

CAT. NO.
26-3153

*Computer Learning Lab Program:
© 1981 The Image Producers, Inc.
Licensed to Tandy Corporation
All Rights Reserved.*

*Computer Learning Lab Program Manual:
© 1981 The Image Producers, Inc.
Licensed to Tandy Corporation
All Rights Reserved.*

It is illegal to reproduce this book or the software used in these lessons for any purpose other than personal convenience. You cannot resell, distribute in any form, or conduct any commercial activity using these materials without permission of the publisher. Please direct requests, questions, or other comments to The Image Producers, Inc., 615 Academy Dr., Northbrook, IL 60062.

No liability is assumed with respect to use of the information herein.

Reproduction or use, without express written permission from Tandy Corporation, of any portion of this manual is prohibited. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the use of the information obtained herein.

Please refer to the Software License on the back cover of this manual for limitations on use and reproduction of this Software package.

10 9 8 7 6 5 4 3 2 1

Time Response Monitoring and TRM Programming are registered trademarks of The Image Producers, Inc.

Computer Learning Lab for the TRS-80 Color Computer

By Dick Ainsworth

A Self-Teaching System of
Software, Experiments, and
Programming Guides

Radio Shack[®]


 A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102

Table of Contents

How to Use Computer Learning Lab	i
SECTION 1	
The following introduction and Lessons 1-12 show how to enter programs and how the computer uses special words to describe each step.	
Introduction to Computing	1
Instruction: PRINT, GOTO	
Line Number: ENTER	
Program: RUN, LIST	
1. Mathematician	5
Arithmetic: + - * / =	
Keyboard: INPUT	
2. Coin Flipper	11
Random Numbers: RND(X)	
Decisions: IF/THEN	
3. Guessing Game	17
Comparison: MORE >, LESS <, EQUAL =	
4. Average Calculator	23
Program Loop: FOR/NEXT	
Formulas	
5. Expressway	29
Graphics: POINT (H,V), PRINT (TAB), CHR(\$)	
Sound: SOUND(F,D)	
Motion: JOYSTICK(N)	
6. Counting Machine	37
Counting: FOR/NEXT/STEP	
7. Kaleidoscope	45
Graphics: SET (H,V,C)	
8. Decision Maker	49
Branching: ON/GOTO	
Keyboard: INKEY\$	
9. Area Calculator	55
Formulas, Special Calculators	
10. Interest Calculator	61
Printing Tables, Rounding Off Numbers	
Integers: INT(X)	
11. Coloring Box	67
Graphics: COLOR, SHAPE	
Double loops: FOR/FOR, NEXT/NEXT	
12. Time Machine	73
Time Delay, Multiple Lines	
Comparison: NOT EQUAL < >	

SECTION 2

Lessons 13-22 show how your computer can be programmed to create many different kinds of games, educational software, music, and art.

13. Probability	79
Random Numbers, Probability Curves	
14. Sorting	89
Arrays, Data Processing Techniques	
15. Temperature Converter	103
Formulas, Special Calculators	
16. Cipher	113
Program and Game Design	
17. Math Teacher	125
Time Response Monitoring®	
18. Hangperson	139
String Manipulation, Game Design	
19. Music Teacher	155
Music Instruction, Game Design	
20. Car Calculator	173
Branching, Special Calculators	
21. Graphics	183
Video Art, Graphics Characters	
22. Player Piano	193
Musical Instruments	

SECTION 3

Lessons 23-30 illustrate programming guides that can be copied and used to create many kinds of software quickly and easily.

23. Menus	199
Program Titles and Starting Screens	
24. Program Restarts	203
Program Endings with Automatic Restart	
25. Time Delays	207
Controlling Time Intervals	
26. Inputs	209
Using the Keyboard to Control Programs	
27. Music and Sound Effects	215
From Bach to BANG!	
28. Rounding Off Numbers	219
Making the Digits Fit; Printing Dollars and Sense	
29. Scoreboards	221
Who Won?	

30. Dynamic Debugger	223
Finding and Fixing Problems	
Buzzwords	229
Index	231

How to Use Computer Learning Lab

How to Use Computer Learning Lab

This interactive teaching system is not like an ordinary book or class. The text and tape software are designed specifically for use with all versions of the TRS-80 Color Computer, your television, and a standard audio cassette recorder. The programs you enter from the tapes or from the keyboard appear on the TV and are the illustrations or pictures for the text.

SECTION 1 contains an introduction to your computer and 12 self-correcting lessons that show you how BASIC words such as PRINT and GOTO are used.

SECTION 2 shows how your computer can be programmed to create games, solve formulas, play music, sort information, and teach. Each lesson contains a software example and experiments to help you design and write your own programs.

SECTION 3 includes programming guides and other tools that make writing programs easier, faster, and more fun.

Look Here for Help

From time to time you may run into a situation you don't understand or a problem you don't know how to solve. Check this section now so that you will know what help is here. If you run into difficulty, check this section again and see if you can find the answer. Here are some typical problems and their solutions:

Reverse Video simply means that you're typing with letters in reverse — light letters on a dark background. If this happens, the computer won't understand anything you type. To change from normal to reverse letters (or to change back again), press and hold down the **SHIFT** key while you type a zero (**0**).

O and 0 may look alike to you, but the letter O and the number zero are very different to your computer. If an instruction won't work, check for an O where a 0 belongs. Also, be careful not to reverse the number 1 and the letter I.

Spaces between letters and numbers are not critical in most instances with your Color Computer. You can enter PRINT 2 + 3 or enter PRINT2+3 and the computer will answer: 5. We have used spaces in front of and behind all command words in the examples. This makes programs easier to read.

How to Use Computer Learning Lab (continued)

Line Numbers must be in the correct numerical sequence for your program to work. You can use 10, 20, 30 or 1, 2, 3, or 100, 172, 203 for your line numbers. We write programs with line numbers 10, 20, 30, etc. so that there will be room for additional instructions. This way, you could add more instructions numbered 14, 15, 17 between lines 10 and 20.

Don't Use LET in your programs. If you are familiar with other versions of BASIC, you might try using a statement such as: LET A=5. With COLOR BASIC, the word LET is not necessary and should not be used. Simply use: A=5.

How It Works

Sections with this title occur through the book. Please include or refer to these sections whenever you wish to learn more or understand the material in greater detail.

If you are interested in how the Learning Lab detects typing errors, for example, you might want to know that special software is recorded on each Section 1 cassette, along with the program. When you load these tapes, the computer reads the computer program used in that lesson and stores it in memory. After each instruction is typed, and the **ENTER** key is pressed, the computer compares the input from the keyboard with the correct instruction. If you have made a typing error that would prevent the program from running correctly, the computer shows you the mistake. After a correct program has been typed, this special software "disappears" and you can run, list, or modify the lesson.

The Section 1 tapes containing this monitoring feature were programmed by Jim Amling in assembly language. He also assisted in creating lesson programs specifically designed to be clear and easy to understand. Dick and Jim have worked as a design/programming team on many other projects, including PINBALL, an Instant-Load Program Pack for the TRS-80 Color Computer.

ACKNOWLEDGEMENT

Special thanks to Al, Bill, Christal, Copper, Etta, Jan, Jim, Leanne, Marty, Natsuko, Paula, Phyllis, Sat Tara, and Tom.

Introduction to Computing

This introduction will show you how to communicate with your computer by typing instructions on the keyboard. You will also create a short computer program by entering instructions into the computer's memory. When you run your program, the computer will follow these instructions exactly.

Experiment 1: HELLO

If your computer, TV, and cassette tape recorder are not already connected and working, follow your instruction manual for details. When you turn on your computer and TV, you will see a small square changing color (called the cursor) and this message:

```
COLOR BASIC
(C) 1980 TANDY
OK
```

The important things to notice are the OK and the cursor. They tell you that the computer is ready and waiting for your next instruction. Now type this message on the keyboard:

```
HELLO
```

Press the **ENTER** key. When you press **ENTER**, the computer will respond by printing ?SN ERROR on the screen. This is an error message which tells you the computer doesn't know what HELLO means. There are only a few words in the computer's vocabulary and HELLO is not one of them.

Now type this instruction, using PRINT — one of the words in your computer's vocabulary. To type quotation marks, press **SHIFT** while you type **2**. If you make a typing error, press the left arrow key (**←**) to back up and make the correction.

```
PRINT "HELLO"
```

Now press **ENTER**. This time the computer does what you requested and prints the word HELLO on the screen. PRINT is one of the words in Color Basic, your computer's programming language.

Experiment 2: Write a Program

A computer program is a numbered list of instructions. Each of these instructions begins with a number and is similar to the instructions you just entered. Try this example:

```
10 PRINT "HELLO"
```

Introduction to Computing (continued)

When you press **ENTER**, the computer stores your instruction in its memory and moves the cursor down for your next instruction. Now enter this second instruction:

```
20 GOTO 10
```

When you press **ENTER**, the computer adds the second instruction to the program stored in the computer memory. Type **LIST** and press the **ENTER** key to see the complete program listed on the screen. Your screen should show:

```
10 PRINT "HELLO"  
20 GOTO 10
```

If your screen doesn't match the example, type your instructions again.

The word GOTO is another word or command in your computer's vocabulary. It is always written as one word and tells the computer to go to a particular line number to continue running the program.

Here's what your program will do: The computer will read your instructions in numerical order. When it reads instruction number 10, it will print HELLO on the screen. Then it will read instruction number 20 and again go back to instruction number 10. This will repeat over and over as the computer prints a constant stream of HELLO on the left side of the screen. When you're ready to run your program, type **RUN** and press **ENTER**. After you've run the program, press the red **BREAK** key to stop it.

Experiment 3: HELLOHELLOHELLOHELLO

List your program again by typing **LIST** and pressing **ENTER**. Your screen should show:

```
10 PRINT "HELLO"  
20 GOTO 10
```

You can change an instruction in a program by entering another instruction with the same line number. This erases the old instruction, replacing it with the new one. Now type a new instruction for Line 10, adding a semi-colon at the end, like this:

```
10 PRINT "HELLO";
```

When you press **ENTER**, this new instruction will replace the old Line 10. You will see the effect of this change when you run your program. Later, you will learn many other ways to use PRINT in your programs.

Check your new program by listing it again. Type **L I S T** and press **ENTER**. Your program will now look like this:

```
10 PRINT "HELLO";  
20 GOTO 10
```

Run your new program by typing **R U N** and pressing **ENTER**. With the semicolon added, your computer will print HELLO over and over again on the same line. When the printing gets to the right edge of the screen, it will move down to the next line automatically. You probably can't see HELLO because it's moving too fast.

You can stop the printing on the screen at any time by pressing **SHIFT** and the **@** key. Try this now to stop the program and see what is being printed. Notice that there is no OK or cursor on the screen. The program has only paused, and you can't enter new instructions at this time. Now press any other key on the keyboard to continue printing.

Stop the program by pressing the **BREAK key.** The blinking cursor and OK are back again, and you can now enter new instructions.

Experiment 4: HELLO NAME

If you don't see the cursor and OK, press **BREAK** to stop your program. Now list it again by typing **L I S T** and pressing **ENTER**. Your screen will show:

```
10 PRINT "HELLO";  
20 GOTO 10
```

This time you will add a new instruction to your program between Lines 10 and 20. Line numbers such as 10, 20, and 30 are often used because this leaves room between these lines to add new instructions.

You can use an added instruction to print any name you like. Notice that I have used a space before and after the name and that there is a semicolon at the end. This keeps all the letters from running together. Now think of someone you would like to say hello to and put their name in a print instruction, like this:

```
15 PRINT " LEANNE ";
```

Now type **L I S T** and press **ENTER** again. Your screen should show:

```
10 PRINT "HELLO";  
15 PRINT " LEANNE ";  
20 GOTO 10
```

Introduction to Computing (continued)

Now the computer will print HELLO, leave a space, print the name you've picked, leave a space, and repeat as before. Try it by typing **RUN** and pressing **ENTER**. You can pause by pressing **SHIFT** and **@**, and then continue by pressing any key. You can also stop your program by pressing **BREAK**.

Type **NEW** and press **ENTER**. This removes your program from the computer's memory. Now experiment on your own by writing a short program, using PRINT and GOTO instructions. You can put anything you like inside the quotation marks and print it on the screen. Here's an example of an interesting pattern:

```
10 PRINT "#-----";  
20 GOTO 10
```

Try printing other patterns with your program by changing Line 10 or by adding more lines. Use **LIST** to check your program and see any changes, then use **RUN** and see what your program does. When you've finished experimenting, press **BREAK** to stop and go to Lesson 1.

Lesson 1: Mathematician

Lesson 1: Mathematician

Arithmetic: + - * / =

Keyboard: INPUT

In this lesson you will use the calculating power of your computer to solve arithmetic problems. You will begin by typing in a short program that adds, subtracts, multiplies, and divides two numbers.

Begin by loading Computer Learning Lab and Lesson 1 from the cassette by following this procedure:

1. Place the cassette in the recorder, rewind, and press PLAY.
2. Type **CLOADM** and press **ENTER**. When the program starts loading you will see a blinking F and the word: COLORMON.
3. When you see OK and the color cursor, press the STOP button on the tape, then type **EXEC** and press **ENTER**.
4. Follow the directions on the title frame for Lesson 1.

Enter Mathematician

Enter this program into your computer by typing these eight instructions on your keyboard. Remember to press the **ENTER** key after you finish typing each line. If you make a typing error, you can back up to correct it by pressing the left arrow key (**←**). When you press the **ENTER** key, Computer Learning Lab will automatically check the line you've typed and show you any errors. To correct a line, just type it over again and press **ENTER**.

If you're in a hurry you can type **AUTO** and press **ENTER**, and Learning Lab will enter the program for you.

```
10 INPUT "A";A
20 INPUT "B";B
30 PRINT "A+B=";A+B
40 PRINT "A-B=";A-B
50 PRINT "A*B=";A*B
60 PRINT "A/B=";A/B
70 INPUT "GO AGAIN (Y,N)";K$
80 IF K$="Y" GOTO 10
```

Lesson 1: Mathematician (continued)

Run Mathematician

After you've entered all eight lines of your program into the computer, type **RUN** and press **ENTER**. The computer will print A? on the screen. Type **5** and press the **ENTER** key. The program will ask for the second number by printing B?. Type **2** and press **ENTER** again.

The computer will print four simple arithmetic problems and their answers on the screen, using 5 for the letter A and 2 for the letter B. Notice that the computer uses an asterisk (*) for the multiplication sign and a slash (/) to show division.

```
A+B=7
A-B=3
A*B=10
A/B=2.5
```

After the problems and answers are printed on the screen, the program will ask if you want to go again. Press **Y** and then press **ENTER** to signal yes.

When the computer asks for number A, type **25** and press **ENTER**. Then type **31** for B and press **ENTER** again. The computer will print the answers, as before.

Continue entering pairs of numbers and seeing the results. You can use numbers with decimals such as 21.5 and 13.03, if you like. If you type a zero for B, the computer will signal an error, as explained in Experiment 1.

How Mathematician Works

This section goes into more detail about your computer and explains what is happening when you run the program. You can continue with this lesson at this time, or go on to Lesson 2 and see how random numbers are used in programs.

To learn more about this program, stop it and list the instructions on the screen. After you've finished calculating numbers, type **N** and press **ENTER** to signal no when the question GO AGAIN (Y,N)? appears. This will stop the program. Now type **LIST** and press **ENTER** to list the program on the screen.

The flowchart diagram on page 10 shows what's happening when your program runs. Follow the chart and see if it helps you understand how this program works. The program uses the letters A and B to add, subtract, multiply, and divide two numbers. When you enter two numbers from the keyboard, the computer sets these letters (called variables) equal to the values you type.

Lesson 1: Mathematician (continued)

Next, the computer does the arithmetic and prints the answers on the screen. You see the numbers you typed added, subtracted, multiplied, and divided.

The question GO AGAIN (Y,N)? gives you an opportunity to repeat the program if you choose. This decision point in the program is shown by a diamond shaped box in the flowchart. If you type the letter **Y**, the program goes to Line 10 and repeats. If you type **N**, the program stops.

If you are interested in writing your own programs, follow the line-by-line description to see how each of the instructions works together in creating the program you've been using.

LINE 10 prints A? on the screen. The letter A is set equal to the number you type on the keyboard.

LINE 20 prints B? on the screen and sets B equal to the number you type.

LINE 30 prints A+B= and then prints the value of A plus B.

LINE 40 prints A-B= and the value of A minus B.

LINE 50 prints A*B= and the value of A times B.

LINE 60 prints A/B= and the value of A divided by B. Notice how the computer uses the symbols +, -, *, and / for addition, subtraction, multiplication, and division.

LINE 70 prints a message on the screen so that you can indicate whether or not you wish to try another problem. The variable K\$ is set equal to the letter you type on the keyboard.

LINE 80 sends the computer back to Line 10 if the letter you type is a **Y**. If you type any other answer, the program stops.

Experiment 1: Computer Messages

The computer is designed to print a message if it cannot follow your instructions. These messages are called Error Codes and they can be very helpful in understanding what might be wrong. In this program it's possible to make an error by entering a zero for letter B. When the computer tries to divide by zero, it will print an error message because division by zero won't work, even with a computer.

Lesson 1: Mathematician (continued)

Run your program again by typing **R|U|N** and pressing **ENTER**. Type any number for A and press the **ENTER** key. Enter a zero for B by typing **0** and pressing **ENTER**. When the computer gets to Line 60 and tries to divide by zero, it will stop and print: `?/0 ERROR IN 60`. This means that you tried to divide by zero in Line 60.

Now try a different mistake, and get a different error message. Run your program again by typing **R|U|N** and pressing **ENTER**, as before. This time, enter a letter instead of a number when the computer asks: `NUMBER A=?`. The message `?REDO` means do it again. Now type a number, press **ENTER**.

Experiment 2: Scientific Notation

For very large or very small numbers, your computer uses a special method for showing how many zeros there are. Run your program again and try these numbers for A and B to see an example:

Number A= 500000

Number B= 300000

The correct answer for 500000 times 300000 is 150000000000. The computer prints this number and all large numbers in Scientific Notation, like this: `1.5E+11`.

Actually, the number 150000000000 and the number `1.5E+11` are equal to the same value, even if they are written differently. If you start with the number 1.5 and move the decimal point 11 spaces to the right, you will get 150000000000.

To convert from scientific notation to regular numbers, just write the number to the left of the letter E, then move the decimal point as many times as the number on the right. In this example, the number to the left of the E is 1.5 and the decimal point is moved eleven places to the right (+11) to create: 150000000000.

Try this example:

NUMBER A= 5

NUMBER B= 4000

This time the division created a small number and the computer printed it in scientific notation. The answer for $5/4000$ is 0.00125 and the computer printed this number as `1.25E-03`. If you write 1.25 and move the decimal point three places to the left (-03), you will get the same answer: 0.00125.

Experiment 3: Command Mode Arithmetic

You don't have to write a program to do arithmetic on your computer. If OK is on the screen with the cursor (colored square) flashing, you are in Command Mode. You can type instructions (RUN, LIST, PRINT), and the computer will do as you ask.

If you don't have the computer in command mode with OK on the screen, press the red **BREAK** key in the upper right corner to stop your program.

Now print the answers to some arithmetic problems with these instructions. Just type these instructions and press **ENTER** after each one.

```
PRINT 2+3
```

```
PRINT 2 *5
```

```
PRINT 15+3+6.5
```

Experiment 4: Multiply or Divide, then Add or Subtract

Computers always do multiplication and division first, then addition or subtraction. Look at this problem and figure out the answer, then let the computer print the result by typing the instruction and pressing **ENTER**.

```
PRINT 9+6/3
```

Did you get the same answer as the computer? If you remembered the rule and divided before you added, you would have said: 9 plus 6 divided by 3 is the same as 9 plus 2, or 11. Get the rule backwards and your answer would have been 5.

If you remember that the computer does arithmetic in this order, you can't go wrong:

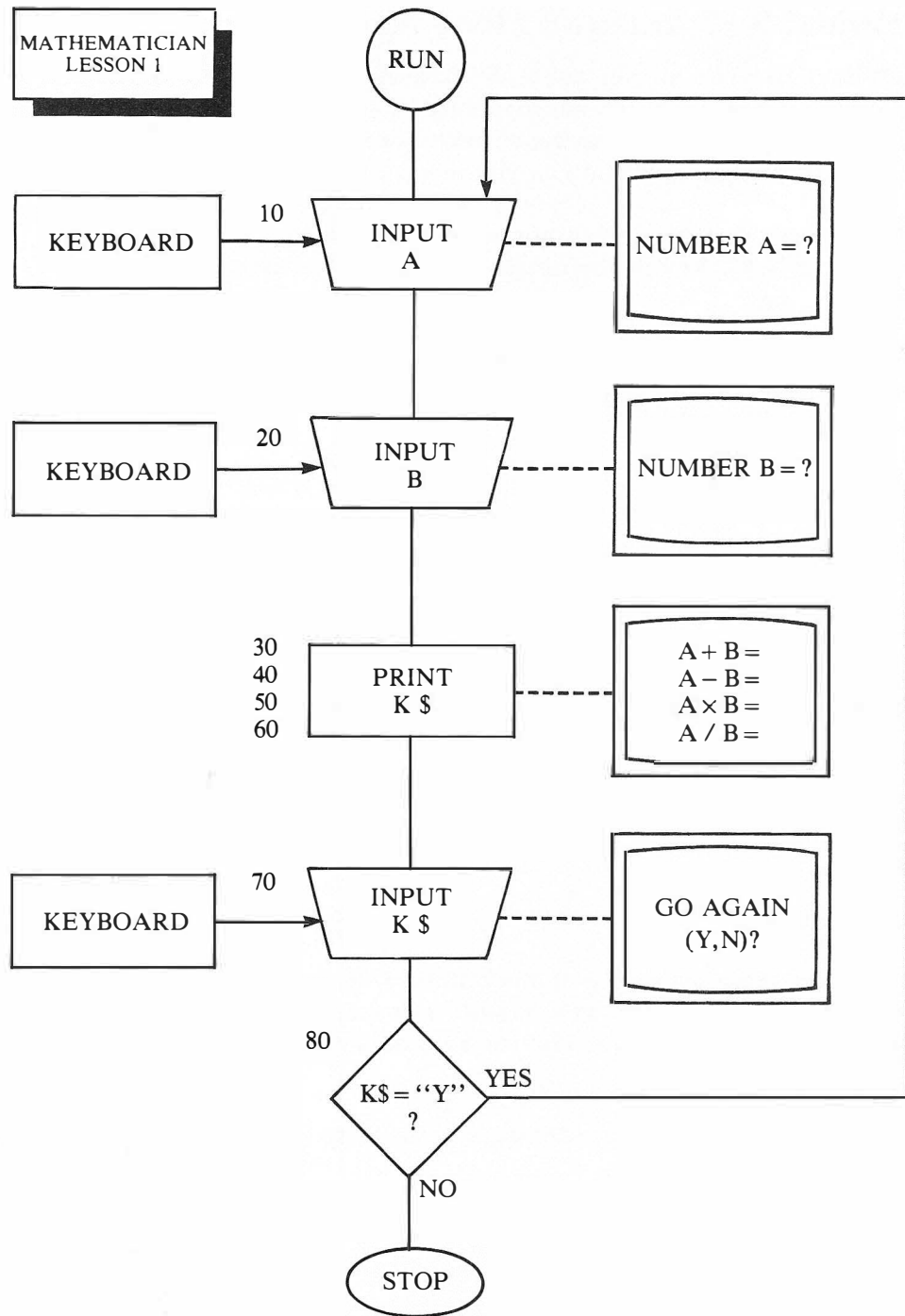
```
Multiply
Divide
Add
Subtract
```

Programmers sometimes keep this straight by remembering: My Dear Aunt Sally.

You can change the order in which the arithmetic is performed by using parentheses. All operations inside the parentheses are done first. In this example, the addition inside the parentheses is done, then the number is divided. The computer adds 9 and 6, then divides 15 by 3.

```
PRINT (9+6) / 3
```

Lesson 1: Mathematician (continued)



Lesson 2: Coin Flipper

Random Numbers: RND(X)

Decisions: IF/THEN

In this lesson you will see how your computer picks numbers by chance and how it makes decisions. You will use both of these functions many times in writing programs.

Random numbers are very useful whenever you're designing a game or creating a pattern and want the results to be different each time the program runs. Coin Flipper shows how the computer can simulate flipping a coin by picking heads or tails by chance.

The ability to make decisions is one of the main advantages that computers have over other types of machines. In this example the computer will decide what results to print on the screen after the coin has been tossed electronically.

Load Computer Learning Lab and Lesson 2 from the cassette with **CLOADM** and **EXEC**. When you see the program title, press **ENTER** to begin the lesson.

To program your computer to simulate flipping a coin, enter these eight instructions: If you would rather have Computer Learning Lab enter the program for you, type **AUTO** and press **ENTER**.

```
10 H=0
20 T=0
30 C=RND(2)
40 IF C=1 THEN H=H+1
50 IF C=2 THEN T=T+1
60 PRINT "HEADS: ";H,
70 PRINT "TAILS: ";T
80 GOTO 30
```

Run Coin Flipper

To run this program, type **RUN** and press the **ENTER** key. The computer will print a column for HEADS and a column for TAILS with their current totals. There isn't much for you to do but watch the

Lesson 2: Coin Flipper (continued)

scores add up. See whether heads or tails comes up more often. Then press the **BREAK** key to stop the program.

Run the program again and see if heads or tails comes up more often. If you run this program many times, or just let it run for a long time, the average number of heads and tails will be about the same.

How Coin Flipper Works

After you've seen enough, stop the program with the **BREAK** key. Type **L I S T** and press **ENTER** to list the program on the screen.

The flowchart diagram on page 16 will help you see what the computer is doing when this program runs. When you first run the program, the computer sets both heads and tails to zero. Then the computer picks a random number that's either a one or a two. If it's one, the total number of heads is increased. If it's two, the total number of tails is increased. The score is printed on the screen, showing the total number of heads and tails.

The program repeats and picks another random number, increases the total, and prints the score again. Each time the score is printed, the data on the screen moves up one line. This program loops or repeats over and over again until you stop it.

Each line in the program is described below to show how these instructions work together to simulate flipping a coin.

LINE 10 sets the variable H to zero. This letter is used to store the total number of heads. We could use any letter we wish, however H is a logical choice.

LINE 20 sets the variable T to zero. This letter stores the total number of tails.

LINE 30 creates a random number that's either a one or a two. The variable C is set equal to the number that the computer picks. This variable keeps track of the coin because we are using one for heads and two for tails.

LINE 40 makes the first decision: If C equals one, then the computer increases H, the total number of heads, by one. If C does not equal one, Line 40 is ignored.

LINE 50 makes the second decision and increases T if the variable C equals two. If C is not two, this instruction is ignored by the computer.

LINE 60 prints the first half of the scoreboard. The word "HEADS:" is printed, followed by the total number of times that heads has come up.

The computer prints the letters inside the quotation marks, then prints the value of H.

LINE 70 prints the second half of the score with the word "TAILS:" and the value of T. After this line is printed, the computer automatically moves the data on the screen up one line. When the program runs, the numbers appear to move up the screen while the names do not change position.

LINE 80 creates a "program loop" by sending the computer back to Line 30 to pick a new random number for C.

Experiment 1: Change the Loop

Notice that the program loop in Line 80 sends the computer back to Line 30 for another random number. What would happen if the program looped back to the beginning? It's easy to change Line 80 and find out. Type this new instruction for Line 80. When you press **ENTER**, this line replaces the original:

```
80 GOTO 10
```

Now type **LIST** and press **ENTER** to see if the new program matches this example:

```
10 H=0
20 T=0
30 C=RND(2)
40 IF C=1 THEN H=H+1
50 IF C=2 THEN T=T+1
60 PRINT "HEADS: ";H,
70 PRINT "TAILS: ";T
80 GOTO 10
```

The Computer Learning Lab only monitors your typing while you enter the original program. If you make an error while trying one of these experiments, just type your instruction again, and list your program to check it.

Now run your program and see the change. The program now goes back to Line 10 and sets H and T equal to zero each time the program loops. Since the totals always start at zero, the values for H and T are never larger than one.

If you prefer the original version of the program, change Line 80 back again. Stop the program with **BREAK**, type a Line 80 exactly the way that it was originally, and press **ENTER** to replace the instruction.

Lesson 2: Coin Flipper (continued)

Experiment 2: Scoreboard

The original program prints the totals on the bottom line of the screen. This causes the data to scroll or move up the screen each time the program loops. You can also print data anywhere else you like by using the PRINT @ command.

Type carefully because Learning Lab is no longer monitoring your input to the computer. Add the PRINT @ instruction to your program and print the totals over and over again on the line that's next to the bottom of the screen.

```
55 PRINT @447
```

When you press **ENTER**, this line is added to your program. Now type **LIST** and press **ENTER** to see your new program. If your added instruction doesn't match, just type it in again. Your screen should show:

```
10 H=0
20 T=0
30 C=RND(2)
40 IF C=1 THEN H=H+1
50 IF C=2 THEN T=T+1
55 PRINT @447
60 PRINT "HEADS: ";H,
70 PRINT "TAILS: ";T
80 GOTO 30
```

Now run this version and see a scoreboard that doesn't move. Line 55 starts the printing near the bottom of the screen each time, so the printing doesn't scroll. You can use the PRINT @ command to print at other locations by changing the number 447.

Experiment 3: Random Numbers

In this experiment you will see how the random number generator works. Stop your program with the **BREAK** key. Now type **NEW** and press **ENTER**. This clears your old program from the computer's memory.

Now enter this new program. Type carefully and press **ENTER** after each instruction. Don't forget the semicolon (;) at the end of the first instruction.

```
10 PRINT RND (3);
20 GOTO 10
```

Now list and check your program. When you run this program it will pick random numbers between one and three and print them on the screen. Run it now and see the result. You should see the screen fill with the numbers 1, 2, and 3.

Press **BREAK** to stop your program. Type **LIST** and press **ENTER** to see the program on the screen. Now change Line 10 again and replace the number 3 with some other number. When you run the new program, the computer will print random numbers between one and the number you have selected. Pick any number you like, and see what happens.

