of it as looking like this.

DIM A(5,5,5)



And you can have arrays in four or more dimensions, too. If you can picture them, you are ready to explain the theory of relativity.

String Arrays

You can assign strings to arrays, as

DIM A$(5,5)

but there are a few rules here, too.

1. When you DIMension the string array, you delete any previous string arrays AND any previous simple string variables with the same name.
2. In two dimensions, you can think of the first number as being the identifier of each string, or "word," and the second as the number of letters in each word.
3. Assignment to the string variable elements is Procrustean, which means that the strings are filled in from character #1 up through a number of characters equal to the second subscript, and
   a. If the string is too long, it will be truncated—cut off—from the end.
   b. If the string is too short, the spaces will be filled in with blanks.

Why "Procrustean"? The method is named for a legendary innkeeper who wanted to make sure his guests fit the beds, with no wasted space. If they were too short, he stretched them on a rack; if they were too tall, he cut off their legs at the appropriate length!

Array A$

	1	2	3	4	5	
1	S	P	A	D	E	
2	H	E	A	R	T	
3	D	I	A	M	O	N D
4	C	L	U	B		
5	J	O	K	E	R	

You access the strings by using the first subscript only, and individual letters by using the second subscript as well.

PRINT A$(3) returns DIAMO

(two NDs are truncated) and

PRINT A$(4) returns CLUB

(including a blank space character). Also,

PRINT A$(3,2) returns I

—the second letter of A$(3).

PRINT A$(4,4) returns the blank space.

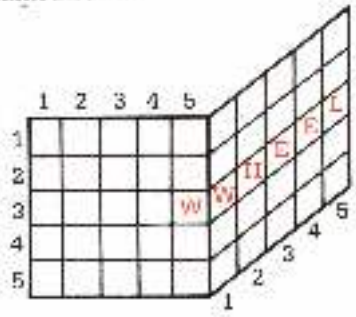
You can also use *string slicing* (more about this in a moment) to obtain a portion of a string, as

PRINT A$(1,2 TO 4) returns LUB
PRINT A$(3,3 TO 5) returns AMO

String Arrays in Three or More Dimensions

You can have string variable arrays in as many dimensions as you like. However many numbers (dimensions) are separated by commas, the *last number* identifies the number of characters in each string, and the other numbers serve to specify the string by its location. In three dimensions

PRINT A$(3,5,5) would return WHEEL in the array diagrammed below:



Again, it is not easy to picture string arrays in more than three dimensions, but just remember that the last subscript specifies the number of characters in each string in the array, while the other subscripts locate the string.

Slicing Strings

Running the program:

```
10 LET A$="HAM AND EGGS"
20 PRINT A$
30 PRINT A$( )
40 PRINT A$(6)
50 PRINT A$(1 TO 3)
60 PRINT A$(TO 3)
70 PRINT A$(9 TO 12)
80 PRINT A$(9 TO)
```

demonstrates string slicing. Line 20 prints the string. Line 30 shows that empty brackets after the string variable name means that the entire string is to be subscripted.

Line 40 selects as a substring only one character —the sixth (don't forget to count the spaces in the string).

Line 50 prints characters 1 to 3 of the string. Line 60 shows that you can omit the first digit and the first character is implied. Line 70 shows how to print the last four characters, and line 80 shows that you can omit the last number and "last character" is implied.

How would you print "AND" out of that string?

Now, we can use string slicing to save space in a program by assigning a long string to a variable name and then cutting pieces out of it, rather than assigning a lot of variables.

Saving and Loading Arrays on Tape

You can save an array on tape using the SAVE and DATA commands:

```
SAVE "TABLE" DATA A( )
```

would save under the name "TABLE" a numeric array that has been created and named A. among other things, this gives you the ability to store arrays of data under more descriptive names than are allowed single letter, and to store and find more than 26 arrays (since you can re-use the letters of the alphabet).

You need to include the parentheses, though you don't have to fill in the numbers that are part of the array's original name.

You reload the saved array with:

```
LOAD "TABLE" DATA A( )
```

You would normally do this with a program already in the computer, which will operate on the data. LOAD with DATA does not erase what is already in the computer (unless there is another array with the same letter name).

String arrays are handled the same way except for using the $ in the array name:

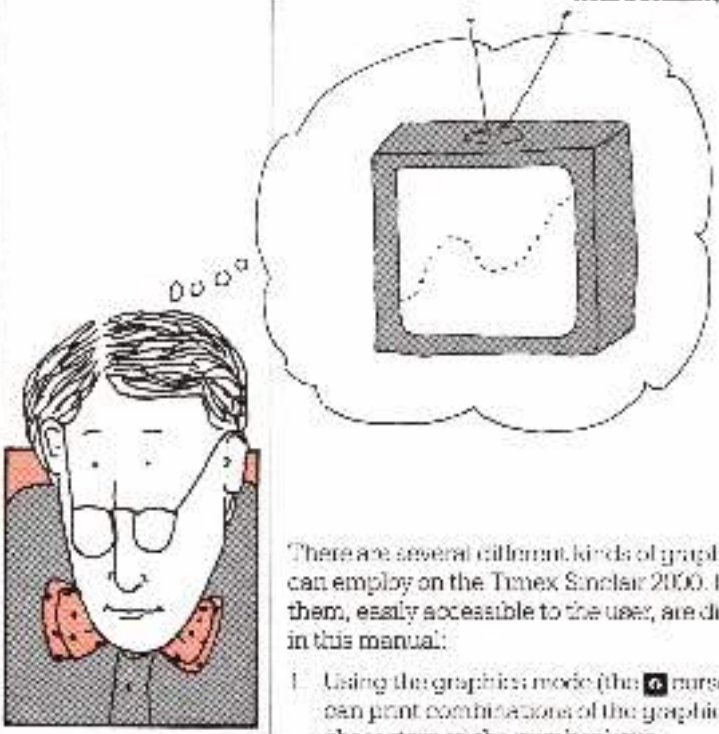
Summary

1. An array is made up of a number of elements, all with the same array variable name, and distinguished from each other by means of subscripts.
2. The name of a numeric array must be a single letter, there can also be a simple variable using the same letter as a name.
3. The name of a string variable must be a single letter followed by $, there cannot be a simple string variable with the same name in the computer's memory.
4. Before assigning values to the elements of an array, you must reserve space for it in the computer with the DIM statement.
5. You can "slice" strings and obtain substrings by using the TO statement.
6. Data arrays (numeric or string) can be SAVEd and LOADed using a command including the DATA statement.

Graphics 17

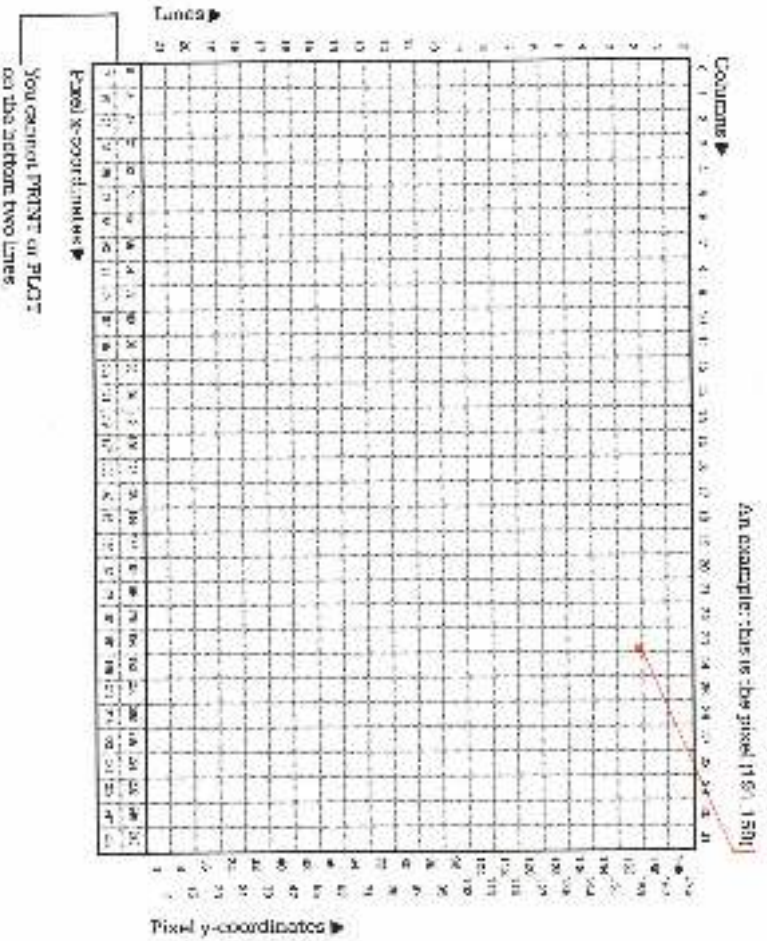
Chapter Preview

A complete look at making pictures on the screen with the PRINT and PLOT statements, and saving them on tape with SCREEN$.



There are several different kinds of graphics you can employ on the Timex Sinclair 2000. Four of them, easily accessible to the user, are discussed in this manual:

1. Using the graphics mode (the **G** cursor), you can print combinations of the graphic characters on the number keys.
2. You can design your own graphics and store them for use by pressing a letter key in graphics mode. We'll do this in the next chapter.
3. As we saw back in Chapter Four, you can use the DRAW and CIRCLE functions.
4. You can use the **PLOT** function, which we introduced in Chapter Four, and will explore further in this chapter.


Lines ▶
Columns ▶
An example: this is the pixel (150, 159)
Pixel x-coordinates ▶
Pixel y-coordinates ▶
You cannot PRINT or PLOT on the bottom two lines

Three additional graphics modes that really explore the power of the TS 2000 are available to advanced programmers (or anybody really into graphics) using machine code.

First, you can expand the 32 × 24 display to a full 64 × 24 (or even more than 64 if you design your own characters) using 512 pixels across the screen.

Second, you can use two 32 × 24 displays and flip between them for great animation effects.

Finally, the TS 2000 will allow, in Extended Color Mode, up to eight choices of color for each character position — each row of 1 × 8 pixels can have a different color. With Extended Color Mode you can create very high color resolution effects for some of the best possible home computer education, entertainment and business graphics effects. See Appendix C.

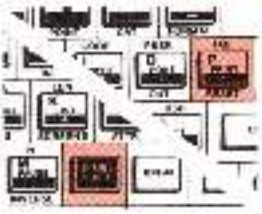
Using the Graphics Symbols with PRINT Statements

The graphic symbols on the number keys are part of the TS 2000's basic character set — see Appendix B. They are placed using PRINT statements, and use the 32 × 24 character screen.

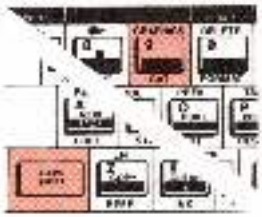
```
 5 REM PROGRAM    BARGRAPH
 10 FOR I = 1 TO 3
 20 PRINT AT 5,0;" INPUT 1 ABEL (10
CHARACTERS)#";I;: INPUT A$
 30 PRINT AT 1,10*I - 10;A$
 40 NEXT I
 50 PRINT AT 5,0;"

 60 FOR I = 1 TO 3
 70 PRINT AT 20,0;" ENTER VALUE (0
TO 15)#";I;: INPUT A
 80 FOR B = 19 TO 19 - A STEP - 1
 90 PRINT AT B,10*I - 7;"■"
100 NEXT B
110 NEXT I
120 PRINT AT 21;"
```

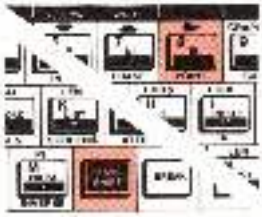
Type in and run the program above. It prints a bar chart. The graphic symbol in line 80 is the inverse of the black square that appears on the 8 key. Here's how you get it, after you've typed the semicolon.



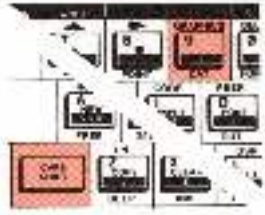
1. Press SYMBOL SHIFT P for the quotation marks.



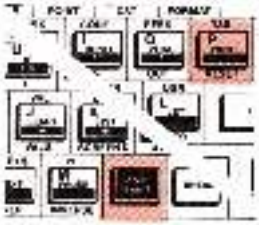
2. Press CAPS SHIFT 9 for graphics mode.



3. Press SYMBOL SHIFT 8 for the black square.



4. Press CAPS SHIFT 9 to leave graphics mode.



5. Press SYMBOL SHIFT P to close the quote.

Other notes on the program:

1. INPUT is spelled out in the prompts in line 20, and is the keyword on the I key in lines 20 and 70.
2. The number of spaces between the quotes in line 60 is equal to the number of characters — including spaces — in the prompt in line 2, plus one character for the variable i; that is, 30 spaces.

There needs to be enough spaces in line 120 similarly, to erase all of line 70. Count them to be sure.

This all-purpose program lets you chart almost any comparison you like. For our illustration, we've input the names of months as our labels, and values that could represent anything that happened each month — sales, expenses, trips to the zoo — to prepare the chart. Just use your own inputs; try several executions of the program.

(Later in the chapter we'll see how to SAVE a screen display like this chart on tape, and in Chapter 23 we'll see how to print it out on a printer.)

Can you figure out how the involved numbers in the PRINT AT statements, lines 30 and 90, place the labels and the bars?

Notice the prompts being inserted and erased by lines 20, 60, 70 and 120.

There is a nested pair of loops late in the program.

Do you see how line 80 prints the bars vertically — and, for effect, from bottom to top?

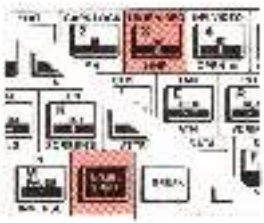
Can you change the program to add colors to the bars?

Can you figure out how to make the bars double or triple width?

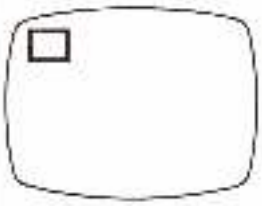
How would you print horizontal bars instead? (**Hint:** it's easier...)

You can use the various keyboard graphic symbols, in combinations of **PRINT** statements, to draw figures on the screen. Try (don't forget **NEW** first to delete the previous program):

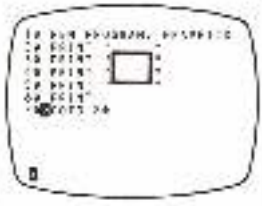
```
10 REM PROGRAM — GRAPHICS
20 PRINT "
30 PRINT "
40 PRINT "
50 PRINT "
```



Hint: In line 20, after the first quotation marks, enter graphics mode, then **SYMBOL SHIFT** 8 (for the inverse of the character that shows on the 8 key) five times, then exit graphics mode and close the quotes.



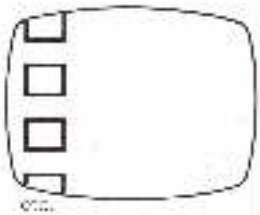
In line 30, again get into graphics mode and use **SYMBOL SHIFT** 8 for the left-hand graphic symbol, then type in three spaces. Finally, type 8 — this time without **SYMBOL SHIFT** — for the right symbol.



Line 40 repeats 30, and in line 50, you'd use the graphic on the 8 key, but without **SYMBOL SHIFT**.

You can also use graphic drawings more than once in a program. Add to the above program, the lines:

```
60 PRINT
70 GOTO 20
```

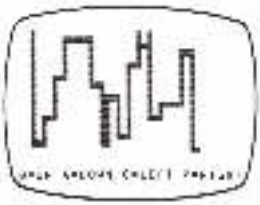


and RUN it.

What would it look like without line 60? How about if you GOTO 30 instead?

Try drawing other, more elaborate figures with the graphic symbols. Notice how they can be made to fit together.

Try this program, just for fun. (Be sure you're in G mode or it won't run.) Can you alter it so you can choose the ink color instead of having it selected by RND?



```
 5 REM PROGRAM—COLORSKETCH
10 PRINT AT 20,0;"Q = UP A = DOWN O = LEFT
P = RIGHT"
20 LET A = 0: LET B = 0
30 PRINT AT A,B;INK INT (RND*7) + 1;"■"
40 IF INKEY$ = "Q" THEN LET A = A + 1
50 IF INKEY$ = "A" THEN LET A = A - 1
60 IF INKEY$ = "O" THEN LET B = B - 1
70 IF INKEY$ = "P" THEN LET B = B + 1
80 GOTO 30
```

Be careful not to run off the screen in any direction; not only will the program stop with a report but you may wind up with a string of characters (due to auto-repeat) at the bottom of the screen.

There is a way to overcome that: add lines to stop the line at the borders.

```
45 IF A< 0 THEN LET A = 0
55 IF A> 21 THEN LET A = 21
65 IF B< 0 THEN LET B = 0
75 IF B> 31 THEN LET B = 31
```

(INKEY$—the N key in F mode—is explained fully in Chapter 19.)

Try running the program after changing the PAPER colors. Use different BORDER colors.

High Resolution Graphics with the PLOT Statement

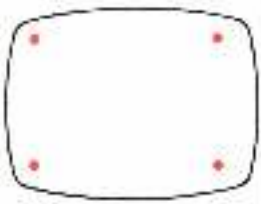
The format of the PLOT statement is

PLOT x, y

with x being a number from 0 to 255, from *left* to *right* on the screen, and y being a number from 0 to 175, from *bottom* to *top* on the screen.

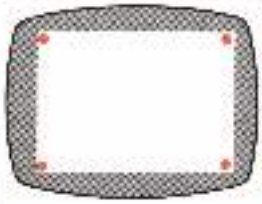
The PLOT command fills in with a black box so tiny it looks like a dot, the *pixel* (picture element) located by the two co-ordinates.

Here's a program that defines the screen for the PLOT command, placing a dot at each corner:



```
10 PLOT 0,0
20 INPUT A$
30 PLOT 0, 175
40 INPUT A$
50 PLOT 255,0
60 INPUT A$
70 PLOT 255, 175
```

After typing in the program, press RUN and ENTER and the first dot will be located—you'll have to look hard to spot it. The program will wait for an INPUT before plotting each of the other corners—but notice, it won't *do* anything with the input. We can use INPUT this way as a device until we are ready to proceed. (You can just press ENTER in response to the INPUT prompt.)

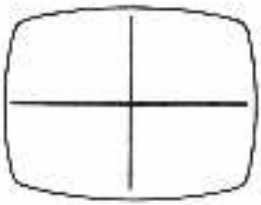


Check the locations of your four dots against the border by pressing

BORDER 4 ENTER

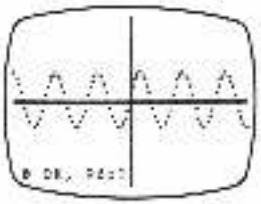
You can locate the x and y axes on the screen with this little program:

```
10 REM PROGRAM: X/Y AXES
20 FOR X = 0 TO 255
30 PLOT X,87
40 NEXT X
50 FOR Y = 0 TO 175
60 PLOT 127,Y
70 NEXT Y
```



Because the axes cross in the middle of the screen at 127,87, it becomes a bit complicated to plot mathematical graphs (for which the point where the axes cross is defined as 0,0, to allow for negative numbers below the x-axis and to the left of the y-axis).

Here's a program to plot a *sine wave*. Notice that we incorporate the program above to draw the axes.



```
5 REM PROGRAM—SINEWAVE
10 FOR X = 0 TO 255
20 PLOT X,87
30 NEXT X
40 FOR Y = 0 TO 175
50 PLOT 127,Y
60 NEXT Y
70 FOR I = -20 TO 20 STEP .03
80 PLOT I*6 + 127,87 + 20*SIN I
90 NEXT I
```

Line 70 dimensions the graph to fit the screen. We experimented until we hit on the range −20 to +20, and STEP often to place a PLOT point.

When you shape a curve for best display on the screen, you have to be careful to properly label the x and y axes, so that the values for different points on the curve are accurate.

The 6 in line 60 is just for convenience in making the curve easy to see. You need to be sure that using that number does not distort the graph. Fortunately, you can dimension the x-axis according to the period of a sine wave—the distance between two similar spots in succeeding cycles. This is 2. You need to mark the
x-axis accordingly.

The amplitude is 20, since a wave of amplitude 1 is y = SIN x. This accounts for the 20 in line 80, and is the guide for dimensioning the y axis.

The 127 and 87 in line 80 are, of course, to center the plot points on the center of the screen and of the axes as drawn.

You can, of course, also graph parabolas, straight lines, and all kinds of mathematical functions.

Leave the sine wave on the screen; we'll use it to practice saving a display on tape.

Saving Screen Displays with SCREEN$

Any time you have a screen display you want to save—and this will almost always be some kind of artistic or graphic display—you can use the SCREEN$ command.

To save the sine wave graph, you would type:

```
SAVE "Sinewave" SCREEN$
```

using the SAVE keyword on the S key, typing the name of the display—you can give it any name you like—in quotes, and then adding the function SCREEN$ (shifted K with the E cursor on the screen).

To recall the display you would use

LOAD 'Sinewave' SCREEN$

Except for the addition of SCREEN$, the process is the same as that outlined in Chapter Four for saving and loading programs.

LOADing a screen display with SCREEN$ does not erase a program in the computer.

Note that VERIFY does not work with SCREEN$.

Summary

1. Graphic symbols on the number keys (using both TRUE VIDEO and INVERSE VIDEO to double the available symbols) are placed on the 32 x 22 screen using PRINT statements.

   PRINT AT 10,16;"*"

2. PLOT places a black — or other INK color — dot in a pixel (picture element) defined by two numbers separated by a comma: 0-255 from left to right across the screen and 0-175 from bottom to top.

   PLOT 255,175

3. SCREEN$ is added to a SAVE or LOAD statement in order to store or recall a screen of information. The screen can be LOADed by itself or for use with a program.

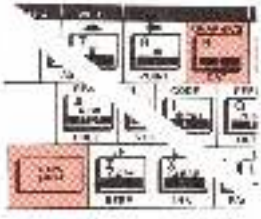
User Defined Graphics

18

Chapter Preview

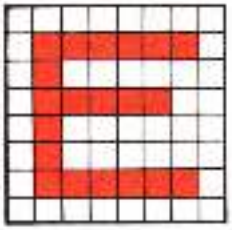
You can design your own symbols and characters with BIN, and place them in the T/S 2000's memory with POKE and USR, then recall them using the graphics mode.

Besides the graphics on the number keys 1-8, you can create your own graphics and store them "under" any of the letter keys A-U. They can then be accessed by pressing the appropriate key while in the graphics mode.



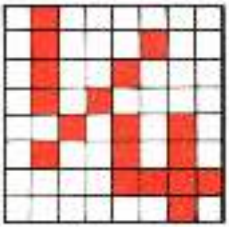
With the **K** cursor on the screen, press CAPS SHIFT 9 and obtain the **G** cursor. Press any of the number keys 1-8.

Now press any of the letter keys A-U. The result is the same as you would get in **G** mode. But you can change the characters in these memory locations.



Each character—letters, numbers, graphic symbols—is made up of *pixels* (miniature squares) in an eight by eight matrix. For instance the capital E is:

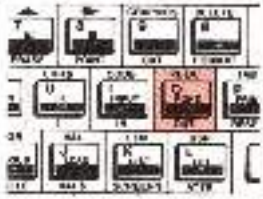
You can think of the blacked-in areas as 1's and the white areas as 0's, and the E is then represented as



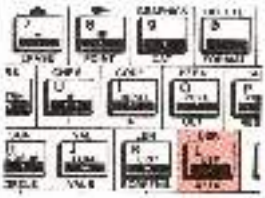
Suppose you wanted to place a character that looks like this in the computer (it is best to design it first on graph paper)

You could translate the black areas to 1's, as we did above, and represent the figure as

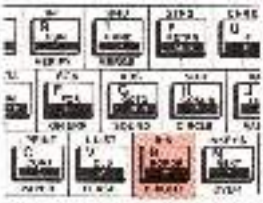
Each eight-digit (eight *bit*) line of the grid represents a specific location in memory. You can place values like this in memory using **POKE**.



POKE, the keyword on the O key, is a command that is usually followed by two decimal numbers, the first being a memory address and the second, a number to place at that address.



USR is the function above the L key: with "e", it becomes the first number in the POKE command and refers to the location where the User Defined Graphic for that key is stored.



BIN (for BINARY), the function above the B key, simply signals the computer to expect a *byte* of binary digits (*bits*) instead of a decimal number.

```
POKE USR "e", BIN 00300000
```

will store the first line of our character at the address of the first line of the user defined graphics area for the "e" key.

It takes a while to enter the entire character. The next line is entered by

```
POKE USR "e" + 1,BIN 01000100
```

Continue to enter all eight lines, using addresses up to "e" + 7, and the rows of 1's and 0's from our chart.

Then, if you press CAPS SHIFT 9 and obtain the G cursor, and press the "e" key, you'll get your symbol!

Once you've entered a user-defined graphic symbol to a letter key, it will be there any time you press that key in graphics mode, until

(a) you enter a different character, or
(b) you turn off the computer. (NEW or CLEAR will not erase it.)

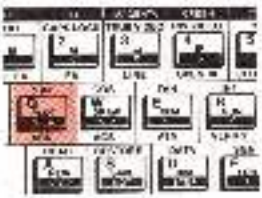
You can make it a little easier to enter graphics by typing in this program:

```
10 FOR n = 0 TO 7
20 INPUT a: POKE USR "q" + n, a
30 NEXT n
```

Then RUN the program and, each time it calls for INPUT, type in a line. For this example, use

```
BIN 10010000
```

for every one of the eight lines.



When the program stops asking for input, get the cursor on the screen and press the Q key. Can you guess what you'll get?

Design other graphics and use the program to input them. You'll have to change the "q" to some other letter each time you enter a new graphic. And you ought to keep a list of what graphic is on what key.

Since the user graphics disappear whenever you turn off the computer, you may want to try to invent a program that contains all your graphics and enters them automatically. Can you do this?

Hint: Try using READ and DATA

With such a program, it is possible to create and store alternate alphabets ("type faces"), figures to place on the screen for games, and symbols you may need that are not available in the TI 99/4A's character set, like the √ we just entered.

Summary

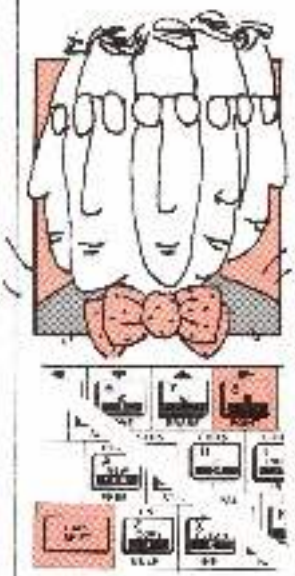
1. POKE places information in the computer's memory; it is followed by two numbers separated by a comma. The first number is the address, the second is the information.
2. BIN followed by eight binary digits—1's or 0's—creates one of eight lines needed to make up a user defined graphic symbol.
3. POKE USR "a", BIN11111111 stores a line of blocks (or, actually, INK colored dots) on the first line of eight as part of a user defined graphic on the A key. POKE USR "a" + 1, BIN00000000 stores a row of white or blank spaces on the second line.

Try out the new skills by typing in the OCTOPUSSELECT.TI program on page 167.

Time and Motion 19

Chapter Preview

This chapter covers ways to make things move, including INKEY$ and STICK. PAUSE makes everything hold still for a while.



The command INKEY$ is like INPUT except that it does not wait for you. INKEY$ scans the keyboard to see which key is being pressed—if any—and the computer then takes action according to the program.

The program below is a game of tag. Lines 20-70 place a single square in the centre of the screen (to get the INK-coloured square in line 70, press GRAPHICS and then CAPS SHIFT 0, get out of graphics mode to close the quotes), and put you in control of a chequered square (graphic 8) in the upper left-hand corner.

```
  5 REM PROGRAM — TAU
 10 BORDER 2: PAPER 2: CLS
 20 LET X = 15
 30 LET Y = 10
 40 LET A = 2
 50 LET B = 2
 60 PRINT AT A,B;"■"
 70 PRINT AT Y,X; INK INT (RND*7)+1;"■"
 80 FOR N = 1 TO 17
 90 NEXT N
100 PRINT AT Y,X; INK 6;" "
110 PRINT AT A,B; INK 0;" "
120 LET C = INT (RND*4)+1
130 IF C = 1 AND X<=29 THEN LET X = X+2
140 IF C = 2 AND X>=2 THEN LET X = X-2
150 IF C = 3 AND Y<=19 THEN LET Y = Y+2
160 IF C = 4 AND Y>=2 THEN LET Y = Y-2
170 IF INKEY$ = "O" AND B>=1 THEN LET
B = B-1
180 IF INKEY$ = "A" AND A<=20 THEN LET
A = A+1
190 IF INKEY$ = "Q" AND A>=1 THEN LET
A = A-1
200 IF INKEY$ = "P" AND B<=30 THEN LET
B = B+1
210 IF A = Y AND B = X THEN GOTO 900
220 PRINT AT Y,X; BRIGHT 1; INK INT (RND*6)
+2;"■"
230 PRINT AT A,B; INK 7;"■"
240 GOTO 80
900 FOR N = 1 TO 20: PRINT AT 0,0; PAPER 7; INK
2;"YOU WON!"
910 BORDER 6
920 PRINT AT A,B; BRIGHT 1; INK INT (RND*6)
+2; FLASH 1;"■"
930 PRINT AT A,B; INK 7; FLASH 1;"■"
940 BORDER 2
950 NEXT N
960 PAUSE 1000: STOP
```

Lines 120-160 use a random number generator to move the black square, two spaces at a time in any direction.

Lines 170-200 move the checkered square, according to your direction. You can only move one space at a time, but, presumably you are moving more purposefully.

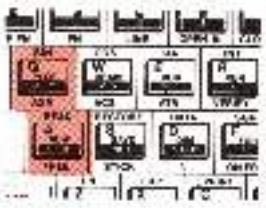
The second portion of each line from 120 to 200 keeps your square and the target square from going off the edge of the screen.

Lines 220 and 230 print the two squares in their newly-calculated locations, while lines 100 and 110 "erase" the squares from their old locations (by printing over them with an INK color—black—the same as the PAPER color of the screen).

Lines 30 and 90 introduce a short pause in the proceedings while the computer, in effect, counts to 17 very quickly. You can use an empty FOR/NEXT loop in this way.

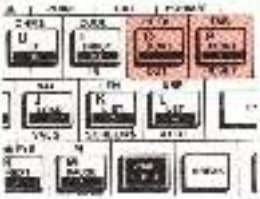
Line 210 and the celebration routine at line 500 are executed if you "bag" the black square with your checkered one, that is, if their coordinates are the same.

When you play this game, remember that INKEY$ checks the keyboard to see what key is being pressed when that program line comes by. Rather than tapping the keys, you ought to hold down the key that corresponds to the direction you wish to move to close in on the black square.



To play, you press the following keys:

Q to move up
A to move down



O to move left
P to move right

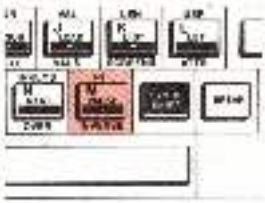
(Note that you could designate any four keys you feel comfortable with to move your square—the arrow keys 5,6,7,8 or any pattern of other keys.)

Don't press two keys at once (for instance down and left) because INKEY$ can only read one at a time.

Since the letters INKEY$ will react to in lines 170-200 are capitals, you must be in C mode — CAPS LOCK on — when you play this game.

Here's a short program to show you that INKEY$ waits for no one. If you don't type a character (press any key) very quickly, the computer prints a blank space and scrolls merrily onward and upward. Use the BREAK key when you want to stop it.

```
10 PRINT INKEY$
20 GOTO 10
```



PAUSE

The PAUSE command does just what you'd expect it to, and you can set it using a numerical value.

PAUSE 60 is about one second. More is a higher number and less is lower. Add to the above program:

```
15 PAUSE 60
```

and it will be a lot easier to keep up with the scrolling.

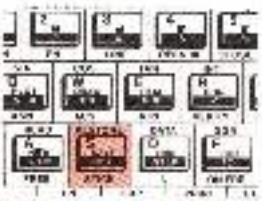
A PAUSE command will be terminated if a key is pressed, notice that the program will move as fast as you do, and not wait for a full second between entries.

Simulate a typewriter by adding a semicolon at the end of line 10.

Here is a diabolical program that changes the length of PAUSE in response to your success at the game! Your task is to "echo" the computer's output on the screen. When the computer prints a random digit between 0 and 9, you have to type that digit before the computer goes on to the next one.

The good news is that, if you miss some, the computer will slow down for you. The bad news is that if you get some right, the pace will speed up.

```
10 REM PROGRAM — ECHO
20 LET T = 60
30 LET A$ = CHR$ INT (RND*10 + CODE "0")
40 PRINT A$
50 PAUSE T
60 LET B$ = INKEY$
70 IF B$ = A$ THEN GOTO 100
80 LET T = T+1.1
90 GOTO 30
100 LET T = T*0.9
110 GOTO 30
```



STICK

The STICK command (located under the 5 key and accessed with either SHIFT while in extended mode) "reads" the position of a device connected to the ZX 2068's joystick port. It treats the input much as INKEY$ does for keyboard input.

This is most useful if you want to write your own graphic games (or other software), and generally will be used to move a cursor or other object with the IF command.

```
IF STICK (1,2) = 1 THEN LET X = X + 1
```

would move a figure on the screen upwards by one print or plot position (assuming you were using x to define the position of the character on the vertical axis).

The STICK function requires two numbers in parentheses after the word STICK. The first number specifies the "device type" you want to check—1 is the joystick itself, 2 is the pushbutton.

The second number identifies the "player" (in other words, which of two joysticks is being investigated)—1 or 2. You may think of 1 as the left one and 2 as the right, but don't get your wires crossed!

Executing this function returns a value which tells you what is going on. If you are reading the pushbutton, there are only two possible answers: you'll get a 1 if the button is being pushed *at the time the reading is being taken* and 0 if it is not.

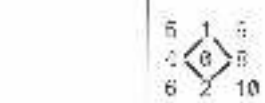
Things are a bit more complicated if you are reading the stick itself. In the example above, the "1" after the first = sign meant that the stick was in the "up" position (this is why we LET X move up the screen). The complete table of values (reading counter-clockwise) is:

0—on center (not moving)
1—up
2—up and to the left (diagonal)
4—left
6—left and down
8—down
10—down and right
8—right
9—up and right

This is not as odd as it looks; the four main directional values are organized like this:

```
   1
4  0  8
   2
```

using binary numbers, and the diagonal directions are read by combining (adding) the adjacent values (up left—5—is up—1—plus left—4).



Animation

You can make figures move about by using INKEY$.

You can also make individual user-defined graphics seem to move by switching between two slightly different designs (such as placing a figure of a man as both G and H and having the legs in slightly different positions, then making him "walk" by alternating the two characters—and moving the figure with INKEY$ or program statements).

Summary

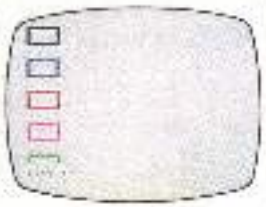
1. INKEY$ reads the keyboard and inputs as a character string any key being pressed. (If no key is being pressed at the time, it "reads" the empty or "null" string. PRINT INKEY$ will therefore like PRINT alone; if no key is being pressed, the computer will print a blank line.)
2. PAUSE causes a program to wait a specified length of time (60 = one second) before continuing to the next line. If a key is pressed, the PAUSE ends.
3. STICK "reads" the position of a joystick attached to the TS 2068 and handles the result similarly to INKEY$.

Colours available on the TS 2000, using this program.

```
10 FOR I = 0 TO 21: READ A
20 FOR J = 0 TO 31
30 PRINT INK A; AT I,J; "■"
40 NEXT J
50 NEXT I
60 DATA 1,7,2,7,3,7,4,7,5,7,6,1,6,2,6,3,6,4,6,5,6,6
```

```
10 REM PROGRAM—STARS
20 PAPER 7: BRIGHT 1: BORDER 0
30 FOR I = 0 TO 5: INK I
40 PLOT 30 + 50*I,20 + 20*I: DRAW 20,20: 300
50 NEXT I
```

Try this with values other than 300 in line 40.



Adding color to the GRAPHICS program on page 166

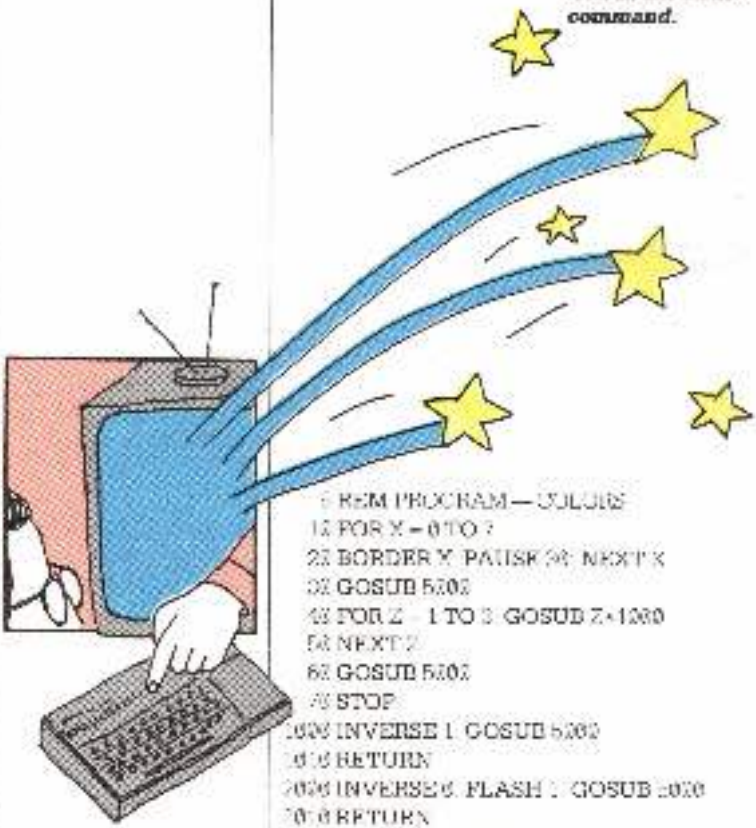
```
5 REM PROGRAM—COLORED CONCENTRIC
CIRCLES
10 FOR I = 1 TO 5
20 FOR J = 0 TO 6
30 BORDER 1: INK J: CIRCLE 127, 87, 5 + J*4
40 NEXT J
50 NEXT I
```

```
5 REM PROGRAM—COLORED STRIPES
10 FOR N = 0 TO 7
20 BORDER N: PRINT PAPER N + 1; INK 9;
" + + + + + + + + ": PAUSE 50
30 NEXT N
40 GOTO 10
```

Color 20

Chapter Preview

This chapter goes over BORDER, INK, and PAPER again, and shows you how to enhance your colors with BRIGHT, FLASH, and INVERSE. We also look at the OVER command.



```
 0 REM PROGRAM—COLORS
 1 FOR X = 0 TO 7
 2 BORDER X: PAUSE 20: NEXT X
 3 GOSUB 5000
 4 FOR Z = 1 TO 3: GOSUB Z*1000
 5 NEXT Z
 6 GOSUB 5000
 7 STOP
1000 INVERSE 1: GOSUB 5000
1010 RETURN
2000 INVERSE 0: FLASH 1: GOSUB 5000
2010 RETURN
3000 FLASH 0: BRIGHT 1: GOSUB 5000
3010 BRIGHT 0: RETURN
5000 FOR Y = 0 TO 7: PAPER Y
5010 PRINT INK 9; "TIMEX SINCLAIR 2000"
5020 PAUSE 20: NEXT Y
5030 PRINT
5040 RETURN
```

Back in Chapter Three, we introduced the color commands BORDER, PAPER and INK. Now we'll revisit them, and some TS 2000 features we haven't covered.

Type in the above program and RUN it. Let's look at it.

Line 20 simply takes the border through the available colors, with each staying on the screen for about a half second.

Line 30 directs the program to the subroutine starting at line 6000.

Line 6000 sets paper color to a different value each time through the loop.

Line 6010 supports printing in INK color 8. But there is no color above the 9 keyl (Color 9 is an instruction to choose a color that can be used for either INK or PAPER for maximum contrast. Either black or white will be selected, as appropriate.

Line 6020 waits a half second then moves to the next repeat of the loop.

After the program has completed eight printed items, line 6030 prints a blank line to separate this group of eight from successive groups. Line 6040 ends the subroutine.

Line 40 operates a loop made up of subroutine calls. They are

Line 1000 sets INVERSE 1 and then requests the above subroutine at line 5000.

Line 2000 turns INVERSE off and turns FLASH on, and recalls subroutine 5000 again.

Line 3000 turns FLASH off and BRIGHT on. Notice that, before leaving the subroutine, line 3010 turns BRIGHT off.

After line 50 completes the loop, line 50 takes us through the subroutines at 5000 one more time, and line 70 STOPs the program.

When you RUN the program, the first eight entries that appear on the screen illustrate:

1. How PAPER, on its own within a program, applies only to areas which print is done.
2. How INK 9 works. In the first four entries white INK is automatically used over a dark PAPER color; for the last four black INK goes over the light PAPER colors.

The second eight entries illustrate INVERSE. Inside the computer the INK and PAPER colors remain the same, but the dots in each 8 × 8 character are reversed (those which had been INK become PAPER and vice versa).

INVERSE 1 turns on the INVERSE function; INVERSE 0 (as in line 5000) turns it off.

The third eight entries, of which the first four appear on the first screen, illustrate the FLASH function. It, as you can see, is simply a rapid alternation of INVERSE 1 and INVERSE 0.

FLASH 1 turns on the FLASH function, and FLASH 0 turns it off.

Press Y or ENTER or any key but BREAK or N in response to the scroll? prompt, and see the rest of the program output.

First are the remainder of the FLASHing items.

The next eight entries illustrate the BRIGHT function (you may have to look closely; look especially at the white letters and background to see the difference).

BRIGHT 1 turns the function on, BRIGHT 0 turns it off.

The last eight entries repeat the original eight, for you to compare with BRIGHT.

You will in most likely to want to use BRIGHT or FLASH to call attention to something on the screen, a label or prompt.

OVER

Another function that is turned on with 1 and off with 0 is OVER. When OVER is on, you can overprint one character with another. Type this in and RUN it:

```
10 PRINT AT 5,5; "OOOOOOOOOO"
20 OVER 1: PAUSE 30
30 PRINT AT 5,5; "          "
40 OVER 0: PAUSE 30
50 PRINT AT 5,5; "**********"
```

In line 10, use capital O, not numeral zero. In line 30, use the dash, SYMBOL SHIFT J.

Line 10 PRINTs the O character, line 20 turns OVER on, and line 30 then overprints the dash to make a row of zeta thetas, more or less.

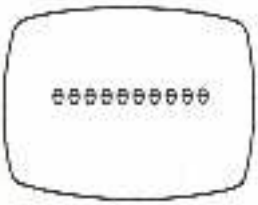
Line 40 turns OVER off, and then when Line 50 prints asterisks, they are printed in place of the previous line of characters.