If you select the number zero, the random number generator will pick numbers between zero and one. Try this program and notice that all the numbers are between zero and one. If you leave off the semicolon the numbers will be printed on separate lines.

```
10 PRINT RND(0)
```

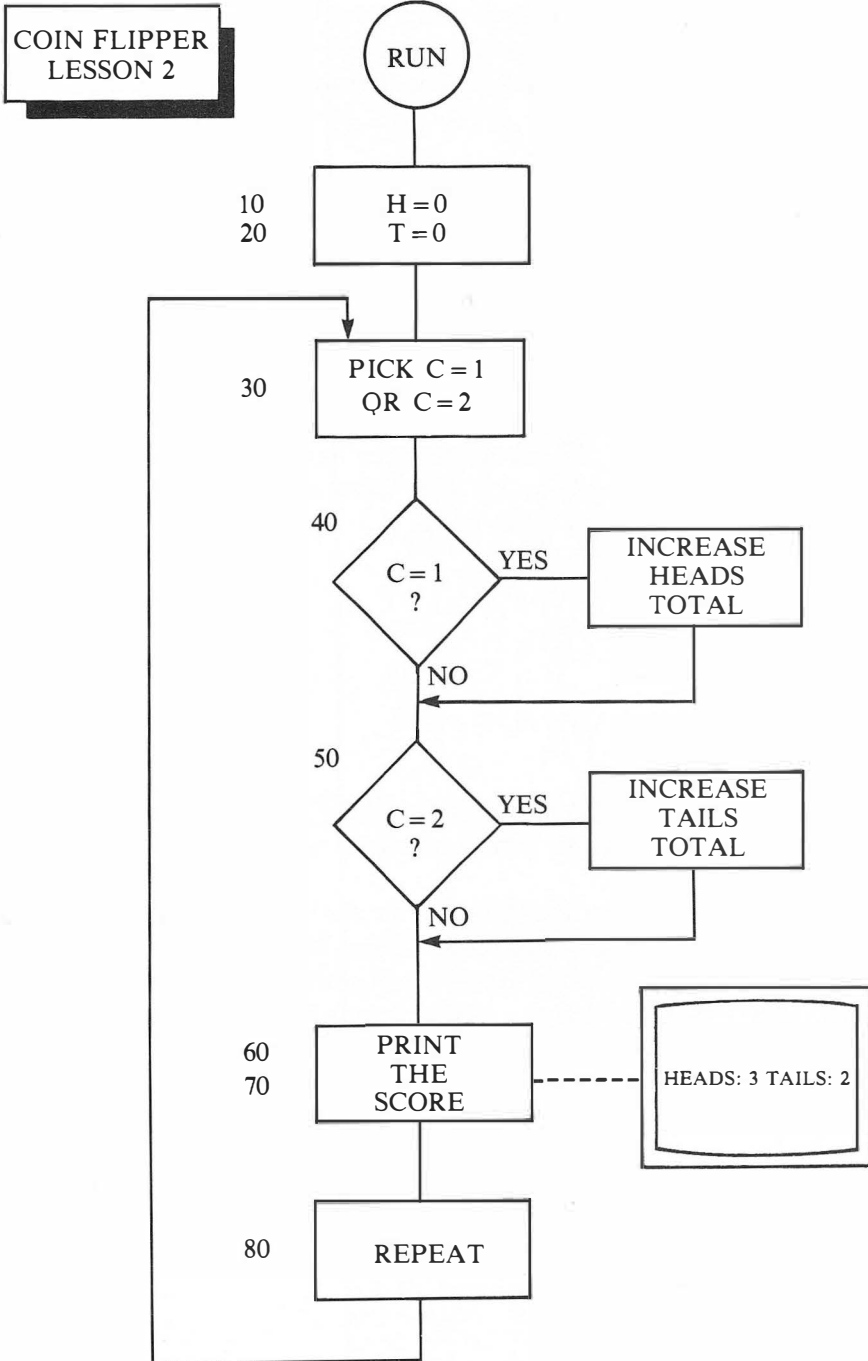
```
20 GOTO 10
```

Experiment 4: Halt!

With the last program running, press and hold **SHIFT** while you press the **@** key. This stops the program. Press any key to start it again.

There's no particular reason why the **@** key was chosen for this feature. If you can remember what it does, this trick makes it easy to stop a program or a listing and see what's going on without having to use the **BREAK** key. One word of caution. If you use the Halt! feature, you must press a key and start the program again before the computer will respond to any other commands.

Lesson 2: Coin Flipper (continued)



Lesson 3: Guessing Game

Comparison: > More Than
< Less Than
= Equal

This simple program lets you play a game with the computer. The rules are simple. When the computer picks a number, you try to guess what it is. If you guess wrong, the computer will give you a hint.

Load Computer Learning Lab and Lesson 3 from the cassette with **CLOADM** and **EXEC**. Press **ENTER** when you see the title frame for this lesson, then enter these seven instructions.

In this program, Line 20 is too long to fit on the screen in a single line. The last few characters will move down to the next line automatically. So don't press **ENTER** until you've finished typing the instruction.

Now type these instructions carefully, and press **ENTER** after each one. If you're in a hurry, you can type **AUTO** and press **ENTER** and Computer Learning Lab will enter the program for you.

```
10 X=RND(10)
20 INPUT "GUESS A NUMBER (1-10)":G
30 IF G=X GOTO 70
40 IF G>X THEN PRINT "LESS"
50 IF G<X THEN PRINT "MORE"
60 GOTO 20
70 PRINT G: "IS RIGHT!"
```

Run Guessing Game

Type **RUN** and press **ENTER** to run the program. The computer will pick a number from one to ten. Type your guess and press **ENTER**. If you're right, the computer prints the number you guessed and the words IS RIGHT! Guess too high, and the computer prints LESS; guess too low, it prints MORE.

After you've guessed the answer, run the program again. Each time the program runs, there's a new number for you to try to guess. Plan your strategy so that you can get the answer in the smallest number of guesses.

How Guessing Game Works

Stop the program by guessing the right answer or by pressing **BREAK**. Now type **LIST** and press **ENTER** to see the instructions on the screen. Compare these instructions with the flowchart diagram on page 21 and see what the computer is doing when the program runs.

When you first run the program, the computer picks a random number from one to ten. The variable X is set equal to this number. We could use any letter to store the computer's number, and X is as good as any.

Now it's your turn. The computer prints: GUESS A NUMBER (1-10) and waits for your input. The number you type is stored in G (for Guess). If your guess is correct, the number is printed with the words: IS RIGHT! Get the answer wrong, and you'll get a hint before the program loops back for another input.

Notice that the program loops until $G=X$ then it stops. The only way to end this program — other than with the **BREAK** key — is to guess the right number.

Here's what each instruction does:

LINE 10 sets the variable X equal to a random number from one to ten.

LINE 20 inputs a number for G and prints GUESS A NUMBER (1-10). When you type a number and press **ENTER**, the computer sets G equal to the number you type.

LINE 30 sends the program to Line 70 if you're correct because G equals X .

LINE 40 prints the message LESS if your guess is too high because G is greater than X ($G > X$).

LINE 50 prints MORE if you're too low because G is less than X ($G < X$).

LINE 60 sends the program back to Line 20. If you guess too high or too low, this instruction takes the computer back for another keyboard input.

LINE 70 is for winners only. This instruction prints the number you've guessed and the words IS RIGHT! when you finally get it.

There are several ways you can modify or improve this program. Start by repeating the next problem automatically and then add some more features.

Experiment 1: Autostart

You can add a decision at the end so that you won't have to type **RUN** each time you wish to repeat the game. These two instructions will print a message and give you the option to start again. Type these new instructions carefully and press **ENTER** after each one:

```
80 INPUT "TRY AGAIN?(Y,N)";A$  
90 IF A$="Y" GOTO 10
```

Type **LIST** and press **ENTER** to see the modified program on the screen. Check the added lines; if there are any errors, just enter them again. Now type **RUN** and press **ENTER** to run your new program. With these two lines your program will cycle automatically. When you see the message on the screen, type **Y** and press **ENTER** to go again.

Experiment 2: Clear Screen

Another nice feature is to clear the screen each time the program starts. The clear screen command (**CLS**) will do the trick. This instruction can be added between Lines 10 and 20, like this:

```
15 CLS
```

Now run your program again and see if you like this addition.

Experiment 3: Change the Odds

Now that you're getting pretty good at guessing the correct number, let's make the game a little more challenging. You can make Guessing Game as difficult as you choose by increasing the range of numbers used for picking the computer's random number.

The random function in Line 10 can be modified to select a number between one and fifty with this change:

```
10 X=RND(50)
```

Before you try this new program, it's a good idea to also change the message in Line 20 so that you will know what range to guess, like this:

```
20 INPUT "GUESS A NUMBER (1-50)";G
```

The computer won't be affected by your message and the program will work no matter what the words inside the quotation marks are. Notice

Lesson 3: Guessing Game (continued)

that the possible inputs are enclosed in parentheses in Line 20 (1-50) and in Line 80 (Y,N). This tells the user that a range of numbers from one to 50 can be used and that Y or N is expected as the answer to TRY AGAIN?

Experiment 4: Select the Difficulty

If you would like to be able to select the game's difficulty, you can add that feature with these modifications:

```
10 INPUT "DIFFICULTY (10-100)";D
17 X=RND(D)
19 PRINT "GUESS A NUMBER BETWEEN 1 AND";D;
20 INPUT G
```

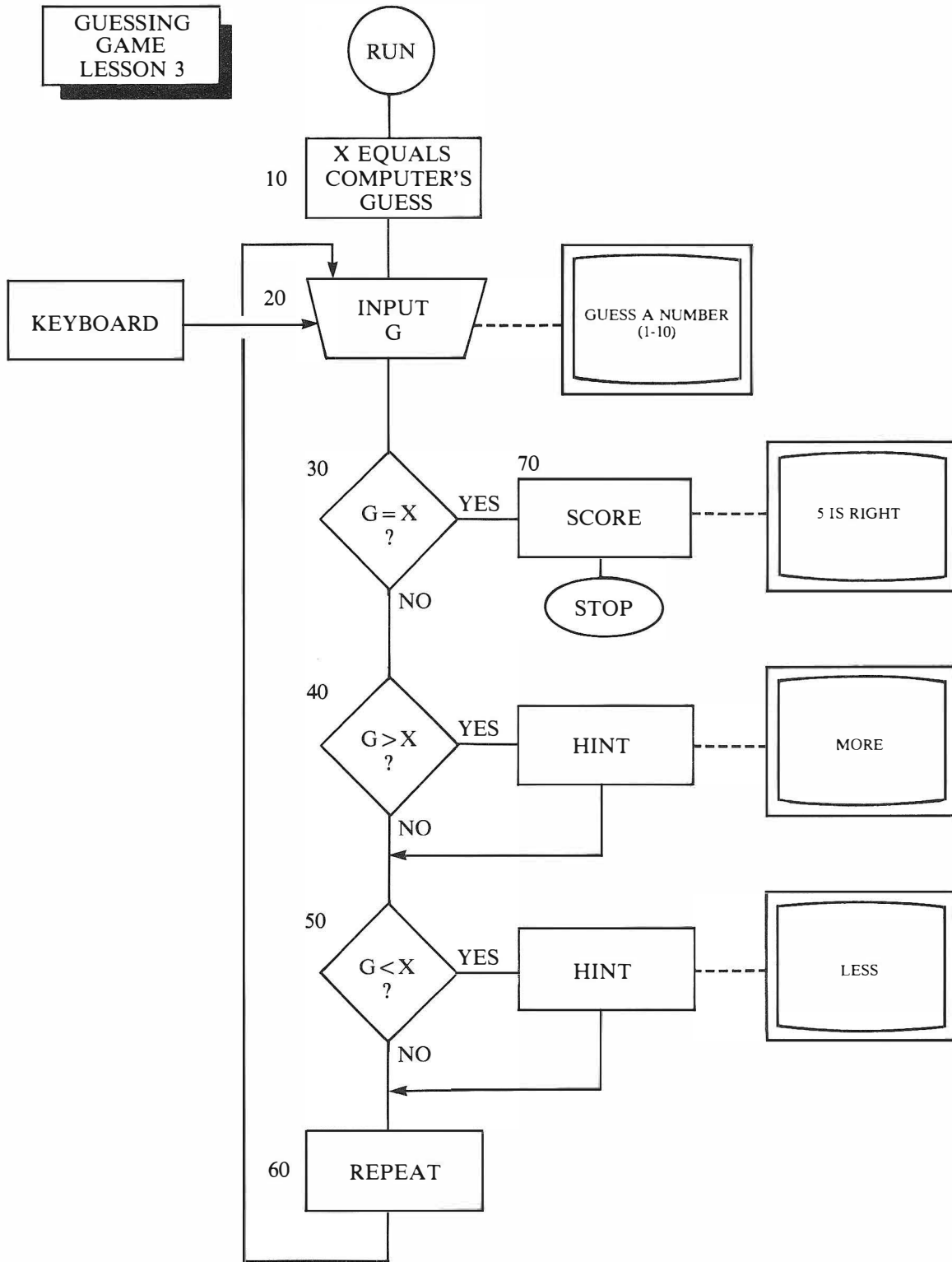
Here's how your complete program will look with all the changes we've suggested:

```
10 INPUT "DIFFICULTY (10-100)";D
15 CLS
17 X=RND(D)
19 PRINT "GUESS A NUMBER BETWEEN 1 AND";D;
20 INPUT G
30 IF G=X GOTO 70
40 IF G>X THEN PRINT "LESS"
50 IF G<X THEN PRINT "MORE"
60 GOTO 20
70 PRINT G; "IS RIGHT!"
80 INPUT "TRY AGAIN (Y,N)"; A$
90 IF A$="Y" GOTO 10
```

The print statement in Line 19 is necessary because you want to print a custom message to let the user know what the range is.

In these experiments you have taken a short, simple program and added several features. This is a good way to develop a program. It's often easier to get the central idea working, then add refinements.

Lesson 3: Guessing Game (continued)



Lesson 4: Average Calculator

Lesson 4: Average Calculator

Program loop: FOR/NEXT

Formulas

If you ever have to average a group of numbers, this program can be a big help. Just tell the computer how many numbers you have, enter each one, and get the average automatically. The key feature of this program is its ability to loop or repeat for each input you enter.

Load Computer Learning Lab and Lesson 4 from the cassette with **CLOADM** and **EXEC**. Press **ENTER** when you see the title frame for this lesson, then enter these ten instructions. If you make a mistake, the lab will show you exactly what to correct. If you would rather enter the program automatically, type **AUTO** and press **ENTER**.

```
10 CLS:T=0
20 PRINT "... THE AVERAGE CALCULATOR ..."
30 INPUT "HOW MANY NUMBERS":X
40 FOR L=1 TO X
50 PRINT "NUMBER":L;
60 INPUT "VALUE":Y
70 T=T+Y
80 NEXT L
90 A=T/X
100 PRINT "THE AVERAGE IS":A
```

Run Average Calculator

After you've entered the program, type **RUN** and press **ENTER** to begin. Pick a small number, like **5**, and then press **ENTER**. The program will request a value for the first number to be averaged. Type a value and press **ENTER** again. Continue until you've entered a value for each of the numbers you wish to average together.

When you've finished, the program will print the average of all the numbers you've entered. You can use decimals and even negative numbers in computing the average. Compute several averages. Then type **LIST** and press **ENTER** to see the complete program on the screen.

How Average Calculator Works

The important feature to notice is the program loop that repeats for each number you input. If you wish to average five numbers, for example, this loop cycles five times.

Each time the program loops it will request a number and wait for your input. Then the value is added to the total. If there are more numbers, the loop repeats. When the loop has cycled for the number of times you requested, the program continues with the next instruction and prints the result.

The FOR statement in Line 40 and the NEXT statement in Line 80 create the loop. All instructions between these lines are repeated for each new number to be averaged. The variable X is used to decide how many times the loop repeats.

LINE 10 is actually two instructions, separated by a colon. This is the same as:

```
10 CLS
```

```
11 T=0
```

You can put several instructions on a line, if you wish. Since both of these are used to initialize or start the program, they're typed on the same line for convenience.

LINE 20 adds a title to the program. The computer doesn't know or care what is printed inside the quotation marks, but titles make things look neat and orderly on the screen.

LINE 30 prints a message and sets the variable X equal to the number you type. This variable is used in the FOR/NEXT loop (Line 40), so the loop will stop when all numbers have been entered.

LINE 40 combines with Line 80 to form the loop. The variable L keeps track of how many times the loop has run. On the first pass, L is set equal to one. On each successive pass, L is increased by one. When L is larger than X, the loop stops and the program goes on to Line 80.

LINE 50 prints NUMBER and the value of L. This tells you which number you are entering.

LINE 60 sets the variable Y equal to the number you type.

LINE 70 adds the value of Y to the total, T. In this way, the numbers you enter are added together each time the loop circulates. After you've typed in all the values for Y, their total will be stored in T.

Lesson 4: Average Calculator (continued)

LINE 80 completes the loop that was started in Line 40. When the computer reads NEXT L, it checks to see if L is greater than X in Line 40. If not, the L is increased by one and the program returns to Line 40. If L is greater than X, the loop is finished and the computer goes to the next line in the program.

LINE 90 computes the average. Remember that T is the total of all numbers entered and X is the number of terms.

Experiment 1: FOR/NEXT

Clear the program from your computer by typing **NEW** and pressing **ENTER**. Now enter this program. Type each instruction carefully because Learning Lab is no longer monitoring your input and telling you about any typing errors.

```
10 FOR Q=1 TO 10
20 PRINT Q
30 NEXT Q
```

List your program and check it. When you run this program it will loop ten times, adding one to the variable Q each time. When $Q > 10$, the loop stops. Run it and see the numbers 1-10 on the screen.

Experiment 2: Loop Music

Now enter this program. The loop will operate as before, except that the variable will start at 30 and increase to 200. Each time the loop cycles, Z is increased by one.

The command SOUND is used to play notes from the TV speaker. The pitch or frequency is determined by Z in the command: SOUND Z,1.

```
10 FOR Z=30 TO 200
20 PRINT Z
30 SOUND Z,1
40 NEXT Z
```

Check your listing for accuracy as before, adjust the TV volume to about mid range, and run this program. If you would like to experiment with the SOUND command, replace the numbers 30 and 200 in Line 10 with two other numbers. The first number must be lower than the second. Values between 1 and 255 must be used or the program will not work. List your program, change Line 10 to the values you want, and run it again to hear the effect.

Lesson 4: Average Calculator (continued)

Experiment 3: Other Counters

We use FOR/NEXT loops because they are convenient. There are other ways to program the same effects. For example, the program above could also be written like this:

```
10 Z=30
20 PRINT Z
30 SOUND Z,1
40 Z=Z+1
50 IF Z<=200 GOTO 20
```

Try this program and see that it produces the same effect as the FOR/NEXT loop. The variable Z begins at 30 and increases to 200, creating a rising pitch. When Z is increased to 201, the IF condition in Line 50 is not met and the program stops.

Experiment 4: Auto Average

In Average Calculator you had to decide how many numbers you wanted to average together. With this program you can input as many numbers as you want, then get the average.

One way to stop and average all the inputs is to check for a special number. In this example we use the number 999 to tell the program that you're through and would like the answer. If you are averaging grades, for example, you could enter as many as you like. When you enter the number **999** for a grade, the program will realize that you are through and would like the average of all grades entered so far.

Clear your computer with a **NEW** command, then enter this program carefully. When you've finished, type **LIST** and press **ENTER** to check it.

Learning Lab is no longer checking your typing. If you make an error in one of the instructions, just type it again and press **ENTER** to replace it in the program.

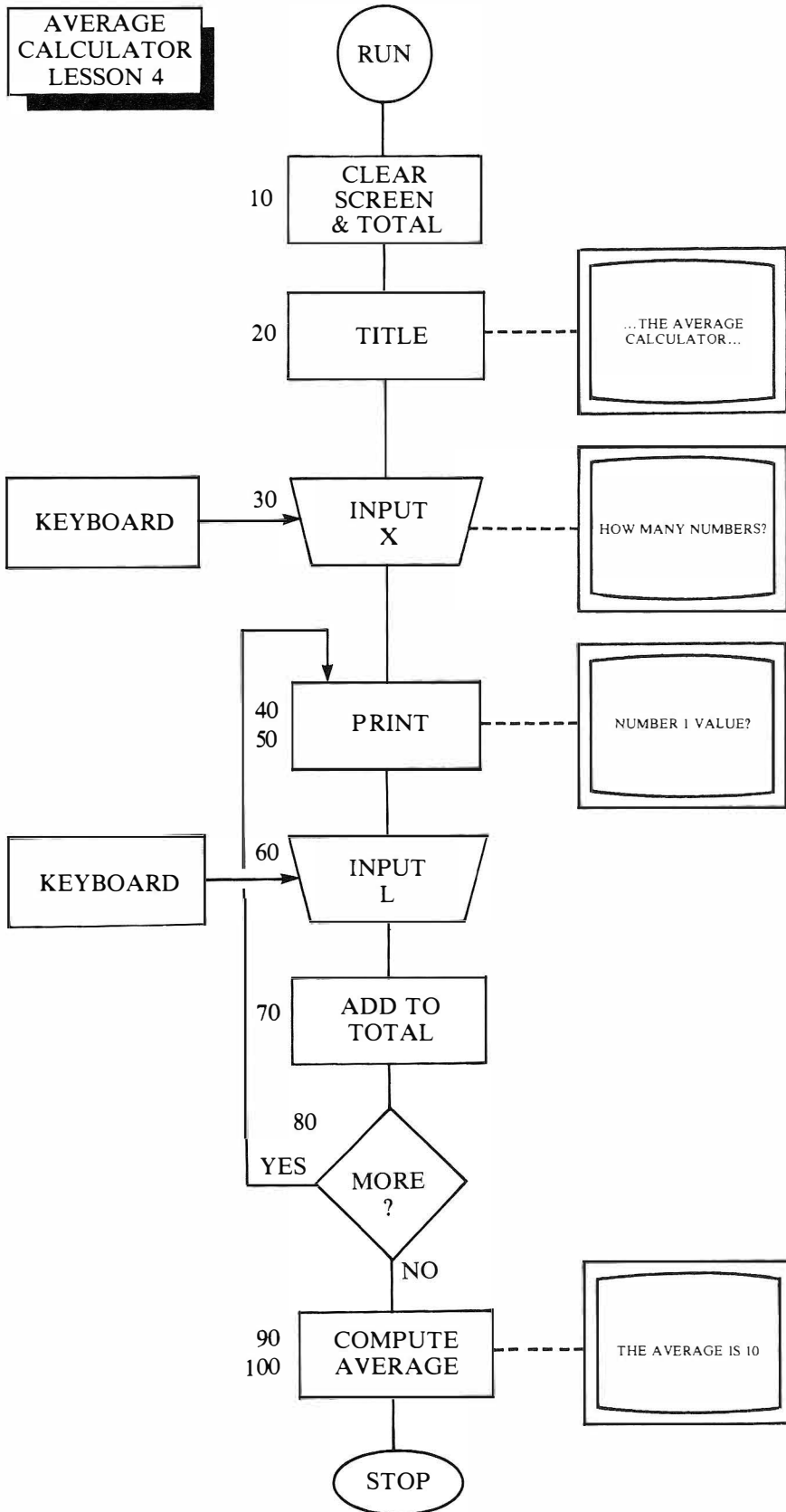
```
10 CLS:T=0:N=0
20 INPUT "VALUE";V
30 IF V=999 GOTO 70
40 T=T+V
50 N=N+1
60 GOTO 20
70 A=T/N
80 PRINT "THE AVERAGE IS";A
```


Lesson 4: Average Calculator (continued)

List your program and check it carefully. Then run it and enter several numbers. When you want to know the average, enter the number **999**. This signals the computer to go to Line 70 where the average is computed and printed.

After you've run this program a few times, see if you can draw a flowchart diagram to show how it works. The variable T stores the total of all numbers entered and N stores the number of terms that have been added so far. The decision in Line 30 should be drawn in a diamond shaped box with two outputs. If V equals 999, one path is followed. If V is not equal to 999, then another path is taken by the computer.

Lesson 4: Average Calculator (continued)



Lesson 5: Expressway

Graphics: POINT(B,V), PRINT TAB, CHR\$(N)

Sound: SOUND(F,D)

Motion: JOYSTK(N)

This program and your computer create a video game. You will use the right joystick to control a car on the expressway. The computer shows you a bird's-eye view from above the car as you zig and zag, trying to miss the traffic. Each time you collide with another car or run into the wall on either side, you will add points to your score. You can try and hit as many cars as possible, or see how long you can go without having an accident. The computer will create an endless stream of traffic for you to dodge and will give you a printout of your current score.

If you don't have joysticks, there is a special addition to this program that lets you use the arrow keys on the keyboard instead of the joystick. See Experiment 4.

Load Computer Learning Lab and Lesson 5 from the cassette with **CLOADM** and **EXEC**. To test your driving skill against the computer, enter these twelve instructions. If you would rather enter the program automatically, type **AUTO** and press **ENTER**.

```
10 X=15
20 T=0
30 A=INT(JOYSTK(0)/22)-1
40 X=X+A
50 Y=POINT(X*2,16)
60 IF Y>0 THEN X=X-A:T=T+1:SOUND 150,1
70 PRINT @256+X.CHR$(128);
80 PRINT @480.T;TAB(4) CHR$(207);
90 PRINT TAB(RND(26)+3) CHR$(175);
100 PRINT TAB(30) CHR$(207)
110 GOTO 30
```

Run Expressway

To run this program, type **RUN** and press the **ENTER** key. Use the right joystick and control your car by moving left or right. (If you don't have joysticks, please go to Experiment 4, page 34 now and modify the program so that the arrow keys on the keyboard can be used for driving.)

The program is designed so that the car on the screen "drives" much like a real car. Notice that the car will be moving left, not moving, or moving to the right. If the joystick is near the center, the car goes straight. Moving the joystick to either side causes the car to turn and continue turning until the stick is centered again.

Adjust the volume control on your TV to hear the "BEEP" each time you hit another car or the edge of the expressway.

The numbers on the left show your score. If your score goes above 99, however, the game no longer works properly because the program only allows for two digits in the number that represents your score.

To stop, just press the **BREAK** key. Each time you run Expressway you will start with a clean driving record and zero accidents. To make Expressway a competitive game, take turns with a friend and see who will cause the lowest number of collisions in a given time period.

How Expressway Works

Now see how your computer can be programmed to create a video game. If you haven't already stopped the program, press the **BREAK** key. Type **LIST** and press **ENTER** to display a list of the instructions on your screen. These eleven instructions are all that's required. Before you learn what each of these instructions does and how they work together in the game, look at the flowchart diagram on page 35 and see what the computer is doing while the game runs.

You have already seen that the car begins in the middle of the screen and that the score starts at zero when you type **RUN**. This step is shown in the first block in the diagram. The rest of the program is inside a loop; that is, the steps that begin with the joystick input and go through the final printing on the screen are repeated over and over again. This cycle or loop runs continuously until you press **BREAK** and stop the program.

Each time the loop runs, the computer moves the cars and prints a new picture on the TV, much like an animated cartoon or a movie. Now let's look at the loop in detail and see how each frame in this "movie" is made. The program begins each frame by checking the joystick to see if you are steering to the left, center, or right. Depending upon your input, the car may move right or left.

Lesson 5: Expressway (continued)

The first question in the program is "Have you hit anything?" To find out, the program checks the spot directly in front of your car to see if it is occupied by a colored block. If you're about to run into another car or the edge of the expressway, the program moves the car back again and scores an accident. In this way, you will appear to bounce as you hit obstacles.

When an accident is scored, the total is increased by one, and a "BEEP" is played in the speaker. Whether you have hit something or not, the car is drawn in its new position as the computer prints a blue box on the screen.

Now, several print commands are used to create the score, the left edge of the expressway, a new car in a random location, and the right edge. All of this data is printed on one line at the bottom of the screen. As this line is added at the bottom of the picture the entire screen moves up one line.

The effect that causes the expressway to "move" is created by the line feed after the last line is printed on the screen. This is similar to the carriage return on a typewriter, rolling the paper up one line. In this program the line feed moves the picture up one line each time the program loops. To see how this effect works, just press the **ENTER** key several times and see the listing move up the screen.

LINE 10 sets the value of X so that your car will begin in the center of the expressway: 15 spaces from the left edge of the screen. The variable X keeps track of the car's position. As X changes, the car moves left or right.

LINE 20 sets the total number of accidents, T, to zero.

LINE 30 sets A equal to the position of the joystick. The variable A will be 0 if the joystick is near the center, -1 if the joystick is moved to the left, and +1 if the joystick is moved to the right. You can copy these instructions whenever you wish to use the joystick in other programs.

LINE 40 adds the change in direction from the joystick, A, to your car's position, X. If the joystick is near the center, A equals zero and X does not change. Move the joystick to either side and X will change by one position each time the program loops.

LINE 50 examines the point directly ahead of your car and sets Y equal to the number of any colored block at that location. If the screen is blank at that point, Y is zero. There are twice as many dots or locations on the screen with the point command as there are with the print command. To examine the screen at column X and row 8, (the location directly below the car), POINT(X*2, 16) is used.

Lesson 5: Expressway (continued)

LINE 60 is a test to see if you have hit anything. Since Y is equal to the color number of any object in front of your car, you can test Y and see if you have had an accident. When the background color is present, Y is zero. If Y is greater than zero, the point you're examining is a color other than the background. This means you've hit another car or the edge. If there is an accident, three things happen. The joystick input is subtracted from the new location to keep you from driving through the edge and off the road. The total number of accidents, T, is increased by one. A "BEEP" is played to let you know that you've hit something.

LINE 70 prints a black square representing your car. The location 256 is on line number eight, at the left edge. Adding X to 256 plots a point on line eight that is X units from the left. The symbol CHR\$ is used with PRINT to place a colored box on the screen. Three values are used in this program with CHR\$(175) printing a blue box, CHR\$(128) printing a black box, and CHR\$(207) printing a white box.

LINE 80 prints the total number of accidents at location 480, which is the lower left corner of the screen. This is followed by a white square to mark the left edge of the expressway.

LINE 90 prints a blue square at a random location on the expressway. This is how the computer generates traffic for you to dodge. The PRINT TAB command tabs or moves to the right a certain number of places from the left margin before printing. In this instruction the number of tabs or spaces from the left is random so that the new traffic can appear anywhere on the bottom line.

LINE 100 prints a white box at the right side of the screen, 30 spaces from the left edge. This box prints the right-hand boundary. Notice that the semi-colon (;) is not used after this print command. A semi-colon would suppress or prevent the line feed after each line is printed. In Lines 70, 80, and 90 the semi-colon is used so that the screen **won't** move up automatically. After Line 100 we want the screen to scroll upwards, so the semi-colon is not used at the end.

LINE 110 sends the computer back to Line 30 to read the joystick, input a new direction, and create the next frame in Expressway.

Experiment 1: Time Limit

You can make Expressway a competitive game if you add a time limit. With this change the program will loop for a certain number of times and then stop automatically. You can compare scores with a friend or play against the computer for the lowest number of accidents. To do this, you will need a new variable to keep track of the number of times the program has cycled. You can use any letter you like, such as C, for "clock." Begin the program by setting the clock to zero:

```
15 C=0
```

The computer will count each time the program loops if you add this instruction:

```
45 C=C+1
```

Finally, add this test to stop the program after it has cycled 300 times.

```
105 IF C>300 THEN END
```

Run and test the program and see if you like the length of time it runs before stopping with a final score. If you want to run each trial for a longer or shorter time, just change the number 300 in this instruction.

Look again at the flowchart and notice that the first instruction that sets the clock to zero (Line 15) only operates once, when the program is first run. The two other instructions that add one to the time and check to see if the time is up are both inside the program loop.

Experiment 2: Add a Report Card

Add a scoreboard to show the results at the end of the game with these additions:

```
105 IF C>300 GOTO 120
```

```
120 PRINT "CONGRATULATIONS, YOU HAD ";T;"ACCIDENTS"
```

Now run the program. After 300 frames, the message is printed with the total number of accidents for that trial.