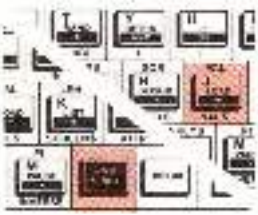
If you printed enough characters in the same place with OVER on, would you end up with a solid black square?

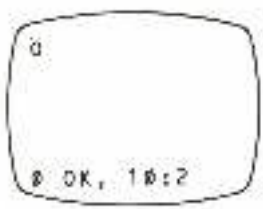
You can also use OVER with CHR$ 8, which is a backspace, to make compound figures. Make an "ü" with an umlaut this way:

```
10 OVER 1:PRINT "u";CHR$ 8;"¨"
```

(Remember, open quote, two more quotes to print one, and close quote.)







There will be more on CHR$ in Chapter 22. But for now, if you look in Appendix B you'll find that code #8 is defined as "cursor left" (or backspace). You can call for any action or character by its code, with CHR$. (Instead of PRINT "$" you could type PRINT CHR$ 36.)

Some Notes on BORDER, PAPER, and INK

BORDER can be specified either as a command line in immediate mode, or in a program line. The BORDER color selected remains until a new color is specified or the computer is turned off.

When PAPER is specified as a command line, the entire display screen is changed to the new color when ENTER is pressed twice. The color remains until changed, as with BORDER.

In a program line, PAPER takes effect only when something is printed, and underlies only the characters which are printed. Try:

```
10 PAPER 7
20 PRINT INK 7;"LOOK AT THIS"
```

press RUN and ENTER. Notice that the PAPER color is shown only under the INKed characters. The PAPER color will be extended to the whole screen after a CLS command, or after an ENTER to recall the program listing to the screen (because that, in effect, clears the screen before showing the listing).

Press ENTER again. Then press

PAPER 7 ENTER ENTER

and we are back to "normal." Add to the program

```
15 CLS
```

and see what happens when you RUN it.

INK, as a command line, changes the ink color. But you can't see it work until you PRINT something. Try (after NEW and ENTER)

```
INK 2      ENTER
```

and then

```
PRINT "LOOK AT THIS"
```

In a program line, by itself, INK will also select a color that will remain until it is changed by a subsequent line, or until the computer is turned off.

But as part of a PRINT command, either INK or PAPER will specify colors *for only that command*, after which the previous generally-specified color (perhaps the default black-on-white) returns. Remember how the INK changed back to black when you pressed ENTER a second time and turned the whole screen to PAPER color? That's because INK 7 was specified in a PRINT statement (line 20) of the little program we were using.

Summary

1. INVERSE reverses the INK and PAPER colors to print inverse characters.

   INVERSE 1 turns it on
   INVERSE 0 turns it off

2. FLASH causes characters to flash by rapidly switching between normal and inverse video.

   FLASH 1 turns it on
   FLASH 0 turns it off

3. BRIGHT makes characters brighter on the screen.

   BRIGHT 1 turns it on
   BRIGHT 0 turns it off

OVER prints a character over whatever is already on that position, not erasing the previous character.

OVER 1 turns it on
OVER 0 turns it off

Sound and Music 21

Chapter Preview

This chapter covers the SOUND command, and how to use it to write three-part harmonies.

Back in Chapter Seven, we played some music with the **BEEP** command.

Now we want to investigate the **SOUND** command. It allows you to compose music in harmony, with three channels (voices) of music at your disposal. It can also produce some interesting sound effects to add to your programs.

The **SOUND** command is followed by pairs of numbers, the pairs separated by semicolons and the individual numbers within the pairs by commas.

You can include up to 16 pairs of numbers in each **SOUND** statement. In each pair, the first designates one of fifteen registers—storage locations—within the special sound/music synthesizer chip. The second is a value to put into the register. These registers control the pitch, duration, and volume of the sound being produced.

Registers 0 and 1 control the pitch of a tone produced on Channel A. 1 is given a "coarse tune" value and 0 a "fine tune." We've produced a chart showing the values to place in each register for eight octaves' worth of notes.

Suppose we want to play an A note (in the fifth octave), through Channel A (it doesn't matter which channel you use). First, type in:

```
10 SOUND 0,124:1,0;      (DON'T ENTER YET!)
```

which puts the values, taken from our chart, of 124 into the "fine tune" register and 0 into the "coarse tune" register.

Next we need to add an amplitude, or volume. For Channel A, we use register 8 and value 13 (13 from the available range of 0-15):

```
10 SOUND 0,124:1,0:8,13;      (DON'T ENTER!)
```

And we need to enable a sound—Channel A by using register 7. For now, use value 62—we'll explain later. Your SOUND statement should now look like this:

```
10 SOUND 0,124:1,0:8,13:7,62
```

Now you can ENTER it. But don't RUN it yet!

One attribute of a note we haven't yet entered is duration. Enter the following line:

```
20 PAUSE 0:      ENTER
```

Now, run the program and play A for one second.

Note: Upon program termination and after every immediate command the TS 2000 will turn the sound chip off for you.

If you want to turn off the sound channels within programs, you will have to do as follows:

```
30 SOUND 3,0,7,63
```

Note: The first set of numbers turns the volume to zero and the second set turns off the channel irrespective of the volume setting; therefore only one set is necessary to turn off the sound.

Table of Values of Notes

Note	Freq.	Course	Fine
C	32.703	13	16
C#	34.648	12	81
D	36.709	11	183
D#	38.891	10	252
E	41.205	10	94
F	43.654	9	201
F#	46.249	9	69
G	48.999	8	194
G#	51.913	8	56
A	55	7	186
A#	58.27	7	65
B	61.735	6	235
C	65.406	6	136
C#	69.296	6	42
D	73.416	5	209
D#	77.782	5	125
E	82.406	5	47
F	87.308	4	228
F#	92.498	4	156
G	97.998	4	92
G#	103.826	4	20
A	110	3	226
A#	116.54	3	172
B	123.47	3	117
C	130.812	3	68
C#	138.592	3	21
D	146.832	2	232
D#	155.564	2	191
E	164.812	2	151
F	174.616	2	114
F#	184.996	2	79
G	195.998	2	46
G#	207.652	2	14

Note	Freq.	Coarse	Fine
A	220	1	241
A#	233.08	1	213
B	246.94	1	185
C	261.624	1	162
C#	277.184	1	138
D	293.664	1	115
D#	311.126	1	95
E	329.624	1	75
F	349.232	1	57
F#	369.992	1	39
G	391.952	1	23
G#	415.304	1	7
A	440	0	240
A#	466.16	0	224
B	493.88	0	221
C	523.248	0	209
C#	554.360	0	197
D	587.320	0	186
D#	622.256	0	175
E	659.248	0	165
F	698.464	0	156
F#	739.984	0	147
G	783.904	0	139
G#	830.608	0	131
A	880	0	124
A#	932.32	0	117
B	987.76	0	110
C	1046.496	0	104
C#	1108.736	0	98
D	1174.656	0	93
D#	1244.512	0	87
E	1318.496	0	82
F	1396.928	0	78
F#	1479.968	0	73
G	1567.808	0	69
G#	1661.216	0	65
A	1760	0	62
A#	1864.64	0	58
B	1975.52	0	55
C	2092.992	0	52
C#	2217.472	0	49
D	2349.312	0	46
D#	2489.024	0	43
E	2636.992	0	41
F	2793.856	0	39
F#	2959.936	0	36
G	3135.936	0	34
G#	3322.432	0	32

Note	Freq	Coarse	Fine
A	3520	0	31
A#	3729.28	0	29
B	3951.01	0	27
C	4185.981	0	26
C#	4434.941	0	24
D	4698.621	0	23
D#	4978.048	0	21
E	5273.984	0	20
F	5587.712	0	19
F#	5919.872	0	18
G	6271.872	0	17
G#	6644.961	0	16
A	7040	0	15
A#	7458.56	0	14
B	7902.02	0	13

There is, admittedly, a lot of footwork involved to play just one note. And there is much more if you plan to compose a symphony. But take heart: every register does not have to be turned on and off for each note.

Let's illustrate by building a chord of three notes. From NEW and ENTER. Type in the following program. We'll walk through it and then RUN it.

```
10 SOUND 7,56
20 SOUND 8,58,1,3,8,12
30 PAUSE 50
40 SOUND 9,161,3,2,9,12
50 PAUSE 50
60 SOUND 4,46,5,2,10,12
70 PAUSE 300
80 SOUND 0,0,1,0,2,0,3,0,4,0,5,0
```

Let's start our analysis with line 20 and come back to line 10 last.

Line 20 contains the fine tune and coarse tune register values for the C note. For volume we've changed the value in register 8 to 12.

Line 30 "holds the note" for one second. Actually, the note will go on until stopped, as we've seen; line 30 actually waits one second before going to line 40.

Line 40 adds a second note to the mix; looking at the chart of registers and the chart of note values, we find that we are playing an E through Channel B — and also turning the volume control up to 12.

Line 50 lets us listen to the two notes for another second.

Line 60 adds the third note of a C chord — a G — in Channel C. Line 70 lets us listen to the whole chord for five seconds, and then line 80 turns off the tone on all three channels.

Notes:

1. We don't have to turn off the other registers (volume, envelope, etc.), but can leave them engaged for the next note or chord.
2. We didn't have to use lines 50 and 60; eliminating them will play the entire chord immediately.
3. Similarly, we could use line 80 to change the notes being played instead of turning them off.

Now, to line 10: loading 56 into register 7 turns on all three tone (music) channels. (We are not using the noise register for this exercise; it can be used by itself for sound effects or mixed with one or more tone channels to change the timbre of the sound.)

If you imagine register 7 holding the value 63 when all six channels are off, and then subtracting the following numbers from that for each channel you wish turned on, you'll be able to mix it easily.

Music: Channel A	1	Noise: Channel A	8
Channel B	2	Channel B	16
Channel C	4	Channel C	32

Combinations of numbers can be subtracted to enable more than one channel, as we've seen subtracting all 63 gives 0 and enables all six channels.

The envelope: Registers 11, 12 and 13 program the envelope to control the total sound from whichever channels are enabled.

There are both a fine tune and a coarse tune register (11 and 12) for the envelope period. The available range of values for each register is 0-255.

The envelope is the overall "shape" of the sound being produced: whether it swells, leaves, until leave rate.

The period is the duration of one "cycle" of the envelope shape. The shape—register 13—is determined by a value of 0-15, as follows:

Attack: Add 4 to cause the sound to swell from zero to peak volume over the duration of one cycle. If you leave this at 0, the sound will "decay" instead: fade from peak to zero volume over one cycle.

Alternate: Add 2 and the patterns described for "attack" will alternate; that is, the sound will swell for one cycle, then decay for one cycle (or vice versa) and will neither start nor stop abruptly.

Hold: Add 1 to limit the period to one cycle of either attack or decay. Sound will then remain at peak or zero volume until a new command changes it.

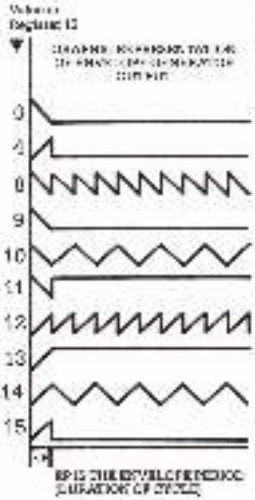
Continue: If this is left at 0, whatever is programmed by the other three parameters will last for only one cycle and volume will then drop to (or stay at) zero. Add 8 and whatever is set up by the other three parameters will repeat until terminated by another command. This may be a repeat of *Attack*, *Decay* or *Alternate* patterns, or a *Hold* as described above.

The value of 0 to 15 for the envelope shape register 13, is then assembled from various combinations of the four pattern parameters. For instance, a pattern could be made up of 8 (for *Continue*, so the note doesn't shut off immediately) plus 2 (for *Alternate*, so the volume will rise and fall somewhat gradually) plus 4 (for *Attack*, so that it rises first) — a total value of 14.

Register Chart for SOUND Command

Register	Function	Values
0	Fine tune, Channel A	0-255
1	Coarse tune, Channel A	0-15
2	Fine tune, Channel B	0-255
3	Coarse tune, Channel B	0-15
4	Fine tune, Channel C	0-255
5	Coarse tune, Channel C	0-15
6	Noise (Higher value = lower frequency)	0-31
7	Enable (Subtract from 63 to enable): Tone A = 1 B = 2 C = 4 Noise A = 8 B = 16 C = 32	0-63
8	Amplitude (volume) Channel A	0-15
9	Amplitude Channel B	0-15
10	Amplitude Channel C (Value 16 enables envelope)	0-15

Register	Function	Values
11	Fine tune Envelope period	0-255
12	Coarse tune Envelope period	0-255
13	Envelope shape (Add to zero to enable)	0-15
	Hold 1	
	Alternate 2	
	Attack 4	
	Continue 8	



Envelope Shape Diagram: The following patterns are created by loading the stated values into register 13

If the value in any channel's amplitude register is from 0 to 15, you will remove it from the control of the envelope. The channel will play its note continuously.

If you change the value to a number from 0 to 15 *plus 16*, you will let the envelope control the "shape" of the note but you will have specified a maximum volume for it to reach within that envelope. This is how you will "play" the TIS 2000 louder or softer.

If the value is exactly 16, the full range of volume will be available under the control of the envelope

You can obviously spend a lot of time learning to program the SOUND command. The best way to learn is to practice it like any musical instrument. The amplitude registers, envelope shape and period, and enable register may make it possible for you to create music with just the SOUND command (without needing, for example, PAUSE in your programs).

Here is the program we used to obtain the values for octaves and tone registers by inputting the frequency for a note. You can use it for any frequency within the synthesizer's capacity.

```
10 REM PROGRAM : SOUND
20 PRINT "NOTE" TAB 5 "FREQ." TAB 12,
"COARSE" TAB 20, "FINE"
30 INPUT N$
40 INPUT F
50 LET X = 1.75*(10**5)
60 LET X = X*1000000
70 PRINT N$;TAB 4;F;TAB 14 INT (X/256), TAB 27
INT X - INT (X/256)*256
```

The Noise Generator

The noise generator can be used with the tone channels or by itself, again the best way to get a "feel" for the possible effects is to experiment.

You can create sound effects by using registers 6-13, leaving 0-5 set to 0. Here are a few to try out, with some explanations:

```
10 ZUNSHOT
20 SOUND 6,15,7,248,16,9,15,10,16,12,15,13,0
30 PAUSE 60
40 GOTO 10
```

Register 6 (Noise) can be given a value between 0 and 31; the higher the value, the lower the frequency of the sound. Value 7 in the Enable register (7) turns on Channels A,B and C for noise only.

Registers 8,9 and 10 Amplitude for Channels A,B and C, are set for envelope control of the full range of volume. Register 11 (Fine Tune) is left at 0. 12 (Coarse Tune) is set to 16 for the envelope period, and register 13 (Envelope Shape) is set at 0, decay for one cycle.

Fire a new gunshot without waiting for the full pause to elapse by pressing any key. Stop the gunshots with break.

EXPLOSION
```
10 SOUND 6,6,7,7,8,16,9,16,10,16,11,12,56,13,8
20 PAUSE 99
30 SOUND 8,0,9,0,10,0
```

Note the similarity of many of the settings. Noise period (6) has been reset. Envelope period (12) has been increased and Envelope shape (13) has been changed.

WHISTLING BOMB
```
10 SOUND 7,67,8,15
20 FOR I = 50 TO 120
30 SOUND 6, I: PAUSE 3
40 NEXT I
```

Can you create a routine to follow the "Whistling Bomb" effect with the "Explosion" effect?

A very useful major project would be to design a program allowing you to "play" the TS 2000 via the keyboard.

Checking Up 22

Chapter Preview

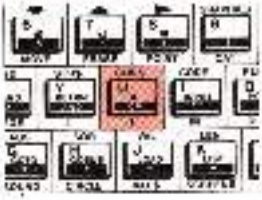
You can find out lots of things by asking the right questions. This chapter covers the FREE, POINT, ATTR, CODE, and CHR$ functions.

In this chapter, we'll explore some ways to obtain information from the TS 2000.

CHR$ and CODE

We talked earlier about the TS 2000's "extended alphabet" in which all the letters and numbers (and even keyword commands) are ranked in a 256-character listing.

Each character or keyword has a code number (a number between 0 and 255—see Appendix B for a list of them) and each of those code numbers has, of course, a character which we call a CHR$ or character *string* (it's a very short character string . . .).



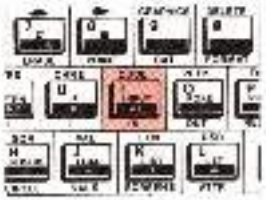
CHR$ (using the function CHR$ located above the U key) applies to a number given the single character string whose code is that number. Here's a program to find the character when you know the code:

```
10 REM PROGRAM — CHARACTERS
20 PRINT "ENTER A NUMBER FROM 32 TO 255"
30 PRINT
40 INPUT A
50 PRINT CHR$ A
60 GOTO 40
```

We used 32 to 255 in line 20, because most of the characters below 32 are tokens for which the TS 2000 has nothing it can display on the screen (see the Appendix on The Character Set). Worse, some of the color commands will stop this program if we try to include them.

Each time you input a number between 32 and 255, you are shown the corresponding character. In many cases, you'll be shown a question mark: this is what the computer will put on the screen if it does not have a symbol it can print for that number. To produce our sample output, we keyed in the codes 50, 100, 150, 200 and 250.

Notice the semi-colon at the end of line 50. Feel free to change it if you like.

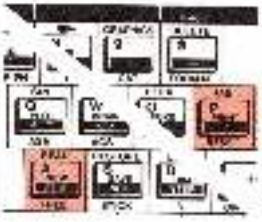


Here's one to go the other way: input a character from the keyboard, and get its code. Use the function CODE located above the I key.

```
10 REM PROGRAM — CODES
20 PRINT "PRESS ANY KEY"
30 PRINT
40 INPUT A$
50 PRINT CODE A$
60 GOTO 40
```

In the sample run, we've obtained the code for, respectively,

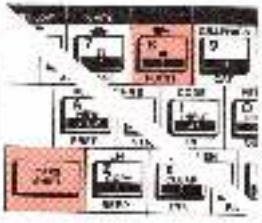
Y	Lower case Y
CAPS SHIFT Y	Capital Y
SYMBOL SHIFT Y	AND
GRAPHICS Y	Pound sign (defined graphic)
EXTENDED MODE, Y	STR$
EXTENDED MODE, SHIFT Y	Left bracket



FREE

At any time—within a program or, more often, as a command in the immediate mode—you can press PRINT FREE (extended mode, SHIFT A) and ENTER.

The computer will respond with the number of bytes of the computer's internal memory you have available to work with.



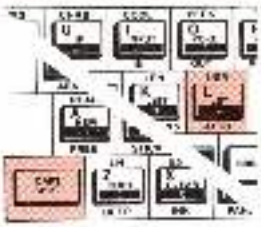
POINT

The function POINT, located under the 8 key and reached with SHIFT from the extended mode, is followed by two numbers separated by commas. The numbers correspond to the PLOT position of the point in question.

```
PRINT POINT (25,175)
```

The response from the TS 2000 will be:

1 if the pixel specified is INK color, or
0 if the pixel is PAPER color.



ATTR

The ATTRibute function, located under the I key and accessed with SHIFT while in extended mode, returns a number which encodes a number of attributes of the PRINT position specified.

```
PRINT ATTR (10,10)
```

In binary, bit 7 is 1 if the position is flashing, 0 if not. Bit 6 is 1 if bright, 0 if normal. Bits 3-5 define the PAPER color, in the same way as bits 0-2 define the INK color.

Converted to decimal, the single number can be decoded as follows:

1. If it contains (is larger than or equal to) 128, the position is flashing. If not, it isn't.
2. Subtract 128, if possible. If the number then contains 64, it is bright. If not, it is normal.
3. Subtract 64, if possible. The INK and PAPER colors can be determined from the table below (for example, if the remainder is 0, both ink and paper are black; if it is 21, the paper must be red and the ink cyan — no other combination will yield that number):

COLOR	INK	PAPER
Black	0	0
Blue	1	8
Red	2	16
Magenta	3	24
Green	4	32
Cyan	5	40
Yellow	6	48
White	7	56

Summary

1. CHR$ is applied to a number, and returns the character for which that number is the code.

```
PRINT CHR$ 220
```

2. CODE is applied to a character, and returns the code for that character.

   PRINT CODE "z

3. FREE lets you know how many bytes of memory you have available for programs or variables.

   PRINT FREE

4. POINT tells you whether the PLOT point specified by the coordinates chosen is PAPER color (if the response is 0) or INK color (if the response is 1).

   PRINT POINT (255,175)

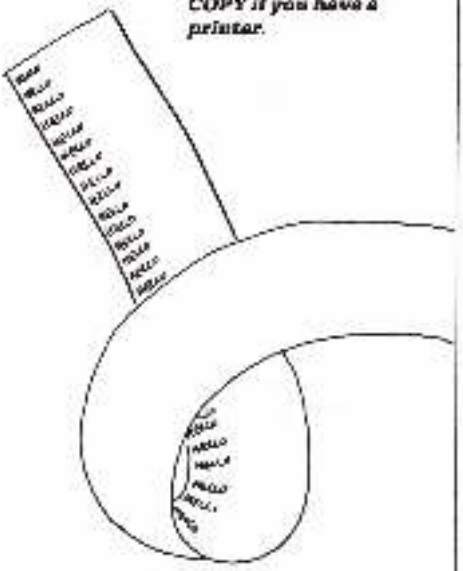
5. ATTR returns a decimal number between 0 and 255, which can be broken down to reveal the INK and PAPER color of the specified print position, and whether it is bright and/or flashing.