Experiment 3: Color the Traffic

Multi-colored traffic will replace the blue boxes if you use the random functions to set the color of the car that is added with each frame. This single change in Line 90 will do it:

```
90 PRINT TAB(RND(26)+3) CHR$(127+RND(8)*16);
```

The PRINT TAB works as before and spaces a random number of squares from the left margin. Instead of printing a single-color box, however, this new instruction prints a randomly colored box.

Here is a list of the color numbers that are picked in Line 90 by the instruction (127+RND(8)*16):

```
143 green  
159 yellow  
175 blue  
191 red
```

Lesson 5: Expressway (continued)

```
207 white
223 light blue-green
239 pink
255 orange
```

Notice that color 143 (green) is the same color as the background. This green-on-green phantom car can nonetheless be hit, so be careful.

Experiment 4: Keyboard Control

If you don't have joysticks you can use the left and right arrow keys to control the car's motion. Just type these instructions and press the **ENTER** key after each one. Adding these lines to your program will let you use the left and right arrow keys to control the car.

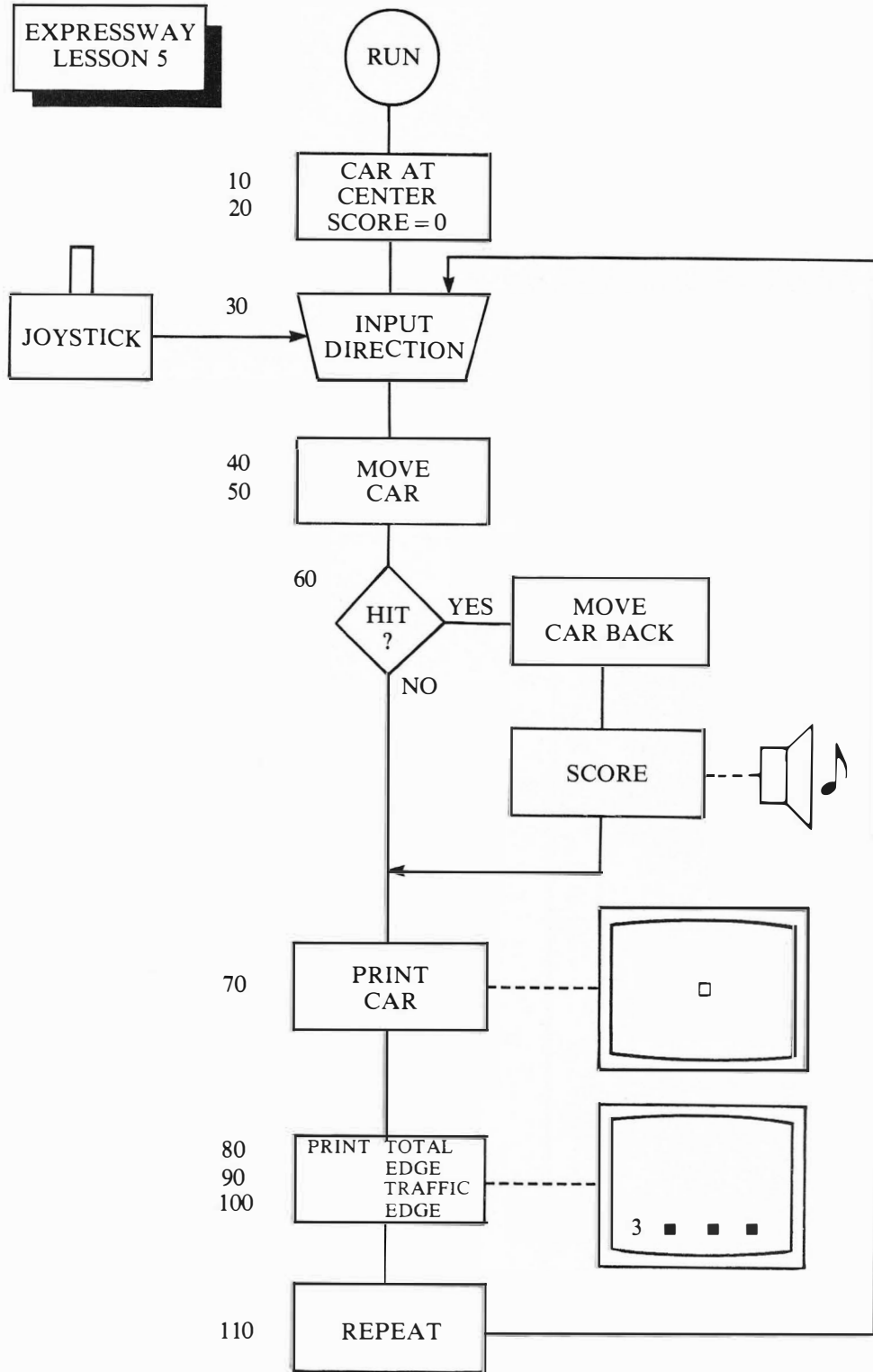
```
30 K$=INKEY$
32 A=0:IF K$=" " GOTO 40
34 IF ASC(K$)=8 THEN A=A-1
35 IF ASC(K$)=9 THEN A=A+1
```

Line 34 checks to see if a left arrow was pressed. The next instruction checks for a right arrow. The variable A is adjusted accordingly.

With this addition to the program, the variable A is adjusted with the arrow keys and with the joystick.

Lesson 5: Expressway (continued)

EXPRESSWAY
LESSON 5



Lesson 6: Counting Machine

Lesson 6: Counting Machine

Program Loop: FOR/NEXT/STEP

In this lesson you will see how to program your computer to count. Since the computer does many things using numbers, this ability will be useful in a wide variety of programs.

The method often used for counting up or down is the "program loop." With a loop, your computer will repeat some of the instructions in a program over and over again. The FOR/NEXT commands make it easy to create loops in your programs. With counting loops you can increase or decrease numbers easily.

Load Lesson 6 from the cassette using **CLOADM** and **EXEC**. Enter these 11 instructions yourself, or type **AUTO** and Computer Learning Lab will enter the program for you.

```
10 CLS
20 INPUT "START":A
30 INPUT "END":B
40 INPUT "STEP":S
50 PRINT "FOR L =":A;"TO":B;"STEP":S
60 FOR L=A TO B STEP S
70 PRINT L;
80 NEXT L
90 PRINT
100 INPUT "GO AGAIN (Y,N)":A$
110 IF A$="Y" GOTO 10
```

Run Counting Machine

When you've entered the complete program, run it. The program begins by asking you to input a number for the variable START. Type **1** and press **ENTER**. Next, type the number **50** for END and press **ENTER**. Finally, input the number **1** again for STEP and press **ENTER**.

When you enter the final number, the program will automatically begin counting from 1 to 50. With the STEP set at 1, the program counts by ones: 1, 2, 3, 4, etc.

Lesson 6: Counting Machine (continued)

This program contains an automatic repeat feature. The computer will ask you if you wish to go again. Type a **Y** and press **ENTER** to run the program again. Now enter these values, and remember to press the **ENTER** key after each one.

START? 10

END? 200

STEP? 5

This time the computer counts from 10 to 200 by fives. Now try counting backwards. Select **Y** to go again and enter the following numbers. Notice that the START is higher than the END and that the STEP is negative.

START? 250

END? 1

STEP? -4

One more example will show you how to step by a fraction. When you try these numbers, the printing will scroll off the screen as the computer counts from 5 to 70 by .5, or one-half, steps.

START? 5

END? 70

STEP? .5

Try any other combinations you like for the START, END, and STEP. If you select a wide range, the computer can take days to print all the numbers. To cause the program and the printing on the screen to pause, just press **SHIFT** and the **@** key. Pressing any other key continues the program again.

After you've tried several combinations, stop the program by selecting **N** when you have the option to repeat.

How Counting Machine Works

The first part of the program clears the screen, inputs the variables for the counting loop, and prints a remainder on the screen so that you can see what the START, END, and STEP are.

All of the counting is done in the FOR/NEXT loop (Lines 60, 70, and 80). The variable L is used to keep track of the loop. When the program first reaches Line 60, L is set equal to the starting value:

Lesson 6: Counting Machine (continued)

START. Using numbers from the first example, L would be set to 1. We have chosen the variable name A to hold this value. We could have used any other letter or name, such as "B", "BEGIN," "FIRST," or even "JOE."

Then the computer prints the value of L on the screen.

When the computer reads NEXT L in Line 80, it compares L and B. If L is greater than B, the loop is finished, and the program continues with the next instruction: Line 90. If L is not greater than B, the computer repeats the loop by going back to Line 60 and adding the value of S to L.

After the loop has finished counting, the program continues. The message GO AGAIN (Y,N)? is printed, and the variable A\$ is set equal to the letter you type. Type **Y**, press **ENTER** and the program repeats.

Notice that the variable A is used to store a number, and the variable A\$ is used to store a letter.

LINE 10 clears the screen.

LINE 20 sets the variable A equal to the keyboard input.

LINE 30 sets the variable B equal to the keyboard input.

LINE 40 sets the variable S equal to the keyboard input.

LINE 50 prints a message to show how the instruction for the FOR/NEXT loop will look with the numbers that have been entered.

LINE 60 begins the FOR/NEXT loop. The variable L is set equal to A when the loop begins. Each time the loop operates, the value S is added to L.

LINE 70 prints the value of L on the screen. The semicolon (;) causes the next number to be printed one space to the right of this number.

LINE 80 completes the loop. When the computer reads NEXT L it returns to the line where FOR L was used. If L is less than or equal to the final value, it is increased by the number following the word STEP in Line 60. If L is more than the final value, the computer does not loop back to Line 60; it goes to the next line in the program.

LINE 90 prints a new line after all numbers have been printed on the screen. This makes it easier to read the message at the end.

Lesson 6: Counting Machine (continued)

LINE 100 prints GO AGAIN (Y,N)? and sets the variable A\$ equal to the letter you type. You could have used K\$, B\$, MORE\$, or any other name that ends in a dollar sign to hold the letter you input from the keyboard.

LINE 110 goes to Line 10 and repeats the program if you input the letter **Y**. With any other input the program stops.

You will use loops like this in many different kinds of programs. Begin these experiments by adding sound so that the loop is more fun. Then try writing program loops of your own.

Experiment 1: Loop Music

Add this instruction to your program, and the computer will play a note when it prints a number on the screen. The SOUND command converts a number between 1 and 255 into a tone. You will use the variable L to set the pitch of the tone. The duration will be set at 1.

```
75 SOUND L,1
```

When you press **ENTER**, this new instruction is added to your program. Type **L I S T** and press **ENTER** again, and see that the new program contains this added instruction.

Adjust the volume on your TV to a normal level. Run the program and try these inputs:

```
START? 1
```

```
END? 255
```

```
STEP? 1
```

Experiment 2: More Music

You just heard the full range of the SOUND command. As L varied from 1 to 255, the sound varied from low to high. Try the next loop and reverse the process. Notice that this time we're only playing every fifth step.

```
START? 255
```

```
END? 1
```

```
STEP? -5
```

Experiment 3: Illegal Action

Now it's time to try something that won't work and see what happens when you make a mistake. You can't harm your computer by typing in the wrong instruction. So always feel free to experiment, like this:

```
START? 1
```

```
END? 500
```

```
STEP? 2
```

When you run this loop, the computer will count to 257 and then print ? FC ERROR IN 75 on the screen. This is an error message from the computer and it tells you that you have an "illegal function call in Line 75."

This message doesn't mean you've broken the law. It simply means that you tried to use a number that's out of range for the SOUND command. When L reached 257, the command SOUND L,1 wouldn't work because the pitch was out of range. The error message told you exactly what was wrong.

There are two ways to fix this problem. You can limit the range of the counting loop to the numbers 1 through 255, or you can change Line 75.

Experiment 4: Make it Legal

Here's a way to keep the numbers for SOUND between 1 and 255. This first instruction creates a new variable SND that converts the number in L to a number between zero and 254. The next instruction adds 1 to SND and uses this for the SOUND command. In this way, we avoid the problem of a function call error. Add these lines to your program:

```
75 SND=L-INT(L/254)*254
```

```
77 SOUND SND+1,1
```

List your program and check these lines carefully, then run the same experiment again. This time the sound will stay within range, and there won't be any messages from the computer.

```
START? 1
```

```
END? 500
```

```
SKIP? 2
```

Experiment 5: Space Sounds

When a program doesn't work quite right, we say that it has a "bug" or a mistake. The problem you had with the FC error is a bug that you've fixed by changing the program. Now you can count forwards or backwards and step by any amount you choose, and the program will always keep the number for the pitch within the correct range. Try these examples, then experiment with your own ideas:

```
START? 1      END? 9000   STEP? 51
```

```
START? 1      END? 9000   STEP? 53
```

```
START? 5000   END? -5000  STEP? -67
```

```
START? 5000   END? -5000  STEP? -101
```

Experiment 6: Writing Loops

The program for this lesson has shown you how loops operate. When writing loops for your own programs, you usually won't need the extra instructions for inputting data and repeating the program. There are several ways you could duplicate the examples with short programs.

Clear your computer by typing **NEW** and pressing **ENTER**, then run these programs and see that they duplicate the results of the first examples in this lesson. We used the variable L in the lesson. You can use any variable you like, for example:

```
10 FOR Z=1 TO 50
20 PRINT Z;
30 NEXT Z
```

```
10 FOR X=10 TO 200 STEP 5
20 PRINT X;
30 NEXT X
```

```
10 FOR COUNT =250 TO 1 STEP -4
20 PRINT COUNT;
30 NEXT COUNT
```

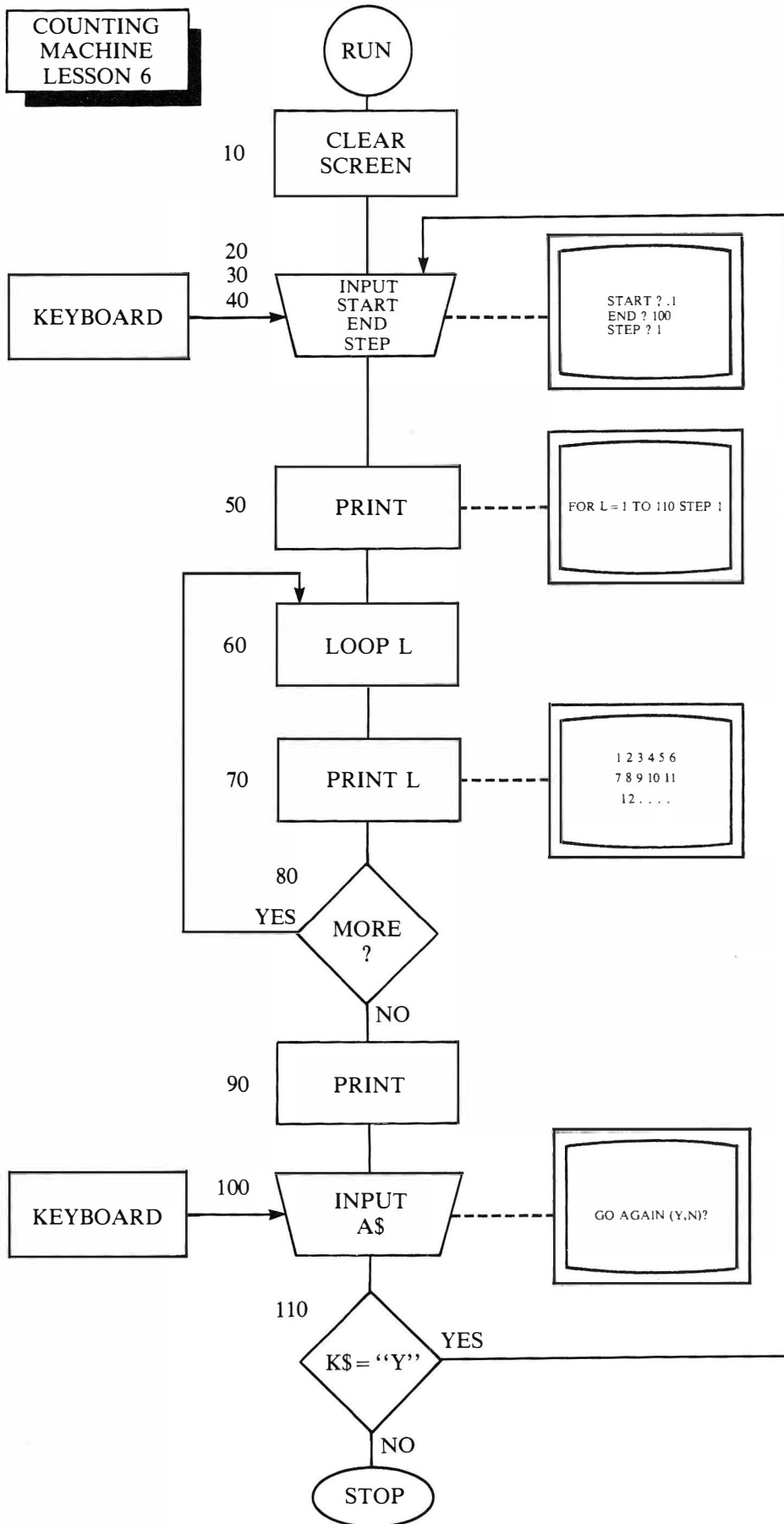
```
100 FOR NUMBER=5 TO 70 STEP .5
101 PRINT NUMBER;
102 NEXT NUMBER
```


Experiment 7: 99 Bottles

Here is a favorite loop program from my friend Ted Nelson:

```
10 CLS
20 FOR N=99 TO 1 STEP -1
30 SOUND N,5
40 PRINT @0,N;"BOTTLES OF BEER ON THE WALL"
50 PRINT N; "BOTTLES OF BEER."
60 PRINT " IF ONE OF THOSE BOTTLES"
70 PRINT " SHOULD HAPPEN TO FALL . . ."
80 PRINT N-1; "BOTTLES OF BEER ON THE WALL"
90 NEXT N
100 PRINT "ALL GONE!"
```

Lesson 6: Counting Machine (continued)



Lesson 7: Kaleidoscope

Graphics: SET(H,V,C)

Color

One of the most interesting things you can do with your color computer is to create patterns and designs. Artists who use computers in their work often program randomness into their art and use symmetry to create a balanced appearance. In this lesson you will use a program to create a symmetrical pattern that never repeats. Randomness is used in forming the design and in selecting the colors.

Load Lesson 7 from the cassette with **CLOADM** and **EXEC**. Press **ENTER** when you see the title, then enter these nine instructions. Be sure to press **ENTER** after each one. If you want to enter the program automatically, type **AUTO** and press the **ENTER** key.

```
10 CLS 0:C=3
20 X=RND(32)-1
30 Y=RND(16)-1
40 SET (X,Y,C)
50 SET (X,31-Y,C)
60 SET (63-X,Y,C)
70 SET (63-X,31-Y,C)
80 IF RND(20)=1 THEN C=RND(8)
90 GOTO 20
```

Run Kaleidoscope

When you run this program, the screen will clear, and a symmetrical pattern of colored dots will appear. As the pattern builds, notice that the screen is actually divided into four sections and that the same pattern repeats in each section. The screen is acting like a mirror with the pattern reflected in the four corners.

Gradually the colors will begin to change, and the pattern will fill the screen. You can stop the pattern at any time by pressing **SHIFT** and the **@** key at the same time. To continue the program, press any other key. When you would like to see a different pattern, press **BREAK**, type **RUN**, and press **ENTER**.

How Kaleidoscope Works

We are using the random number generator in the computer to create the pattern. As the program cycles, a single colored dot is placed in the top-left section of the screen. Then this position is repeated in each of the other three sections.

The decision to select a new color is based on a 1-in-20 probability. Each time the program cycles, the computer generates a random number between 1 and 20. If the number is 1, a new color is picked at random; for any other number, the color stays the same. This probability creates, on the average, about 20 dots of any color before a new color is picked. Actually, since the computer may pick the same color again, the rate of change is slightly less than this.

The random number generator is used to pick values for X and Y . Then these variables control the positions of all four dots that are plotted on the screen.

The SET command places a dot of any color anywhere on the screen. The screen is 64 dots or positions wide and 32 dots high.

LINE 10 clears the screen and colors it black (color 0).

LINE 20 picks a random number for X that is between zero and 31.

LINE 30 picks a random Y between zero and 15.

LINE 40 plots a dot in the top-left section of the screen. If X and Y are zero, the dot is in the top-left corner. If both variables are at their maximum (31 and 15), the dot is next to the center of the screen.

LINE 50 plots a matching dot in the top-right section of the screen.

LINE 60 plots a dot in the lower-left section. This dot is in the lower-left corner if X and Y are zero and near the center if they are maximum.

LINE 70 plots a matching dot in the bottom-right section.

LINE 80 creates a random number between 1 and 20. If this number is 1, the computer picks a new color number between 1 and 8.

LINE 90 sends the computer back to Line 20 to pick two new numbers for X and Y .

One of the ways you can vary the appearance of this program is to alter the rate at which the colors change. Increasing the number 20 in Line 80 will cause the colors to change less often because the odds of the random number being equal to 1 are less. Here are some other ways you can modify this program:

Experiment 1: Auto Repeat

Try adding a loop around the entire program. When the loop is finished, the computer will go to the next line in the program and clear the screen. Add these lines carefully, then list your program to check it.

```
15 FOR N=1 TO 100  
  
90 NEXT N  
  
100 GOTO 10
```

Here is how your complete program should look:

```
10 CLS 0:C=3  
15 FOR N=1 TO 100  
20 X=RND(32)-1  
30 Y=RND(16)-1  
40 SET (X,Y,C)  
50 SET (X,31-Y,C)  
60 SET (63-X,Y,C)  
70 SET (63-X,31-Y,C)  
80 IF RND(20)=1 THEN C=RND(8)  
90 NEXT N  
100 GOTO 10
```

Now the program will loop between Line 15 (FOR) and Line 90 (NEXT) in a loop. After the loop has cycled 100 times, the program will go on to Line 100. As you can see, this will cause the program to start over again at Line 10. Try this change and see if you like the program clearing and starting over after 100 positions have been plotted.

Now consider changing Line 15 so that the pattern builds for a longer time before clearing. I prefer the pace when the program cycles 500 times. Try it with this instruction:

```
15 FOR N= 1 TO 500
```

Experiment 2: Add a Tune

Computer art is not limited to visual effects. Try adding a tune with this instruction:

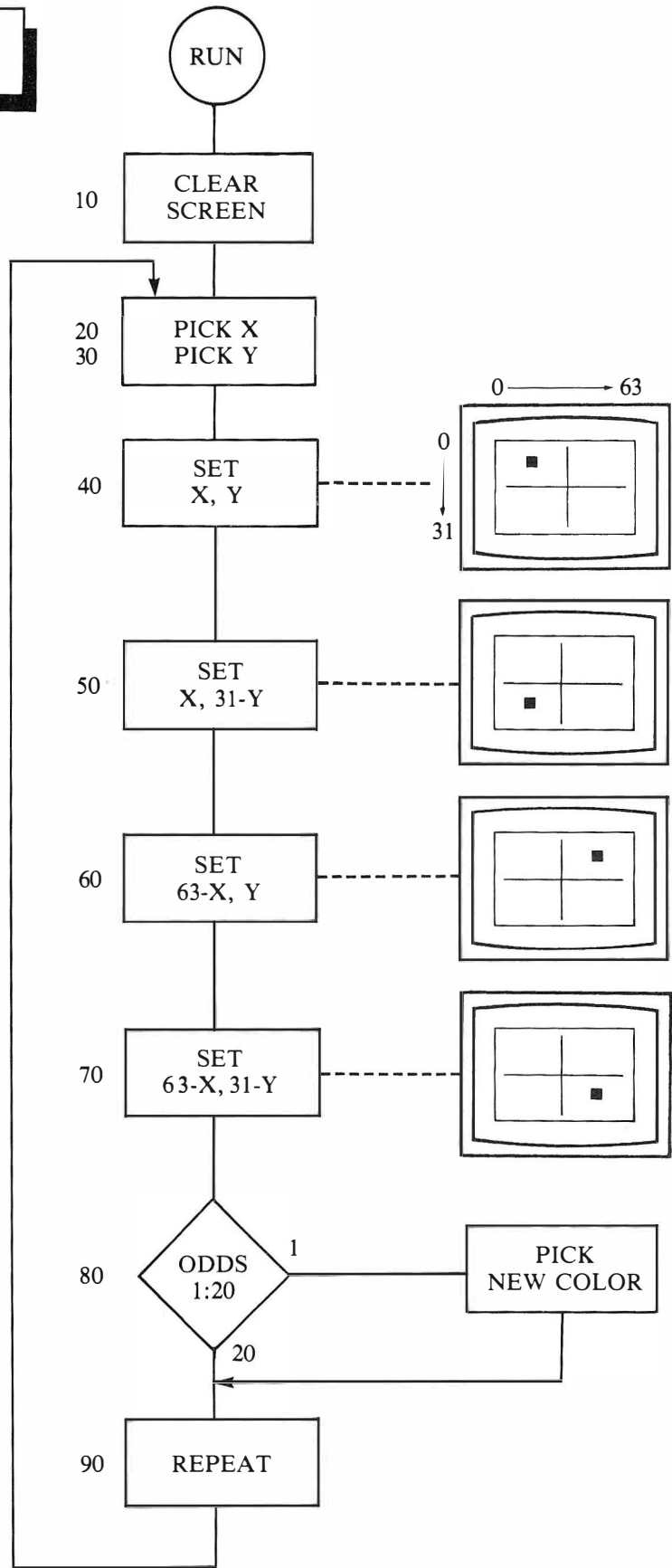
```
75 SOUND 6*X+20,RND(3)
```

With this instruction your program will create a note that varies in pitch as X varies the position of the dots. The second number after the SOUND command determines the duration of the tone. In Line 75 this duration is set to a random number between 1 and 3 to create a rhythm.

Some people like the tune that this program plays, and some don't. Try your own recipe and see if you can find a method for generating computer music that appeals to you.

Lesson 7: Kaleidoscope (continued)

KALEIDO-
SCOPE
LESSON 7



Lesson 8: Decision Maker

Branching: ON-GOTO

Keyboard: INKEY\$

Many methods for determining the course of future events are based on chance. From Tarot cards to I Ching, random events are combined with intuition to help us see our fate. While the computer is short on intuition, it's a master of random events.

In this lesson we will use two new commands to create a computerized information service or advisor. The ON-GOTO command makes it easy to branch to a wide variety of possibilities in a program. We will use this capability to create the messages displayed on the screen. With the INKEY\$ command we can read a key from the keyboard without stopping a program. Unlike the INPUT command, INKEY\$ lets you type a single key and effect a running program.

Load Lesson 8 from the cassette with **CLOADM** and **EXEC**. To consult your own computer, just enter these instructions. You can also type **AUTO** and press **ENTER** if you would like Computer Learning Lab to enter the program for you.

```
10 CLS
20 PRINT @133, "... ASK THE COMPUTER ..."
30 IF INKEY$="" GOTO 30
40 ON RND(4) GOTO 50,60,70,80
50 PRINT @239, "YES":GOTO 90
60 PRINT @240, "NO":GOTO 90
70 PRINT @238, "MAYBE":GOTO 90
80 PRINT @237, "NO WAY":GOTO 90
90 FOR DLY=1 TO 500:NEXT DLY
100 GOTO 10
```

Run Decision Maker

When you first run the program, the screen clears and prints the title. Think of a question you wish to have answered, then tap the space bar gently. The computer will answer your inquiry and clear the screen again.

Lesson 8: Decision Maker (continued)

Now think of your question, type `RUN`, and press `ENTER`. When you're ready, press the space bar. If you don't like the answer you get, press the space bar again. The computer's advice, on the average, is correct about 50% of the time.

How It Works

The first two instructions clear the screen and print the title.

The keyboard is read by `INKEY$`. That is, the computer checks the keyboard and automatically inputs any key that is pressed. If the input from the keyboard is a null string (meaning that no key has been typed), the program loops back to the same instruction and reads the keyboard again. In this way, the program "hangs" or simply waits for you to press any key. After you press a key, the program continues.

The `ON-GOTO` command simply transfers the program to one of several lines, depending on the value of `RND(4)`. We use this feature to select which message to print.

After printing one of the messages, the computer pauses for a short delay. Then the program restarts, prints the title, and waits for you to press any key. As you can see by the program, it doesn't matter which key you press to get your answer.

LINE 10 clears the screen.

LINE 20 prints the title starting at screen location 133.

LINE 30 reads the keyboard. If no key has been pressed, this instruction is repeated. As soon as any key is pressed, the computer goes to the next line in the program.

LINE 40 sends the computer to one of the listed lines, depending on the value of `RND(4)`. If this value is one, the computer goes to the first line in the list: Line 50. If it is a two, the computer goes to the second line number in the list: Line 60. Similarly, a value of three or four will select Lines 70 or 80.

The value that selects the line does not have to be `RND(N)`. Here are other forms of the `ON-GOTO` command:

```
ON A GOTO 20,30,400,55
ON X-3 GOTO 300,400,500
ON CHOICE GOTO 1000,2000,7000
```

LINE 50 prints YES starting at location 239, then goes to Line 90.

LINES 60-70-80 print their messages and go to Line 90.

LINE 90 causes a time delay while the variable DLY is increased from one to 500. This FOR/NEXT loop is two separate instructions, written on the same line and separated by a colon.

LINE 100 sends the computer back to the beginning of the program.

We can add a lot to this program with some convincing sound effects and a wider variety of answers. If Decision Maker is still running, stop the program by pressing **BREAK**. When OK appears, type **LIST** and press **ENTER** to see the program on your screen. Now you can add to or modify these instructions.

Experiment 1: Music to Think By

Let's make the computer sound like it's actually trying to think up an answer with these additional lines. Type carefully and don't forget to press **ENTER** after each line.

```
35 FOR S=1 TO 10  
36 SOUND RND(100)+150,RND(6)  
37 NEXT S
```

Type **LIST** and press **ENTER** to see that you've entered these lines correctly. If not, just type them over and try again.

This short FOR/NEXT loop will play a tune of ten notes. Both the frequency and the duration are randomized. Now run the program again. See if the computer's answers are more convincing with this addition.

Experiment 2: Beeeeeeeeeeep!

The time delay in Line 90 will work in any program situation where you wish a pause. By changing the number of times the loop operates, you can create delays of less than a second to many hours.

If you want a musical background, however, you can use the SOUND command to create a delay. Use **BREAK** to stop your program. Then list it with **LIST** and the **ENTER** key.

Change Line 90 to create a long musical tone. The second number in the SOUND command controls the duration of the tone. Notice that the duration (25) is much higher than before with this new instruction:

```
90 SOUND RND(100),25
```

Lesson 8: Decision Maker (continued)

Now run the complete program and see if the sounds make it more interesting.

Experiment 3: Vocabulary Building

There's no limit to the things you can have your computer say, and you can add as many words or phrases as you wish. Just add the line numbers to the ON-GOTO list and increase the range of random numbers.

As an example, here's how to add three more comments. Change Line 40 two ways: increase the 4 to a 7 and add the three new line numbers to the list. The new lines with additional words will be numbered 45, 55, and 65.

```
40 ON RND(7) GOTO 50,60,70,80,45,55,65
```

All seven possible numbers for RND(7) will now branch or go to a different line in the program.