   PRINT ATTR (15,16)

Using the Printer 23

Chapter Preview

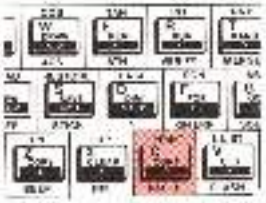
You can get programs or output on paper with LPRINT, LLIST, and COPY if you have a printer.



```
10 REM PROGRAM — PRINTER
20 LPRINT "THIS PROGRAM"
30 LLIST
40 LPRINT
50 LPRINT "PRINTS OUT THE
CHARACTER SET"
60 FOR N = 32 TO 255
70 LPRINT CHR$ N;
80 NEXT N
```

You can obtain copies of your programs, and their results, on paper, by attaching Timex Sinclair 2040, a printer, to your TS 2000. This is called a "hard copy" because it will be around for a while, as opposed to what you see on your screen.

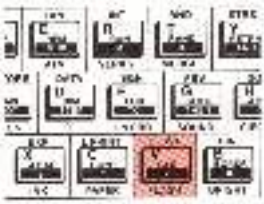
The Timex 2040 printer is an inexpensive device that attaches to the back of your 2000 and is then operated by just three simple commands.



LPRINT

LPRINT (press the O key while the E cursor is on the screen), is the same as PRINT, except that the material to be printed is sent to the printer instead of the screen.

The "L" stands for "line printer," which is what the Timex Sinclair 2040 is — it prints an entire line at a time — although the command is now used for any kind of "hard copy" printer. When BASIC was invented, the usual display was a kind of electric typewriter rather than a TV screen, so PRINT really did mean print. If you wanted a lot of output fast, a line printer would turn it out more quickly than a character-at-a-time typewriter.



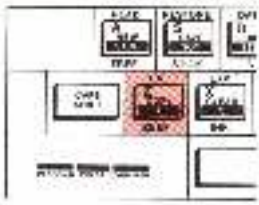
LLIST

In the same way, LLIST (press the V key while the E cursor shows) lists the program currently in the computer's memory on the printer instead of the screen.

LLIST can be used to "print a listing" without putting a program line number in front of the command. In other words, if you know you want a copy of the program the computer has in it, just press LLIST and ENTER, and you'll get it. If you add a program line number, as in LLIST 90 — the listing will be printed starting with that line.

Or, LLIST can be used within a program like the one above — now is a good time to try it out, if you haven't already. By the way, it doesn't print the entire character set, strictly defined (that runs from 0 to 255) because it cannot handle the color definition commands that reside between 0 and 31 on the printer.

(LPRINT, of course, can also be used either way. Usually, you'll use LPRINT within a program — it needs to LPRINT "SOMETHING" — while you'll more often use LLIST outside the program, just to get a copy of it to refer to.)



COPY

The third printer command is COPY (keyword on the Z key). This simply makes a copy, on the printer, of whatever is on the screen when COPY is pressed. You can use COPY as a program line or as a separate command anytime you want to copy the screen.

Try this program, which graphs some imaginary, if encouraging, sales figures:

```
10 REM PROGRAM  GRAPH
20 PRINT "FIRST QUARTER SALES"
30 PRINT
40 PRINT "JANUARY" TAB 12 "FEBRUARY" TAB
   22;"MARCH
50 LET X = 12: LET N = 3
60 FOR A  15 TO X STEP  1
70 PRINT AT A,N; "■"
80 NEXT A
90 LET X = X  3: LET N = N + 10
100 IF N> 23 THEN GOTO 120
110 GOTO 60
120 COPY
130 STOP
```

Questions:

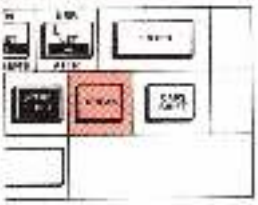
Will you get the same effect by eliminating line 120 and then pressing COPY and ENTER after the program has stopped running?

Do you see what each line in the program does?

Hint: you need GRAPHICS and SYMBOL SHIFT 8 to get the black box.

Will you get the same effect by pressing LIST and ENTER as you get by using ENTER to get the program listing on the screen and then pressing COPY and ENTER? Can you COPY a program listing?

Can you COPY the output from the bargraph program we worked with in Chapter 17?



Stopping the Printer with BREAK

When the printer is running, you can stop it with BREAK. You might want to save paper if, for example, you are using COPY to print out the five line program listing above. The command will run out all 24 lines' worth of paper, unless you press BREAK when you see line 130 appear.

If you try to execute any of the three printer commands without a printer attached, the program will usually proceed to the next line without printing anything. Sometimes, however, the computer will get hung up and you'll need to press BREAK to rescue it.

Print Format Statements with the Printer

All but one of the screen printing format commands will work with LPRINT. The comma, semicolon and TAB can be used, but AT does not work. To illustrate, try this:

```
10 REM PROGRAM — LETTERS
20 FOR N = 31 TO 0 STEP – 1
30 PRINT AT 31 – N N;CHR$ (CODE "O" + N)
40 NEXT N
```

RUN the program. You'll see a diagonal row of letters working its way down the screen, until it stops with report code 5: out of screen.

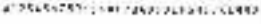
Next, change AT 31 – N N in line 30 to TAB N. RUN it again and you'll get the same effect, except that it will stop with "scroll?"

Okay, go ahead and scroll.

Be patient, we'll get to the point.

Now, change PRINT in line 30 to LPRINT. The program will RUN, and the pattern will continue for ten more lines since the printer can't be "full" as the screen can. You'll get no report code or scroll message.

Finally, change TAB N to AT 21, N. N. RUN the program one more time.



This time the printer shows a single row of characters! This is simply because AT does not send a "line feed" to the printer. It will move the print position over a column, but not down a line.

The printer will print a line:

1. After an LPRINT statement that does not end in a comma or semicolon.
2. When a comma or TAB statement requires a new line to be started.
3. At the end of a program, if there is anything "left over" to be printed.

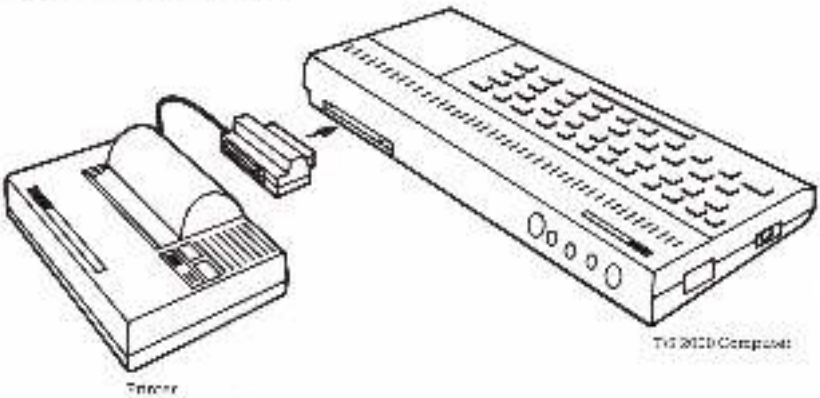
4. Any time the "buffer" is full. The buffer is the area where characters to be printed are stored until they are printed. The buffer is exactly one line (32 characters) long, so unless one of the events above (1, 2 or 3) occurs first, the printer will print a line when a full line is in the buffer (among other things, it needs to empty the buffer before new characters can be stored there).

   Remember how the printer in the first program of the chapter seemed to hesitate before printing each line of the character set? It was filling the buffer to a full line and printing those letters.

Summary

1. LPRINT prints on the printer just like PRINT prints on the screen.
2. LLIST sends a program listing to the printer just as LIST sends it to the screen.
3. COPY duplicates on the printer whatever is showing on the screen.
4. BREAK stops the printer when it is running, or interrupts printer commands which may cause a problem.
5. TAB, comma and semicolon can be used to format LPRINT statements.

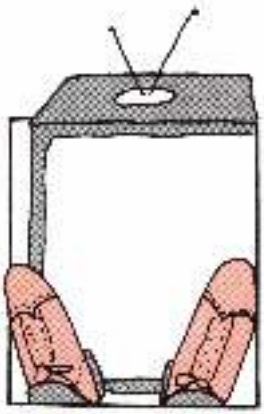
Diagram of T/S 2000 Computer Connection to Printer



Input and Output 24

Chapter Preview

A look at the commands to control input and output, including PEEK, POKE, IN, OUT, OPEN, CLOSE, FORMAT, ERASE, CAT, MOVE, and RESET.



Anything that goes into the computer is, logically enough, called "input." And, of course, anything that comes out of it is called "output."

Throughout this manual, we've been looking at ways to provide input by way of the keyboard, Timex Command Cartridges, or a cassette recorder, and to produce output to the TV screen, the printer, or the cassette recorder.

Let's look briefly at a few other aspects of input and output.

PEEK and POKE

When you type:

LET A = 150

the computer takes the number and stores it in some specific memory location. You don't know where it is, by the TS 2000 can find it—and does if you then type:

PRINT A

You can deal directly with memory locations using the PEEK and POKE commands; POKE puts any number from 0 to 255 at any location from 0 to 65535, and PEEK looks to see what number is stored in a given location. For instance:

PRINT PEEK 23070

will give you the answer 0. But then enter:

POKE 23070, 150

and then repeat the PEEK command. The 150 is now stored at address 23070.

You have to be careful where you POKE (you can PEEK anywhere safely). You won't do the computer any harm, but you can ruin any programs you have in the computer by changing one byte of information in the middle of it.

Here's a program to explore what kind of numbers are stored at various locations:

```
10 FOR I = 23530 TO 24000 STEP 10
20 PRINT PEEK I " ";
30 NEXT I
```

The space in line 20 is so that you can tell where a number ends and the next one begins. You can explore the entire set of addresses from 0 to 65535 by making changes in line 10.

You can check the program by POKEing a number into an address (best to use addresses above 24000 for this) and then having that address included in the range in line 10. Start the program with GO TO 10 instead of RUN to make sure you don't erase anything you've POKEd into the area for storing variables.

What the numbers at the various ROM addresses mean to the computer has to do with machine code and the "operating system" — the program that controls the ZX Spectrum itself — and is a subject for another time. . .

IN and OUT

IN and OUT are used to read and write to "port addresses" external to the ZX 2000's memory (this includes the keyboard as well as the ports for printers and future peripheral devices).

```
PRINT IN 49150
```

tells you what is "coming in" from that port address.

```
OUT 49150,150
```

sends "write" 150 to the device connected to that port address. Note that when you try this with the address 49150 the 150 goes "out" to "nowhere". In this case, it is because you cannot write to that address. 49150 "contains" a quantity that tells the computer if any key in the half-row from H to ENTER is being pressed.

If you are quick enough, you can have some fun with this. If you press ENTER firmly, you'll see 254, which means ENTER is being pressed. But if you can jab ENTER quickly enough so that it is not being pressed by the time the TS 2000 receives and executes the command, you'll see 255, which means no key in that half-row is being pressed. And if you are even faster, and can jab ENTER and then press another of those keys, you'll see the code for it!

(If you are interested in developing new peripheral hardware or software to use with the Timex Sinclair 2000, contact Timex Computer Corporation for port address assignments.)

Commands for Future Peripherals

A number of commands that appear on the keyboard will be used with peripheral devices yet to come. IN and OUT are among them, as are some other commands we've discussed: LOAD, SAVE, MERGE, VERIFY, INPUT and RESET are among them.

Some which will be used only with peripherals are "file manipulation" commands for use with storage devices other than cassette recorders.

FORMAT will prepare a disk or other storage medium to work with the TS 2000.

OPEN will open a file to be read or written to; CLOSE will, of course, close the file; in so doing it will make sure there is no stray information still on the way to the file.

MOVE will transfer or rename a file.

CAT — for "catalog" — will show a menu, or list, of the files available on a given storage device.

ERASE, as you might suspect, will erase a specified file from storage.

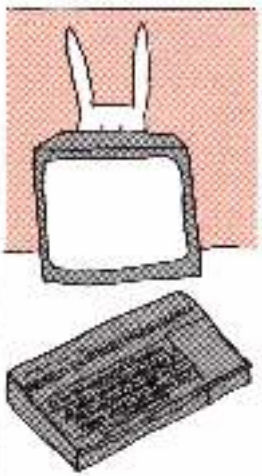
RESET, used with peripherals, will initialize or "turn on" a particular device.

Your Timex Sinclair 2068 can accept up to 2 peripherals directly connected to the rear edge (expansion) connector. For example, the TS 2040 Printer can be attached along with the TS 2050 Modem to allow you to make permanent records of data accessed from telecommunications services like The Source and CompuServe.

The cassette recorder, TV, high resolution monitor and joysticks can always be attached no matter how many other peripherals are connected.

As future Timex peripherals (including bulk storage devices and serial or parallel port interfaces for these and other products) are announced, combinations of two of the peripherals can be added to suit your growing needs.

Appendix A: Review of T/S 2000 Basic



The Keyboard

The 2000 characters comprise not only the single symbols (letters, digits etc.) but also the compound tokens (keywords, function names, etc.) and all these are entered from the keyboard rather than being spelled out. To obtain all these functions and commands some keys have two or more distinct meanings, given partly by shifting the keys (i.e. pressing either the **CAPS SHIFT** key or the **SYMBOL SHIFT** key at the same time as the required one) and partly by placing the machine in different modes.

The mode is indicated by the cursor, a flashing letter that shows where the next character from the keyboard will be inserted.

K (for keywords) mode automatically returns **L** mode when the machine is expecting a command or program line rather than INPUT data, and knows its position on the line it answers it should expect a line number or a keyword. This is at the beginning of the line or just

Appendix A:
Review of T/S 2000 BASIC

after THEN, or just after (except in a string). If unshifted, the next key will be interpreted as either a keyword written on the key(s), or a digit.

L (for letters) mode normally occurs after other times. If unshifted, the next key will be interpreted as the main symbol on that key, in lower case for letters.

In both **K** and **L** modes, SYMBOL SHIFT and a key will be interpreted as the character or a black blank on the key and CAPS SHIFT with a digit key will be interpreted as the control function written in black above the key. CAPS SHIFT with other keys does not affect the keywords in **K** mode, and in **L** mode it converts lower case to capitals.

C (for capitals) mode is a variant of **L** mode in which all letters appear as capitals. CAPS LOCK causes a change from **L** mode to **C** mode or back again.

E (for extended) mode is used for obtaining further characters, mostly tokens. It occurs after both shift keys are pressed together and lasts for one key depression only. In this mode, a letter gives one character or token (shown above it) unshifted, and another (shown below it) if pressed with either shift. A digit key gives a token if pressed with SYMBOL SHIFT, otherwise it gives a color control sequence.

G (for graphics) mode occurs after GRAPHICS (CAPS SHIFT and 9) is pressed, and lasts until it is pressed again. A digit key will give a mosaic graphic, and GRAPHICS or DELETE, and each of the letter keys apart from V, W, X, Y and Z, will give a user-defined graphic.

If any key is held down for more than a short one, will end, it will start repeating.

Keyboard input appears in the bottom half of the screen, as it is typed, each character (single symbol or compound token) being inserted just before the cursor. The cursor can be moved left with CAPS SHIFT and 5, or right with CAPS SHIFT and 8. The character before the cursor can be deleted with DELETE (CAPS SHIFT and 0). (Note, the whole line can be deleted by typing EDIT (CAPS SHIFT and 1) followed by ENTER.)

Appendix A: Review of T/S 2000 BASIC

When ENTER is pressed, the line is executed or stored into the program, entered as INPUT data as appropriate, unless it contains a syntax error. In this case a flashing ? appears next to the error.

As program lines are entered, a listing is displayed in the top half of the screen. The manner in which the listing is produced is rather complicated, and explained more fully in Chapter 2. The last line entered is called the *current line* and is indicated by the symbol >, but this can be changed by using the keys ↓ (CAPS SHIFT and 6) and ↑ (CAPS SHIFT and 7). If EDIT (CAPS SHIFT and 1) is pressed, the current line is brought down to the bottom part of the screen and can be edited.

When a command is executed or a program run, output is displayed in the top half of the screen and remains until a program line is entered, or ENTER is pressed with an empty line, or ↑ or ↓ is pressed. In the bottom part appears a report giving a code (digit or letter) referring you to Appendix B, a brief verbal summary of what Appendix B says, the number of the line containing the last statement executed (or 0 for a command) and the position of the statement within the line. The report remains on the screen until a key is pressed (and indicates K mode).

In certain circumstances, CAPS SHIFT with the BREAK key acts as a BREAK, stopping the computer with report D or L. This is recognised

(i) at the end of a statement while a program is running, or
(ii) while the computer is using the cassette recorder or printer.

The television screen

This has 24 lines, each 32 characters long, and is divided into two parts. The top part is at most 22 lines and displays either a listing or program output. When printing in the top part has reached the bottom, it all scrolls up one line, if this would involve losing a line that you have not had a chance to see yet, then the computer stops with the message scroll? Pressing the keys N, BREAK or STOP will make the program stop with report D BREAK — CONT repeats, any other key will let the scrolling continue. The bottom part is used for inputting commands, program lines, and INPUT

Appendix A:
Review of T/S 2000 BASIC

code, and also for displaying reports. The bottom part starts off as two lines (the upper one blank), but it expands to accommodate whatever is typed in. When it reaches the current print position in the top half, further expansions will make the top half scroll up.

Each character position has attributes specifying its PAPER (background) and INK (foreground) colors, a two-level brightness, and whether it flashes or not. The available colors are black, blue, red, magenta, green, cyan, yellow and white.

The edge of the screen can be set to any of the colors using the BORDER statement.

A character position is divided into 8 × 8 pixels and high resolution graphics are obtained by setting the pixels individually to show either the INK or PAPER color for that character position.

The attributes at a character position are adjusted whenever a character is written there or a pixel is plotted. The exact manner of the adjustment is determined by the *current parameters*, of which there are two sets (called permanent and *temporary*) of six: the PAPER, INK, FLASH, BRIGHT, INVERSE and OVER parameters. Permanent parameters for the top part are set up by PAPER, INK, etc., statements, and last until further notice. (Initially they are black ink on white paper, with normal brightness, no flashing, normal video and no overprinting). Permanent parameters for the bottom part are the BORDER color as the PAPER color, with a black or white contrasting INK color, normal brightness, no flashing, normal video and no overprinting.

Temporary parameters are set up by PAPER, INK, etc., items, which are embedded in PRINT, LPRINT, INPUT, PLOT, DRAW and CIRCLE statements, and also by PAPER, INK, etc., control characters when they are printed to the television. They are followed by a further byte to specify the parameter value. Temporary parameters last only to the end of the PRINT (or whatever) statement, or, in INPUT statements, until some INPUT data is needed from the keyboard, when they are replaced by the permanent parameters.

PAPER and INK parameters are in the range 0 and 9. Parameters 0 to 7 are the colors used when a character is printed:

0 black
1 blue
2 red
3 magenta
4 green
5 cyan
6 yellow
7 white

Parameter 8 ('transparent') specifies that the color on the screen is to be left unchanged when a character is printed.

Parameter 9 ('contrast') specifies that the color in question (PAPER or INK) is to be made either white or black to show up against the other color.

FLASH and BRIGHT parameters are 0, 1 or 8: 1 means that flashing or brightness is turned on, 0 that it is turned off, and 8 ('transparent') that it is left unchanged at any character position.

OVER and INVERSE parameters are 0 or 1.

OVER 0	new characters obliterate old ones
OVER 1	the bit patterns of the old and new characters are combined using an exclusive or operation (*overprinting*)
INVERSE 0	new characters are printed as INK color on PAPER color (*normal video*)
INVERSE 1	new characters are printed as PAPER color on INK color (*inverse video*)

When a TAB control character is encountered, two more bytes are expected to specify a tab stop n (less significant byte first). This is reduced modulo 32 (divide by 32 and use only the remainder) to n_2 (say), and then sufficient spaces are printed to move the printing position into column n_2.

When a comma control character is received, then sufficient spaces (at least one) are printed to move the printing position into column 9 or column 16.

When an apostrophe or ENTER control character is encountered, the printing position is moved on to the next line.

Appendix A: Review of T/S 2000 BASIC

The printer

Output to the printer is via a buffer one line (32 characters) long, and a line is sent to the printer:

(i) when printing spills over from one line to the next,
(ii) when an ENTER character is received,
(iii) at the end of the program, if there is anything left unprinted,
(iv) when a TAB control or comma control moves the printing position on to a new line.

TAB controls and comma controls output spaces in the same way as on the television.

The AT control changes the printing position using the column number and ignores the line number.

The printer is affected by INVERSE and OVER controls (and also statements) in the same way as the screen is, but not by PAPER, INK, FLASH or BRIGHT.

The printer will stop with error B if BREAK is pressed.

If the printer is absent the output will simply be lost.

The BASIC

Numbers are stored to an accuracy of 9 or 10 digits. The largest number you can get is about 10^{38} and the smallest (positive) number is about 4×10^{-39}.

A number is stored in the T/S 2000 in 'floating-point' binary with one exponent byte e ($1 <= e <= 255$), and four mantissa bytes m ($1/2 <= m < 1$). This represents the number $m \times 2^{e-128}$.

Since $1/2 <= m < 1$, the most significant bit of the mantissa m is always 1. Therefore in actual fact we can replace it with a bit to show the sign: 0 for positive numbers, 1 for negative.

Small integers have a special representation in which the last byte is 0, the second is a sign byte (0 or FFH) and the third and fourth are the integer in twos complement form, the less significant byte first.

Numeric variables have names of arbitrary length starting with a letter and continuing with letters and digits. Spaces and color controls are ignored and all letters are converted to lower case letters.

Control variables of FOR/NEXT loops have names a single letter long.

Appendix A:
Review of T/S 2000 BASIC

Numeric arrays have names a single letter long which may be the same as the name of a simple variable. They may have arbitrarily many dimensions of arbitrary size. Subscripts start at 1.

Strings are completely flexible in length. The name of a string consists of a single letter followed by $.

String arrays can have arbitrarily many dimensions of arbitrary size. The name is a single letter followed by $ and may not be the same as the name of a string. All the strings in a given array have the same fixed length, which is specified as an extra, final dimension in the DIM statement. Subscripts start at 1.

Slicing. Substrings of strings may be specified by using *slices*. A slice can be

(i) empty or
(ii) a numerical expression or
(iii) optional numerical expression TO optional numerical expression

and is used in expressing a substring either by

(a) string expression (slice)

(b) string array variable (subscript, ..., subscript, slice)

which means the same as

string array variable (subscript, ..., subscript) (slice)

In (a), suppose the string expression has the value s$.

If the slice is empty, the result is s$ considered as a substring of itself.

If the slice is a numerical expression with value n, the result is the nth character of s$ (a substring of length 1).

If the slice has the form (iii), then suppose the first numerical expression has the value m (the default value is 1), and the second, n (the default value is the length of s$).

If 1 <= m <= n <= the length of s$, then the result is the substring of s$ starting with the mth character and ending with the nth.

If 0 <= n < m then the result is the empty string.

Otherwise, error 3 results.

Slicing is performed before functions or operations are evaluated, unless brackets dictate otherwise.

Appendix A: Review of T/S 2000 BASIC

Substrings can be assigned to (see LET).

If a string quote is to be written in a string literal, then it must be doubled.

Functions

The argument of a function does not need brackets if it is a constant or a (possibly subscripted or sliced) variable.

Function	Type of argument (x)	Result
ABS	number	Absolute magnitude.
ACS	number	Arccosine in radians. Error A if x not in the range −1 to +1.
AND	binary operation, right operand always a number	
	Numeric left operand	A AND B = $\begin{cases} A \text{ if } B<>0 \\ 0 \text{ if } B=0 \end{cases}$
	String left operand	A$ AND B = $\begin{cases} A\$ \text{ if } B<>0 \\ "" \text{ if } B=0 \end{cases}$
ASN	number	Arcsine in radians. Error A if x not in the range −1 to +1.
ATN	number	Arctangent in radians.
ATTR	two arguments, x and y, both numbers, enclosed in brackets	A number whose binary bits codes the attributes of the x, column y on the television. Bit 7 (most significant) is 1 for flashing, 0 for not flashing. Bit 6 is 1 for bright, 0 for normal. Bits 5 to 3 are the paper color. Bits 2 to 0 are the ink color.

Appendix A: Review of T/S 2000 BASIC

Function	Type of argument	Result
		Error B unless 0<= x <= 23 and 0<= y<= 31
BIN		This is not really a function, but an alternative notation for numbers: BIN followed by a sequence of 0s and 1s is the number with such a representation in binary.
CHR$	number	The character whose code is x, rounded to the nearest integer.
CODE	string	The code of the first character in x (or 0 if x is the empty string).
COS	number (in radians)	Cosine of x.
EXP	number	e^x
FN		FN followed by a letter looks up a user-defined function (see DEF). The arguments must be enclosed in parentheses; even if there are no arguments, the parentheses must still be present.
FREE	none	Returns number of bytes available for BASIC programs and variables.
IN	number	The result of inputting a processor level from port x (0<= x <= FFFFh) (loads the bc register pair with x and does the assembly language instruction IN a,(c)).
INKEY$	none	Reads the keyboard. The result is the character representing (in L or C mode) the key pressed if there is exactly one; else the empty string.
INT	number	Integer part (always rounds down).
LEN	string	Length.
LN	number	Natural logarithm (to base e). Error A if x<= 0.
NOT	number	0 if x<> 0, 1 if x = 0. NOT has priority 4.

Appendix A: Review of T/S 2000 BASIC

Function	Type of argument	Result
OR	binary operator, both operands numbers	a OR b = $\begin{cases} 1 \text{ if } b<>0 \\ a \text{ if } b=0 \end{cases}$ OR has priority 2.
PEEK	number	The value of the byte in memory whose address is x (rounded to the nearest integer). Error B if x is not in the range 0 to 65535.
PI	none	π (3.14159265 ...)
POINT	two arguments x and y, both numbers, enclosed in brackets	1 if the pixel at (x,y) is ink color, 0 if it is paper color. Error B unless 0<= x <= 255 and 0<= y<= 175.
RND	none	The next pseudorandom number in a sequence generated by taking the powers of 75 modulo 65537, subtracting 1 and dividing by 65536. 0<=y<1.
SCREEN$	Two arguments, x and y, both numbers enclosed in brackets	The character that appears, either normally or inverted, on the television at line x, column y. Given the empty string if the character is not recognized. Error B unless 0<= x <= 23 and 0<= y<= 31.
SGN	number	Signum: the sign (−1 for negative, 0 for zero or +1 for positive) of x.
SIN	number (in radians)	Sine x.
SQR	number	Square root. Error A if x<0

Appendix A: Review of T/S 2000 BASIC

Function	Type of argument	Result
STICK	two arguments x and y, both numbers enclosed in parentheses	Returns number derived from reading input from device attached to joystick port. x = 1 is joystick, x = 2 is button; y = 1 is left device, y = 2 is right. See Chapter 19.
STR$	number	The string of characters that would be displayed if x were printed.
TAN	number (in radians)	Tangent.
USR	number	Calls the machine code subroutine whose starting address is x. On return, the result is the contents of the hl register pair.
USR	string	The address of the bit pattern for the user-defined graphic corresponding to x. Error A if x is not a single letter between a and u, or a user-defined graphic.
VAL	string	Evaluates x (without its bounding quotes) as a numerical expression. Error C if x contains a syntax error, or gives a string value. Other errors possible, depending on the expression.
VAL$	string	Evaluates x (without its bounding quotes) as a string expression. Error C if x contains a syntax error or gives a numeric value. Other errors possible, as for VAL.
—	number	Negation.

Appendix A: Review of T/S 2000 BASIC

Function	Result
The following are binary operators:	
+	Addition (on numbers), or concatenation (on strings)
−	Subtraction
*	Multiplication
/	Division
↑	Raising to a power. Error if the left operand is negative
=	equals
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
Both operands must be of the same type. The result is a number 1 if the comparison holds and 0 if it does not.	
Functions and operations have the following priorities:	
Operation	Priority
Subscripting and slicing	12
All functions except NOT and unary minus	11
↑	10
Unary minus (i.e. minus preceded by a right-hand operand)	9
*, /	8
+, − (minus used to subtract one number from another)	6
>, <, <=, >=, =, <>	5
NOT	4
AND	3
OR	2

Statements

In this list,

a	represents a single letter
v	represents a variable
x,y,z	represent numerical expressions
m,n	represent numerical expressions that are rounded to the nearest integer
e	represents an expression
t	represents a string-valued expression
s	represents a sequence of statements separated by colons :

Appendix A:
Review of T/S 2000 BASIC

Function	Result
	c represents a sequence of color items, each terminated by commas or semi-colons. A color item can be one of a PAPER, INK, FLASH, BRIGHT, INVERSE or OVER statement.
	Note that arbitrary expressions are allowed everywhere (except for the line number at the beginning of a statement).
	All statements except INPUT, DEF, and DATA can be used either as commands or in programs (although they may be more sensible in one than the other). A command or program line can have several statements, separated by colons (:). There is no restriction on where or how often a line any particular statement can occur — although GOTO and REM.
BEEP x, y	Sounds a note through the loudspeaker for x seconds at a pitch y semitones above middle C (or below if y is negative).
BORDER n	Sets the color of the border of the screen and also the paper color for the lower part of the screen. Error K if n is not in the range 0 to 7.
BRIGHT	Sets brightness of characters subsequently printed. 0 = normal, 1 for bright, 8 for transparent. Error K if n is not 0, 1 or 8.
CAT	For use with peripherals.
CIRCLE x, y, z	Draws an arc of a circle: center (x,y), radius z.
CLEAR	Deletes all variables, freeing the space they occupied. Does RESTORE and CLS, resets the PLOT position to the bottom left hand corner and clears the GO SUB stack.
CLEAR n	Like CLEAR, but also changes the system variable RAMTOP to n and puts the new GO SUB stack there.
CLOSE #	For use with peripherals.

Appendix A:
Review of T/S 2000 BASIC

Function	Result
CLS	(Clear Screen). Clears the display file.
CONTINUE	Continues the program (starting where it left off last time it stopped with report other than 0. If the report was 0 or 1, then continues with the following statement (taking jumps into account); otherwise repeats the one where the error occurred. If the last report was in a command line then CONTINUE will attempt to continue the command line and will either go into a loop if the error was in 0:1, give report 0:2 if it was in 0:2, or give error N if it was 0:3 or greater. CONTINUE appears as CONT on the keyboard.
COPY	Sends a copy of the top 22 lines of display to the printer, if attached; otherwise does nothing. Note that COPY can not be used to print the automatic listings that appear on the screen. Report D if BREAK pressed.
DATA e_1, e_2, e_3	Part of the DATA list. Must be in a program.
DEF FN $a(a_1, \ldots, a_k) = e$	User-defined function definition; must be in a program. Each of a and a_1 to a_k is either a single letter or a single letter followed by '$' (for string argument or result). Takes the form DEF FN a() = e if no arguments.
DELETE x, y	Deletes program lines x through y.
DELETE x,	Deletes from program line x through end of program.
DELETE ,y	Deletes from beginning of program through line y.
DIM $a(n_1, \ldots, n_k)$	Deletes any array with the name a and sets up an array a of numbers with k dimensions $n_1, \ldots, n_k$. Initializes all the values to 0.
DIM $a\$(n_1, \ldots, n_k)$	Deletes any array or string with the name $a\$$ and sets up an array of characters with k dimensions $n_1, \ldots, n_k$. Initializes all the values to " ". This can be considered as an array of strings of fixed length n_k with k − 1 dimensions $n_1, \ldots, n_{k-1}$.

Appendix A: Review of T/S 2000 BASIC

Function	Result
	Error 4 occurs if there is no room to fit the array in. An array is undefined until it is dimensioned in a DIM statement.
DRAW x, y	DRAW x, y, 0.
DRAW x, y, z	Draws a line from the current plot position, moving x horizontally and y vertically relative to it while turning through an angle z. Error 8 if it runs off the screen.
ERASE	For use with peripherals.
FLASH	Defines whether characters will be flashing or steady. n = 0 for steady, n = 1 for flash, n = 8 for no change.
FOR *a* = x **TO** y	FOR *a* = x TO y STEP 1
FOR *a* = x **TO** y **STEP** z	Defines any simple variable *a* and sets up a control variable with value x, limit y, step z, and looping address referring to the statement after the FOR statement. Checks if the initial value is greater (if step > = 0) or less (if step <0) than the limit, and if so then skips to statement NEXT*a*, giving error 1 if there is none. See NEXT. Error 4 occurs if there is no room for the control variable.
FORMAT i	For use with peripherals.
FREE	Returns number of bytes of RAM available for BASIC programs and variables.
GOSUB n	Pushes the line number of the GOSUB statement onto a stack, then as GO TO n. Error 4 can occur if there are not enough RETURNs.
GO TO n	Jumps to line n (or, if there is none, the first line after that).
IF x **THEN** s	If x true (non-zero) then s is executed. Note that s comprises all the statements up to the end of the line. The form 'IF x THEN line number' is not allowed.

Appendix A: Review of T/S 2000 BASIC

Function	Result
INK n	Sets the ink (foreground) colour of characters subsequently printed. n is in the range 0 to 7 for a colour, n = 8 for transparent or 9 for contrast. Error 7 if n not in the range 0 to 9.
INPUT	The '...' is a sequence of INPUT items (separated on input by commas, semicolons or apostrophes). An INPUT item can be (i) Any PRINT item not beginning with a bracket (ii) A variable name, or (iii) LINE, then a string type variable name The PRINT items and separators in (i) are treated exactly as in PRINT, except that everything is printed in the lower part of the screen. For (ii) the computer stops and waits for input of an expression from the keyboard. The value of this is assigned to the variable. The input is echoed in the usual way and syntax errors give the flashing ?. For string type expressions, the input buffer is initialized to contain two string quotes (which can be erased if necessary). If the first character in the input is STOP, the program stops with error H. (iii) is like (ii) except that the input is treated as a string item without quotes, and the STOP mechanism doesn't work. to stop if you must type ➡ instead.
INVERSE n	Controls inversion of characters subsequently printed. If n = 0, characters are printed in normal video, ink color on paper colour. If n = 1, characters are printed in inverse video, paper colour on ink color. Error K if n not 0 or 1.
LET v = e	Assigns the value of e to the variable v. LET cannot be omitted. A simple variable is undeclared until it is assigned to in a LET, READ or INPUT statement. If v is a subscripted string variable or a sliced string variable (substring), then the assignment is: Truncated, if need be, if the string value of e is either truncated or filled out with spaces on the right, to make it the same length as the variable v.

Appendix A:
Review of T/S 2000 BASIC

Function	Result
LIST	LIST 0
LIST n	Lists the program to the upper part of the screen, starting at the first line whose number is at least n, and makes n the current line.
LLIST	LLIST 0
LLIST n	Like **LIST** n, but using the printer.
LOAD f	Loads program and variables.
LOAD f **DATA** ( )	Loads a numeric array.
LOAD f **DATA** $( )$	Loads character array.
LOAD f **CODE** m,n	Loads at most n bytes, starting at address m.
LOAD f **CODE** m	Loads bytes starting at address m.
LOAD f **CODE**	Loads bytes back to the address they were saved from.
LOAD f **SCREEN$**	LOAD f CODE 16384,6912. Searches for file of the right sort on cassette tape and loads it, deleting previous versions in memory. See Chapter 22.
LPRINT	Like PRINT but using the printer.
MERGE f	Like LOAD f, but does not delete old program lines and variables except in such way as new ones with the same line number or name.
MOVE f_1 f_2	For use with peripherals.
NEW	Starts the BASIC system all anew, deleting program and variables, and using the memory up to and including the byte whose address is in the system variable RAMTOP, and preserves the system variables UDG, P RAMT, RASP and PIP.

Appendix A: Review of T/S 2000 BASIC

Function	Result
NEXT *a*	(i) Finds the control variable *a* (ii) Adds its step to its value (iii) If the step> = 0 and the value> the limit; or if the step<0 and the value< the limit, then jumps to the looping statement. Error 2 if there is no variable *a*. Error 1 if there is one, but it's not *a* control variable.
ON ERR GOTO line number **ON ERR CONT** **ON ERR RESET**	These statements allow the programmer to disable automatic program termination upon encountering an error condition. The ON ERR GOTO line number allows the programmer to cause the transfer to the specified line number to handle the encountered error. The error number and line number on which it occurred are available by PEEKing the locations (23736) and (23738). The statement number within the line that caused the error is stored in location (23739). The ON ERR CONT statement causes the program to resume execution at the statement in which the error originally occurred. If an ON ERR CONT statement is encountered and an error has not occurred, then the command is ignored. The ON ERR RESET command disables this feature causing the program to report errors and terminate in the usual manner.
OPEN *#*	For use with peripherals.
OUT *m,n*	Outputs byte *n* to port *m* at the processor level. (Loads the bc register pair with *m*, the *n* register with *n*, and does the assembly language instruction: out (c),*n*.) 0<= m <= 65535 255<= n <= 255, else error 5
OVER *n*	Controls overprinting for characters subsequently printed. If *n* = 0, characters obliterate previous characters at that position. If *n* = 1, then new characters are mixed in with old characters to give ink color wherever either (but not both) had ink color, and paper color if they were both paper or both ink color. Error 5 if *n* not 0 or 1.

Appendix A: Review of T/S 2000 BASIC

Function	Result
PAPER n	Like INK, but controlling the paper (background) color
PAUSE n	Stops computing and displays the the display file for n frames (at 60 frames per second) until a key is pressed. 0<= n<= 65535, else error B If n = 0 then the pause is not timed, but lasts until a key is pressed
PLOT x;y,n	Prints an ink point (subject to OVER and INVERSE) at the pixel (x, y). (x, y) moves the PLOT position. Unless the color items specify otherwise, the ink color at the character position containing the pixel is changed to the current permanent ink color, and the other (paper, color, flashing and brightness) are left unchanged 0<= |x| <= 255, 0<= |y| <= 175, else error B.
POKE m,n	Writes the value n to the byte in store with address m. 0<= m<= 65535, -255<= n<= 255, else error B.
PRINT	The list is a sequence of PRINT items, separated by commas , semicolons ; or apostrophes ' and they are written to the display file for output to the television. A semicolon ; between two items has no effect. It is used purely to separate the items. A comma , outputs the comma control character, and an apostrophe outputs the ENTER character. At the end of the PRINT statement, if it does not end in a semicolon, or comma, or apostrophe, an ENTER character is output. A PRINT item can be (i) empty, i.e. nothing (ii) a numerical expression First a minus sign is printed if the value is negative. Now let x be the modulus of value. If x<= 10^-5 or x>= 10^13, then x is printed using scientific notation. The mantissa part has up to eight digits (with no trailing zeros), and the deci-

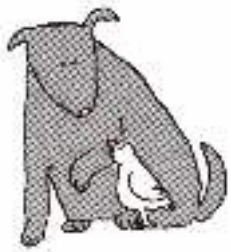
Function	Result
	real point (except if only one digit is after the E). The exponent part is E, followed by + or − followed by one or two digits. Otherwise x is printed in ordinary decimal notation with up to eight significant digits, and no trailing zeros after the decimal point. A decimal point at the beginning is always followed by a zero, so for instance .03 and 0.3 are printed as such. 0 is printed as a single digit 0.
	(iii) a string expression The tokens in the string are expanded, possibly with a space before or after. Control characters have their control effect. Unrecognised characters print as ?.
	(iv) AT m,n Outputs an AT control character followed by a byte for m (the line number) and a byte for n (the column number).
	(v) TAB n Outputs a TAB control character followed by two bytes for n (one significant byte last). Like TAB in PRINT.
	(vi) A color item, which takes the place of a PAPER, INK, FLASH, BRIGHT, INVERSE or OVER statement.
RANDOMIZE	RANDOMIZE 0
RANDOMIZE n	Sets the system variable (called SEED) used to generate the next value of RND. If n < > 0, SEED is given the value n, if n = 0 then it is given the value of another system variable (called FRAMES) that counts the frames so far displayed on the television, and so should be fairly random. RANDOMIZE appears as RAND on the keyboard. Error B occurs if n is not in the range 0 to 65535.

Appendix A:
Review of T/S 2000 BASIC

Function	Result
READ $v_1, v_2, \ldots, v_n$	Assigns to the variables using successive expressions in the DATA list. Error C if an expression is the wrong type. Error E if there are variables left to be read when the DATA list is exhausted.
REM . . .	No effect. . . . can be any sequence of characters except ENTER. This can include :, so no statements are possible after the REM statement on the same line.
RESET	For use with peripherals.
RESTORE	RESTORE 0
RESTORE n	Restores the DATA pointer to the first DATA statement in a line with number at least n; the next READ statement will start reading there.
RETURN	Takes a reference to a statement off the GOSUB stack, and jumps to the line after it. Error 7 occurs when there is no statement reference on the stack. There is some mistake in your program; GOSUBs are not properly balanced by RETURNs.
RUN	RUN 0
RUN n	CLEAR, and then GOTO n.
SAVE "	Saves the program and variables.
SAVE " **LINE** n	Saves the program and variables so that if they are loaded there is an automatic jump to line n.
SAVE : **DATA** ( )	Saves the numeric array.
SAVE : **DATA** S( )	Saves the character array S.
SAVE : **CODE** m,n	Saves n bytes starting at address m.

Function	Result
SAVE (SCREEN$	SAVE (CODE 16384,6912. Saves information on cassette giving it the name " Error F if file empty or has length eleven or more. See Chapter 20.
SOUND x,y x,y... x,y	Controls 3-channel sound synthesizer, where x is any of up to 15 registers, and y is a value placed in the register. See Chapter 21.
STICK	Returns number served from reading input from device attached to joystick port. See Chapter 19.
STOP	Stops the program with report 9. CONTINUE will resume with the following statement.
VERIFY	The same as LOAD except that the data is not loaded into RAM, but compared against what is already there Error R if one of the comparisons shows different bytes

Appendix B: The Character Set



This is the complete TS 2068 character set, with codes in decimal and hex. If one imagines the codes as being Z80 machine code instructions, then the right hand columns give the corresponding assembly language mnemonics. As you are probably aware if you understand these things, certain Z80 instructions are compounds starting with CBh or EDh. The two right hand columns give these.