Write your own comments or use these "answers." The PRINT @ number is adjusted so the words will print in the center of the screen.

Here are some suggestions:

```
45 PRINT @234,"ASK AGAIN":GOTO 90
55 PRINT @235,"BEATS ME":GOTO 90
65 PRINT @238,"NEVER":GOTO 90
```

Try these additions to the program, then add as many answers as you like. For each new answer change the random function and add to the list in Line 40.

Experiment 4: Long Listings

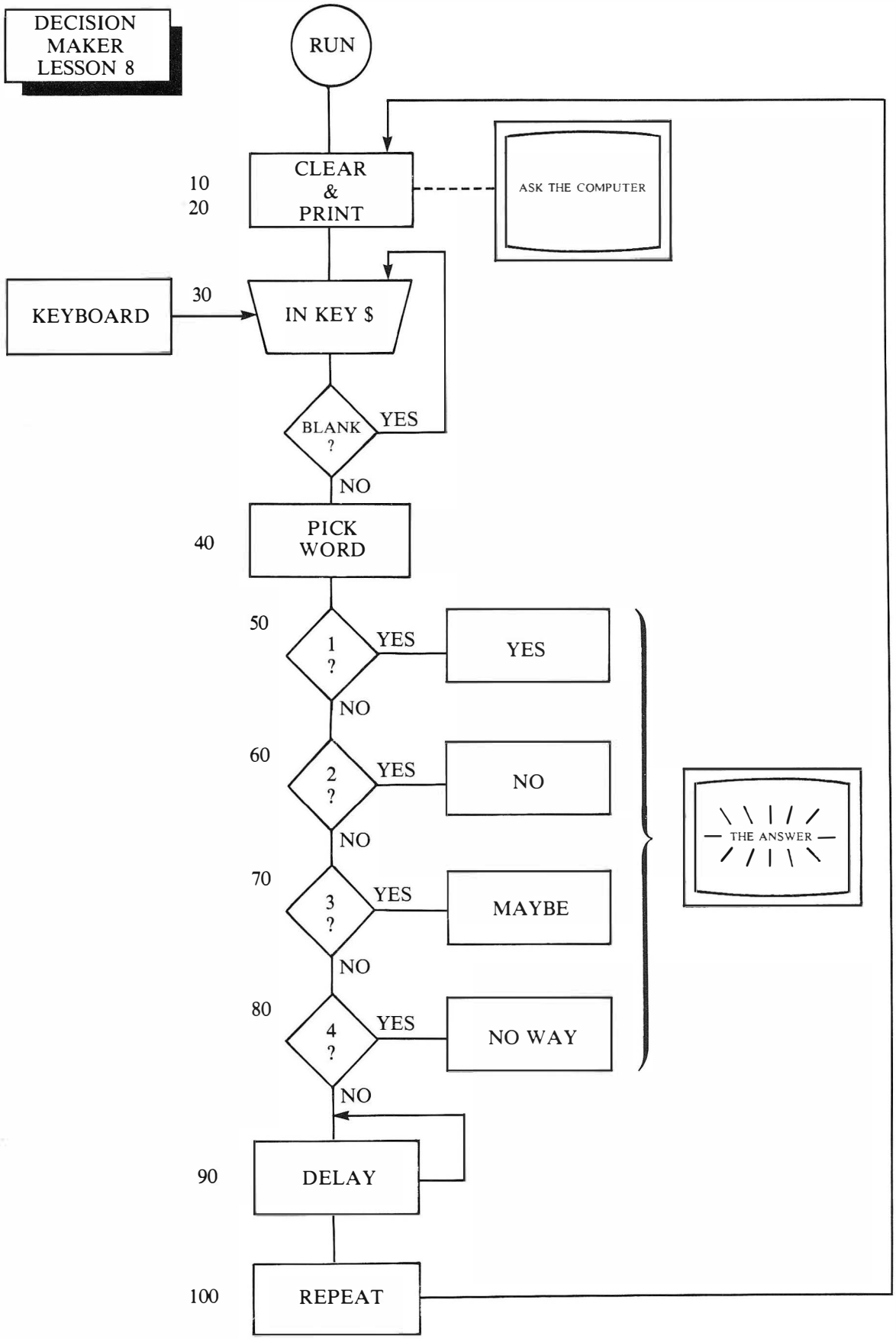
By now your program has grown longer than the screen. Try listing it and see that the first few lines scroll off the top.

You can list the first section, or any other part of a program by adding the first and last line numbers of a section to the LIST command. Note how the hyphen (-) is used in these examples. Try each one and see the effect.

```
LIST 10-60 lists lines 10 through 60.
LIST -50 lists all lines up through Line 50.
LIST 50- lists Line 50 and all following lines.
```

Lesson 8: Decision Maker (continued)

DECISION
MAKER
LESSON 8



Lesson 9: Area Calculator

Formulas

Special Calculators

This calculator program computes areas and is very useful whenever you are estimating how much paint, carpeting, wallpaper, tile, grass seed, or other material to buy. To find the total area to be covered, you will enter the dimensions of smaller sections. The computer will calculate the area of each section and keep a running total.

Not only areas, but also volumes and other quantities may be calculated with this programming model. In the Experiments section you will see how to create a similar program from any mathematical formula.

Load Lesson 9 from the cassette using **CLOADM** and **EXEC**. Type each of these ten instructions and press **ENTER** after each one. You can also type **AUTO** and press **ENTER** if you would like Computer Learning Lab to enter the program for you.

```
10 T=0
20 CLS
30 INPUT "LENGTH";L
40 INPUT "WIDTH";W
50 A=L*W
60 PRINT "THIS AREA:";A
70 T=T+A
80 PRINT "TOTAL AREA:";T
90 INPUT "MORE (Y,N)";A$
100 IF A$="Y" GOTO 20
```

Run Area Calculator

Now type **RUN** and press **ENTER**. The screen will clear, and you will be asked to enter a number for the length.

Type **20** or some other small number for the length and press **ENTER**. Now type another small number like **35** for the width and press **ENTER** again. The area you've specified is calculated and

Lesson 9: Area Calculator (continued)

displayed along with the total area. Since this is the first calculation, this area and the total area are the same.

Now select **Y** to go again and enter two more dimensions. This time, the area you specified is shown along with the total area for both surfaces.

Here's how to use this program in computing a total surface area, such as the amount of carpeting required for a home. Begin by thinking of the surface as a combination of squares or rectangles. If there are several rooms, for example, consider each room separately. If a room isn't a rectangle, divide the room into two or more rectangles that you can measure easily.

Pick the units (feet, meters, yards, etc.) you will use to get the answer you want. To find the number of square yards of carpeting to buy, measure the dimensions of each small area in yards. To get a total area in square meters, measure all dimensions in meters.

Stop the program by selecting **N** or by pressing **BREAK**. Then run the program again. This sets the total area to zero (Line 10).

Now enter the dimensions of each rectangle that makes up the total area you wish to calculate. As you enter the dimensions for each small area, the total area will increase.

How Area Calculator Works

Stop your program by pressing **BREAK**. List it on the screen by typing **LIST** and pressing **ENTER**. Look at the flowchart diagram on page 60 and the listing on the screen to see what the computer is doing while this program runs.

The total area is set to zero when the program is first run. Then the screen is cleared. The numbers you input for length and width are used by the program to compute the area with the formula: $A=L*W$. The screen shows the result as THIS AREA. Total area is computed next by adding the area just calculated to the previous total, T. Then the new total area is printed as: TOTAL AREA.

Finally, the program asks if you have any additional areas to add. If you enter **Y**, the program loops back to clear the screen and input new dimensions. Notice that the program doesn't loop back to the beginning because this would reset the total area to zero.

LINE 10 sets the variable T to zero. This variable is used to store the total area.

LINE 20 clears the screen.

Lesson 9: Area Calculator (continued)

LINE 30 prints LENGTH? and sets the variable L equal to the number you input.

LINE 40 prints WIDTH? and sets the variable W equal to the number you input.

LINE 50 computes the area with the formula: $A=L*W$.

LINE 60 prints THIS AREA and the value of A.

LINE 70 computes the total area with the formula: $T=T+A$.

LINE 80 prints TOTAL AREA and the value of T.

LINE 90 prints MORE (Y,N)? and sets the variable A\$ equal to the letter you type.

LINE 100 sends the computer to Line 20 to clear the screen and input a new set of dimensions if you type **Y**. Notice that T (the total area) is not set to zero when the program repeats.

Programs that calculate specific things can be as simple or as elaborate as you choose. We will add a feature to this program and then use this programming concept to build other special calculators.

Experiment 1: Add or Subtract

It's often convenient to subtract an area when computing totals. If you're estimating how much paint is required to cover the walls, for example, it's easier to add all the wall areas together and then subtract the windows and doors. If you're estimating how much grass seed to buy, you might find it easier to add together all the area of your lot and then subtract the area of the house and driveway.

These additional instructions make it easy to subtract areas. First, stop your program by pressing **BREAK**. Then list it by typing **LIST** and pressing **RETURN** again. Now you can add this subtraction feature to your program by adding these two instructions. Type carefully and press **ENTER** after each instruction.

```
55 INPUT "ADD OR SUBTRACT (+,-)";B$
```

```
57 IF B$="-" THEN A=-A
```

Now list your program and see the changes. It should look like this:

```
10 T=0
20 CLS
30 INPUT "LENGTH";L
40 INPUT "WIDTH";W
```

Lesson 9: Area Calculator (continued)

```
50 A=L*W
55 INPUT "ADD OR SUBTRACT (+,-)";B$
57 IF B$="-" THEN A=-A
60 PRINT "THIS AREA";A
70 T=T+A
80 PRINT "TOTAL AREA";T
90 INPUT "MORE (Y,N)";A$
100 IF A$="Y" GOTO 20
```

If your program matches the example, type **RUN** and press **ENTER** to run it. Now the computer will also ask if you wish to add or subtract each area you enter. Try estimating the wall area of the room you're in by entering the dimensions of each wall. Then subtract each window or door by entering its dimensions and subtracting its area from the total.

Experiment 2: Change the Formula

The general form of this program may be used to build a special calculator for solving any mathematical problem, if you know the formula. For example, if you want to find the volume of a cube, you can use this formula:

$$\text{Volume}=\text{Length}\times\text{Width}\times\text{Height}$$

Try writing a program that inputs the length, width, and height and then prints the volume of a cube. If you have difficulty getting this to work or would like some help, read the rest of this experiment for a solution.

In writing a program, you would choose variables to represent each item in the formula. We used L and W for length and width, so let's continue and use H for the height.

First, erase the program you now have by typing **NEW** and pressing **ENTER**. Now you can write a new program that will begin by clearing the screen. Then the instructions will input each variable with a message to describe what it is. Start with these instructions.

```
10 CLS
20 INPUT "LENGTH";L
30 INPUT "WIDTH";W
40 INPUT "HEIGHT";H
```


List the program to be sure that it matches the example. Then run it and see what it does. The program is not complete, but the part you have entered should print each word and wait for you to input a number. After you've entered a number for each word, the program will stop. Now complete your program by adding the formula and the printout instruction:

```
50 V=L*W*H  
60 PRINT "THE VOLUME IS:";V
```

Your program should now be complete. List it and check to see that all instructions are correct, then run it and see that it computes the volume accurately. Here's what the complete program should look like:

```
10 CLS  
20 INPUT "LENGTH";L  
30 INPUT "WIDTH";W  
40 INPUT "HEIGHT";H  
50 V=L*W*H  
60 PRINT "THE VOLUME IS:";V
```

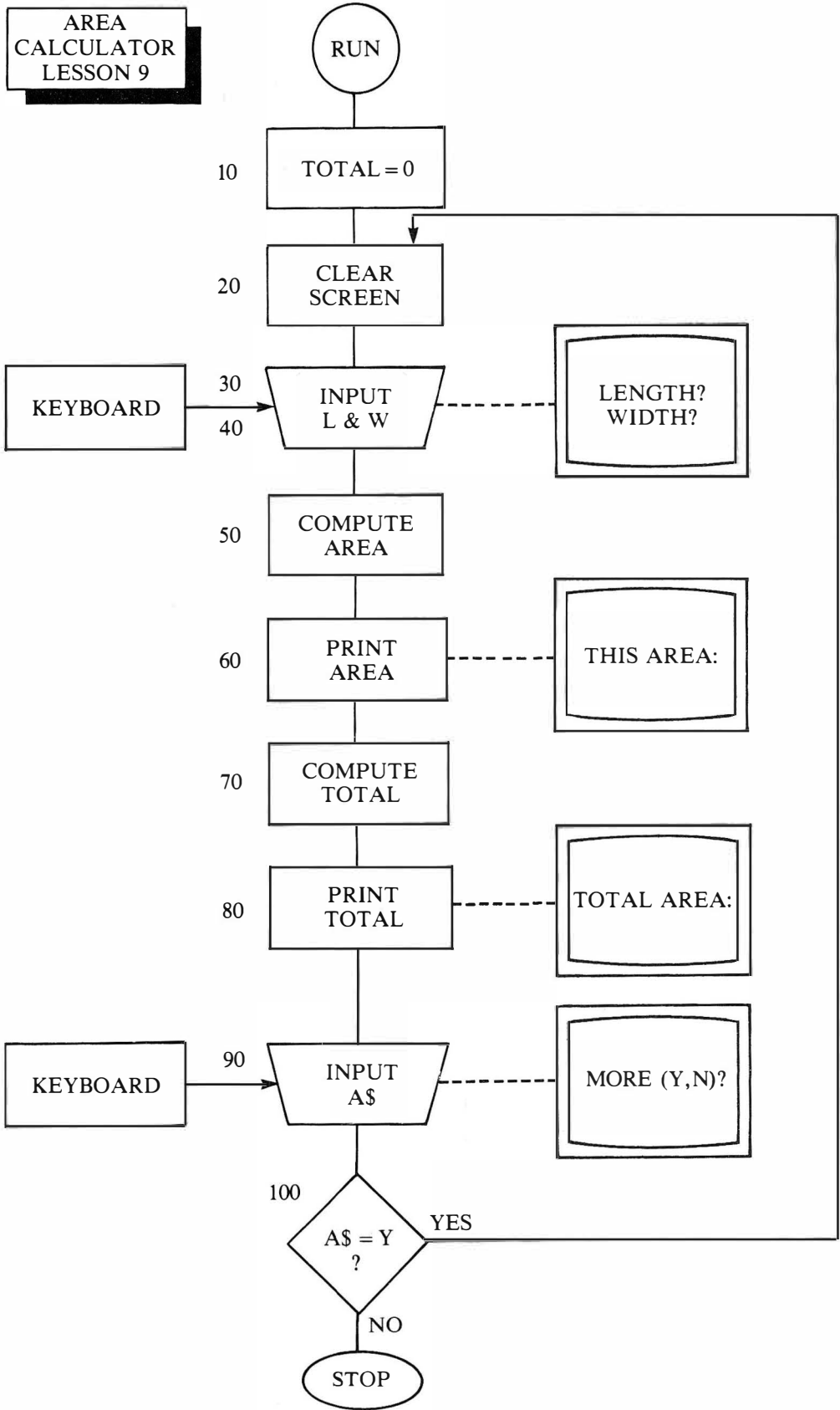
If you like, you could also add instructions to compute the total of several volumes, or have the program ask if you would like to repeat a calculation.

Experiment 3: Try Your Own

Pick any simple formula such as the area of a triangle or volume of a sphere. Write a program to calculate the answer automatically when you input the data. As a final model, here's a program to compute gas mileage when distance and fuel are known:

```
10 CLS  
20 INPUT "DISTANCE TRAVELED";D  
30 INPUT "GAS CONSUMED";G  
40 MPG=D/G  
50 PRINT "GAS MILEAGE =" ;MPG;"MGP"
```

Lesson 9: Area Calculator (continued)



Lesson 10: Interest Calculator

Lesson 10: Interest Calculator

Printing Tables and Rounding Off Numbers

Integers: INT(X)

It's often useful to print charts and tables with your computer. In fact, that's the primary function of many business machines that compute and print tables of financial and statistical information.

In this lesson you will see that this type of computing is very easy to do. If you have a printer, you can create your own business machine with little effort. The example we will use is actually quite useful if you want to know how much money you would earn at a particular interest rate. You can also calculate and print the effects of depreciation on your capital.

Whenever you print dollar amounts with your computer, you will have to round off the figures to the nearest penny. Otherwise, you could wind up with figures like \$12.1302928 instead of simply \$12.13.

Load Lesson 10 from the cassette with **CLOADM** and **EXEC**. Type these eleven instructions into your computer after you see the title frame. If you would rather enter the program automatically, type **AUTO** and press **ENTER**.

```
10 CLS
20 INPUT "PRINCIPLE" ($);T
30 INPUT "YEARLY INTEREST (%)";I
40 INPUT "NUMBER OF MONTHS";M
50 PRINT
60 PRINT "MO . . INTEREST . . . . . TOTAL . . . . . ."
70 FOR L=1 TO M
80 A=INT(I*T/12+.5)/100
90 T=INT((T+A)*100)/100
100 PRINT L;A;T
110 NEXT L
```

Run Interest Calculator

To run this program, just type **RUN** and press **ENTER**. Let's pretend that you have \$17,000 to invest for nine months and that you can earn a yearly interest rate of 11.9 percent.

When you see PRINCIPLE (\$) on the screen, type **17000** and press **ENTER**. Notice that you do not use commas when typing in large numbers. Now type **11.9** for the yearly interest in percent, and press **ENTER** again. Select **9** months, press **ENTER**, and see how your \$17,000 investment would grow over the next nine months at 11.9% yearly interest rate.

In the first month, you would earn \$168.58 interest and your total assets would be the original investment plus the interest, or \$17,168.58. Now your new assets will earn interest for the second month. Since you have more capital, your interest for the second month would grow to \$170.26.

You can try another calculation by running the program again. Just type **RUN** and press **ENTER**. The program will prompt or remind you as you enter the data. If you select more than nine months, the chart won't fit on your screen. To stop the chart while it's printing, press **SHIFT** and the **@** key. You can resume printing by pressing any other key.

How Interest Calculator Works

List your program and look at the flowchart diagram on page 66 to see what the computer does when you run Interest Calculator.

After clearing the screen, the computer inputs your data: principle, yearly interest, and number of months.

A title is printed for the chart with headings for each column.

The FOR/NEXT loop repeats once for each month. On each cycle, the program computes the added amount for each month. This income is rounded off to the nearest penny (\$/100). The total is increased by the income to become the principle for the following month's calculation. After the data is printed, the screen scrolls up one line.

If there are any more months to compute, the program loops again. If not, it stops.

LINE 10 clears the screen.

LINE 20 prints PRINCIPLE (\$) and sets T equal to the number you input.

Lesson 10: Interest Calculator (continued)

LINE 30 prints YEARLY INTEREST (%)? and sets I equal to your input. Notice how the information inside the parentheses tells you exactly what to type.

LINE 40 prints NUMBER OF MONTHS? and sets M equal to your input.

LINE 50 prints a space to separate the input data from the chart.

LINE 60 prints a heading for the chart. Lines 50 and 60 make the interest rate table easier to read and understand.

LINE 70 begins the FOR/NEXT loop. This loop continues until L is greater than M, the total number of months.

LINE 80 computes the added income for the month and rounds off the answer to two decimal places.

LINE 90 increases the total by adding the new amount to the previous total.

LINE 100 completes the FOR/NEXT loop. If L is less than M, the program loops back to Line 70 where L is increased and the next month is calculated.

This program works very well as it is and can calculate income from virtually any investment when yearly interest is known.

Experiment 1: Daily Compounding

If interest is figured every day instead of each month, your investment will increase faster. The reason is that you won't have to wait until the end of the month for your interest to start earning money. The difference between monthly and daily calculation (or compounding) isn't large, but it does change the figures.

Begin this experiment by calculating the total capital when \$12000 is invested for 12 months at 10% interest. Write down the interest and the total for the final month.

Now change your program so that you will compound the interest daily. This will be a quick test. We won't change all the lines in the program, only the one that actually does the calculation. Here's the original instruction:

```
80 A=INT(I*T/12+.5)/100
```

Now change this line to calculate daily interest by changing T/12 to T/365, like this:

```
80 A=INT(I*T/365+.5)/100
```

Lesson 10: Interest Calculator (continued)

With this change the program computes daily, not monthly, interest. Run this version and remember that you are actually counting days, not months. If this change were permanent, we would also change the printing on the screen. Pretend that MONTHS actually means DAYS and enter this data:

```
PRINCIPLE ($) ? 12000
YEARLY INTEREST (%) ? 10
NUMBER OF MONTHS ? 365
```

When the program prints out the results, you will see the interest for a complete year, calculated one day at a time. Watch the days go by in the left column, and see the daily interest grow from \$3.29 at the start to \$3.63 per day after 358 days.

If you have written down the previous figures, you will see that this method of calculating interest actually pays higher dividends. Here are the comparisons:

```
Daily compounding: $13,261.52
Monthly compounding: $13,256.55
Difference: $4.97
```

As you can see, it does make a difference whether your bank or other savings institution compounds monthly or daily.

If you don't want to keep a record of this program, go on to the next experiment. If you wish to save this program on tape and use it again, please change these lines so that the information on the screen is accurate:

```
40 INPUT "NUMBER OF DAYS";M
60 PRINT "DAY . INTEREST . . . . . TOTAL . . . . ."
```

Experiment 2: Integers

Begin by clearing the computer. Type **NEM** and press **ENTER**. Now enter this short program:

```
10 INPUT N
20 PRINT "NUMBER :";N
30 PRINT "INTEGER: ";INT(N)
40 GOTO 10
```

Now run your program and enter these numbers:

```
4.44444
2.22222
```

```
4.56789
111.8888888
12345.6789
```

Notice what the computer prints. The integer of a number, `INT(X)`, is the whole number with any fraction or decimal portion thrown away. Also notice that the computer rounds off the ninth digit when printing any number on the screen.

Try some more numbers if you like, and see that the integer is always the number minus any fractional part.

Experiment 3: Printing Money

When we use the computer to calculate money, we often round off the answers to the nearest penny. The integer function allows us to round off a quantity to any number of decimal places we choose.

Press **BREAK** to stop your program, then add this line to your program to print the number you input to two decimal places:

```
35 PRINT "MONEY$ $:":INT(N*100+.5)/100
```

Now run the program again, enter these numbers, and see that Line 35 rounds off the answer to the nearest penny. The computer will round off the second number in the list below because it has more than nine digits.

```
123.456789
1000000.666666
1.98765
```

In Line 35 (and in Lines 80 and 90 of Interest Calculator) the integer function rounds off the number by first multiplying it by 100, then adding again. This produces a number that contains no more than two decimal places and is increased by 0.01 if the third figure to the right of the decimal point is greater than $1/2$ ($.5/100$ or 0.005).

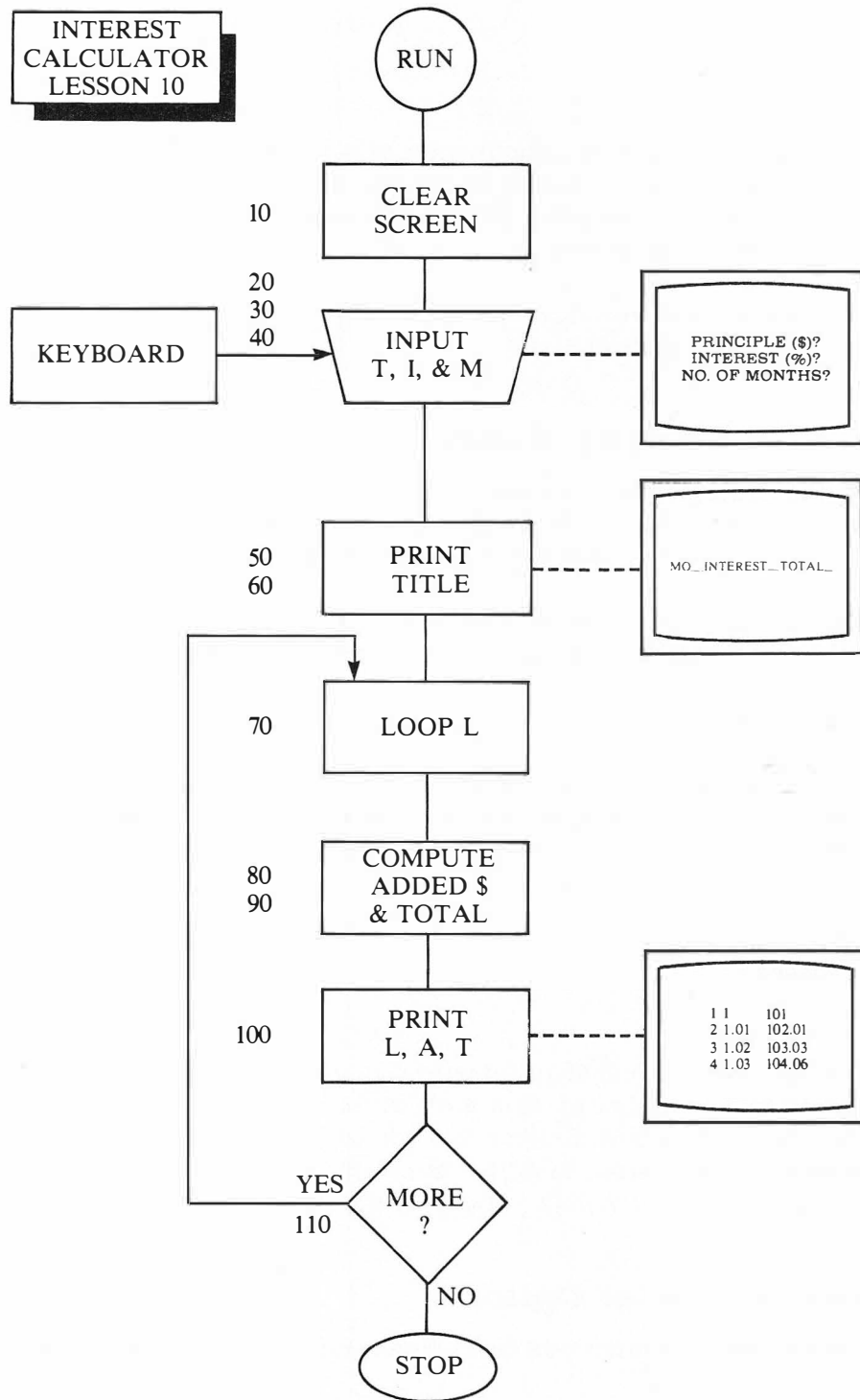
Experiment 4: Printer Option

If you have a hard copy printer, add these lines and create printed interest rate tables:

```
61 PRINT#-2,"DAY . INTEREST . . . . . TOTAL . . . . ."  
101 PRINT#-2,L:A,T
```

Now the results will be sent to your printer as well as displayed on the screen.

Lesson 10: Interest Calculator (continued)



Lesson 11: Coloring Box

Graphics: COLOR, SHAPE

Double loops: FOR/FOR, NEXT/NEXT

This lesson illustrates how graphic shapes of several colors may be placed anywhere on the screen. You will use this feature whenever you write a program to print a graph, make a drawing, add a border, or create a video game. You can mix letters and words with colored shapes to form drawings or charts with text. You will also see how two FOR/NEXT loops may be combined in a program. This technique is used in this lesson to fill an area of the screen.

Load Lesson 11 from the cassette with **CLOADM** and **EXEC**. Follow the directions on the title frame for Lesson 11, then enter these instructions. You can also type **AUTO** and press **ENTER**, and Computer Learning Lab will enter the program for you.

```
10 CLS
20 INPUT "COLOR (1-8)";C
30 INPUT "SHAPE (0-15)";S
40 A=128+(C-1)*16+S
50 PRINT @50,CHR$(A)
60 FOR Y=4 TO 13
70 FOR X=5 TO 26
80 PRINT @Y*32+X,CHR$(A)
90 NEXT X:NEXT Y
100 INPUT "MORE (Y,N)";K$
110 IF K$="Y" GOTO 10
```

Run Coloring Box

The program begins by asking you to select one of eight colors. Type any number from one to eight and press **ENTER**.

Now the program is requesting a number for the shape. There are 16 separate graphic shapes you can print on the screen. These shapes are numbered 0-15. Select **9** to print a pattern of small squares.

Lesson 11: Coloring Box (continued)

When you press **ENTER**, the computer prints a single graphic character in the shape and color you've selected. Then this same character is printed over and over again to fill a large square on the screen. This shows you what that character looks like when printed alone and when it is used to cover a large area.

To see another combination of color and shape, type **Y** and press **ENTER**. Now enter two numbers as before, pressing **ENTER** after each one. Try several combinations to see what all the shapes and colors look like. You will find that shape number zero always prints a black block, and that shape number 15 is a block of the color you've selected. In the experiments section you will see how to write programs like this and print colored graphic characters anywhere on the screen.

How Coloring Box Works

Stop your program by selecting **N**. Now type **LIST** and press **ENTER** to list it on the screen. Compare the flowchart with the instructions and see what the computer is doing when you run this program.

When the program begins it clears the screen. Then you are asked to input numbers for the color and shape of a graphic character. Each graphic character is plotted on the screen with $\text{CHR}\$(N)$, where N is the number of your character. A sample is printed near the top of the screen, so you can see what a single character looks like.

Using loop X , the character you've selected is printed in a horizontal row that is 22 characters wide. Then, using loop Y , this row is repeated 10 times to fill a square. When you run the program, you can see the computer filling each row.

After all rows are filled, the program asks if you would like to see more. Answer with Y and the program repeats. Any other answer will stop the program.

LINE 10 clears the screen.

LINE 20 prints COLOR (1-8)? and sets the variable C to the number you enter.

LINE 30 prints SHAPE (0-15)? and sets the variable S to the number you enter.

LINE 40 is a formula that sets A equal to the number of the graphic character you specified with C and S . This instruction can be used in any program to print a color and shape.

LINE 50 prints character number A at location number 50. This location is near the top of the screen (see diagram). The symbol `CHR$(A)` is used to print the character whose number is A.

LINE 60 begins loop Y. This loop prints from line 4 to line 13.

LINE 70 begins loop X. This loop prints from column 5 to column 26. In this program, two loops are used. As X goes from 5 to 26, a horizontal row is printed. As Y goes from 4 to 13, this row is repeated 10 times to fill the square.

LINE 80 prints graphic character number A at the position specified by X and Y. As shown in the diagram, the expression $Y*32+X$ converts X and Y to a position number.

LINE 90 completes both loops. Notice that loop X moves the character from left to right. Each time this loop is finished, loop Y moves down one line, and the process repeats. After both loops are complete, the area on the screen is filled and the program continues with Line 100.

LINE 100 asks: MORE (Y,N)?. Type `Y` and the program repeats.

These experiments show you more about printing colors and shapes, using the `CHR$(N)` command. You will also see how to locate any position on the screen.

Experiment 1: The Cast of Characters

Clear your program from the computer's memory by typing `NEW` and pressing `ENTER`. Now enter this program, and don't forget to press `ENTER` after each instruction:

```
10 CLS
20 FOR A=32 TO 255
30 PRINT CHR$(A);
40 NEXT A
```

Now type `LIST` and press `ENTER`. Check each instruction and see that there are no errors. Run the program as before. The computer prints all the characters from 32 to 255 on the screen. As you can see, these characters include the symbols, letters of the alphabet, and shapes in all eight colors. To print any one of these graphic characters, use `CHR$(A)`, where A is the number of the character, like this:

```
PRINT CHR$(249)
```

Lesson 11: Coloring Box (continued)

When you press **ENTER**, this instruction prints shape number 9 in color number 8.

Experiment 2: Character Numbers

This experiment shows how to find the character number when you know the color and shape you want. A formula for finding the number was used in Line 40 of Coloring Box. You entered numbers for color C and shape S. The computer calculated the character number A with this instruction in Line 40:

$$A=128+(C-1)*16+S$$

Let your computer do the arithmetic. Type these instructions and print the answer. Remember to press **ENTER** after each instruction. You will not use line numbers, instead, the computer will follow each instruction when you press **ENTER**.

```
C=8
S=9
PRINT 128+(C-1)*16+S
```

The computer printed 249, the character number for shape 9 and color 8. Now print the graphic character number 249 with this instruction:

```
PRINT CHR$(249)
```

Experiment 3: Screen Position

Now that you can print anything you want, learn how to position a character or letter anywhere on the screen. There are 512 positions to choose from. The screen is 32 columns wide and 16 rows high. Notice that the top row and the column on the left are both numbered zero. To find the number of the third row, for example, count: 0, 1, 2.

Another way to find a position number is to use the formula from Coloring Box, Line 80. If X and Y are the column and row, the position number is $32*Y+X$. If it is easier for you to find the position you want by looking at the chart, use that method. If you would rather decide on a column and row, use the formula.

In Coloring Box we printed the graphic character in the 19th space of the second row (Line 50). Find this position on the chart and read its number. Or use the formula with $Y=1$ (second row) and $X=18$ (19th column). Remember that the row and column numbers start with zero, not one.

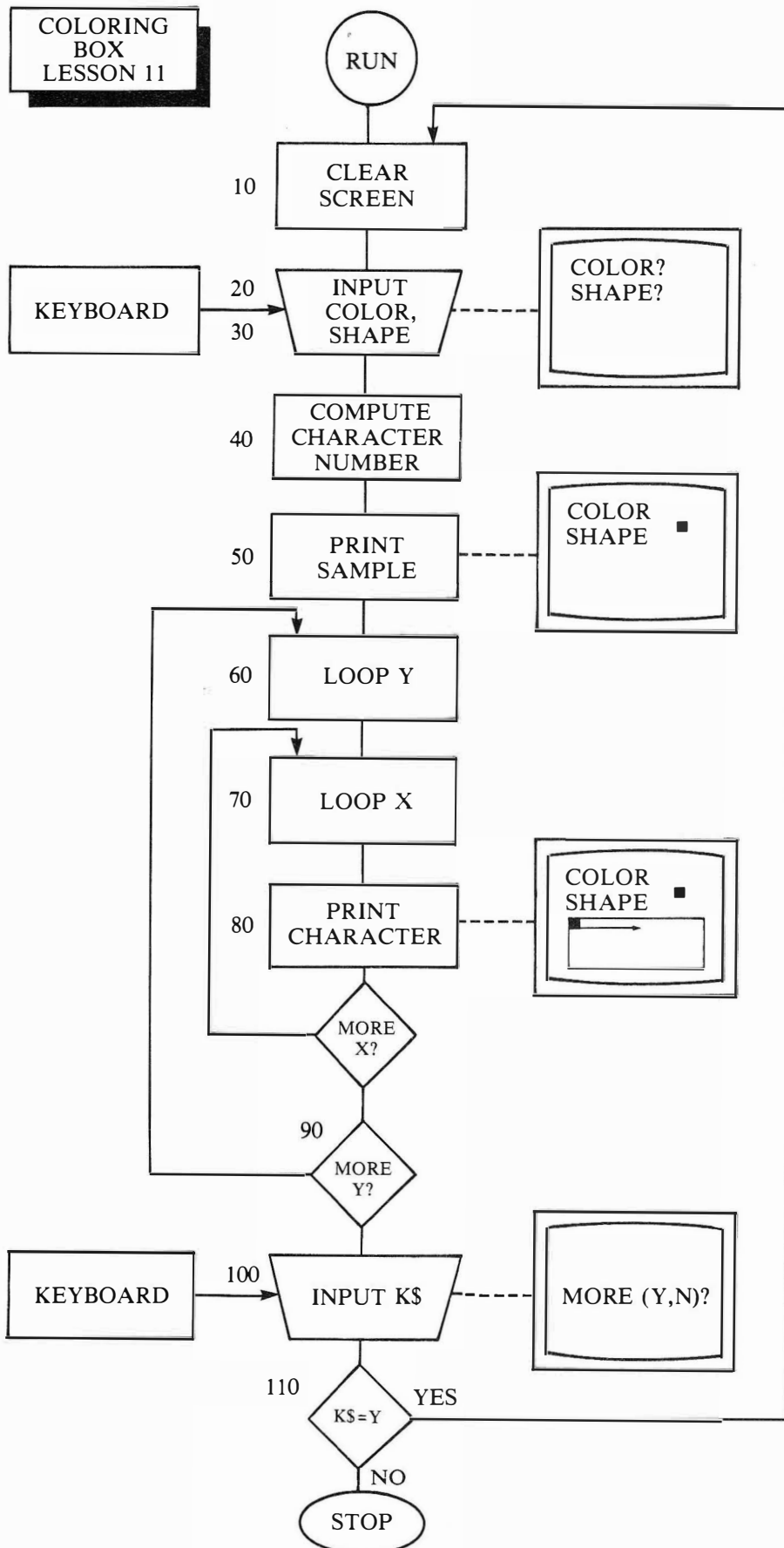
```
Y=1
X=18
PRINT 32*Y+X
```

Lesson 11: Coloring Box (continued)

Either way, using the position number or the formula, you should get the same answer: 50. To print a character at position 50, use this instruction:

```
PRINT @50.CHR$(249)
```

Lesson 11: Coloring Box (continued)



Lesson 12: Time Machine

Time Delay

Comparison: Not Equal < >

If you've been wanting a digital stop watch, this program will create one. It prints the hours, minutes, and seconds on the screen and updates the display ten times each second.

You will see how to create a time delay that causes a program to pause for a specific length of time. The PRINT@ command is used to print the data on the screen so that the words stay in place while the numbers change. The space bar is used to stop the program and the clock.

Load Lesson 12 with **CLOADM** and **EXEC**. Then enter the program or use the **AUTO** feature.

```
10 H=0:M=0:S=0
20 CLS
30 S=S+1
40 IF S=600 THEN S=0:M=M+1
50 IF M=60 THEN M=0:H=H+1
60 PRINT @0, "HOURS   ";H
70 PRINT "MINUTES";M
80 PRINT "SECONDS";S/10;" "
90 FOR T=1 TO 19:NEXT T
100 IF INKEY$<>" " GOTO 30
```

Run Time Machine

This program couldn't be easier to operate. Just type **RUN** and press **ENTER**. The computer sets the time to zero and starts counting the tenths of seconds. After sixty seconds, the number of minutes increases by 1 and the number of seconds starts over again at zero. If you wait for an hour, you will see the hours increase by 1 and the minutes and seconds both return to zero.

After a minute or two, press the space bar and stop the program. You can begin counting again by typing **RUN** and pressing **ENTER**. Doing this starts the program over from the beginning with the hours, minutes, and seconds each set to zero.

How Time Machine Works

In this section you will see how the computer is programmed to create a clock. Stop your program now by pressing the space bar. Type **L I S T** and press **ENTER** to list the program on the screen.

The flowchart diagram on page 78 shows what the computer is doing as the program runs. Notice that the program loops or repeats constantly until the space bar is pressed. Each time the program loops, 0.1 seconds are added to the clock and the time is printed on the screen. When the program begins, the computer sets the hours, minutes, and seconds to zero. Then the screen is cleared. In the next step, 0.1 second is added to the clock. If 60 seconds have passed, one minute is added to the clock. If 60 minutes have passed, one hour is added to the clock. (Remember that 60 minutes is 600 1/10 seconds, the speed of your clock.)

In the next step, all data is printed on the screen. Previous times are erased as new times are printed. The words HOURS, MINUTES, and SECONDS appear to stay the same because they are printed over and over again in the same place on the screen.

The time delay in Line 90 causes a brief pause so that the computer takes one-tenth of a second to do all the instructions inside the loop. Each time the program loops, the computer checks the keyboard to see if the space bar is pressed. If so, the program ends. If not, it loops back to Line 30 to add 0.1 second to the clock and continue keeping time.

LINE 10 sets the variables H, M, and S to zero. These letters are used in this program to represent the number of hours, minutes, and tenths of a second that have passed. Each time the program is run the hours, minutes, and seconds start at zero.

LINE 20 clears the screen.

LINE 30 adds 1 to the variable S. This letter represents the tenths of seconds that are shown on the screen. When S equals 1, one-tenth of a second has passed. If S=10, one second has passed. If S=100, ten seconds have passed.

LINE 40 checks to see if one minute has passed. Since the variable S is increasing ten times each second, a minute is up when S=600. If S=600 now, the computer resets S to zero and adds 1 to M. If S is not equal to 600, the computer skips Line 40 entirely and goes to the next line in the program.

LINE 50 checks the variable M to see if sixty minutes have passed. If M=60 then the computer sets the minutes to zero and adds 1 to the hours, H. If M does not equal 60, the computer skips Line 50.

LINE 60 prints HOURS, followed by two spaces, and then prints the value of H. The three spaces after the word are added so that the numbers of hours, minutes, and seconds will line up on the screen.

The symbol @ after the word PRINT tells the computer to print at a particular location on the screen. PRINT @0 means print at location zero, which is on the left at the top of the screen.

LINE 70 prints MINUTES and the value of M. This line is printed directly under the previous line.

LINE 80 prints SECONDS and the number of seconds. Remember that the variable S is incrementing (counting) ten times each second. To print the number of seconds, the computer divides the value of S by 10.

After printing the number of seconds, the program prints two spaces " ". This erases any previous numbers that were left on the screen. For example, when the clock goes from 59.9 seconds to 0 seconds, the two blank spaces erase the .9 portion of the number.

LINE 90 creates the time delay. This instruction tells the computer to count from 1 to 19 before going to the next line in the program. The number 19 is chosen so that the entire program will loop or repeat ten times each second.

Remember that 0.1 second is added to the clock each time the program loops. With a number that is larger than 19, the computer would take longer and the clock would run too slowly. Similarly, counting to a number that is smaller than 19 would take less time, and the clock on the screen would run too fast.

LINE 100 checks to see if the space bar is pressed. If you don't press the space bar, the program goes back to Line 30 and repeats. When you press the space bar, the program stops.

The computer symbol for the keyboard is INKEY\$ and if this is not equal to a space " ", the computer goes to Line 30. These two symbols <> mean "not equal to" and are used here to branch to Line 30 if the space bar is not pressed. If the space bar has been pressed, then INKEY\$ = " ". In this case, the computer skips the instruction in Line 100, and the program stops.

You can add or change instructions and modify Time Machine in many ways. Here are a few ideas you might wish to try:

Experiment 1: Tick, Tick, Tick

Since most clocks make noise, you may wish to add a ticking sound as the clock runs. Stop the program and add this new instruction:

```
35 IF S=INT(S/10)*10 THEN SOUND 235.1
```

Lesson 12: Time Machine (continued)

Type **LIST** and press **ENTER** to see the complete program with your addition. Now type **RUN**, press **ENTER**, and adjust your TV volume to hear the beep.

Line 35 adds a "BEEP" to the clock every second. To do this, the program uses the integer (INT) function to see if S is a number that can be divided evenly by ten, such as 10, 20, 30, etc. If S is a multiple of ten, then one full second has passed, and a sound is played.

Experiment 2: Timing Accuracy

The addition of a ticking sound will change the accuracy of your clock and make it run slower. You can adjust the delay loop in Line 90 so that the clock will run at the correct speed.

Type **LIST 90** and press **ENTER** to see the present Line 90. To make the clock run faster, use a smaller value for the number 19 in this line:

```
90 FOR T=1 TO 19:NEXT T
```

You may have to try several values to get your clock to keep accurate time.

Experiment 3: Alarm Clock

Now change the program to create an alarm clock. There are several ways to do this. You might start by deciding how long the clock should run before the alarm sounds. Then use an IF statement to check and see whether or not the time is up.

For example, you could add an instruction to make a buzz after a certain length of time. Stop your program and add this line:

```
85 IF M=3 THEN SOUND 20,20:END
```

When you run the new program, you will hear a buzz as soon as three minutes are up.

Experiment 4: Timer

You could set the alarm clock by using a variable (such as the letter A for alarm) instead of the number three in Line 85. At the beginning of the program, input a value for the variable so that you can set the alarm for any number of minutes you choose. Stop your program again and add these two lines:

```
15 INPUT "HOW MANY MINUTES ";A
```

```
85 IF M=A THEN SOUND 20,20:END
```

When you run this version of your program it will ask you how many minutes you wish. Enter a number and wait that many minutes to hear the result.

If you have a precise sequence of events such as developing photographs or doing yoga, you might wish to make a multiple alarm clock that alerts you with tones at specific intervals. Just add as many IF conditions as you wish, with a sound for each one.

Experiment 5: Stop Watch

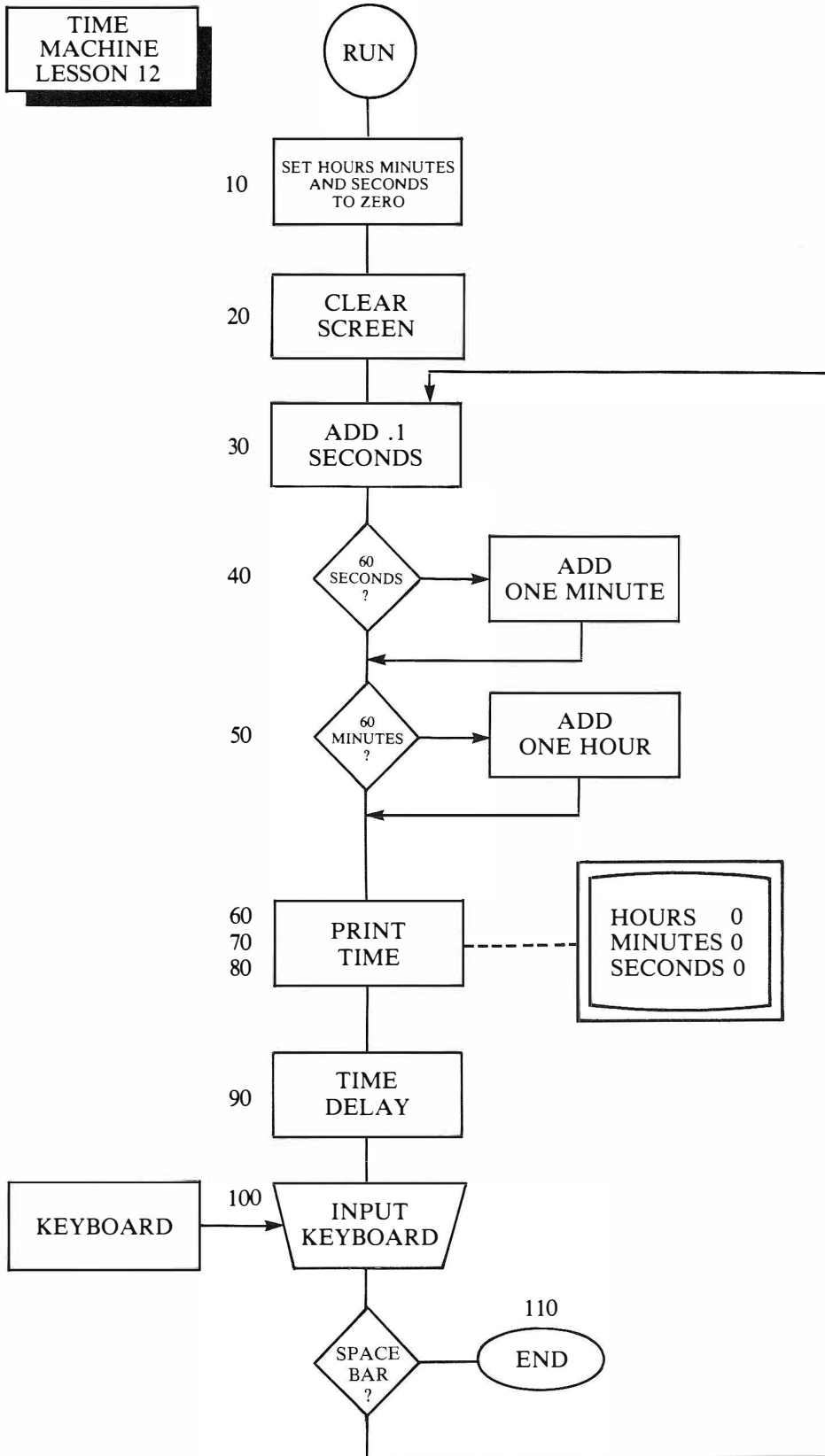
Here's one final idea to create a start/stop option for your program. Add the following lines; then use the space bar to stop, the **G** key to go, and the **R** key to reset the time to zero.

After you press the space bar, the program will wait for you to press **G** or **R** before proceeding.

```
110 K$=INKEY$  
120 IF K$="G" GOTO 30  
130 IF K$="R" GOTO 10  
140 GOTO 110
```

Line 110 sets the variable K\$ to the letter typed on the keyboard. If this letter is G or R, the clock resets. If not, the program returns to Line 110 to get another key from the keyboard.

Lesson 12: Time Machine (continued)



Lesson 13: Probability

Random Numbers and Probability Curves

In this lesson you will use the computer to simulate random events. Randomness is often used in designing computer games and for creating patterns in music and art.

You will begin by writing a short program to print random numbers from 1 to 6, like a six-sided die. This programming technique can also be used to create random dice with any number of sides, or possible combinations. Each experiment will add a section to your program and add a feature, such as storing the results or printing a graph.

The final program is recorded on the Lesson 13 cassette. This computer model of two dice displays a graph showing how often each combination is rolled.

Experiment 1: Random Numbers

The easiest way to understand random numbers is to try an experiment. Type this instruction into your computer:

```
PRINT RND(6)
```

When you press **ENTER**, the computer prints a number between 1 and 6. This number is selected by chance and could be a 1, 2, 3, 4, 5, or 6. If you type this instruction many times, you will eventually see each of these numbers printed on the screen.

Experiment 2: Write a Program

An easier way to print random numbers is to write a short program. Type **NEW** and press **ENTER** to clear the computer of any other programs or data. Type these instructions and press **ENTER** after each one.

```
110 A=RND (6)
```

```
120 PRINT A
```

```
130 GOTO 110
```

Type **RUN** and press **ENTER** to run your program. The computer will set the variable A equal to a random number from 1 to 6. Then the computer will print the value of A on the screen. As the computer adds new numbers at the bottom of the screen, the printing will move up.

Lesson 13: Probability (continued)

Press **BREAK** to stop the program. Then type **LIST** and press **ENTER** to list it on the screen, like this:

```
110 A=RND (6)
120 PRINT A
130 GOTO 110
```

The random function in Line 110 may be changed to create numbers between 1 and any positive number you select. You can also use this function to create random numbers between 0 and 1. Here are some of the ways you can change Line 110 to print different sets of random numbers, with the result shown to the right of the instruction:

```
110 A=RND(10)    numbers from 1 to 10.
110 A=RND(100)   numbers from 1 to 100.
110 A=RND(2)-1   numbers that are 0 or 1.
110 A=RND(0)     numbers from 0 to 1.
```

To see what each of these changes do, just type the new line and press **ENTER** to change the program, and then run it.

Experiment 3: Dice Simulation

This program simulates a pair of dice by creating two random numbers between 1 and 6, adding them together, and printing the result. Type each of these instructions:

```
110 A=RND(6)
120 B=RND(6)
130 PRINT A+B;
140 GOTO 110
```

Run your program. The computer will fill the screen with numbers from 2 to 12. You can stop the printing on the screen by pressing **SHIFT** and **@**. Press any key to start the printing again.

Notice that the numbers 6, 7, and 8 come up very often and that the numbers 2 and 12 are seldom printed. As with real dice, this computer simulation will roll a 7 much more often than a 2 or 12. This is because there are several combinations of two dice that will total 7 (6+1, 5+2, 4+3, 3+4, 2+5, and 1+6). There is only one combination that will total 2 (1+1) or 12 (6+6).

Experiment 4: Store and Print the Results

To keep a total of the results and see which combinations of two dice come up more often, store the numbers in an array with these instructions:

```
20 DIMX(12)
```

```
140 X(A+B)=X(A+B)+1
```

Line 20 dimensions an array to hold 13 numbers. They are $X(0)$, $X(1)$, $X(2)$, and so on up to $X(12)$. Line 140 adds 1 to a number in the array to record how often each total is rolled. If the total is 2, for example, the number in $X(2)$ is increased. If the total is 4, then $X(4)$ is increased.

These next two instructions create a FOR/NEXT loop that cycles 200 times. With this loop, the computer will roll the dice 200 times, storing the results of each roll in the array.

```
90 FOR N=1 TO 200
```

```
290 NEXT N
```

These last four instructions clear the screen and print the numbers stored in the array. As N cycles from 2 to 12, the computer will print N and the value of $X(N)$.

```
300 CLS
```

```
310 FOR L=2 TO 12
```

```
320 PRINT L,X(L)
```

```
330 NEXT L
```

Type **LIST** to see your program on the screen. Here is how your complete program should look. If you see any errors, just enter the line again, using the same line number, to replace it.

```
20 DIMX(12)
```

```
90 FOR N=1 TO 200
```

```
110 A=RND(6)
```

```
120 B=RND(6)
```

```
130 PRINT A+B;
```

```
140 X(A+B)=X(A+B)+1
```

```
290 NEXT N
```

```
300 CLS
```

```
310 FOR L=2 TO 12
```

```
320 PRINT L,X(L)
```

```
330 NEXT L
```

Lesson 13: Probability (continued)

Run this version of the program. You will see the total for each throw on the screen as the computer rolls the dice 200 times. Then the results are displayed with the totals on the left, with the number of times each total was rolled. Your results will vary, but the screen should show something similar to this:

2	5
3	20
4	12
5	24
6	18
7	39
8	29
9	23
10	13
11	11
12	6

In the example shown above, “snake eyes” came up 5 times and “boxcars” came up 6 times. The number 7 came up most often: 39 times. Run your own experiment several times and see your results.

Experiment 5: Watch the Action

With these two lines you can add a continuous printout that shows the results of each roll:

```
30 CLS1
```

```
220 PRINT @64," DIE 1=";A;" DIE 2=";B;" TOTAL=";A+B
```

Line 30 clears the screen. Line 200 prints the score. The @ symbol means to print at a particular location. PRINT @64 is used to print at location 64, the beginning of the second line on the screen.

Type **130** and press **ENTER** to delete Line 130.

When you run this program you will see a scoreboard with DIE 1, DIE 2 and their TOTAL printed on the second line of your screen. After 200 rolls, the results are displayed as before.

Experiment 6: Count the Rolls

Add a new variable R to keep track of the number of rolls with these two instructions:

```
10 R=0
```

```
130 R=R+1
```

```
230 PRINT " NUMBER OF ROLLS=";R
```


There is no important reason for keeping track of this number; it simply makes the scoreboard more fun to watch. Line 10 sets R to 0 when the program starts. Line 130 adds 1 to R each time the dice are tossed. Line 230 prints the results on the third line on the screen. Run the program and see a complete scoreboard.

Experiment 7: Plot a Graph

This change in the program will replace the section that printed the array with a section that prints a graph of the results. Add this line to print a dot on the screen for each roll. Type carefully; we will explain how this works later.

```
310 PRINT @(A+B+2)*32+X(A+B)+2,CHR$(142)
```

Now change the program loop. Instead of using the FOR/NEXT loop in Lines 90 and 290, the program will use a conditional branch in Line 320. With the FOR/NEXT loop, the program cycled 200 times. The branching instruction in Line 320 checks to see if our graph has reached the right edge of the screen. If it has not, the program loops back to Line 100 to roll the dice again. Instead of rolling the dice 200 times, the program will now roll the dice until the graph fills the screen.

Remove these lines by entering their line numbers, pressing **ENTER** after each one:

90

290

300

330

Then change Line 320 to form a loop that goes back to Line 110 if the graph is less than 28 units wide.

```
320 IF X(A+B)<28 GOTO 110
```

Here is how the complete program should now look. List your program and check these lines carefully:

```
10 R=0
20 DIMX(12)
30 CLS 1
110 A=RND(6)
120 B=RND(6)
130 R=R+1
140 X(A+B)=X(A+B)+1
```

Lesson 13: Probability (continued)

```
220 PRINT @64," DIE 1=";A;" DIE
2=";B;" TOTAL=";A+B
230 PRINT " NUMBER OF ROLLS=";R
310 PRINT @(A+B+2)*32+X(A+B)+2,C
HR$(142)
320 IF X(A+B)<28 GOTO 110
```

When you run this version of the program, you will see the scoreboard at the top of the screen and a graph of the results. With each roll of the dice, the computer prints a dot on the screen. Notice that the dots in the middle rows add up faster. When any row contains 28 dots, the program stops.

Experiment 8: Format the Screen

As you might have guessed, the rows of dots correspond to the possible totals for the two dice. With these instructions you will add a title at the top and a row of numbers from 2 to 12 along the left edge of the screen to label each row.

```
40 PRINT:PRINT " . . . PROBABILITY CURVE . . ."
50 PRINT:PRINT
60 FOR N=2 TO 12:PRINT N:NEXT N
330 GOTO 330
```

Line 330 goes to Line 330. This is not a mistake; this special instruction creates a hold in the program by repeating Line 330 over and over again. The program stays in this holding pattern on Line 330 until you press **BREAK**. The reason for this line is to avoid having the program stop after the graph is printed. This avoids the "OK" signal and flashing cursor on the screen.

Now run the complete program and see the computer simulate rolling two dice, with a continuous readout of the results. Compare the shape of the graph with a curve. While the probability of rolling a 7 is the highest, not all runs of this experiment will show row number 7 as the longest. When you wish to stop the program, press **BREAK**.

For a complete listing of the program, see the next experiment.

Experiment 9: Probability

The program for this experiment is recorded on the Lesson 13 cassette. You can load this program from the cassette using **CLOAD**, or enter it yourself, following the experiments in this lesson.

Lesson 13: Probability (continued)

When you run Probability you will see a computer simulation of two dice. With each roll, the computer prints the value for each die, their total, and the number of rolls so far. For each roll, the computer prints a dot on the graph, indicating which totals come up most often. To see how many times the dice total 3, for example, just count the dots in row 3. When any row has totaled 28 dots, the program goes into a holding pattern. To stop the program, press **BREAK**.

Here is a complete listing of the program. Remarks in Lines 1, 100, 200, and 300 are used to make the program easier to read and understand. The listing shows spaces between groups of instructions to help you see how each section works.

```
1 REM ... PROBABILITY ...
10 R=0
20 DIMX(12)
30 CLS1

40 PRINT:PRINT " ... PROBABILITY CURVE ..."
50 PRINT:PRINT
60 FOR N=2 TO 12:PRINT N:NEXT N

100 REM ... ROLL DICE ...
110 A=RND(6)
120 B=RND(6)
130 R=R+1
140 X(A+B)=X(A+B)+1

200 REM ... PRINT SCORE ...
220 PRINT@64, " DIE 1=";A;" DIE 2=";B;" TOTAL=";A+B
230 PRINT "NUMBER OF ROLLS=";R

300 REM ... PLOT GRAPH ...
310 PRINT@(A+B+2)*32+X(A+B)+2,CHR$(142)

320 IF X(A+B)<28 GOTO 100

330 GOTO 330
```

How It Works

Look at the flowchart diagram on page 88 and see how the seven sections of this program work together in creating the final result. To see the program instructions in each section, type **LIST** and the lines you wish to see on the screen. For example, type **LIST 1-30** and the computer will list the first four lines of the program.

Lesson 13: Probability (continued)

LINES 1-30 initialize the program by setting the variable R to 0, dimensioning the array X to hold 13 numbers, and clearing the screen.

LINES 40-60 format the screen. These instructions print the title at the top, skip two lines, and print the numbers 2 through 12 along the left edge.

LINES 100-140 use the random function to simulate two dice. The variables A and B are set to a random number from 1 to 6. The number of rolls, R, is increased by 1. One of the locations in array X is increased. If A+B is 7, location X(7) is increased.

These instructions are repeated for each roll of the dice.

LINES 200-230 print the score board. At location 64 (the beginning of the second line on the screen) the computer prints DIE 1 and the value of A, DIE 2 and the value of B, and TOTAL followed by the value of A+B. On the next line, NUMBER OF ROLLS is printed, followed by the value of R.

LINES 300-310 plot the graph. To see how Line 310 works, type this instruction:

```
PRINT @90,CHR$(142)
```

Notice that this instruction has no line number and will not be added to the program. When you press **[ENTER]**, the computer will print a black dot near the top-right corner of your screen. This location is number 90 and CHR\$(142) is the graphics character you see on the screen.

LINE 310 uses the expression $(A+B+2)*32+X(A+B)+2$ to position the dot on the screen. This expression is a programming module that you can use to position letters or numbers anywhere you like.

LINE 320 checks to see if the number of dots in the line is larger than 28. If not, the program goes back to Line 110 for another roll of the dice. If there are 28 dots, the program goes to the next instruction. The number 28 was picked because that many dots fill the screen

LINE 330 creates a holding pattern by repeating over and over. This prevents the "OK" and the flashing cursor that normally appear when a program is finished. To stop the program, press **[BREAK]**.