Code	Character	Hex	Z80 assembler	-after CB	-after ED
0	Not used	00	nop	rlc b	
1	Not used	01	ld bc,NN	rlc c	
2	Not used	02	ld (bc),a	rlc d	
3	Not used	03	inc bc	rlc e	
4	Not used	04	inc b	rlc h	
5	Not used	05	dec b	rlc l	
6	PRINT comma	06	ld b,N	rlc (hl)	
7	EDIT	07	rlca	rlc a	

Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
8	Cursor Left	08	ex af,af'	rrc b	
9	Cursor Right	09	add hl,bc	rrc c	
10	Cursor Down	0A	ld a,(bc)	rrc d	
11	Cursor Up	0B	dec bc	rrc e	
12	DELETE	0C	inc c	rrc h	
13	ENTER	0D	dec c	rrc l	
14	Number (shug)	0E	ld c,N	rrc (hl)	
15	Not used	0F	rrca	rrc a	
16	INK Control	10	djnz DIS	rl b	
17	PAPER Control	11	ld de,NN	rl c	
18	FLASH control	12	ld (de),a	rl d	
19	BRIGHT Control	13	inc de	rl e	
20	INVERSE Control	14	inc d	rl h	
21	OVER Control	15	dec d	rl l	
22	AT Control	16	ld d,N	rl (hl)	
23	TAB Control	17	rla	rl a	
24	Not used	18	jr DIS	rr b	
25	Not used	19	add hl,de	rr c	
26	Not used	1A	ld a,(de)	rr d	
27	Not used	1B	dec de	rr e	
28	Not used	1C	inc e	rr h	
29	Not used	1D	dec e	rr l	
30	Not used	1E	ld e,N	rr (hl)	
31	Not used	1F	rra	rr a	
32	Space	20	jr nz,DIS	sla b	
33	!	21	ld hl,NN	sla c	
34	"	22	ld (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla l	
38	&	26	ld h,N	sla (hl)	
39	'	27	daa	sla a	
40	(	28	jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	-	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	0	30	jr nc,DIS		
49	1	31	ld sp,NN		

Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
50	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b,(c)
65	A	41	ld b,c	bit 0,c	out(c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc
67	C	43	ld b,e	bit 0,e	ld (NN),bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b,l	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b,a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out(c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2,b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out(c),d
82	R	52	ld d,d	bit 2,d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN),de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out(c),e
90	Z	5A	ld e,d	bit 3,d	adc hl,de
91	[	5B	ld e,e	bit 3,e	ld de,(NN)
92	/	5C	ld e,b	bit 3,h	
93	]	5D	ld e,l	bit 3,l	

Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
94	↑	5F	ld e,(ix+d)	bit 3,(hl)	ldi
95		5F	ld e,a	bit 3,a	ld a,r
96	£	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out(c),h
98	b	62	ld h,d	bit 4,d	sbc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
100	d	64	ld h,h	bit 4,h	
101	e	65	ld h,l	bit 4,l	
102	f	66	ld h,(hl)	bit 4,(hl)	
103	g	67	ld h,a	bit 4,a	rrd
104	h	68	ld l,b	bit 5,b	in l,(c)
105	i	69	ld l,c	bit 5,c	out(c),l
106	j	6A	ld l,d	bit 5,d	adc hl,hl
107	k	6B	ld l,e	bit 5,e	ld hl,(NN)
108	l	6C	ld l,h	bit 5,h	
109	m	6D	ld l,l	bit 5,l	
110	n	6E	ld l,(hl)	bit 5,(hl)	
111	o	6F	ld l,a	bit 5,a	rld
112	p	70	ld (hl),b	bit 6,b	in f,(c)
113	q	71	ld (hl),c	bit 6,c	
114	r	72	ld (hl),d	bit 6,d	sbc hl,sp
115	s	73	ld (hl),e	bit 6,e	ld (NN),sp
116	t	74	ld (hl),h	bit 6,h	
117	u	75	ld (hl),l	bit 6,l	
118	v	76	halt	bit 6,(hl)	
119	w	77	ld (hl),a	bit 6,a	
120	x	78	ld a,b	bit 7,b	in a,(c)
121	y	79	ld a,c	bit 7,c	out(c),a
122	z	7A	ld a,d	bit 7,d	adc hl,sp
123	{ (ONERR)	7B	ld a,e	bit 7,e	ld sp,(NN)
124	STICK	7C	ld a,h	bit 7,h	
125	} (SOUND)	7D	ld a,l	bit 7,l	
126	FREE	7E	ld a,(hl)	bit 7,(hl)	
127	© (RESET)	7F	ld a,a	bit 7,a	
128		80	add a,b	res 0,b	
129	▗	81	add a,c	res 0,c	
130	▖	82	add a,d	res 0,d	
131	▄	83	add a,e	res 0,e	
132	▝	84	add a,h	res 0,h	
133	▐	85	add a,l	res 0,l	
134	▞	86	add a,(hl)	res 0,(hl)	
135	▟	87	add a,a	res 0,a	
136	▘	88	adc a,b	res 1,b	
137	▚	89	adc a,c	res 1,c	

Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
138	▙	8A	adc a,d	res 1,d	
139	▜	8B	adc a,e	res 1,e	
140	▟	8C	adc a,h	res 1,h	
141	▛	8D	adc a,l	res 1,l	
142	▄	8E	adc a,(hl)	res 1,(hl)	
143	■	8F	adc a,a	res 1,a	
144	(a)	90	sub b	res 2,b	
145	(b)	91	sub c	res 2,c	
146	(c)	92	sub d	res 2,d	
147	(d)	93	sub e	res 2,e	
148	(e)	94	sub h	res 2,h	
149	(f)	95	sub l	res 2,l	
150	(g)	96	sub (hl)	res 2,(hl)	
151	(h)	97	sub a	res 2,a	
152	(i)	98	sbc a,b	res 3,b	
153	(j) user	99	sbc a,c	res 3,c	
154	(k) graphics	9A	sbc a,d	res 3,d	
155	(l)	9B	sbc a,e	res 3,e	
156	(m)	9C	sbc a,h	res 3,h	
157	(n)	9D	sbc a,l	res 3,l	
158	(o)	9E	sbc a,(hl)	res 3,(hl)	
159	(p)	9F	sbc a,a	res 3,a	
160	(q)	A0	and b	res 4,b	ldi
161	(r)	A1	and c	res 4,c	cpi
162	(s)	A2	and d	res 4,d	ini
163	(t)	A3	and e	res 4,e	outi
164	(u)	A4	and h	res 4,h	
165	HND	A5	and l	res 4,l	
166	INKEY$	A6	and (hl)	res 4,(hl)	
167	PI	A7	and a	res 4,a	
168	FN	A8	xor b	res 5,b	ldd
169	POINT	A9	xor c	res 5,c	cpd
170	SCREEN$	AA	xor d	res 5,d	ind
171	ATTR	AB	xor e	res 5,e	outd
172	AT	AC	xor h	res 5,h	
173	TAB	AD	xor l	res 5,l	
174	VAL$	AE	xor (hl)	res 5,(hl)	
175	CODE	AF	xor a	res 5,a	
176	VAL	B0	or b	res 6,b	ldir
177	LEN	B1	or c	res 6,c	cpir
178	SIN	B2	or d	res 6,d	inir
179	COS	B3	or e	res 6,e	otir
180	TAN	B4	or h	res 6,h	
181	ASN	B5	or l	res 6,l	

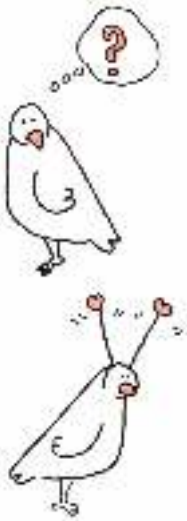
Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
182	ACS	D6	out (hl)	res 6,(hl)	
183	ATN	D7	out a	res 6,a	
184	LN	D8	ret p	res 7,b	ldir
185	EXP	D9	exx	res 7,c	cpdr
186	INT	DA	jp p,d	res 7,d	ind
187	SQR	DB	in a,e	res 7,e	outd
188	SGN	DC	call p,a	res 7,a	
189	ABS	DD	prefixes ix	res 7,l	
190	PEEK	DE	sbc a,(hl)	res 7,(hl)	
191	IN	DF	rst 24	res 7,a	
192	USR	E0	ret po	set 0,b	
193	STR$	E1	pop hl	set 0,c	
194	CHR$	E2	jp po,NN	set 0,d	
195	NOT	E3	ex (sp),hl	set 0,e	
196	BIN	E4	call po,NN	set 0,h	
197	OR	E5	push hl	set 0,l	
198	AND	E6	add a,N	set 0,(hl)	
199	<=	E7	rst 32	set 0,a	
200	>=	E8	ret z	set 1,b	
201	<>	E9	jp (hl)	set 1,c	
202	LINE	EA	jp z,NN	set 1,d	
203	THEN	EB	ex de,hl	set 1,e	
204	TO	EC	call z,NN	set 1,h	
205	STEP	ED	prefixes misc	set 1,l	
206	DEF FN	EE	xor N	set 1,(hl)	
207	CAT	EF	rst 40	set 1,a	
208	FORMAT	F0	ret p	set 2,b	
209	MOVE	F1	pop af	set 2,c	
210	ERASE	F2	jp p,NN	set 2,d	
211	OPEN#	F3	di	set 2,e	
212	CLOSE#	F4	call p,NN	set 2,h	
213	MERGE	F5	push af	set 2,l	
214	VERIFY	F6	or N	set 2,(hl)	
215	BEEP	F7	rst 48	set 2,a	
216	CIRCLE	F8	ret m	set 3,b	
217	INK	F9	ld sp,hl	set 3,c	
218	PAPER	FA	jp m,NN	set 3,d	
219	FLASH	FB	ei	set 3,e	
220	BRIGHT	FC	call m,NN	set 3,h	
221	INVERSE	FD	prefixes iy	set 3,l	
			continuing in		
222	OVER	FE	cp N	set 3,(hl)	
223	OUT	FF	rst 56	set 3,a	
224	LPRINT			set 4,b	

Appendix B:
The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
224	LLIST	E1	pop hl	set 4,c	
225	STOP	E2	jp po,NN	set 4,d	
227	READ	E3	ex (sp),hl	set 4,e	
228	DATA	E4	call po,NN	set 4,h	
229	RESTORE	E5	push hl	set 4,l	
230	NEW	E6	and N	set 4,(hl)	
231	BORDER	E7	rst 32	set 4,a	
232	CONTINUE	E8	ret pe	set 5,b	
233	DIM	E9	jp (hl)	set 5,c	
234	REM	EA	jp pe,NN	set 5,d	
235	FOR	EB	ex de,hl	set 5,e	
236	GO TO	EC	call pe,NN	set 5,h	
237	GO SUB	ED	----	set 5,l	
238	INPUT	EE	xor N	set 5,(hl)	
239	LOAD	EF	rst 40	set 5,a	
240	LIST	F0	ret p	set 6,b	
241	LET	F1	pop af	set 6,c	
242	PAUSE	F2	jp p,NN	set 6,d	
243	NEXT	F3	di	set 6,e	
244	POKE	F4	call p,NN	set 6,h	
245	PRINT	F5	push af	set 6,l	
246	PLOT	F6	or N	set 6,(hl)	
247	RUN	F7	rst 48	set 6,a	
248	SAVE	F8	ret m	set 7,b	
249	RANDOMIZE	F9	ld sp,hl	set 7,c	
250	IF	FA	jp m,NN	set 7,d	
251	CLS	FB	ei	set 7,e	
252	DRAW	FC	call m,NN	set 7,h	
253	CLEAR	FD	prefixes instructions using iy	set 7,l	
254	RETURN	FE	cp N	set 7,(hl)	
255	COPY	FF	rst 56	set 7,a	

Appendix C: Display Modes and Memory



In addition to the "normal" 32-column screen, the TS 2000 allows several "enhanced display modes for use by advanced programmers and designers of software for your enjoyment. The full use of these capabilities is beyond the scope of this manual and will be discussed in a forthcoming *TS 2000 Advanced Programming Concepts Manual*. Use of these modes is made easier by Timex application programs which can use the TS 2000's display capabilities in word processing, financial data display, entertainment, and education.

Display Modes

Use this command to set the full width (64 characters—or more if you redefine the character set) mode and select white INK. The proper complementary color of PAPER (in this case, black) is selected automatically:

OUT 253,[2]

Other INK selection values are shown in the chart below:

Value	Function	INK	PAPER
0	Full-width mode	Black	White
8 + 0		Blue	Yellow
16 + 0		Red	Cyan
24 + 0		Magenta	Green
32 + 0		Green	Magenta
40 + 0		Cyan	Red
48 + 0		Yellow	Blue
56 + 0		White	Black

Use this command to select the alternate display file ("dual screen mode"):

OUT 255,1

Extended color mode, which also uses both display file areas, is selected by this command:

OUT 255,2

You can always return to the "normal," simple screen, 32 column mode with

OUT 255,0

In each of the enhanced display modes (beyond the "normal," 32 column display mode), you need to set up the characters and attributes in the memory areas reserved for these displays. See the chart below. You can do this by proper use of machine code routines combined with system software.

Display Addresses

Address	Hexadecimal	Decimal
Display File 1	4000-577F	16384-22527
Attribute File 1	5800-5AFF	22528-23295
Display File 2	5B00-777F	23575-30719
Attribute File 2	7800-7AFF	30720-31487

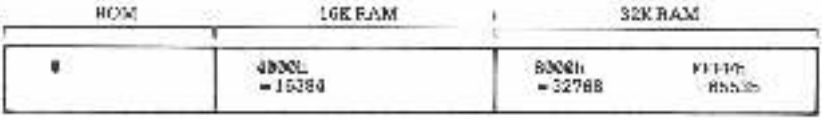
POKE statements must use decimal addresses.

Appendix C:
Display Modes and Memory

The Memory

Deep inside the computer everything is stored as bytes, i.e. numbers between 0 and 255. You may think you have stored away the price of wool in the address of your fertilizer suppliers, but it has all been converted into collections of bytes and bytes are what the computer sees.

Each place where a byte can be stored has an address, which is a number between 0 and FFFFh (so an address can be stored as two bytes), so you might think of the memory as a long row of numbered boxes, each of which can contain a byte. Not all the boxes are the same, however. In the standard 48K RAM machine, the boxes from 4000h to FFFFh are RAM boxes, which means you can open the lid and alter the contents, and those from 0 to 3FFFh are ROM boxes, which have glass tops but cannot be opened. You just have to read whatever was put in them when the computer was made.



To inspect the contents of a box, we use the PEEK function: its argument is the address of the box, and its result is the contents. For instance, this program prints out the first 21 bytes in ROM (and their addresses):

```
10 PRINT "Address": TAB 8; "byte"
20 FOR a = 0 TO 20
30 PRINT a; TAB 8; PEEK a
40 NEXT a
```

All these bytes will probably be quite meaningless to you, but the processor chip understands them to be instructions telling it what to do.

To change the contents of a box (if it is RAM), we use the POKE statement. It has the form

POKE address, new contents

where 'address' and 'new contents' stand for numeric expressions. (Note that the address is given in decimal notation, not hexadecimal.) For instance, if you say

```
POKE 31000,57
```

the byte at address 31000 is given the new value 57 — type

```
PRINT PEEK 31000
```

to prove this. (Try poking in other values to show that there is no checking.) The new value must be between −256 and +255, and if it is negative then 256 is added to it.

The ability to POKE gives you immense power over the computer if you know how to wield it, and immense destructive possibilities if you don't. It is very easy, by poking the wrong value in the wrong address, to lose vast programs that took you hours to type in. Fortunately, you won't do the computer any permanent damage.

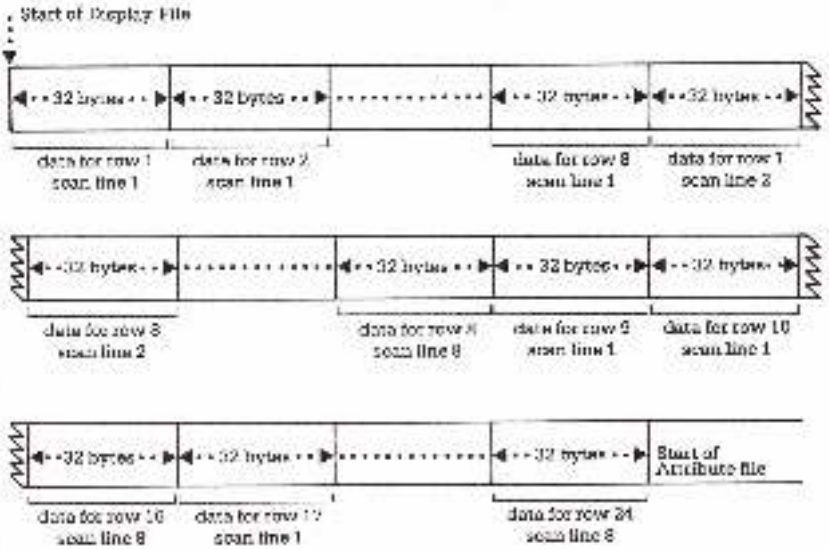
We shall now take a more detailed look at how the RAM is used but don't bother to read this unless you're interested.

The memory is divided into different areas (shown on the big diagram) for storing different kinds of information. The areas are only large enough for the information that they actually contain, and if you insert some more at a given point (for instance by adding a program line or variable) space is made by shifting up everything above that point. Conversely, if you delete information then everything is shifted down.

Pixel Data Organization

Pixel data describes where dots are located on the screen. Data are written to the screen to display characters and by the PLOT and DRAW commands. This data is stored in memory as follows:

Appendix C:
Display Modes and Memory



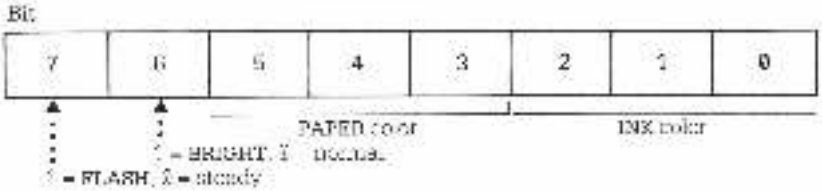
This organisation applies to both display files when used in normal video mode, full watch mode, and dual screen mode.

The display file stores the television picture. It is rather curiously laid out so you probably won't want to PRINT or POKE in it. Each character position on the screen has an 8 x 8 square of dots, and each dot can be either 0 (paper) or 1 (ink), and by using binary notation we can store the pattern as 8 bytes, one for each row. However, these 8 bytes are not stored together. The corresponding rows in the 32 characters of a single line are stored together as a group of 32 bytes, because this is what the electron beam in the television needs as it scans from the left hand side of the screen to the other. Since the complete picture has 24 lines of 8 scans each, you might expect the total of 192 scans to be stored in order, one after the other, you'd be wrong. First come the top scans of lines 0 to 7, then the next scans of lines 0 to 7, and so on to the bottom scans of lines 0 to 7; then the

same as lines 8 to 15; and then the same for lines 16 to 23. The upshot of all this is that if you're used to a computer that uses PEEK and POKE on the screen you'll have to start using SCREEN$ and PRINT AT instead, or PLOT and POINT.

The attributes are the colors and so on for each character position, using the format of ATTR. These are stored line by line in the order you'd expect.

Attribute Byte Format

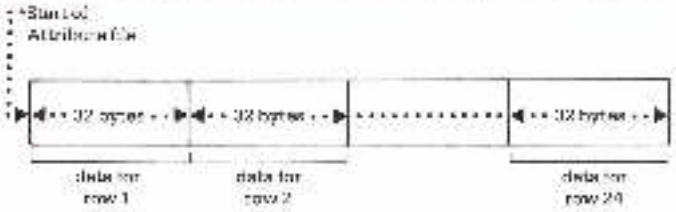


PAPER or INK color

Value		Color
7	111	White
6	110	Yellow
5	101	Cyan
4	100	Green
3	011	Magenta
2	010	Red
1	001	Blue
0	000	Black

Attribute Data Organization

Attribute data describes paper and ink color, flashing or steady, and bright or normal intensity. For each character in normal video mode there is one byte of attribute data. This data is stored in memory as follows:



In Extended Color Mode, the organization of attributes (which reside in memory starting at 9000H) is the same as the organization of pixel data.

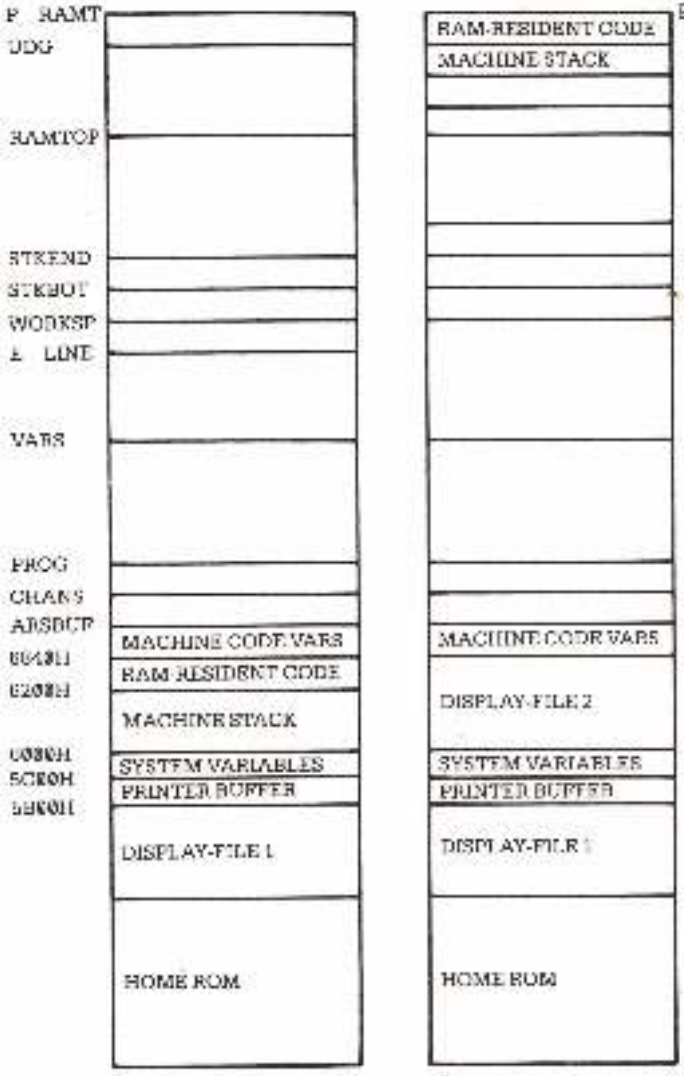
The printer buffer stores the characters destined for the printer.

The system variables contain various items of information that tell the computer what sort of state the computer is in. They are used fully in the next chapter, but for the moment note that there are some (called CHANS, PROG, VARS, E__LINE and so on) that contain the addresses of the boundaries between the various areas in memory. These are not BASIC variables, and their names will not be recognized by the computer.

Appendix C:
Display Modes and Memory

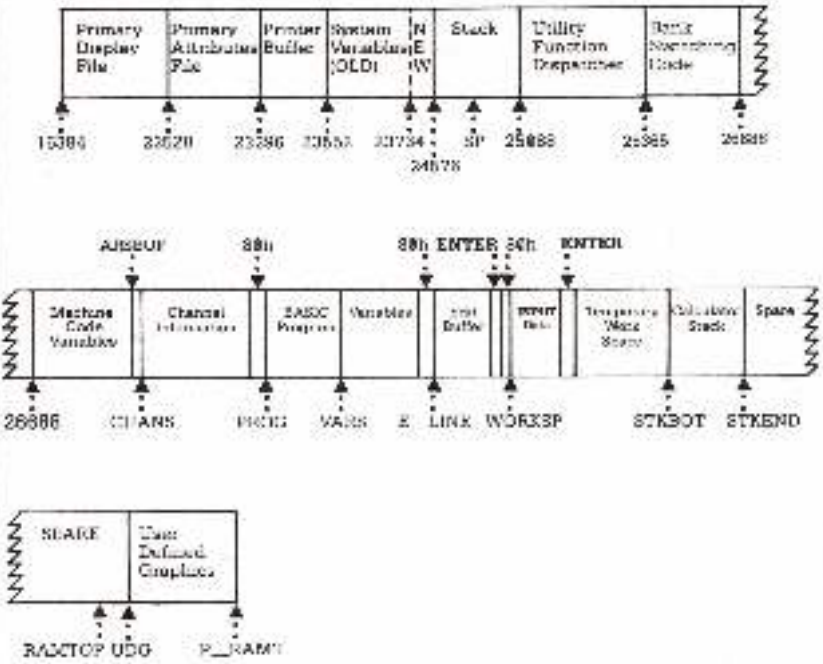
Home Memory Map

(a) One Display File (b) Two Display Files



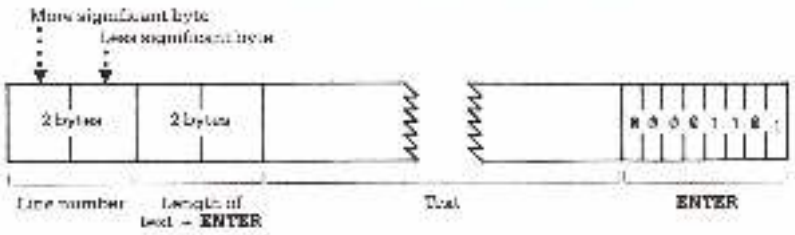
Appendix C: Display Modes and Memory

Figure C-1. Data Structure Layout



Each line of BASIC program has the form illustrated by Figure C-2.

Figure C-2. Basic Program Line Layout



Note that in contrast with all other cases of two-byte numbers in the system, the low-order byte is stored with its most significant byte first, that is to say in the order that you write them down.

A numerical constant in the program is followed by its floating point form, using the character CHR$(4) followed by five bytes for the number itself.

The variables have different formats according to their different natures. The letters of the names should be interpreted as starting out as lower case. This order is illustrated by Figure C-3.

Figure C-3. Number Whose Name Is One Letter Only

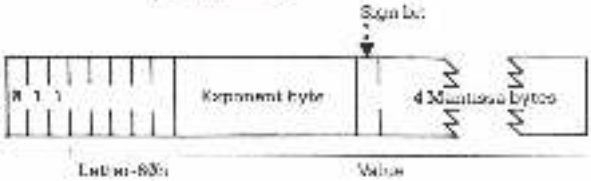
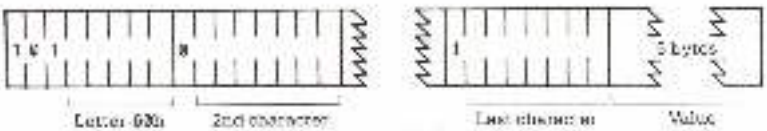


Figure C-4 illustrates a situation where a number whose name is longer than one letter is used.

Figure C-4. Longer Name Data Structure



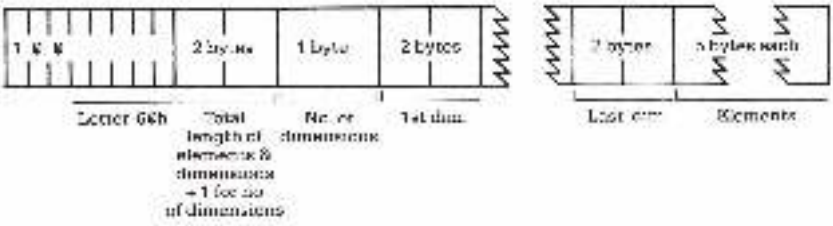
An array of numbers is illustrated by Figure C-5.

The order of the elements is:
first, the elements for which the first subscript is 1
next, the elements for which the first subscript is 2
next, the elements for which the first subscript is 3
and so on, for all possible values of the first subscript.

The elements with a given first subscript are ordered in the same way using the second subscript, and so on down to the last.

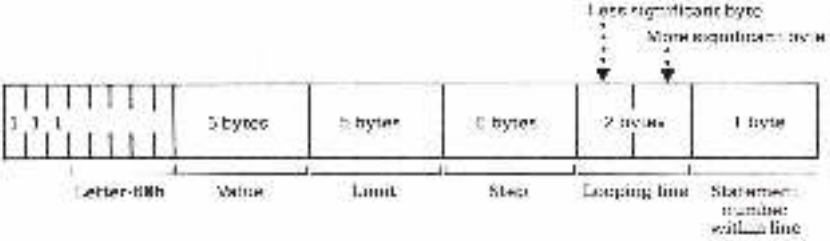
As an example, the elements of the 3×6 array b are stored in the order b(1,1) b(1,2) b(1,3) b(1,4) b(1,5) b(1,6), b(2,1) b(2,2) . . . b(2,6) b(3,1) b(3,2) . . . b(3,6)

Figure C-5. Array Data Structure



Structure of a control variable for a FOR/NEXT loop is illustrated by Figure C-6.

Figure C-6. FOR/NEXT Loop Data Structure



String structures are illustrated by Figure C-7.

Figure C-7. String Data Structure

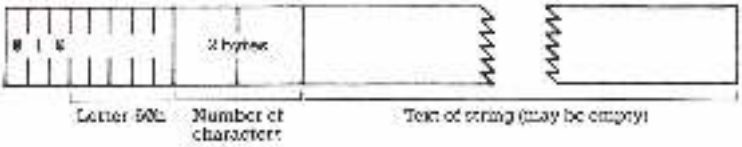
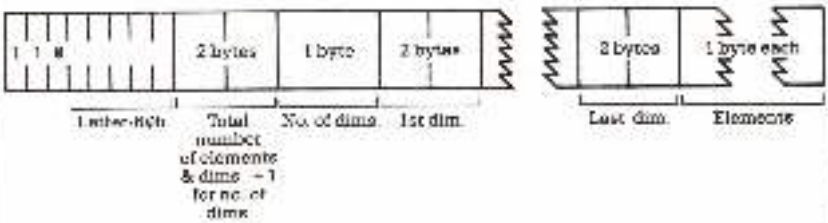


Figure C-8 illustrates the data structure of an array of characters.

Figure C-8. Character Array Data Structure



The calculator is the part of the BASIC system that deals with arithmetic and the numbers on which it is operating are held mostly on the calculator stack.

The spare part contains the space so far unused.

The machine stack is the stack used by the Z80 processor to hold return addresses and so on.

Any number (except 0) can be written uniquely as
$+m\cdot 2^{e}$
where + is the sign
m is the mantissa, and lies between ½ and 1 (it cannot be 1).
and e is the exponent, a whole number (possibly negative)

Suppose you write m in the binary scale. Because it is a fraction, it will have a binary point (like the decimal point in the scale of ten) and then a binary fraction (like a decimal fraction) as in binary.

a half is written .1
a quarter is written .01
three quarters is written .11
a tenth is written .0001100110011001100110011... and so on. With our number m, because it is less than 1, there are no bits before the binary point, and because it is at least ½, the bit immediately after the binary point is a 1.

To store the number in the computer, we use five bytes, as follows:

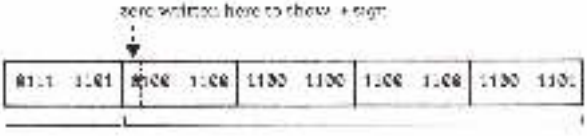
1 Write the first eight bits of the mantissa in the second byte (we know that the first bit is 1); the second eight bits in the third byte; the third eight bits in the fourth byte and the fourth eight bits in the fifth byte.

2. Replace the last bit in the second byte—which we know is 1—by the sign: 0 for plus, 1 for minus.
3. Write the exponent + 128 in the first byte. For instance, suppose our number is 0.9:

$$0.9 = 9/10 \times 2^{-3}$$

Thus the mantissa is .1100110011001100110011001100 (since the 33rd bit is 1, we shall round the 32nd up from 0 to 1), and the exponent is -3. Applying our three rules gives the five bytes illustrated in Figure C-9.

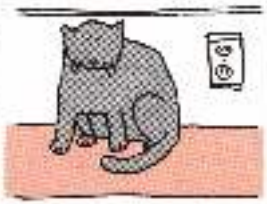
Figure C-9. Zero Written Here To Show + Sign



There is an alternative way of storing whole numbers between -65535 and +65535.

1. The first byte is 0.
2. The second byte is 2 for a positive number, 255 for a negative one.
3. The third and fourth bytes are the less and more significant bytes of the number (or the number + 131072 if it is negative).
4. The fifth byte is 0.

Appendix D: The System Variables



The bytes in memory from 23552 to 23748 are set aside for specific uses by the system. You can PEEK them to find out various things about the system, and some of them can be usefully POKEd. They are listed here with their uses.

These are called system variables, and have names, but do not confuse them with the variables used by the BASIC. The computer will not recognize the names as referring to system variables, and they are given solely as mnemonics for us humans.

The abbreviations in column 1 have the following meanings:

X The variables should not be poked because the system might crash.

N Poking the variable will have no lasting effect.

Appendix D: The System Variables

The number in column 3 is the number of bytes in the variable. For two bytes, the first one is the less significant byte — the reverse of what you might expect. So to POKE a value n to a two byte variable at address a, use

POKE a, n - 256*INT (n/256)
POKE a+1, INT (n/256)

and to PEEK the value, use the expression

PEEK a + 256*PEEK (a+1)

Notes	Address	Name	Contents
N8	23552	KSTATE	Used in reading the keyboard.
N1	23560	LAST K	Stores newly pressed key.
1	23561	REPDEL	Time — in 50ths of a second — that a key must be held down before it repeats. This starts off at 35, but you can POKE in other values.
1	23562	REPPER	Delay — in 50ths of a second — between successive repeats of a key held down: initially 5.
N2	23563	DEFADD	Address of arguments of user-defined function if one is being evaluated; otherwise 0.
N1	23565	K DATA	Stores 2nd byte of colour controls entered from keyboard.
N2	23566	TVDATA	Stores bytes of colour, AT and TAB controls going to television.
X38	23568	STRMS	Addresses of channels attached to streams.
2	23620	CHARS	256 less than address of character set (which starts with space and carries on to the copyright symbol). Normally in ROM, but you can set up your own in RAM and make CHARS point to it.
1	23606	RASP	Length of warning buzz.
1	23609	PIP	Length of keyboard click.
1	23610	ERR NR	1 less than the report code. Starts off as 255 (so PEEK 23610 gives 255).
X1	23611	FLAGS	Various flags to control the BASIC system.
X1	23612	TV FLAG	Flags associated with the television.
X2	23613	ERR SP	Address of item on machine stack to be used as error return.
N2	23615	LIST SP	Address of return address from automatic listing.

Appendix D: The System Variables

Notes	Address	Name	Contents
N1	23617	MODE	Specifies K, L, C, E or G cursor.
2	23618	NEWPPC	Line to be jumped to.
1	23620	NSPPC	Statement number in line to be jumped to. Poking first NEWPPC and then NSPPC makes a GOTO to a specified statement in a line.
2	23621	PPC	Line number of statement currently being executed.
1	23623	SUBPPC	Number within line of statement being executed.
1	23624	BORDCR	Border color *8; also contains the attributes normally used for the lower half of the screen.
2	23625	E_PPC	Number of current line (with program cursor).
X2	23627	VARS	Address of variables.
X2	23629	DEST	Address of variable in assignment.
X2	23631	CHANS	Address of channel data.
X2	23633	CURCHL	Address of information currently being used for input and output.
X2	23635	PROG	Address of BASIC program.
X2	23637	NXTLIN	Address of next line in program.
X2	23639	DATADD	Address of terminator of last DATA item.
X2	23641	E_LINE	Address of command being typed in.
2	23643	K_CUR	Address of cursor.
X2	23645	CH_ADD	Address of the next character to be interpreted: the character after the argument of PEEK, or the NEWLINE at the end of a POKE statement.
2	23647	X_PTR	Address of the character after the ? marker.
X2	23649	WORKSP	Address of temporary work space.
X2	23651	STKBOT	Address of bottom of calculator stack.
X2	23653	STKEND	Address of start of spare space.
N1	23655	BREG	Calculator b register.
X2	23656	MEM	Address of area used for calculator's memory. (Usually MEMBOT, but not always).
1	23658	FLAGS2	More flags.
X	23659	DF_SZ	The number of lines (including one blank line) in the lower part of the screen.
2	23660	S_TOP	The number of the top program line in automatic listings.
2	23662	OLDPPC	Line number to which CONTINUE jumps.
1	23664	OSPCC	Number within line of statement to which CONTINUE jumps.
N1	23665	FLAGX	Various flags.

Appendix D: The System Variables

Notes	Address	Name	Contents
N2	23366	STRLEN	Length of string type destination in assignment.
N2	23368	T ADDR	Address of next item in syntax table.
2	23670	SEED	The seed for RND. This is the variable that is set by RANDOMIZE.
3	23672	FRAMES	3-byte (least significant first), 50/60 counter. Incremented every 20ms.
2	23675	UDG	Address of 1st user-defined graphic. You can change this, for instance to save space, by having fewer user-defined graphics.
1	23677	COORDS	x-coordinate of last point plotted.
1	23678		y-coordinate of last point plotted.
1	23679	P POSN	33-column number of printer position.
1	23680	PR CC	Less significant byte of address of next position for LPRINT to print at (in printer buffer).
1	23681		Not used.
2	23682	ECHO E	33-column number and 24-line number (in lower half) of end of input buffer.
2	23684	DF CC	Address in display file of PRINT position.
2	23686	DFCCL	Like DF CC for lower part of screen.
X1	23688	S POSN	33-column number for PRINT position.
X1	23689		24-line number for PRINT position.
X2	23690	SPOSNL	Like S POSN for lower part.
1	23692	SCR CT	Counts scrolls. It is always 1 more than the number of scrolls that will be done before stopping with scroll? If you keep poking this with a number bigger than 1 (say 255), the screen will scroll on and on without asking you.
1	23693	ATTR P	Permanent current colors, etc. (as set up by color statements).
1	23694	MASK P	Used for transparent colors, etc. Any bit that is 1 shows that the corresponding attribute bit is taken not from ATTR P, but from what is already on the screen.
N1	23695	ATTR T	Temporary current colors, etc. (as set up by color items).
N1	23696	MASK T	Like MASK P, but temporary.
1	23697	P FLAG	More flags.
N30	23698	MEMBOT	Calculator's memory area used to store numbers that cannot conveniently be got on the calculator stack.
2	23728		Not used.
2	23730	RAMTOP	Address of last byte of BASIC system area.
2	23732	P-RAMT	Address of last byte of physical RAM.

Appendix D: The System Variables

Notes	Address	Name	Contents
2	23734	ERRLN	Line number to GOTO on error
2	23736	LLINE	Line number in which error occurred
1	23738	LSTMT	Statement number within line in which error occurred
1	23739	ERRNO	Error number (Report Code)
2X	23740	SYSCOM	Pointer to the System Communication Table
1X	23742	MAXBNK	Number of Expansion Banks in System
1X	23743	CURCBN	Current Channel Bank Number
2X	23744	MSTBOT	Address of location above machine stack
1X	23746	VIDMOD	Video Mode. Non-zero if the second display file is open for use.
7X	23746		Various variables used by BASIC cartridges
1X	23755	STRMNM	Current stream number.

This program tells you the first 22 bytes of the variables area:

```
10 FOR n = 0 TO 21
20 PRINT PEEK (PEEK 23627 + 256 * PEEK 23628 + n)
30 NEXT n
```

Try to match up the control variable n with the descriptions above. Now change line 20 to

```
20 PRINT PEEK (23755 + n)
```

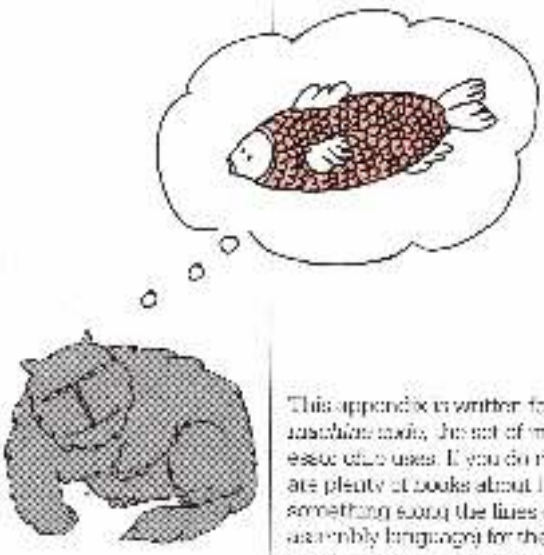
This tells you the first 22 bytes of the program area. Match these up with the program itself.

Display Addresses

Address	Hexadecimal	Decimal
Display File 1	4000-57FF	16384-22527
Attribute File 1	5800-5AFF	22528-23295
Display File 2	6000-77FF	24576-30719
Attribute File 2	7800-7AFF	30720-31487

POKE statements must use decimal addresses.

Appendix E: Using Machine Code



This appendix is written for those who understand Z80 machine code, the set of instructions that the Z80 processor chip uses. If you do not, but would like to, there are plenty of books about it. You want to get one called something along the lines of 'Z80 Machine code for assembly language) for the absolute beginner.'

These programs are normally written in assembly language, which, although cryptic, is not too difficult to understand with practice. However, to run them on the computer you need to code the program into a sequence of bytes—in this form it is called *machine code*. This translation is usually done by the computer itself, using a program called an assembler. There is no assembler built in to the TS 2000, but you may well be able to buy one as an extra. Failing that, you will have to do the translation yourself, provided that the program is not too long.

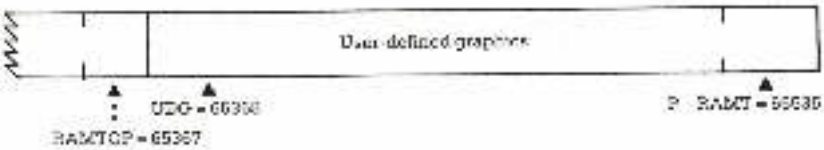
Let's take as an example the program

```
LD BC,NN
RET
```

which loads the bc register pair with NN. This translates into the four machine code bytes 1, NN, 2 (for ld bc, NN) and 201 (for ret). (If you look up 1 and 201 in Appendix B, you will find ld bc, NN—where NN stands for any two byte number—and ret.)

When you have got your machine code program, the next step is to get it into the computer. (An assembler would probably do this automatically.) You need to decide where about in memory to put it, and the best thing is to make extra space for it between the BASIC area and the user defined graphics.

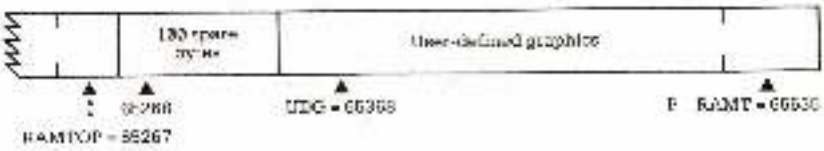
You have a 48K machine, so the top end of RAM has



If you type:

```
CLEAR 65267
```

this will give you a space of 100 (for great measure) bytes starting at address 65268



To put in the machine code program, you would run a BASIC program something like:

```
10 LET a = 65280
20 READ b: POKE a,b
30 LET a = a + 1: GOTO 20
40 DATA [data],[data],[data]
```

(This will stop with report B Out of DATA when it has filled in the four bytes you specified.)

To run the machine code, you use the function USR—but this time with a numeric argument, the starting address. The result is the value of the bc register on return from the machine code program, so if you do

```
PRINT USR 65280
```

you get the answer 99.

The return address to the BASIC is stacked in the usual way, so return is by a Z80 ret instruction. You should not use the iy and i registers in a machine code routine.

You can save your machine code program easily enough with

```
SAVE "name" CODE 65280,4
```

On the face of it, there is no way of saving it so that, when loaded it automatically runs itself, but you can get round this by using a BASIC program.

```
10 LOAD "" CODE 65280,4
20 PRINT USR 65280
```

Do first

```
SAVE "name" LINE 10
```

and then

```
SAVE "xxxx" CODE 65280,4
LOAD "name"
```

will then automatically run the BASIC program, and the BASIC program will load and run the machine code.