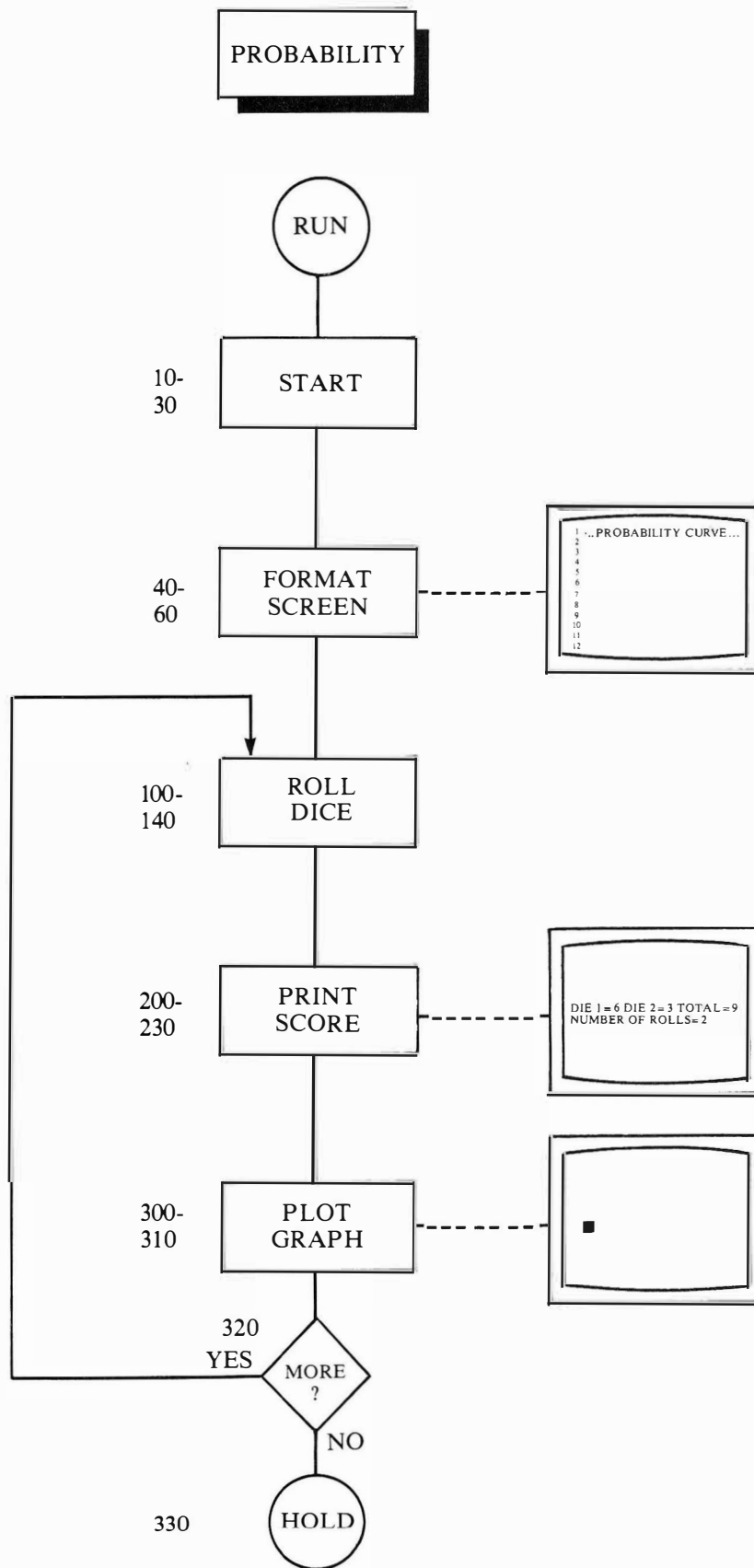
Experiment 10: Musical Dice

If you would like sound effects, add this instruction to your program:

```
315 SOUND (A+B)*10,1
```

The computer will play a note for each dot added to the graph. This slows down the program but adds a nice feature.

Lesson 13: Probability (continued)



Lesson 14: Sorting

Arrays and Data Processing Techniques

Computers used as business machines are frequently programmed to sort information. Data must often be organized in a particular way, and several kinds of sorting programs may be used to do this. A mailing list, for example, might be sorted by alphabetical order to find people quickly by name. This same information could also be sorted by zip code and organized by location. The method used could also vary, with some sorting programs operating faster under certain circumstances than others.

This program shows you how the computer sorts data. At first, numbers in an array will be put in numerical sequence. The final experiments show how you can arrange data in alphabetical, numerical, and even graphical order.

Experiment 1: Pick 10 Numbers

Begin by typing **NEW** to clear your computer of any previous programs. Then enter these instructions to create an array and fill it with five random numbers:

```
10 DIM S(10)
20 FOR L=1 TO 10
30 S(L)=RND(50)
40 NEXT L
```

This array is named S and contains ten locations that can store numbers: S(1), S(2), S(3), S(4), S(5), S(6), S(7), S(8), S(9), and S(10). To see what numbers are stored in these locations, add these instructions:

```
50 FOR L=1 TO 10
60 PRINT S(L)
70 NEXT L
```

Now your program will create an array, fill it with ten random numbers, and print the contents of the array. Run the program and compare your results with the following numbers. The numbers picked for your array will almost certainly be different, but you should see ten numbers between 1 and 50, like this:

```
42
6
30
```

Lesson 14: Sorting (continued)

```
17
21
4
18
31
41
22
OK
```

Experiment 2: Pick the Smallest Number

You can find the smallest number in the array by checking it and comparing it with a new variable, X. Add these instructions to find the smallest number in the array and set X equal to that number:

```
80 X=50
90 FOR L=1 TO 10
100 IF S(L)<X THEN X=S(L)
110 NEXT L
120 PRINT "THE SMALLEST NUMBER IS
";X
```

First the variable X is set equal to 50. Then X is compared with each number in the array. If any number is smaller than X, then X is set equal to that number. After the complete array has been checked, the smallest number is printed.

Here is the complete program. Spaces have been added between sections to match the flowchart in Figure 1 (on page 100). Type **LIST** and check your program against this listing. Also compare this listing with the flowchart and see how the three sections of your program work together to produce the printout on the screen.

```
10 DIM S(10)
20 FOR L=1 TO 10
30 S(L)=RND(50)
40 NEXT L

50 FOR L=1 TO 10
60 PRINT S(L)
70 NEXT L
```



```
80 X=50
90 FOR L=1 TO 10
100 IF S(L)<X THEN X=S(L)
110 NEXT L
120 PRINT "THE SMALLEST NUMBER
IS";X
```

Run the program. You will see ten random numbers with the smallest number printed at the end, like this:

```
14
5
21
43
32
26
40
33
2
15
THE SMALLEST NUMBER IS 2
OK
```

Your numbers will be different, but the format should look the same, with the smallest number selected and printed at the end. Run this program several times, if you like, and see that the smallest number is always selected and printed.

Experiment 3: Smallest Numbers First

Now change the last section of the program, Lines 80-120, to place the smallest numbers first. Your new program will examine each position in the array. If the number being checked is larger than any number in the array, the numbers change places, putting the smallest number first.

Remove Line 80 because it is no longer needed, and replace the last section of the program with these instructions:

```
80
90 FOR L=1 TO 9
100 IF S(L)<S(L+1) GOTO 140
110 T=S(L)
120 S(L)=S(L+1)
130 S(L+1)=T
140 NEXT L
```

Lesson 14: Sorting (continued)

Run the new program and print the array. Now enter GOTO 50 and run the program again, starting at Line 50. Notice that the smaller numbers have moved up in the display and that the larger numbers have moved down. Enter GOTO 50 several more times. Each time you run the program from Line 50 the numbers in the listing will change places to put the smaller numbers first. After running the program several times, you will see that the numbers are in numerical order.

How It Works

When you run the program from Line 50 the computer will print the array and then compare each number, $S(L)$, with the number that comes next in the array, $S(L+1)$. If $S(L)$ is the smaller, the computer goes to Line 140 to compare the next numbers in the array. If $S(L)$ is not the smaller, the numbers in $S(L)$ and $S(L+1)$ are exchanged.

The variable T is used to temporarily hold the number in $S(L)$ while the swap is being made. Here are the steps:

1. The number in $S(L)$ is stored in T .
2. The number in $S(L+1)$ is moved to $S(L)$.
3. The number stored in T is moved to $S(L+1)$.

Experiment 4: Before and After

In these next two experiments you will write a program that repeats automatically to completely sort the numbers in the array. You can enter this program yourself, or skip to Experiment 6 and load the final program from the Lesson 14 cassette.

The program begins with an input section which lets you decide the range of numbers used in selecting the random numbers for the array. Then the array is filled and printed on the screen.

The sorting section uses two program loops to place all numbers in sequence, starting with the smallest numbers. After the sorting process, the last section prints the array.

Enter **NEW** to erase the previous program and enter these instructions:

```
10 REM . . . . SORTING . . .
20 CLS
30 INPUT "START";S
40 INPUT "END";E
```

```
50 INPUT "NUMBER";N
60 DIM S(N)
100 REM . . . FILL ARRAY . . .
110 FOR L=1 TO N
120 S(L)=S+RND(E-S+1)-1
130 NEXT L
140 PRINT "RANDOM:"
150 GOSUB 400
200 REM . . . SORT ARRAY . . .
210 FOR A=1 TO N-1
220 SOUND 200,1
230 FOR B=A+1 TO N
240 IF S(A)<S(B) GOTO 280
250 T=S(A)
260 S(A)=S(B)
270 S(B)=T
280 NEXT B:NEXT A
290 PRINT:PRINT "SORTED:"
300 GOSUB 400
310 END
400 REM . . . PRINTOUT . . .
410 FOR L=1 TO N
420 PRINT S(L);
430 NEXT L
440 RETURN
```

Lesson 14: Sorting (continued)

Run this program and input starting and ending numbers for selecting the random numbers to fill the array. To begin, try **START=1** and **END=9** to fill the array with numbers between 1 and 9. The final question asks how many numbers you wish to put in the array. Try **NUMBER=30** to select 30 random numbers. After printing these numbers, the program sorts the array in numerical order. Adjust the volume to beep as each position in the array is filled. When the sorting is complete, the printout will show these same 30 numbers sorted in numerical sequence, like this:

```
START? 1
END? 9
NUMBER? 30
RANDOM:
 4 9 3 5 5 6 6 2 2 6 9
 4 9 2 9 3 2 1 3 4 9
 1 9 2 9 9 4 2 8 3

SORTED:
 1 1 2 2 2 2 3 3 3
 3 4 4 4 4 5 5 6 6 6
 8 9 9 9 9 9 9 9 9
OK
```

Run the program again and select a new range of numbers to sort. You can request any range you like. Try 32 numbers between 10 and 99 to create a printout like this:

```
START? 10
END ? 99
NUMBER ? 32
RANDOM:
17 16 67 18 75 29 83 29
12 52 43 54 14 98 30 76
93 39 97 67 53 72 73 71
43 53 56 72 42 15 85 31

SORTED:
12 14 15 16 17 18 29 29
30 31 39 42 43 43 52 53
53 54 56 67 67 71 72 72
73 75 76 83 93 98
OK
```

Experiment 5: Dynamic Sorting

With the addition of these instructions, the program will show you the sorting process while it is happening. In this way, you will be able to see exactly how the computer compares each position with the rest of the array and exchanges numbers to place lower numbers first.

Remove Lines 140, 290, and 300 from the program, then add three new printing instructions. These printing commands tell the computer to print characters on the screen. These characters are the numbers, letters, punctuation marks, and graphics symbols represented by their ASCII numbers.

```
140
```

```
290
```

```
300
```

```
275 PRINT @127+A.CHR$(S(A));
```

```
277 PRINT @127+B.CHR$(S(B));
```

```
420 PRINT @127+L.CHR$(S(L));
```

The final program listing is shown at the end of this lesson, along with a complete flowchart in Figure 3 on page 102. When you run this program, use numbers that match ASCII values. Any numbers between 36 and 255 may now be used for START and END. Numbers lower than 36 are blanks, and will not show on the screen. Higher numbers than 255 are not used for graphics characters and will cause an error in Line 420.

How It Works

The flowchart in Figure 2 on page 101 shows three program sections and a subroutine for printing the array. These units in the flowchart work together to display a random and a sorted array.

The input section assigns values to the variables S, E, and N, and dimensions the array. Each position in the array is filled with a random number between S and E. The subroutine at Line 400 is used to print the random numbers.

In the last section, numbers are arranged in sequence, starting with the smallest. Two program loops are used. Loop A begins with the first position and compares all following positions in the array. If any smaller numbers are found, the numbers are reversed so that the smaller number is in the first position. Then the next position is compared with all the rest of the array to see if any numbers are smaller. If so, they are reversed as before, putting the next smallest number in position two.

This process continues until loop A has compared all positions and the numbers are in sequence. This is very much like the previous experiment, with the computer continuing the comparison until the process is complete. The printout subroutine is used again in Line 290 to print the results.

Lesson 14: Sorting (continued)

Experiment 6: Sorting

This program is recorded on the Lesson 14 cassette. You can load the program from the cassette or enter it yourself, as described in Experiments 4 and 5.

Run the program and begin by entering numbers for START, END, and NUMBER. As a suggestion, try these inputs:

```
START? 48
END? 57
NUMBER? 32
```

The computer will load an array with 32 random numbers between 48 and 57. These are the ASCII numbers associated with 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

Watch your screen as the computer sorts these numbers in sequence, starting with the first position and filling it with the lowest number in the array. Notice that the computer moves from one position to the next, filling each position with the lowest number available. Adjust your TV speaker to hear the beeps as each position is filled. After all positions have been checked and adjusted, the program stops with the numbers in perfect sequence.

You can also try sorting letters of the alphabet with these inputs:

```
START? 65
END ? 90
NUMBER ? 128
```

With these instructions, the computer will fill an array with 128 random numbers between 65 and 90. Since 65 is the ASCII number for the letter A and 90 is the ASCII number for the letter Z, this array will contain numbers associated with letters of the alphabet. Notice the computer moves from one position to the next as it fills the array in alphabetical order.

You can select any numbers you choose for this program, provided that the START and END do not exceed 255 and the value for NUMBERS is not larger than 383. Numbers larger than this will cause an error in the program. Numbers smaller than 36 are blanks, and will not be displayed on the screen.

The numbers associated with graphics characters can also be used. To sort 11 lines of characters, input these instructions:

```
START ? 128
END ? 134
NUMBER ? 256
```

With more items in the array, the sorting process takes almost 30 minutes. If you get tired of waiting, press **BREAK** and try another input.

Here is a combination that displays all of the graphics characters, then sorts them by shape and color:

```
START ? 128
END ? 255
NUMBER ? 256
```

How It Works

You can use computers to sort data by numerical sequence, alphabet, or code. This program is a simple sorting algorithm or recipe that works well and lets you watch the process. For sorting large data bases, much faster sorting programs, as well as larger and faster computers, are often used.

The flowchart in Figure 3 shows what is happening while the sorting program operates. After you input the START, END, and the NUMBER of items to sort, the computer fills an array with numbers in random order.

The sorting process begins with the first item in the array. The computer searches the array for the lowest numbered item and places it in the first position. After filling the first position, the computer beeps and moves to the next position. After all positions have been checked, the array is in numerical sequence, and the computer stops.

As the positions in the array are sorted, the number of items the computer searches through are decreased. The search process and the beeps speed up because the computer is spending less and less time looking through the items that are left.

The principles involved are described in the first three experiments. This program is assembled in experiments 4 and 5. Here is a complete listing of Sorting:

```
10 REM ... SORTING ...
20 CLS
30 INPUT "START";S
40 INPUT "END";E
50 INPUT "NUMBER";N
60 DIM S(N)

100 REM ... FILL ARRAY ...
110 FOR L=1 TO N
120 S(L)=S+RND(E-S+1)-1
130 NEXT L
150 GOSUB 400
```

Lesson 14: Sorting (continued)

```
200 REM ... SORT ARRAY ...
210 FOR A=1 TO N-1
220 SOUND 200,1
230 FOR B=A+1 TO N
240 IF S(A)<S(B) GOTO 280
250 T=S(A)
260 S(A)=S(B)
270 S(B)=T
275 PRINT @127+A,CHR$(S(A));
277 PRINT @127+B,CHR$(S(B));
280 NEXT B:NEXT A
310 END

400 REM ... PRINTOUT ...
410 FOR L=1 TO N
420 PRINT @127+LCHR$(S(L));
430 NEXT L
440 RETURN
```

LINE 10 and all other Remarks are ignored by the computer.

LINE 20 clears the screen.

LINES 30-50 print the messages in quotes and set the variables (S, E, and N) to the numbers you input.

LINE 60 dimensions array S. This sets aside positions in the computer's memory for storing N numbers.

LINES 100-130 are a program loop that fills the array with random numbers between S and E.

LINE 150 sends the computer to the subroutine beginning at Line 400. When the computer reads RETURN in Line 440, it will return to the next line in the program following GOSUB, Line 200. This subroutine is used to print the array.

LINES 200-280 form two program loops. Loop A begins the search at the first position in the array and slowly moves to the next to the last position. When this loop is complete, the program ends. Loop B checks the array to see if there are any numbers less than the one being compared. If so, they are reversed to put the lowest number available in position A. This process is repeated for each position.

LINE 220 creates a beep each time loop A cycles. These cycles get shorter as A increases.

LINES 250-270 exchange numbers in the array. This quick shuffle involves an extra variable, T, to store the contents of S(A) while the swap is being made.

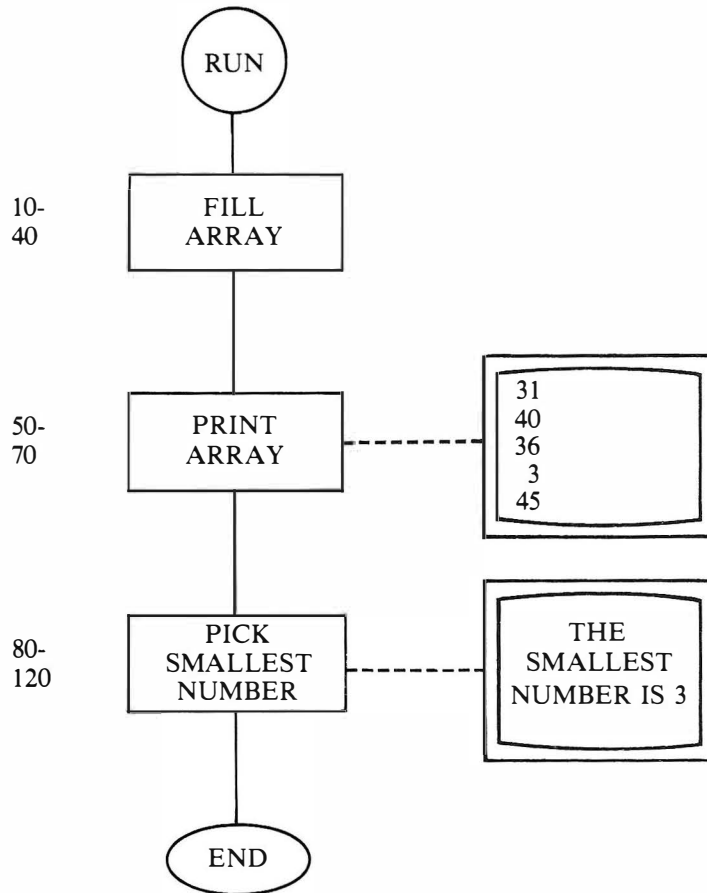
LINES 275-277 are used to print characters. When two numbers in the array change places, these instructions change the corresponding figures on the screen. The location on the screen and the location in the array are set by the same variable. If $A=2$, for example, Line 275 would print at screen location 129. The character printed would be the one whose number is stored in $S(2)$, the second position in the array.

LINE 310 ends the program after loop A is complete.

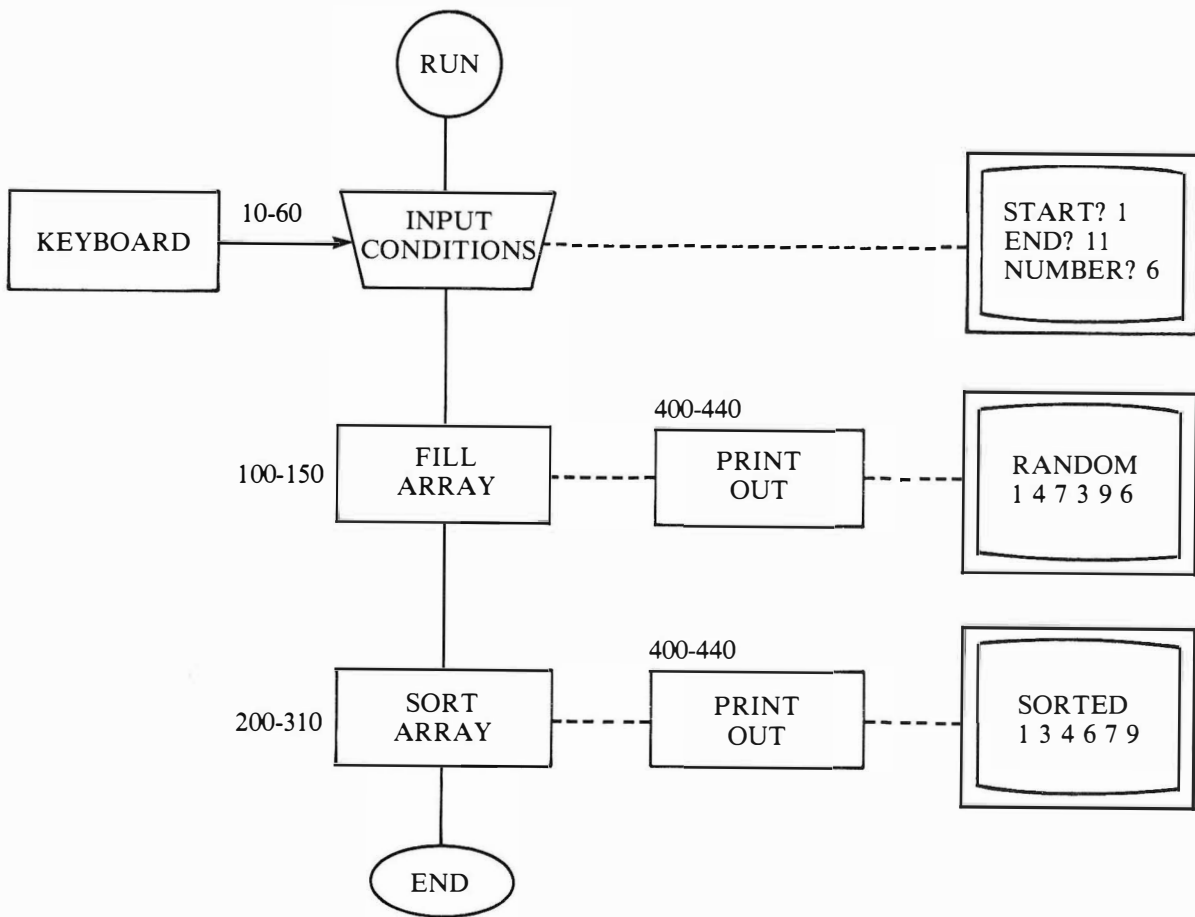
LINES 400-440 are a subroutine that reads the array and prints matching characters on the screen.

Lesson 14: Sorting (continued)

SORTING
FIG. 1

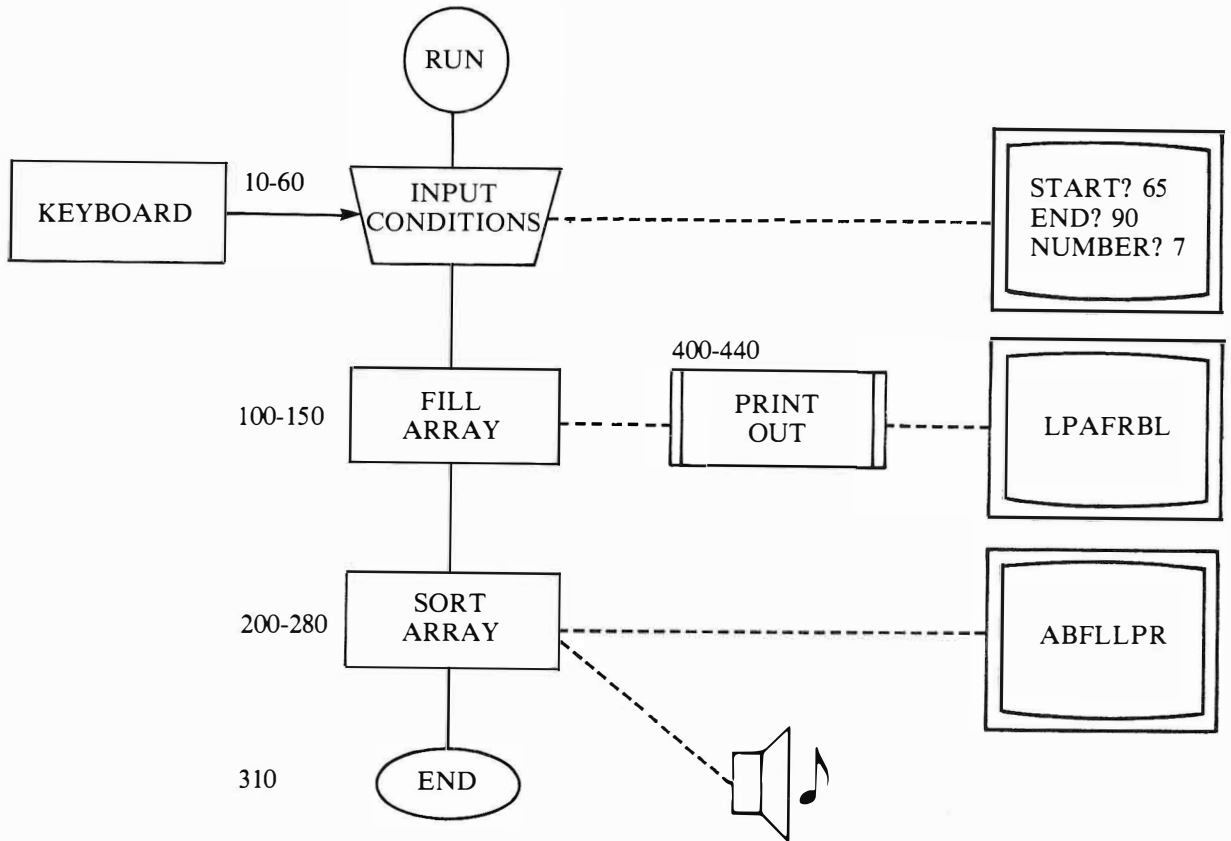


SORTING
FIG. 2



Lesson 14: Sorting (continued)

SORTING
FIG. 3



Lesson 15: Temperature Converter

Lesson 15: Temperature Converter

This lesson uses temperature conversion between English and metric units as an example to show how you can write a program for solving any mathematical problem. You will also see how to round off your answers and how a menu can be used to select any portion of a program.

The menu, one of the programming modules discussed in Section 3, is used in this lesson to select a conversion when either English or metric units are known. You will use this same module with slight modifications to easily select from any number of possibilities at the beginning of other programs you write.

Experiment 1: Convert F to C

The formula for converting a temperature in fahrenheit to celsius is $C=5/9*(F-32)$ where C is the temperature in celsius and F is the temperature in fahrenheit.

Begin by typing **NEW** to clear your computer of your previous program, then enter this new program to convert temperatures from F to C.

```
100 INPUT F
110 C=5/9*(F-32)
120 PRINT C
```

When you run this program, the computer will print a question mark and wait for you to input a value for F. Type **212** and press **ENTER**. You will see the answer, 100, and OK on the screen. Run the program again and enter **32** for F. The celsius equivalent is 0. The temperature at which water boils is defined as 212° F or 100° C. The freezing temperature is 32° F and 0° C.

Now convert 70 degrees F to C by running the program again and inputting **70**. The answer on your screen is 21.1111111. If you convert 80 degrees F to C, you will get 26.6666667 as the answer.

Experiment 2: Rounding Off Answers

When the computer converted 212 and 32 degrees F to C, the answers were 100 and 0. These numbers are integers, or whole numbers with no fractional parts. Converting 70 and 80 degrees to C created the numbers 21.1111111 and 26.6666667. These numbers are not integers because they contain a fractional part. The INTEGER command can be used to print the integer or whole number with the fraction removed.

Lesson 15: Temperature Converter (continued)

Add line 115 to change the program so that it prints the integer or whole number part of the answer.

```
115 C=INT(C)
```

Now run the program two more times to convert 70 and 80 degrees F to C. You will get 21 and 26 as the answers.

Removing the fractional part of the answer is not as accurate as rounding off to the nearest whole number. To round off to the nearest whole number, first add 0.5 to the number, then convert the number to an integer. Change Line 115 to round off to the nearest whole number of degrees with this instruction:

```
115 C=INT(C+.5)
```

Type **LIST** and press **ENTER** to list the complete program:

```
100 INPUT F
110 C=5/9*(F-32)
115 C=INT(C+.5)
120 PRINT C
```

Now run the program and convert 70 and 80 degrees F to C. The answers, 21 and 27, are now rounded off to the nearest integer or whole degree.

This table of temperature conversions is accurate to one degree. Use your program to check this table or to convert any temperature in fahrenheit to celsius, with the answer rounded off to the nearest degree.

Fahrenheit	Celsius
0	-18
32	0
50	10
70	21
80	27
100	38
212	100

Experiment 3: Prompt the User

In order to use this program, you must know that the number you enter is in degrees F and that the number calculated and printed is in degrees C. This works perfectly well, as long as you remember what to do. By changing the input and print instructions, you can print messages on the screen that make this program clear and easy for anyone to follow.

Lesson 15: Temperature Converter (continued)

The actual messages you use can be as simple or elaborate as you like. To add any message to an input, just put the message in quotes, followed by the semicolon and the variable you wish to use. For example, you can replace Line 100 with this instruction:

```
100 INPUT "HOW MANY DEGREES F";F
```

When the computer reads Line 100 it will print the message HOW MANY DEGREES F? and wait for your input.

Similarly, instead of just printing the value of C in Line 130, you can add a message describing the answer with this new instruction:

```
120 PRINT F; "DEGREES F =";C;"DEGREES C"
```

In Line 120 the computer will print the value of F, the words DEGREES F =, the value of C, followed by DEGREES.

List your program. It should look like this on your screen, with messages added to Lines 100 and 120:

```
100 INPUT "HOW MANY DEGREES F";F
110 C=5/9*(F-32)
115 C=INT(C+.5)
120 PRINT F; "DEGREES F =";C; "DEG
REES C"
```

If your program doesn't match this listing, please type the correct instructions and list the program again to check it.

Now run your program and notice that it tells you what to input and describes the result. With these messages added, your program is much clearer, especially for a person using the program for the first time.

Experiment 4: Convert C to F

Converting centigrade to fahrenheit is easy. Just substitute the correct formula, and change the instructions to input C and print F. First, add Line 130 to end the program after the first set of instructions. Then add this conversion to your program with these lines:

```
130 END

200 INPUT "HOW MANY DEGREES C";C

210 F=C*9/5+32

220 F=INT(F+.5)

230 PRINT C; "DEGREES C =";F; "DEGREES F"
```

Lesson 15: Temperature Converter (continued)

Your program now contains two sections, as shown by this listing. We have added a blank line between the sections for clarity.

```
100 INPUT "HOW MANY DEGREES F";F
110 C=5/9*(F-32)
115 C=INT(C+.5)
120 PRINT F; "DEGREES F =";C; "DEG
REES C"
130 END
```

```
200 INPUT "HOW MANY DEGREES C";C
210 F=C*9/5+32
220 F=INT(F+.5)
230 PRINT C; "DEGREES C =";F; "DEG
REES F"
```

One set of instructions beginning at Line 100 converts F to C, and the new lines you have added convert C to F. Check the second section beginning at Line 200 with this instruction:

```
GOTO 200
```

When you press **ENTER**, the computer will begin running the program at Line 200 and print the message: HOW MANY DEGREES C ? Enter the number **100** for degrees centigrade. The computer will print:

```
100 DEGREES C = 212 DEGREES F
```

Experiment 5: Menu Module

In this experiment you will add a menu so that you can convert either fahrenheit or celsius temperatures. This menu is one of the programming modules explained in Section 3. When you select item **1** from the menu, the program will go to Line 100 for a F to C conversion. Select **2** on the menu and the program will go to Line 200 for a C to F conversion.

The extra spaces in front of each line in the menu are used to center the printing on the screen. Add these lines to create the menu:

```
10 CLS:PRINT
20 PRINT "      TEMPERATURE CONVERTER
30 PRINT
40 PRINT "      1. FAHRENHEIT TO CELSIUS"
50 PRINT "      2. CELSIUS TO FAHRENHEIT"
```


Lesson 15: Temperature Converter (continued)

```
60 PRINT
70 INPUT "      SELECT (1-2)";S
80 CLS:PRINT
90 ON S GOTO 100,200
```

These instructions are fully described in Section 3 and can be used, with different words, to create a menu to fit any program. In this example, the menu prints the text on the screen and lets you input a number from the keyboard.

Input a **2** and the computer will go to Line 200, the section that converts C to F. After printing the answer, the computer will stop.

Run the program again and enter a **1** for your selection. The program will convert the fahrenheit temperature you input and then stop.

Figure 1 on page 111 is a flowchart diagram that shows how the program works. The menu module prints the menu and inputs the selection. If a number other than 1 or 2 is selected, an error is detected and the computer repeats the menu. If you select **1**, the program converts F to C. Select **2** and the program converts C to F. Notice that the keyboard is used to input the selection and the numbers for conversion.

Experiment 6: Repeat?

One more program module will make Temperature Converter easier to use over and over again. After the program converts a temperature, have it print: ANOTHER CONVERSION (Y,N) ? The following instructions ask the question, input the answer, and send the computer back to Line 10 if the input is a Y.

```
300 PRINT
310 INPUT "ANOTHER CONVERSION (Y,N)";A$
320 IF A$="Y" GOTO 10
```

Now complete the addition to the program with this new Line 130. This instruction sends the computer to Line 300 and ANOTHER CONVERSION (Y,N) ? after a fahrenheit conversion.

```
130 GOTO 300
```

Lesson 15: Temperature Converter (continued)

Here is a complete listing. Blank lines have been added to separate the listing into sections.

```
10 CLS:PRINT
20 PRINT "      TEMPERATURE CONVERTER"
30 PRINT
40 PRINT "      1. FAHRENHEIT TO CELSIUS"
50 PRINT "      2. CELSIUS TO FAHRENHEIT"
60 PRINT
70 INPUT "      SELECT (1,2)";S
80 CLS:PRINT
90 ON S GOTO 100,200

100 INPUT "HOW MANY DEGREES F";F
110 C=5/9*(F-32)
115 C=INT(C+.5)
120 PRINT F; "DEGREES F =";C; "DEG
REES C"
130 GOTO 300

200 INPUT "HOW MANY DEGREES C";C
210 F=C*9/5+32
220 F=INT(F+.5)
230 PRINT C; "DEGREES C =";F; "DEG
REES F"

300 PRINT
310 INPUT "ANOTHER CONVERSION (Y,N)";A$
320 IF A$="Y" GOTO 10
```

The Figure 2 flowchart on page 112 shows how the menu, F to C conversion, C to F conversion, and the repeat sections work together. The computer prints the menu, inputs your selection, and goes to the section of the program that converts fahrenheit (Lines 100-130) or celsius (Lines 200-230). The number you input is converted, and the results printed are on the screen. After printing the answer, the computer asks if you wish to repeat. If so, the program begins again with the menu.

Experiment 7: Temperature Converter

This program is recorded on the Lesson 15 cassette. You can load this program from the cassette with **CLOAD**, or you can enter it from the keyboard, following the experiments in this lesson.

When you run Temperature Converter you will see a menu with two selections. Type **1** and press **ENTER** and the program will ask for a temperature in fahrenheit. Enter any number you choose. The computer will convert to celsius and print the results.

Lesson 15: Temperature Converter (continued)

Type **Y** and press **ENTER** to try another conversion. This time, select **2** and convert a celsius temperature to fahrenheit.

Here is a complete listing of the program. Remarks in Lines 1, 99, 199, and 299 are used to make the program listing easier to read and understand. Compare these titles with the flowchart, Figure 2.

```
1 REM ... MENU ...
10 CLS:PRINT
20 PRINT "      TEMPERATURE CONVERTER"
30 PRINT
40 PRINT "      1. FAHRENHEIT TO CELSIUS"
50 PRINT "      2. CELSIUS TO FAHRENHEIT"
60 PRINT
70 INPUT "      SELECT (1,2)";S
80 CLS:PRINT
90 ON S GOTO 100,200
95 GOTO 10

99 REM ... F TO C ...
100 INPUT "HOW MANY DEGREES F";F
110 C=5/9*(F-32)
115 C=INT(C+.5)
120 PRINT F; "DEGREES F =";C; "DEG
REES C"
130 GOTO 300

199 REM ... C TO F ...
200 INPUT "HOW MANY DEGREES C";C
210 F=C*9/5+32
220 F=INT(F+.5)
230 PRINT C; "DEGREES C =";F; "DEG
REES F"

299 REM ... MORE ? ...
300 PRINT
310 INPUT "ANOTHER CONVERSION(Y
,N)";A$
320 IF A$="Y" GOTO 10
```

How It Works

Look at the Figure 2 flowchart and see how these four sections of this program work together in creating the Temperature Converter. To see the program instructions in each section, type **LIST** and the lines you wish to see on the screen. Begin by typing **LIST 1-90** to see the menu.

Lesson 15: Temperature Converter (continued)

LINES 1-90 create the menu. This program module is described in Section 3, page 199. The number you input determines the next step in the program. Input a 1 and the program goes to Line 100. Input a 2 and the program goes to Line 200. Input any other number and the program goes to Line 10 to clear the screen and print a new menu.

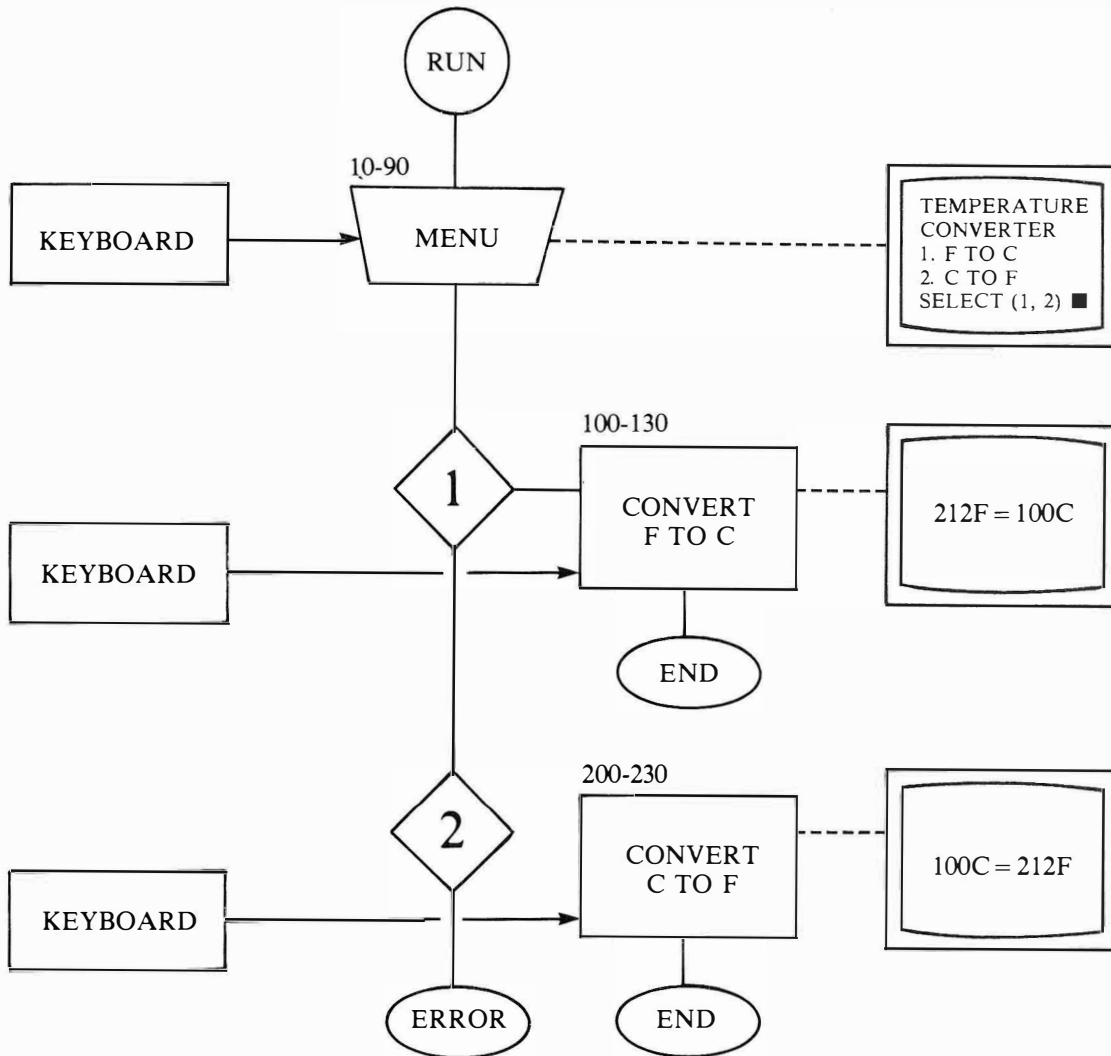
LINES 99-130 request a temperature in fahrenheit, input the number, and convert it to celsius. The program bypasses the next section and goes to Line 300.

LINES 199-230 request a temperature in celsius, input the number, and convert it to fahrenheit.

LINES 299-320 are a program module that asks if you wish to repeat the program. If you input a Y the computer is sent to Line 10. This and other modules used to end programs are described in Section 3.

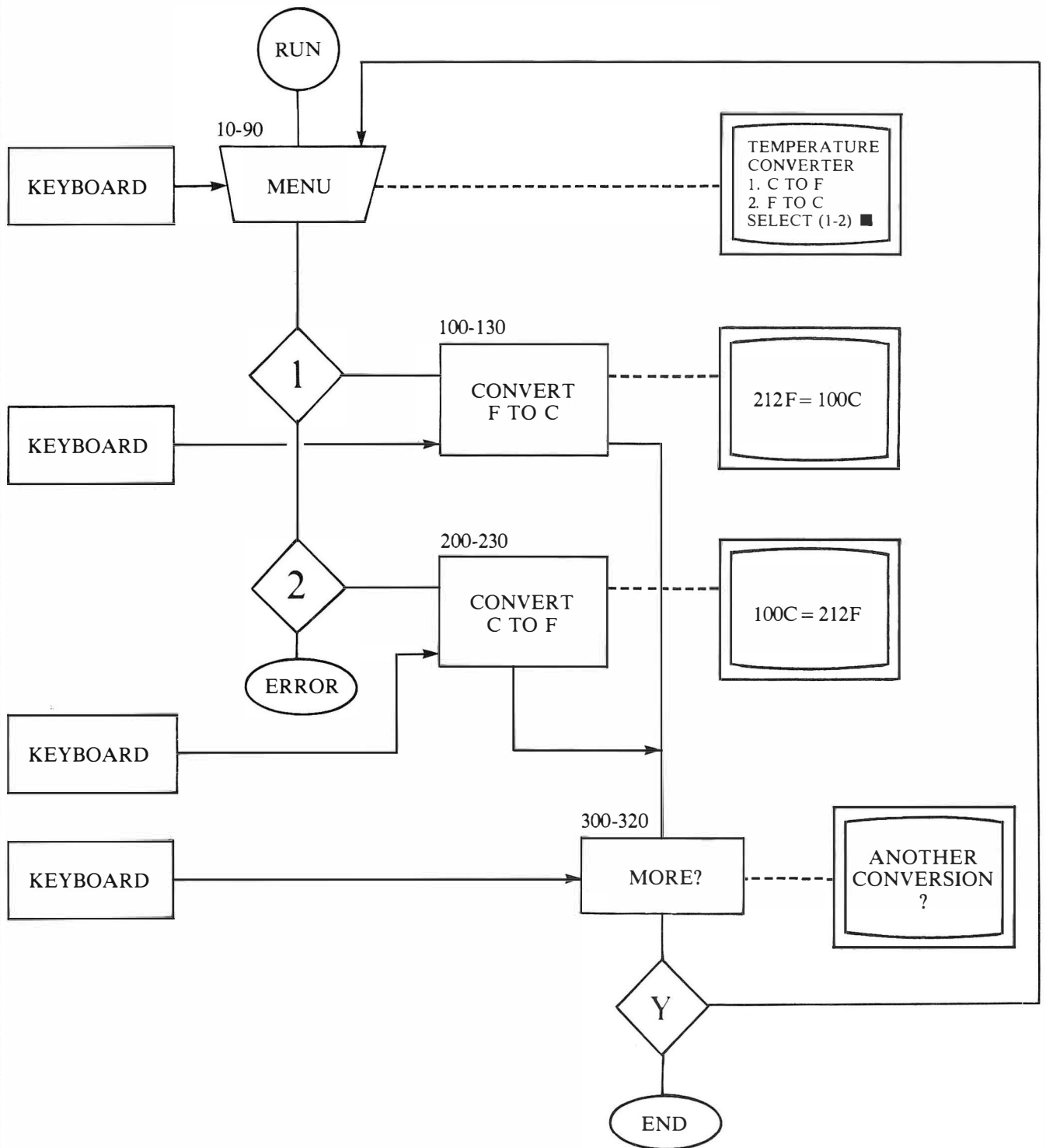
Lesson 15: Temperature Converter (continued)

TEMPERATURE
CONVERTER
FIG. 1



Lesson 15: Temperature Converter (continued)

TEMPERATURE
CONVERTER
FIG. 2



Lesson 16: Cipher

Program and Game Design

Begin this lesson by loading the Cipher program from the Lesson 16 cassette and playing a computer version of a popular strategy game. Then see how this program can be written in small sections, tested, and expanded.

A popular version of this program is available as a board game with colored pegs. You can also purchase several hand-held electronic variations. Commercial software is available for playing this game on many different computers.

The game you will build follows these rules:

- You try to guess a secret code made up of a combination of numbers.
- Each time you guess, the computer will score the accuracy of your guess.
- For each number in your answer that exactly matches an item in the code in both number and position, you are scored one black bar.
- For each number in your answer that matches a number in the code but is not in the correct position, you are scored one black dot.
- The score you get on each trial tells you more about the code. If you plan your strategy perfectly, you can break the code in the minimum number of moves.

This version lets you select up to seven positions and up to nine numbers. With Cipher, you can pick a wide variety of options to test your skill. If you have never played the game, just follow the directions in the first experiment, and see how it works.

Experiment 1: Cipher

Begin by loading the Lesson 16 cassette with **CLOAD**. Then type **RUN** and press **ENTER**.

The computer will ask for the number of positions you want in the code. Type **3** and press **ENTER**. Then the computer will ask how many numbers (from 1 to 9) you want the computer to choose from when it creates the code. Type **3** again and press **ENTER**.

The computer will create a three-digit code, using the numbers: 1, 2, and 3. This code could be any one of the following combinations.

Lesson 16: Cipher (continued)

111	211	311
112	212	312
113	213	313
121	221	321
122	222	322
123	223	323
131	231	331
132	232	332
133	233	333

Now type **123** as your first guess. For each number that matches the code exactly, the computer will print a black bar. Three black bars mean you have guessed the code. For each number that matches the code but is not in the exact position, the computer will print a black dot.

For example, if the computer had picked 112 for the code, and you had guessed the following numbers, this would be the result:

Computer's
Code=112

Your Guess:	Score:	Note:
123	*	= bar * = dot
231	**	
122		
333		
212		
211	**	
121	**	
112		

Notice how a vertical bar (shown as | in the chart above) is displayed each time the correct number appears in the correct position and how a black dot (shown as * is displayed each time the number is correct but not in the correct position.

If your guess matches the code exactly, the computer prints three bars and asks, GO AGAIN? (Y.N). Type **Y** and press **ENTER** to start again with another code.

If you didn't guess the code correctly on your first try, enter another guess. Type three digits and look at your score to see if your second guess matches any numbers in the code.

Continue guessing until your guess matches and you have broken the code. Then the program will ask GO AGAIN? (Y.N). Type **Y** and press **ENTER** to start a new game. If you select more positions or more numbers, the code will be harder to break.

If you want to stop the program before you have solved the code, just press **BREAK**.

How It Works

Look at the flowchart on page 123 and see what the computer is doing while this program runs. The first step sets up two arrays for storing the code and for storing each guess.

The INPUT GAME section works with the keyboard to determine what type of code will be created.

INPUT GUESS also works with the keyboard to input and store each guess.

SCORE POSITIONS contain the instructions for checking each number in the guess against the matching position in the code. Numbers that match are scored with a black bar.

SCORE NUMBERS contain other instructions for comparing each number in the guess with all numbers in the code to see if any black dots should be scored.

The question MORE, shown by a diamond in the flowchart, can be answered two ways. If the code is not broken, the computer returns to input another guess. Solve the code and the computer goes to the next section in the diagram.

The TUNE near the end is played when the code is solved.

TRY AGAIN by typing a **Y** and the program repeats by returning to the INPUT GAME section.

Experiment 2: Initialize and Input Game

In these next experiments you will write Cipher from the beginning. In writing long programs, it is best to write short sections and to test them as you go along.

The first section dimensions arrays to hold the code and the answer, clears the screen, and prints the title. Type **NEW** and press **ENTER** to erase the previous program, then enter these instructions:

```
10 REM    ... CIPHER ...  
20 DIM C(7),H(7)  
30 CLS1  
40 PRINT " ... CIPHER ..."
```

Lesson 16: Cipher (continued)

Line 10 is a remark and is ignored by the computer. Remarks make your programs easier to read and understand, but have no effect on the computer. Line 40 prints the title on the screen.

The array C(7) will store the computer code up to 7 digits long. The second array, H(7), will store the guess. When you write your own programs, pick letters for your variables that help you remember what they contain. Arrays C and H, for example, hold the computer's code and the human's guess.

In the following instructions, the variables P and N hold the values for position and number. Now enter these instructions and press **ENTER** after each one:

```
100 REM . . . INPUT GAME . . .  
110 INPUT "HOW MANY POSITIONS (3-7)";P  
120 INPUT "HOW MANY NUMBERS (2-9)";N
```

Run this part of the program and check it. The screen clears and the title is printed on the top line. Input numbers as requested. The numbers you enter are printed to the right of each question. After you input two numbers, the program stops.

Experiment 3: Select Code

Now add the instructions that create the code you will try to break.

```
200 REM . . . SELECT CODE . . .  
210 FOR L=1 TO P  
220 C(L)=RND(N)  
230 NEXT L
```

This loop cycles once for each position or digit in the code. On each cycle, the computer sets a position in the array equal to a random number. On the first cycle, C(1) is set equal to a random number. On the next cycle, C(2) is set equal to a random number, and on the third cycle, C(3) is set, and so on.

Run your program and select a code with 3 positions and 4 numbers. The computer will input your data, create a code, and stop. You can check to see what the first number in the code is by asking the computer to print it with this instruction:

```
PRINT C(1)
```

Notice that there is no line number. This tells the computer to do the instruction immediately. When you press **ENTER**, the computer prints the number in C(1), which is the first digit in the code. Since you picked 4 numbers to select from, the number in C(1) could be a 1, 2, 3, or 4.

The other numbers in the code are C(2), C(3), and C(4). You can print these and any other variables in a program by using PRINT.

Experiment 4: Input Guess

Now add these instructions to input your guess. The number you enter will be stored in the variable K\$ and printed on the screen. Then each digit will be stored in array H with the first digit in H(1), the second in H(2), and so on.

```
300 REM ... INPUT GUESS ...  
  
305 PRINT  
  
310 FOR L=1 TO P  
  
320 K$=INKEY$:IF K$="" GOTO 320  
  
330 H(L)=ASC(K$)-48  
  
340 PRINT " ";K$;  
  
350 SOUND 92.2  
  
360 NEXT L  
  
370 PRINT " ";
```

The remark in Line 300 makes it easy to find this section in the listing. Line 305 spaces down one line on the screen. This line was added to the program to separate the numbers on the screen.

The FOR/NEXT loop from Line 310 to Line 360 cycles once for each position in the code. Each time this loop cycles, Line 320 creates a holding pattern and the program waits for an input from the keyboard. This is done by repeating Line 320 if the input from the keyboard is a blank "". When you type any key, the variable K\$ is set equal to the ASCII value of the key you type, and the program goes to Line 330.

Line 330 stores each number of your guess in array H. The number of any key is its ASCII value minus 48.

Then the computer prints a space, prints the number you typed, and makes a beep in the speaker. After a number has been entered for

Lesson 16: Cipher (continued)

each position in the code, the program goes to Line 370 and prints a space to separate the score from the guess.

Here is how the program should look on your screen. Blank lines have been added between sections to match the flowchart.

```
10 REM      . . . CIPHER . . .
20 DIM C(7),H(7)
30 CLS 1
40 PRINT " . . . CIPHER . . ."

100 REM . . . INPUT GAME . . .
110 INPUT "HOW MANY POSITIONS (3
-7);P
120 INPUT "HOW MANY NUMBERS (2
-9);N

200 REM . . . SELECT CODE . . .
210 FOR L=1 TO P
220 C(L)=RND(N)
230 NEXT L

300 REM . . . INPUT GUESS . . .
305 PRINT
310 FOR L=1 TO P
320 K$=INKEY$:IF K$="" GOTO 320
330 H(L)=ASC(K$)-48
340 PRINT " ";K$;
350 SOUND 92,2
360 NEXT L
370 PRINT " ";
```

Experiment 5: Score Positions

After you have added this next section, you will be able to play a limited version of the game.

```
400 REM . . . SCORE POSITIONS . . .

410 R=0

420 FOR L=1 TO P

430 IF C(L)<>H(L) GOTO 460

440 R=R+1:SOUND 177,2

450 PRINT CHR$(138);

460 NEXT L
```

The variable `R` is used to keep track of the number of digits in the correct position. The `FOR/NEXT` loop cycles once for each digit and compares the code with the guess. If they are not equal, Line 430 sends the computer to Line 460. If the code and the guess match, `R` is increased, a beep is played through the speaker, and a black bar, `CHR$(138)` — is printed.

You can play Cipher now if you add this instruction:

```
620 IF R<P GOTO 300
```

This instruction comes later in the program and sends the computer back for another guess if the number of correct digits `R` is less than the number of positions in the code `P`. Play this limited version of the game to check your program. The computer will only score black bars, not black dots, and the code will be very hard to guess. After you have checked the program, add the second scoring section.

Experiment 6: Score Number

With these added instructions, your program will score both bars and dots:

```
500 REM ... SCORE NUMBER ...  
510 Y=0  
520 FOR A=1 TO P:FOR B=1 TO P  
530 IF C(A)<>H(B) GOTO 570  
540 Y=Y+1  
550 IF Y>R THEN PRINTCHR$(142)::SOUND 126.2  
560 H(B)=0:B=P  
570 NEXT B:NEXT A
```

In this section, the variable `Y` is used to keep track of all the digits in the answer that match a digit in the code. Each digit in the code is compared with all digits in the answer to see if there is a match. If not, the program goes to Line 570 to try the next combination. If the numbers match, `Y` is increased.

In Line 550, the computer prints a black dot and plays a beep if `Y` is larger than `R`, the number of black bars. A dot is not printed for those matching combinations that have already been scored as black bars.

If you have already added Line 620, you can now play the game with both bars and dots.

Experiment 7: Score Board

This section of the program checks to see if you have solved the code. If not, the computer goes back to Line 300 for another guess. Get the correct answer and this section will play a tune and ask you if you want to go again.

Add these instructions to complete your program:

```
600 REM ... SCORE BOARD ...  
610 PRINT  
620 IF R<P GOTO 300  
640 SOUND 126,6  
642 SOUND 148,4  
644 SOUND 255,1  
648 SOUND 126,2  
650 SOUND 132,2  
652 SOUND 148,4  
660 INPUT "TRY AGAIN (Y,N)";K$  
670 IF K$="Y" GOTO 30
```

If the numbers of bars R is less than the number of positions in the code P , Line 620 sends the computer back for another guess. Get the code right and the sequence of sounds in Lines 640 through 652 play a tune. Then Line 660 asks if you wish to go again. Enter the letter **Y** and the program goes back to the beginning, Line 30. Type any other letter and the program stops.

Here is the complete listing for Cipher. Extra spaces have been added between sections.

```
10 REM ... CIPHER ...  
20 DIM C(7),H(7)  
30 CLS1  
40 PRINT " ... CIPHER ..."  
  
100 REM ... INPUT GAME ...  
110 INPUT "HOW MANY POSITIONS (3  
-7);P  
120 INPUT "HOW MANY NUMBERS (2  
-9);N
```

Lesson 16: Cipher (continued)

```
200 REM ... SELECT CODE ...
210 FOR L=1 TO P
220 C(L)=RND(N)
230 NEXT L

300 REM ... INPUT GUESS ...
305 PRINT
310 FOR L=1 TO P
320 K$=INKEY$:IF K$="" GOTO 320
330 H(L)=ASC(K$)-48
340 PRINT " ";K$;
350 SOUND 92,2
360 NEXT L
370 PRINT " ";

400 REM ... SCORE POSITIONS ...
410 R=0
420 FOR L=1 TO P
430 IF C(L)<>H(L) GOTO 460
440 R=R+1:SOUND 177,2
450 PRINT CHR$(138);
460 NEXT L

500 REM ... SCORE NUMBER ...
510 Y=0
520 FOR A=1 TO P:FOR B=1 TO P
530 IF C(A)<>H(B) GOTO 570
540 Y=Y+1
550 IF Y>R THEN PRINTCHR$(142);
:SOUND 126,2
560 H(B)=0:B=P
570 NEXT B:NEXT A

600 REM ... SCORE BOARD ...
610 PRINT
620 IF R<P GOTO 300
640 SOUND 126,6
642 SOUND 148,4
644 SOUND 255,1
648 SOUND 126,2
650 SOUND 132,2
652 SOUND 148,4

660 INPUT "TRY AGAIN (Y,N)";K$
670 IF K$="Y" GOTO 30
```

Experiment 8: How to Cheat

While you probably wouldn't think of cheating, it doesn't hurt to know how to look inside a program while it is running and see what is going on. In this case, you can stop the program, examine the code to see what it is, and continue running as if nothing had happened. Here is the procedure.

With the program already running and a code selected by the computer, stop the program by pressing **BREAK**. Now you can type instructions and print any variables to see what they contain. To print the first letter in the code, for example, type this instruction:

```
PRINT C(1)
```

Notice that there is no line number. This instruction will be acted on immediately and will not be added to the program.

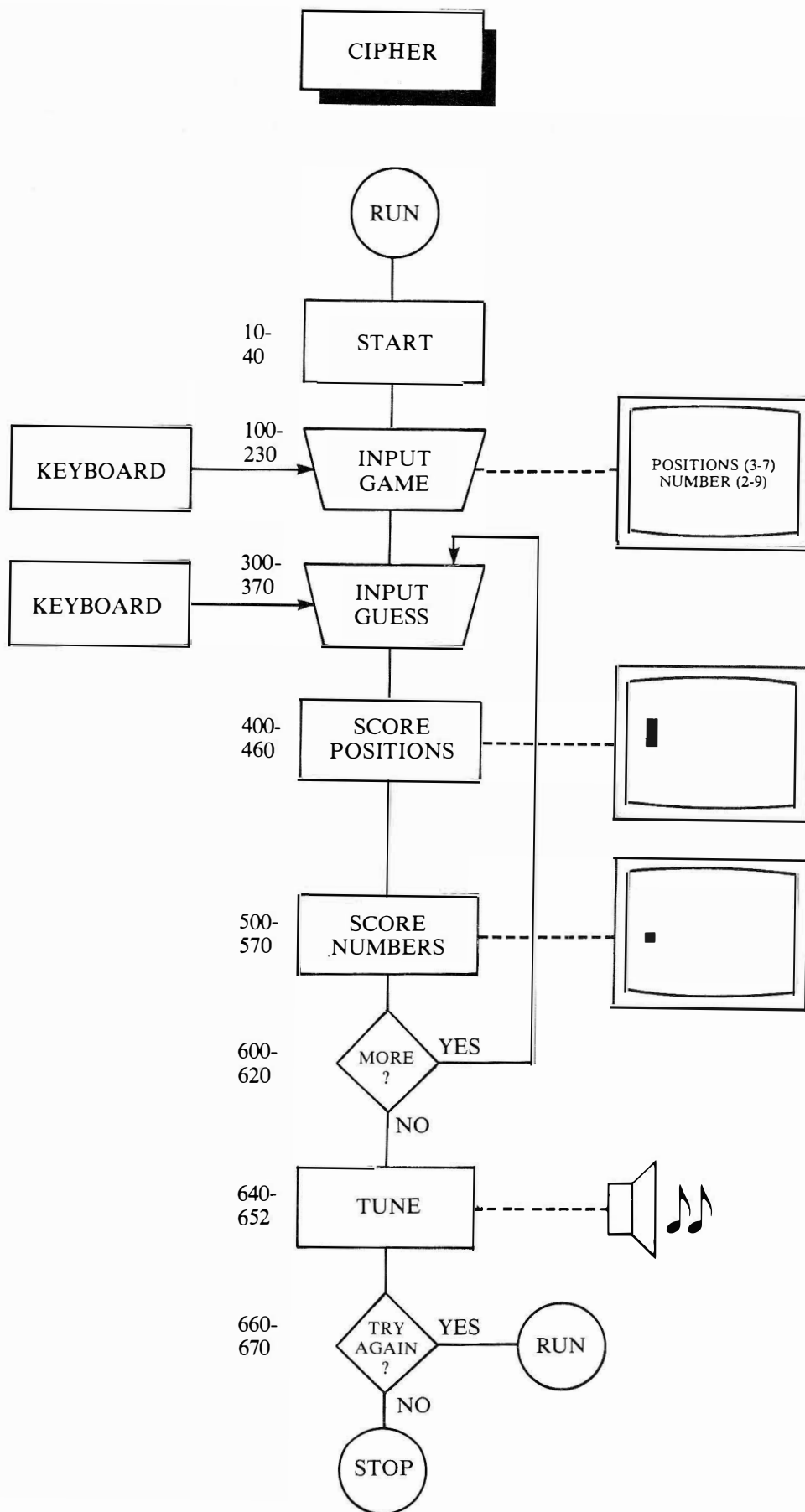
To print all the numbers in the code, use several PRINT statements or use this instruction:

```
FOR L=1 TO P:PRINT C(L):NEXT L
```

Again, there is no line number and this short "program" will print out all the numbers in the code. If you wish to continue with the program, just type **GOTO 370** and press **ENTER**. The program will continue as before.

You can use this method to check any variable in a running program to see what is going on, what might have gone wrong, or to peek at the answers.