Notes:

1. If interrupts are enabled, don't use the iy register.
2. The i register must never hold a value greater than 3Fh.

Appendix F: Keyword Table

Cursor		Definition
K	**Keyword Cursor.**	If a primary key is pressed when the K cursor is displayed, the command (or keyword) imprinted on the key can be utilized (i.e. PRINT).
L	**Letter Cursor.**	If a primary key is pressed when the L cursor is displayed, the letter or special symbol imprinted on the key is taken "literal" by the computer and displayed on the screen (i.e. 8 or A).
C	**Caps Lock Cursor.**	Locks keyboard to produce upper case (capital) letters when a primary letter key is pressed.
G	**Graphics Cursor.**	Initiates graphics mode so the graphics characters imprinted on keys 1-8 can be utilized.
E	**Extended Cursor.**	If a primary key is pressed when the E cursor is displayed, the command (or keyword) imprinted above the primary key can be utilized (i.e. LPRINT) if both the primary key and the SYMBOL SHIFT key are pressed simultaneously. The command (or keyword) imprinted below the primary key can be utilized (i.e. BRIGHT).

Appendix F: Keyword Table

Tokens (Keywords) and Functions	Primary Key Name	Instructions
ABS	G	Need E cursor, then press the Primary Key.
ACS	W	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
AND	Y	Need K, L, or C cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
ASN	C	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
AT	'	Need K, L, or C cursor, hold SYMBOL SHIFT key and then press the Primary Key.
ATN	F	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
ATTR	L	Need K cursor, then hold SYMBOL SHIFT key and press the Primary Key.
BEEP	Z	Need K cursor, then hold SYMBOL SHIFT key and press the Primary Key.
BIN	B	Need L cursor, then press the Primary Key.
BORDR (BORDER)	B	Need K cursor, then press the Primary Key.
BREAK	BREAK	Press the Primary Key.
BRIGHT	8	Need K cursor, then hold SYMBOL SHIFT key and press the Primary Key.
CAPS LOCK	2	Works with L or Need C cursor, then simultaneously press CAPS SHIFT key and Primary Key.
CAT	9	Need L cursor, then hold SYMBOL SHIFT key and press the Primary Key.
CHR$	U	Need L cursor, then press the Primary Key.
CIRCLE	H	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
CLEAR	X	Need K cursor, then press the Primary Key.
CLOSE #	5	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
CLS	V	Need K cursor, then press the Primary Key.
CODE	I	Need E cursor, then press the Primary Key.
CONT	C	Need K cursor, then press the Primary Key.
COPY	Z	Need K cursor, then press the Primary Key.
COS	W	Need E cursor, then press the Primary Key.
DATA	D	Need E cursor, then press the Primary Key.

Appendix F: Keyword Table

Tokens (Keywords) and Functions	Primary Key Name	Instructions
DEF FN	1	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
DELETE	0	Works with **K**, **L**, or **C** cursor, then simultaneously press CAPS SHIFT key and Primary Key.
DIM	D	Need **K** cursor, then press the Primary Key.
DRAW	W	Need **K** cursor, then press the Primary Key.
EDIT	1	Need **K** cursor, then simultaneously press CAPS SHIFT key and Primary Key.
ERASE	7	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
EXP	X	Need **E** cursor, then press the Primary Key.
FLASH	V	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
FN	2	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
FOR	F	Need **K** cursor, then press the Primary Key.
FORMAT	6	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
FREE	A	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
GOSUB	H	Need **K** cursor, then press the Primary Key.
GOTO	G	Need **K** cursor, then press the Primary Key.
GRAPHICS	9	Works with **K**, **L**, or **C** cursor, then simultaneously press CAPS SHIFT key and Primary Key.
IF	U	Need **K** cursor, then press the Primary Key.
IN	I	Need **K** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
INK	X	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
INKEY$	N	Need **K** cursor, then press the Primary Key.
INPUT	I	Need **K** cursor, then press the Primary Key.
INT	R	Need **K** cursor, then press the Primary Key.
INV. VIDEO	4	Works with **K**, **L**, or **C** cursor, then simultaneously press CAPS SHIFT key and Primary Key.
INVERSE	M	Need **K** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
LEN	K	Need **E** cursor, then press the Primary Key.

Appendix F: Keyword Table

Tokens (Keywords) and Functions	Primary Key Name	Instructions
LET	L	Need K cursor, then press the Primary Key.
LINK	S	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
LIST	X	Need K cursor, then press the Primary Key.
LLIST	V	Need E cursor, then press the Primary Key.
LN	Z	Need E cursor, then press the Primary Key.
LOAD	J	Need K cursor, then press the Primary Key.
LPRINT	C	Need K cursor, then press the Primary Key.
MERGE	T	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
MOVE	B	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
NEW	A	Need K cursor, then press the Primary Key.
NEXT	N	Need K cursor, then press the Primary Key.
NOT	S	Need K, L, or G cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
ON ERR	T	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
OPEN #	4	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
OR	U	Need K, L, or G cursor, hold SYMBOL SHIFT key and then press the Primary Key.
OUT	O	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
OVER	N	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
PAPER	C	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
PAUSE	M	Need K cursor, then press the Primary Key.
PEEK	O	Need K cursor, then press the Primary Key.
PI	M	Need E cursor, then press the Primary Key.
PLOT	O	Need K cursor, then press the Primary Key.
POINT	8	Need E cursor, then hold SYMBOL SHIFT key and press the Primary Key.
POKE	O	Need K cursor, then press the Primary Key.
PRINT	P	Need K cursor, then press the Primary Key.
RAND	T	Need K cursor, then press the Primary Key.

Appendix F: Keyword Table

Tokens (Keywords) and Functions	Primary Key Name	Instructions
READ	A	Need **E** cursor, then press the Primary Key.
REM	F	Need **K** cursor, then press the Primary Key.
RESET	P	Need **K** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
RESTORE	S	Need **E** cursor, then press the Primary Key.
RETRN (RETURN)	V	Need **K** cursor, then press the Primary Key.
RND	T	Need **F** cursor, then press the Primary Key.
RUN	R	Need **K** cursor, then press the Primary Key.
SAVE	S	Need **K** cursor, then press the Primary Key.
SCREEN$	K	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
SGN	I	Need **E** cursor, then press the Primary Key.
SIN	Z	Need **E** cursor, then press the Primary Key.
SOUND	U	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
SPACE BAR	SPACE BAR	Press the Primary Key.
SQR	H	Need **E** cursor, then press the Primary Key.
STEP	D	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
STICK	S	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
STOP	A	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
STR$	Y	Need **K** cursor, then press the Primary Key.
SYMBOL SHIFT	SYMBOL SHIFT	Press the Primary Key.
TAB	F	Need **E** cursor, then press the Primary Key.
TAN	J	Need **E** cursor, then press the Primary Key.
THEN	G	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
TO	F	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
TRUE VIDEO	3	Works with **K**, **L**, or **C** cursor, then simultaneously press CAPS SHIFT key and Primary Key.
USR	C	Need **E** cursor, then press the Primary Key.

Appendix F: Keyword Table

Tokens (Keywords) and Functions	Primary Key Name	Instructions
VAL	~	Need **K** cursor, then press the Primary Key
VAL$	J	Need **K** cursor, then press the Primary Key
VERIFY	H	Need **L** cursor, then hold SYMBOL SHIFT key and press the Primary Key
!	1	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key and then press the Primary Key.
"	P	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key and then press the Primary Key.
#	3	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
$	4	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
%	5	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key and then press the Primary Key.
>	T	Need **K**, **L** or **C** cursor, hold SYMBOL SHIFT key and then press the Primary Key.
\	D	Need **K** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
^	H	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
£	X	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
?	C	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key and then press the Primary Key.
/	V	Need **K**, **L** or **C** cursor, hold SYMBOL SHIFT key and then press the Primary Key.
<=	Q	Need **K** **L** or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
>=	E	Need **K** **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
@	2	Need **K**, **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
[	Y	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
]	U	Need **E** cursor, then hold SYMBOL SHIFT key and press the Primary Key.
&	6	Need **K** **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
'	7	Need **K** **L**, or **C** cursor, hold SYMBOL SHIFT key, and then press the Primary Key.

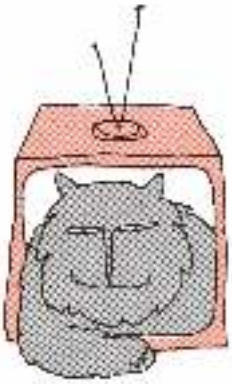
Appendix F: Keyword Table

Tokens (Keywords) and Functions	Primary Key Name	Instructions
(	8	Need K, L, or C cursor, hold SYMBOL SHIFT key and then press the Primary Key.
)	9	Need K, L, or C cursor, hold SYMBOL SHIFT key and then press the Primary Key.
*	B	Need K, L, or C cursor, hold SYMBOL SHIFT key and then press the Primary Key.
+	K	Need K, L, or C cursor, hold SYMBOL SHIFT key and then press the Primary Key.
,	N	Need K, L, or C cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
-	J	Need K, L, or C cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
.	M	Need K, L, or C cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
:	Z	Need K, L, or C cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
;	O	Need K, L, or C cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
=	L	Need K, L, or C cursor, hold SYMBOL SHIFT key, and then press the Primary Key.
▝	1	Need C cursor, then press the Primary Key.
▗	2	Need C cursor, then press the Primary Key.
▐	3	Need C cursor, then press the Primary Key.
▖	4	Need C cursor, then press the Primary Key.
←	5	Works with K, L, or C cursor, then simultaneously press CAPS SHIFT key and Primary Key.
▙	6	Need C cursor, then press the Primary Key.
↓	7	Works with K, L, or C cursor, then simultaneously press CAPS SHIFT key and Primary Key.
▟	8	Need C cursor, then press the Primary Key.
↑	9	Works with K, L, or C cursor, then simultaneously press CAPS SHIFT key and Primary Key.
▜	0	Need C cursor, then press the Primary Key.
→	8	Works with K, L, or C cursor, then simultaneously press CAPS SHIFT key and Primary Key.
▛	8	Need C cursor, then press the Primary Key.

Appendix F: Keyword Table

Tokens (Keywords) and Functions	Primary Key Name	Instructions
<>	W	Need K L or C cursor: hold SYMBOL SHIFT key and then press the Primary Key.
<	R	Need K L or C cursor: hold SYMBOL SHIFT key and then press the Primary Key.

Appendix G: Index



A

ABS 224
ACS 224
AND 128, 131, 224
apostrophe 82, 88, 221
array 143, 156
array data structure 262
array variable name 143, 156, 220
ASN 224
assembly language 269
assignment statement 96, 124
AT 94, 98
ATN 224
ATTR 260, 261, 224
attributes 222
attribute byte format 261
audio cable 8, 9
automatic starting 31

B

BASIC 27
BASIC program
  line layout 251
BEEP 86, 88, 229
BIN 165, 167, 225
BORDER 58, 59, 228, 229
branch 121
branching program 121
BREAK 74, 78, 205, 206, 210
BRIGHT 178, 179, 182, 221, 229

C

C cursor 18, 218
© copyright 75
calculator stack 257
capital letters 73
CAPS LOCK 18, 73, 218
CAPS SHIFT 18, 218
cassette recorder 7, 9, 215
CAT 214, 229
channel selector switch 6, 9
character array
  data structure 257
CHR$ 197, 230, 235
CIRCLE 82, 84, 229
CLEAR 91, 93, 229
CLOSE 214, 229
CLS 91, 93, 230
CODE 196, 201, 225
code numbers 120, 197
colon 87, 88
comma 88, 97
command 27, 71
comparing strings 128
comparing values 120
compiler 148
CONT 75, 78, 230
control variable 114, 119
control variable names 222
COPY 205, 208, 230
copyright notice 9, 13
COS 225
counter 114
cursor arrows 15, 33, 218

D

DATA 108, 112, 226
data structure layout 254
DEF FN 230
DELETE 17, 32, 138, 141, 218, 230
DIM 144, 152, 230
display addresses 240, 263
display file 251
display modes 217
down arrow 81
DRAW 98, 64, 231

E

E cursor 23, 218
E (exponent) 97
EDIT 90, 195, 219
element of an array 143, 150
endless loop 114
ENTER 29, 36, 219
envelope 191, 193
ERASE 214, 231
EXP 226
Extended Color Mode 153
extended mode 23, 35

F

FLASH 178, 179, 182, 221
FN 225
FOR 114, 119, 231
FOR/NEXT loop 114, 119
FOR/NEXT loop data structure 256
FORMAT 214, 231
FREE 199, 231, 225, 231
functions 30, 224

G

G cursor 22
GOSUB 134, 141, 231
GOTO 73, 76, 82, 83, 231
graphics mode 22, 34, 151, 216
hard copy 200

I

IF 121, 125, 130, 231
immediate mode 71
IN 213, 225
INK 58, 59, 60, 105, 228, 232
INKEY$ 180, 175, 225
INPUT 104, 111, 158, 232
INT 102, 225
interference, TV 8
interpreter 242
INVERSE 175, 179, 192, 221, 232
INVERSE VIDEO 29

J

joystick 6, 173, 175, 215
joystick sockets 8

K

K cursor 13, 217
keyboard tutorial program 25
keywords 14, 27, 217

L

L cursor 14, 218
left bracket 25
LEN 225
LET 68, 93, 121, 232
LINE 51, 53, 237
line number 73
LIST 115, 119, 233
LLIST 204, 238, 233
LN 225
LOAD 80, 233
LOAD . . . CODE 233
LOAD . . . DATA 149, 150, 233
LOAD . . . SCREEN$ 233
logical relations 128
LPRINT 204, 238, 233

M

machine code 205
machine stack 257
memory map 252
MERGE 52, 53, 233
modem 215
modules 139
monitor 6, 215
MOVE 214, 230

N

nested loops 115
NEW 32, 36, 78, 93, 233
NEXT 114, 119, 234
noise generator 194
NOT 125, 131, 225

O

ON ERR 234
on/off switch 6, 9
OPEN 214, 234
OR 125, 131, 225
OUT 213, 234
OVER 180, 183, 221, 234

P

PAPER 57, 58, 220, 235
parentheses 97, 132
PAUSE 172, 175, 235
PEEK 212, 225, 250
peripheral devices 213, 214, 215
PI 226
pixel 50, 161, 164
PLAY 41
PLOT 62, 64, 150, 161, 235
PLOT positions 152, 161
POINT 199, 201, 226
POKE 164, 167, 212, 235, 257
power supply unit 7, 9
PRINT 28, 36, 72, 87, 235
PRINT position 88, 152
printer 6, 203, 215, 222
priorities 93, 132, 225

Procrusteen 148
program 3, 4, 71
program cursor 73, 79, 219
program line 73
program name 42, 53
programs on tape cassettes 38
prompt 104
pseudo-random 104

Q

quotation marks 32, 85, 96

R

RAND 101, 230
random number generator 99
READ 108, 111, 237
RECORD 48
registers 186, 192
relations 126, 131
REM 47, 77, 237
report code 30, 42
RESET 214, 237
RESTORE 110, 111, 237
RETURN 134, 141, 237
right bracket 25
RND 99, 102, 228
rounding errors 90
RUN 42, 74, 75, 93, 237

S

SAVE 48, 53, 237
SAVE...CODE 237
SAVE...DATA 149, 153, 237
SAVE...SCREEN$ 238
scientific notation 97, 102
SCREEN$ 188, 191, 225
scroll? 74, 78
semicolon 80, 88
SGN 229
SIN 199, 229
sine wave 199
slicing strings 148, 152, 223

SOUND 185, 238
sound effects 184
SQR 35, 95, 226
STEP 117, 119, 231
STICK 179, 180, 227, 238
STOP 70, 180, 111, 238
string 32, 90, 223
string arrays 148, 150, 223
string array names 223
string variable 90, 93
string variable names 90, 93
STR$ 227
structured programming 133, 141
subroutines 134, 141
subscript 114, 150
substrings 140, 150, 223
superscripts 95
SYMBOL SHIFT 15, 218
syntax error marker 35, 219
system variables 259

T

TAB 83, 88, 207, 230, 231
TAN 227
television cable 6, 8
THEN 125, 136, 231
Tinex Command Cartridges 37
TO 114, 119, 231
transfer switch box 5, 8
TRUE VIDEO 21

U

UHF/VHF matching transformer 8
up arrow 79, 81
user-defined graphics 163, 218
USR 165, 167, 227

V

VAL 227
VAL$ 227
variable 90
variable name 90, 93, 222
VERIFY 49, 53, 238

X

x/y axes 154

Z

Z80 processor chip 265

+ , - , * , / , ^ 98, 101, 228
= , < , > , <= , >= , <> 126, 131, 228
$ 90

Appendix H: Report Codes

These appear at the bottom of the screen whenever the computer stops executing some BASIC and explain why it stopped: whether for a normal reason, or because an error occurred.

The report has a code number or letter so that you can refer to the table here, a list of meanings explaining what happened and the line number and statement number within that are where it stopped. (A command is shown as line 0. Within a line, statement 1 is at the beginning, statement 2 comes after the first colon or THEN, and so on.)

The behavior of CONTINUE does vary much on the reports. Normally, CONTINUE goes to the line and statement specified in the last report, but there are exceptions with reports 0, 2 and D.

Here is a table showing all the reports. It also tells you when command mode the report can occur. For instance, error A Invalid argument can occur with SGN, INT, ABS and ASN.

Appendix H: Report Codes

Code	Situation	Meaning
0	Any	OK Successful completion, or jump to a line number bigger than any existing. This report does not change the line and statement jumped to by CONTINUE.
1	NEXT	NEXT without FOR The control variable does not exist (it has not been set up by a FOR statement) but there is an ordinary variable with the same name.
2	Any	Variable not found For a simple variable this will happen if the variable is used before it has been assigned in a LET, READ or INPUT statement or loaded from tape or set up in a FOR statement. For a subscripted variable it will happen if the variable is used before it has been dimensioned in a DIM statement or loaded from tape.
3	Subscripted variables Substrings	Subscript wrong A subscript is beyond the dimension of the array or there are the wrong number of subscripts. If the subscript is negative or bigger than 65535, then error B will result.
4	LET, INPUT, FOR, DIM, GOSUB, LOAD, MERGE. Sometimes during expression evaluation	Out of memory There is not enough room in the computer for what you are trying to do. If the computer really seems to be stuck in this state, you may have to clear out the command line using DELETE and then delete a program line or two (with the intention of putting them back afterwards) to give yourself room to manoeuvre with — say — CLEAR.
5	INPUT, PRINT AT	Out of screen An INPUT statement has tried to generate more than 23 lines in the lower half of the screen. Also occurs with PRINT AT 22, .
6	Any arithmetic	Number too big Calculations have led to a number greater than about 10^{38}.
7	RETURN	RETURN without GOSUB There has been one more RETURN than there were GOSUBs.
8	Peripheral operations	End of file.

Appendix H: Report Codes

Code	Situations	Meaning
9	STOP	STOP statement After this, CONTINUE will not repeat the STOP but carries on with the statement after.
A	SQR, LN, ASN, ACS, USR (with string argument)	Invalid argument The argument for a function is no good for some reason.
B	RUN, RANDOMIZE, POKE, DIM, GO TO, GO SUB, LIST, LLIST, PAUSE, PLOT, CHR$, PEEK, USR (with numeric argument) Array access	Integer out of range When an integer is required, the floating point argument is rounded to the nearest integer. If this is outside a suitable range then error B results. For array access, see also Error 3
C	VAL, VAL$	Nonsense in BASIC The text of the (string) argument does not form a valid expression.
D	LOAD, SAVE, VERIFY, MERGE, LPRINT, LLIST, COPY. Also when the computer asks 'scroll?' and you type N, SPACE or STOP	BREAK - CONT repeats BREAK was pressed during some peripheral operation. The behavior of CONTINUE after this report is normal in that it repeats the statement. Compare with report L.
E	READ	Out of DATA You have tried to READ past the end of the DATA list
F	SAVE	Invalid file name SAVE with name empty or longer than 10 characters.
G	Entering a line into the program	No room for line There is not enough room left in memory to accommodate the new program line
H	INPUT	STOP in INPUT Some INPUT data started with STOP, or - for INPUT LINE - was pressed. Unlike the case with report 9, after report H CONTINUE will behave normally, by repeating the INPUT statement.

Appendix H: Report Codes

Code	Situations	Meaning
1	FOR	FOR without NEXT There was a FOR loop to be executed no times (e.g. FOR n = 1 TO 0) and the corresponding NEXT statement could not be found.
J	Peripheral operations	Invalid I/O device
K	INK, PAPER, BORDER, FLASH, BRIGHT, INVERSE, OVER also after one of the corresponding control characters	Invalid color The number specified is not an appropriate value.
L	Any	BREAK into program BREAK pressed: this is detected between two statements. The line and statement number in the report refer to the statement before BREAK was pressed, but CONTINUE goes to the statement after (allowing for any jumps to be done); it does not repeat any statements.
M	CLEAR, possibly in RUN	RAMTOP no good The number specified for RAMTOP is either not big or too small.
N	RETURN, NEXT, CONTINUE	Statement lost Jump to a statement that no longer exists.
O	Peripheral operations	Invalid stream
P	FN	FN without DEF User-defined function.
Q	FN	Parameter error Wrong number of arguments, or one of them is the wrong type (string instead of number or vice versa).
R	VERIFY, LOAD or MERGE	Tape loading error A file on tape was found but for some reason could not be read in, or would not verify.

Notes

Notes

335-879000

TIMEX Timex Computer Corporation Waterbury, Connecticut 06720