Lesson 16: Cipher (continued)



Lesson 17: Math Teacher

Time Response Monitoring®

You can program your computer to create arithmetic problems, input the answers, and score the results. In this lesson you will also use a special technique called Time Response Monitoring to adjust the skill level or difficulty of each problem.

Learning the arithmetic facts is one example of a skill that can be aided by a computer program. Although you probably know your arithmetic, you will discover in this lesson that computer aided instruction, or CAI, is a challenging project for your programming skills. You will begin by writing a simple program to create and score addition problems. Then you will add the other functions to build a program that teaches addition, subtraction, multiplication, and division facts. Finally, you will add feedback by timing the responses to each problem and continuously adjusting the skill level or difficulty.

Experiment 1: Problem Solver

Begin by entering this short program. If there are any other programs in the computer, type **NEW** and press **ENTER** to clear the computer's memory. Type these instructions carefully, and press **ENTER** after each one.

```
10 A=RND(10)
```

```
20 B=RND(10)
```

```
30 PRINT A;"+";B;"=";A+B
```

The computer will select a random value between 1 and 10 for A, and a random value between 1 and 5 for B. Then the computer will print the value of A, a plus sign (+), the value of B, an equal sign (=), and the correct sum of A+B.

Type **RUN** and press **ENTER** to run the program and print a random addition problem on the screen. After the program runs, the computer prints OK and waits for your next instruction. If you run the program several times, your screen should look something like this, with different problems:

```
OK
RUN
  8 + 4 = 12
OK
RUN
 10 + 9 = 19
```

Lesson 17: Math Teacher (continued)

```
OK
RUN
  2 + 8 = 10
OK
```

Each time you run the program, a randomly selected addition problem and the correct answer are displayed. With the random function you can generate an endless series of problems like these, selected by chance.

Experiment 2: Answer Please

Instead of having the answer printed, change the program to print the problem only. Then, have the student input the answer from the keyboard. If the answer is correct, instruct the computer to print RIGHT on the screen. First type **L I S T** to see your program:

```
10 A=RND(10)
20 B=RND(10)
30 PRINT A; "+" ;B;"=" ;A+B
```

Change Line 30 so that the problem, not the answer, is printed.

```
30 PRINT A;"+" ;B; "="
```

When you press **ENTER**, this instruction replaces Line 30 in the program. Input and score the answer by adding these instructions:

```
40 INPUT C
50 IF A+B=C THEN PRINT "RIGHT"
```

Run this program a few times, and try guessing right and wrong answers to check it. Here is the complete program:

```
10 A=RND(10)
20 B=RND(10)
30 PRINT A;"+" ;B;"="
40 INPUT C
50 IF A+B=C THEN PRINT "RIGHT"
```

Experiment 3: Branching

One way to expand the program is to repeat the problem if the answer is wrong and create a new problem if the answer is correct. You can add this feature by changing Line 50 and adding a new instruction, Line 60. The new Line 50 is too long to fit on your screen. Just continue typing and the instruction will continue on the next line on your screen. Remember to press **ENTER** at the end of each instruction.

```
50 IF A+B=C THEN PRINT "RIGHT":GOTO 10  
60 GOTO 30
```

Type **LIST** and press **ENTER**. Your complete program should now look like this:

```
10 A=RND(10)  
20 B=RND(10)  
30 PRINT A;"+";B "=";  
40 INPUT C  
50 IF A+B=C THEN PRINT "RIGHT":G  
OTO 10  
60 GOTO 30
```

The flowchart in Figure 1 on page 135 shows how the branching works in this program. Line 50 checks to see if the answer is correct. If so, the computer prints `RIGHT` and goes to Line 10 to select a new problem. If the answer is wrong and $A+B$ is not equal to C , the instructions in Line 50 are ignored and the computer goes to the next line in the program, Line 60. This instruction sends the computer back to Line 30 to print the problem again.

Run this program and notice that it repeats a problem over and over, until the correct answer is given. Then the computer creates a new problem. In this example, the program branches to one of two possible actions, depending on whether the answer is correct or not. The next experiment uses a similar branching technique to select one of four possible problems to solve.

Experiment 4: Electronic Flash Cards

Another change that will make this program more fun to use and more effective as a learning tool is to include subtraction, multiplication, and division problems. This new program uses four short sections to generate the four types of problems. The branching occurs in Line 40 where the computer goes to one of four places in the program, depending on the value of `G`.

To enter this program, press **BREAK** to stop the previous program. Type **NEW** and press **ENTER** to erase the old program from the computer's memory, then enter each of the following instructions:

```
10 A=RND(10)  
20 B=RND(10)  
30 IF B>A THEN 10  
40 G=RND(4)
```

Lesson 17: Math Teacher (continued)

```
50 ON G GOTO 100, 150, 200, 250
100 PRINT A;"+";B;"=";
110 X=A+B
120 GOTO 400
150 PRINT A;"-";B;"=";
160 X=A-B
170 GOTO 400
200 PRINT A;"*";B;"=";
210 X=A*B
220 GOTO 400
250 PRINT A*B;" / ";B;"=";
260 X=A
270 GOTO 400
400 INPUT C
410 IF C=X THEN 10
420 GOTO 40
```

Run the program and enter your answer to the first problem. The program will repeat the problem until you input the correct answer.

How It Works

The flowchart in Figure 2 on page 136 and the following description show how this program creates an electronic version of the familiar flash cards used to teach arithmetic facts.

LINES 10 and 20 select two random numbers for A and B.

LINE 30 repeats the selection if B is larger than A. With this instruction, $A-B$ will always be a positive number. This avoids negative answers to the subtraction problems.

LINE 40 sets G equal to a random number from 1 to 4.

LINE 50 sends the computer to one of four places, depending on the value of G. This branching allows the program to create a random assortment of problems in addition, subtraction, multiplication, and division.

LINES 100-120 print an addition problem and set X equal to the answer.

LINES 150-170 print a subtraction problem and set X equal to the answer.

LINES 200-220 print a multiplication problem and set X equal to the answer.

LINES 250-270 print a division problem and set X equal to the answer. Notice that the answer is the integer A, not a fraction.

LINE 400 inputs the answer from the keyboard.

LINE 410 sends the computer to Line 10 to create a new problem if the answer is correct.

LINE 420 sends the computer to Line 50 to reprint the problem if the answer is not correct.

Experiment 5: Input Module

Having to press **ENTER** after typing in the answer is clumsy. You can avoid the use of the **ENTER** key with this program module that replaces Line 400. These instructions work together and allow you to input any number up to three digits long.

The program flowchart (Figure 2) is unchanged. The INPUT ANSWER function is replaced with the INPUT MODULE, Lines 400-490, below. This and other modules are explained in Section 3.

Press **BREAK** to stop the program, then add these instructions to replace Line 400:

```
400 REM ... INPUT MODULE ...
410 B$=""
420 A$=INKEY$
430 IF A$="" THEN 420
440 PRINT A$;
450 B$=B$+A$
```

Lesson 17: Math Teacher (continued)

```
460 IF X>9 AND LEN(B$)<2 THEN 420
470 IF X>99 AND LEN(B$)<3 THEN 420
480 PRINT
490 IF VAL(B$)=X THEN 10 ELSE 50
```

Now run the program and notice that you don't have to press the ENTER key. You do have to type the same number of digits as the correct answer, however, for the program to work. If the answer is 17, for example, you must type two digits before the computer will respond.

How It Works

In this program, the input module uses the string variable A\$ to store each key you type and B\$ to store the combined keys that make up the number you input.

The INKEY function sets A\$ to the key you type. Unlike the INPUT statement, this function reads the keyboard immediately, and does not wait for the ENTER key.

Line 450, each key you type is combined with the keys in B\$. If you type **3** and **2**, for example, B\$ will contain 32.

The function LEN(B\$) shows how many characters (numbers or letters) are stored in B\$. The checks in Lines 460 and 470 continue inputting keys from the keyboard until your answer has the same number of digits as the correct answer.

Then the two numbers are compared, in Line 490. The function VAL(B\$) turns B\$ into its numeric value. If B\$ contains 25, VAL(B\$) would be the number 25.

Experiment 6: Adjust the Skill Level

One way to adjust the difficulty of the problems this program generates is to adjust the size of the numbers picked for A and B. Stop the program and type **L I S T 1 0 - 5 0** to list these lines of your program.

Add a new instruction to input the skill level, S. Then change RND functions so that the numbers A and B will be selected from a larger range of possibilities. With a low skill level of 5, both A and B will be numbers between 1 and 5. With a higher skill level such as 15, both A and B will range from 1 to 15 and the problems will be much more difficult to solve.

To make this change, add these instructions:

```

5 INPUT "SKILL LEVEL";S

10 A=RND(S)

20 B=RND(S)

```

When you run this version, the computer will ask SKILL LEVEL? If you input a **10**, the program will work as before. To create easier problems, use a lower number. If you select a skill level that is too high, or if you wish to change your selection at any time, just press **BREAK** and run the program again.

How It Works

Here is the complete listing and full description of each instruction. The spaces have been added between lines in this listing so that the sections will match the flowchart in Figure 3.

```

5 INPUT "SKILL LEVEL";S

10 A=RND(S)
20 B=RND(S)
30 IF B>A THEN 10

40 G=RND(4)
50 ON G GOTO 100, 150, 200, 250

100 PRINT A;"+";B;"=";
110 X=A+B
120 GOTO 400
150 PRINT A;"-";B;"=";
160 X=A-B
170 GOTO 400
200 PRINT A;"*";B;"=";
210 X=A*B
220 GOTO 400
250 PRINT A*B;" / ";B;"=";
260 X=A
270 GOTO 400

400 REM ... INPUT MODULE ...
410 B$=""
420 A$=INKEY$
430 IF A$="" THEN 420
440 PRINT A$;
450 B$=B$+A$
460 IF X>9 AND LEN(B$)<2 THEN 420
470 IF X>99 AND LEN(B$)<3 THEN 420
480 PRINT

490 IF VAL(B$)=X THEN 10 ELSE 50

```

Lesson 17: Math Teacher (continued)

LINE 5 inputs the skill level, S.

LINES 10-30 select random numbers from 1 to S for the variables A and B.

LINES 40-50 select the type of problem (+, -, *, or /) and branch to the appropriate section of the program.

LINES 100-260 print the problem and set the variable X equal to the correct answer.

LINES 400-480 are a program module that inputs a number from the keyboard without using the ENTER key.

LINE 490 tests to see if the number typed is equal to the answer, X. If so, the program creates a new problem; and if not, the old problem is repeated.

Press **CLEAR**, type **LIST 100-260**, and press **ENTER**. The computer will list Lines 100 to 200 on the screen. In Figure 3 on page 137 these lines are labeled PRINT PROBLEM. These same instructions also appear in Figure 2, where PRINT PROBLEM is shown as four separate functions. There is no "correct" way to draw a flowchart, and you can combine instructions any way that helps you see the overall pattern. This entire section, for example, could appear as a block labeled Math Teacher in a flowchart of this book.

Experiment 7: Math Teacher

This final program is recorded on the Lesson 17 cassette. Load the program from the cassette with **CLOAD**, and type **RUN**.

Type the answers to the problems on the screen as fast as you can. If you answer quickly and correctly, the problems will include larger numbers and be more challenging. Miss a few answers or slow down in your response, and the problems will get easier again. As you continue to work with this program, the level of difficulty will adjust to match your skill and give you an ideal challenge that is neither boring nor too difficult. After 20 problems, you get a report card and the chance to go again.

This programming technique is called Time Response Monitoring, or TRM. In this program, the response time is monitored and used to control the average difficulty of the problems.

How It Works

The flowchart in Figure 4 on page 138 shows that three program modules are used for INPUT, TRM, and REPORT CARD. These and other modules are described in Section 3.

Lesson 17: Math Teacher (continued)

LINES 1-25 initialize the program by setting the skill level S to 5 and the number of errors E to 0. The FOR/NEXT loop is set to cycle 20 times and create 20 problems before displaying the report card.

The variable S is continuously adjusted to vary the skill level. This level may be increased or decreased after each answer, depending on the response time. Answer quickly and S is increased. Take longer or get a wrong answer and S is reduced.

LINES 29-70 create the problem, using two random numbers A and B . At the start, A and B are random numbers from 1 to 5. If the skill level increases, S will increase, and the random numbers will be selected from a larger range. In Line 70 the program branches to one of four sections, depending upon the random value of G .

LINES 99-260 print the problem on the screen. Four separate sections, beginning at Lines 100, 150, 200, and 250, are used to create addition, subtraction, multiplication, and division problems. In each section, the variable X is set equal to the correct answer.

LINES 400-480 are an input module that gets a number from the keyboard. This module eliminates using the ENTER key after each number.

LINES 500-620 are the TRM module that measures the response time and adjusts the skill level after each problem. This module increases or decreases the variable S that is used in Lines 30 and 40 to select the numbers A and B .

LINE 630 completes the FOR/NEXT loop. If less than 20 problems have been answered, this instruction sends the computer back to Line 25 to create another problem.

LINES 700-780 are a report card that prints the results. This program module also inputs a key from the keyboard. If the key pressed is Y , the program continues with the current skill level and 20 new problems.

```
1 REM ... MATH TEACHER ...
10 S=5:CLS
20 E=0
25 FOR L=1 TO 20

29 REM ... CREATE PROBLEM ...
30 A=RND(S)
40 B=RND(S)
50 IF A<B THEN GOTO 30
60 G=RND(4)
70 ON G GOTO 100,150,200,250

99 REM ... PRINT PROBLEM ...
100 PRINT A;"+";B;"=";
```

Lesson 17: Math Teacher (continued)

```
110 X=A+B
120 GOTO 400
150 PRINT A;"-";B;"= ";
160 X=A-B
170 GOTO 400
200 PRINT A;"*";B;"= ";
210 X=A*B
220 GOTO 400
250 PRINT A*B;" / ";B;"= ";
260 X=A

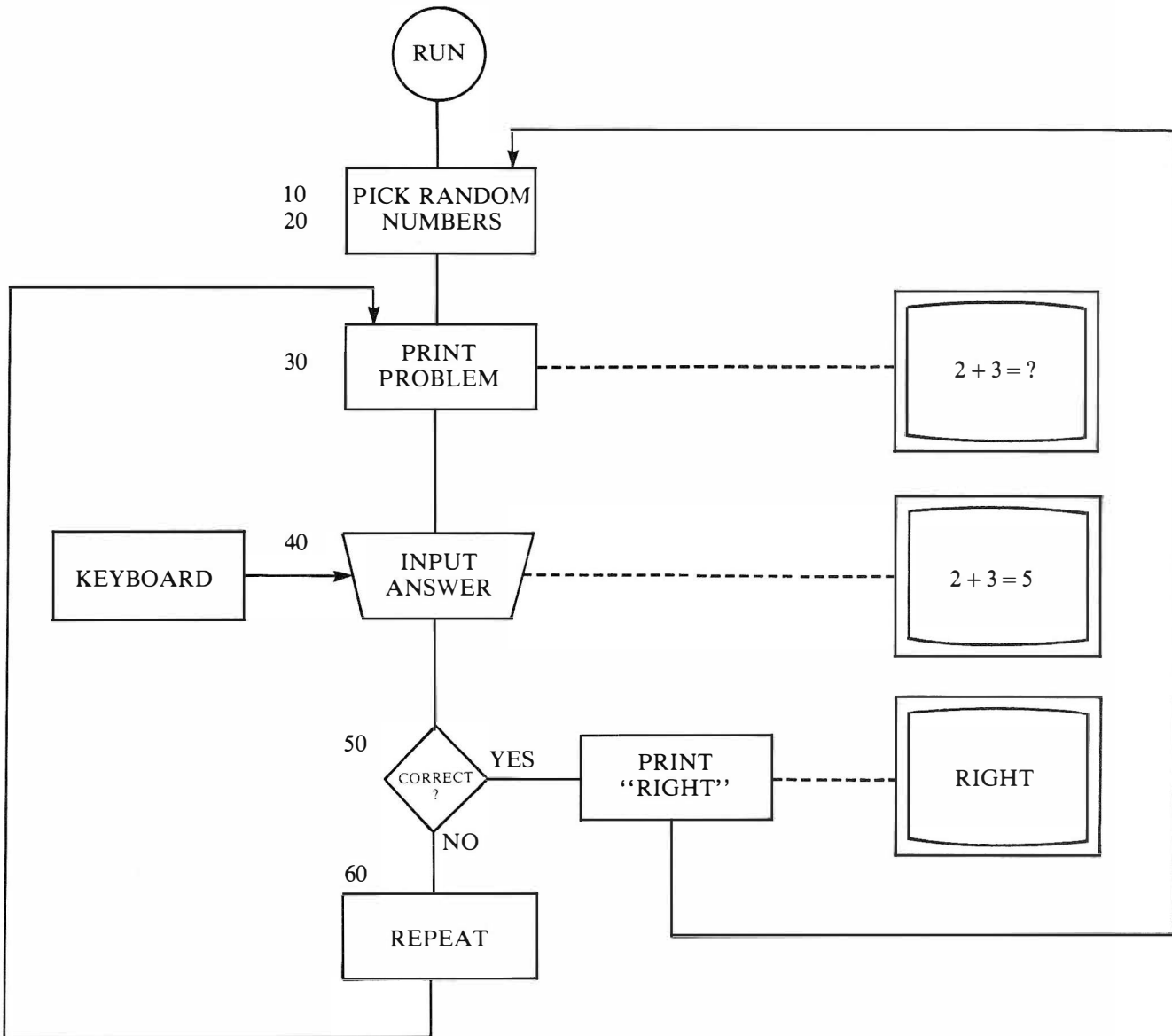
400 REM ... INPUT MODULE ...
410 B$="" :T=0
420 A$=INKEY$
425 T=T+1
430 IF A$="" THEN 420
440 PRINT A$;
450 B$=B$+A$
460 IF X>9 AND LEN(B$)<2 THEN 420
470 IF X>99 AND LEN(B$)<3 THEN 420
480 PRINT

500 REM ... TRM MODULE ...
510 IF VAL(B$)=X THEN 600
520 PRINT "SORRY, THE ANSWER IS ";X
530 E=E+1
540 S=S-4
550 GOTO 70
600 IF T>400 THEN T=400
610 S=S+2-INT(T/100)
620 IF S<5 THEN S=5
630 NEXT L

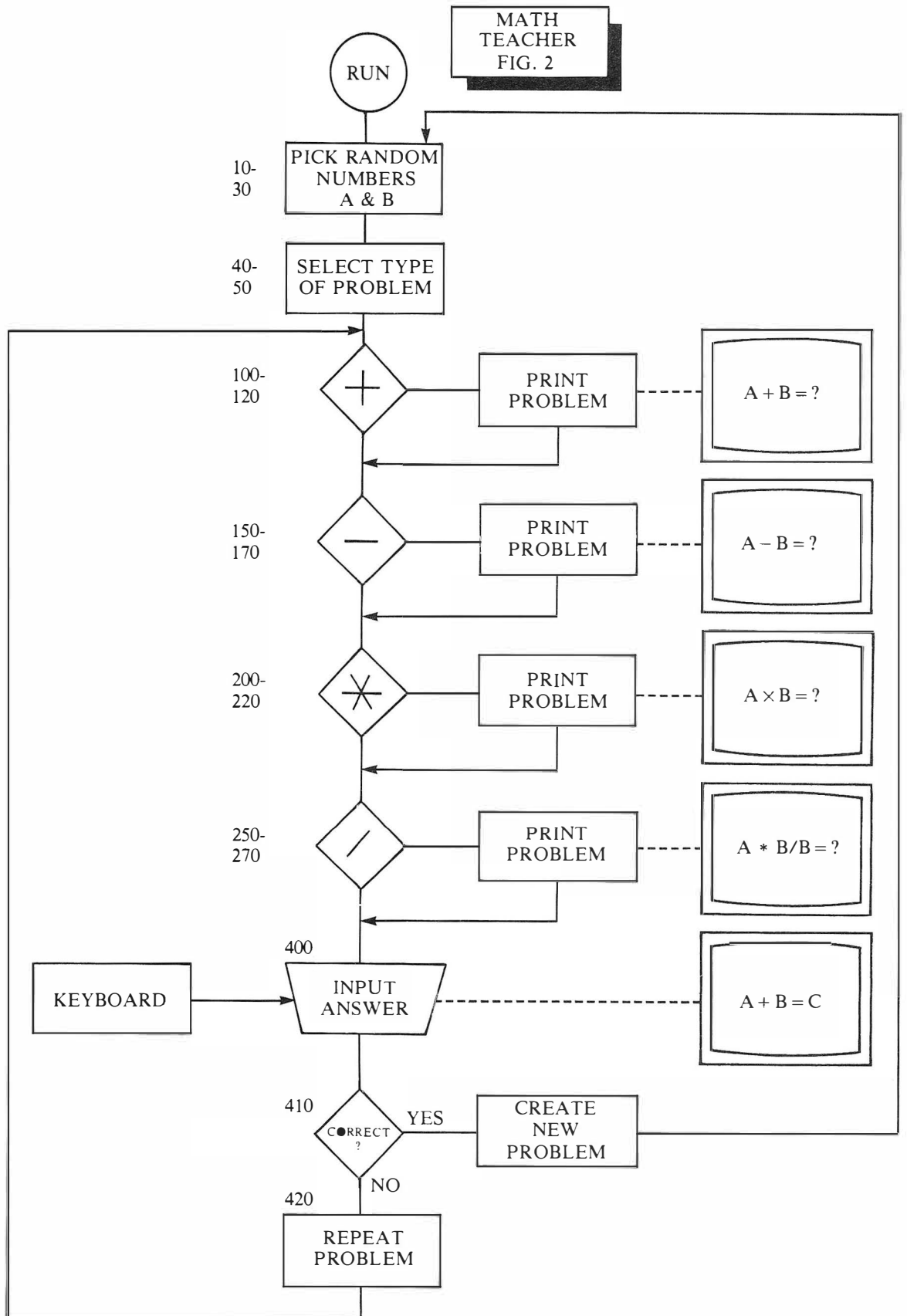
700 REM ... REPORT CARD ...
710 CLS:PRINT
720 PRINT "          ... REPORT CAR
D ...":PRINT
730 PRINT "    YOU GOT";20-E"
740 PRINT "    OUT OF 20 CORRECT"
:PRINT
750 PRINT "    YOUR SKILL LEVEL I
S";S:PRINT
760 PRINT "    SAME PLAYER GO AGA
IN (Y,N)?"
770 Y$=INKEY$:IF Y$="" THEN 770
780 IF Y$="Y" THEN CLS:GOTO 20
```

Time Response Monitoring and TRM programming are registered trade marks of The Image Producers, Inc.

MATH
TEACHER
FIG. 1

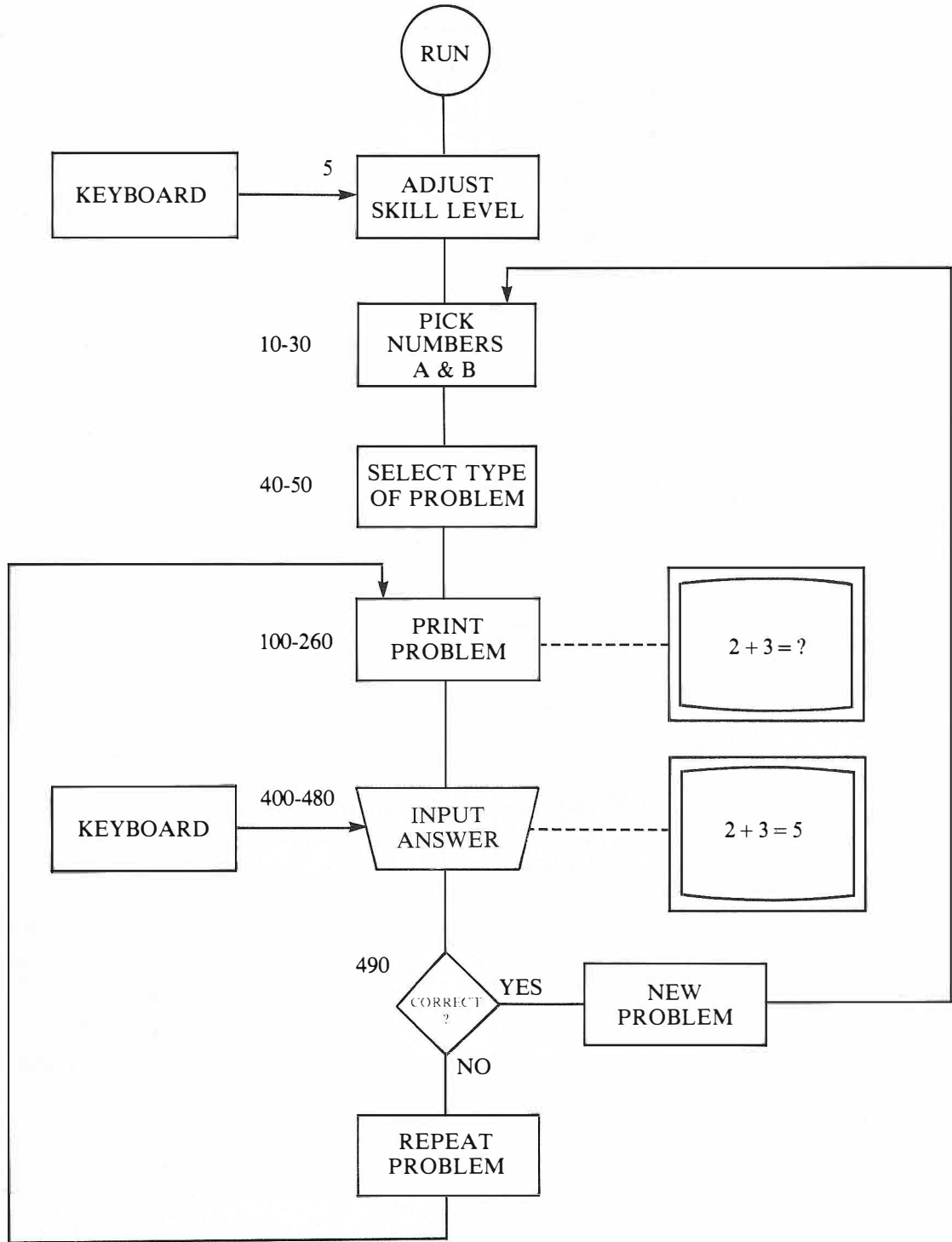


Lesson 17: Math Teacher (continued)



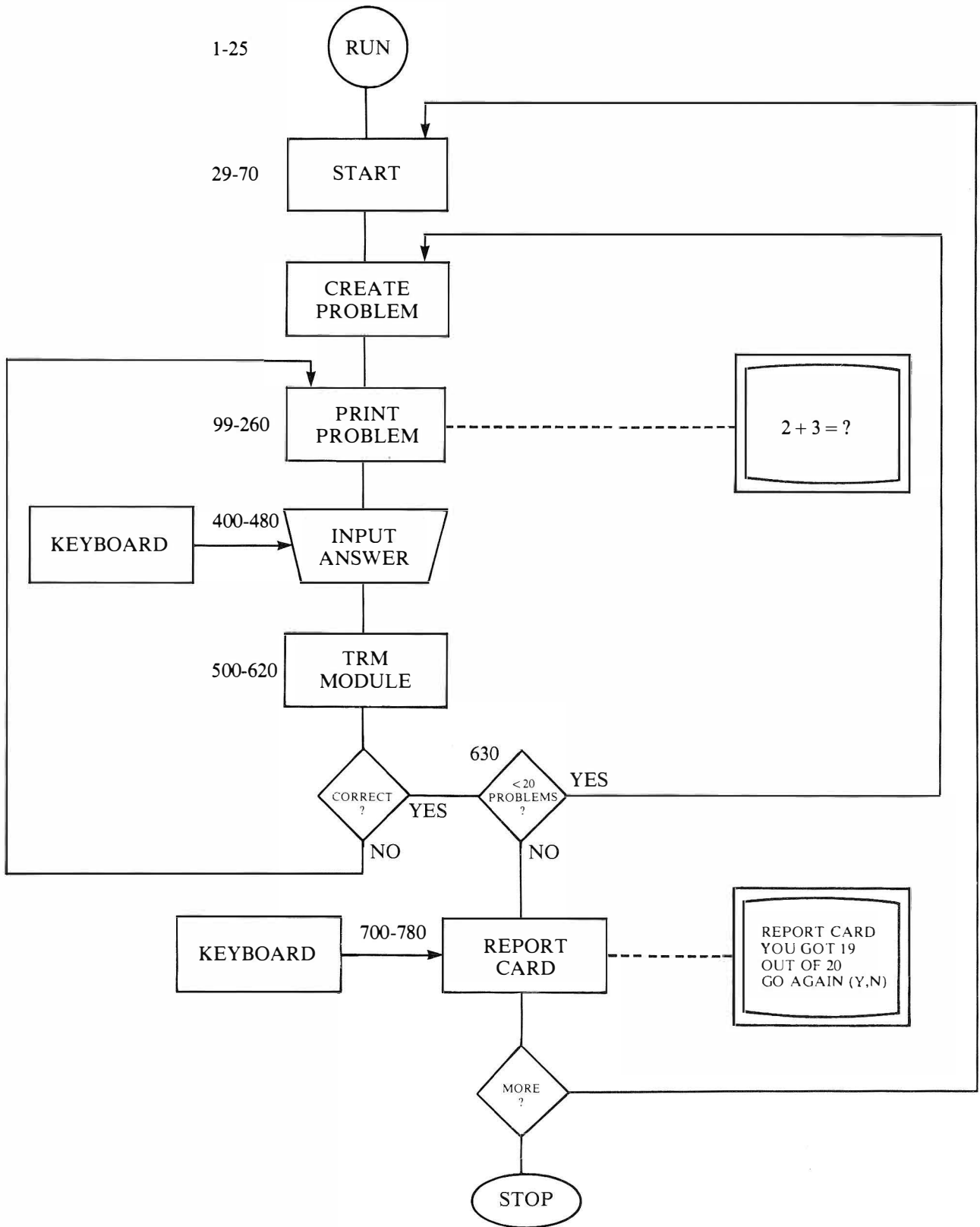
Lesson 17: Math Teacher (continued)

MATH
TEACHER
FIG. 3



Lesson 17: Math Teacher (continued)

MATH
TEACHER
FIG. 4



Lesson 18: Hangperson

String Manipulation and Game Design

In this lesson you will create your own version of a popular logic game. You will begin by building a short program and then add features and improvements to the design.

Each section in the short program is expanded and improved to create the final game. You can begin with Experiment 1 and create the game from scratch, or load Lesson 18 from the cassette and see the final result before you study this lesson in detail.

The first two experiments show how ASCII numbers are used for representing characters on the keyboard and how strings or words can be measured and evaluated. With this background you will then write a short program for playing the game.

Experiments 5 through 9 expand that program and add many features to make it easier and more fun to play.

This programming technique — building a simple version of a program and expanding its features — is a very good design method. Follow along as these experiments show you how to create Hangperson — a word guessing game for two people.

Experiment 1: ASCII Numbers

Your color computer uses a number code to represent the keys on the keyboard. You can find the number associated with each key with this short program. Clear your computer with **NEW** and enter these instructions:

```
10 INPUT X$
20 PRINT X$; ASC(X$)
30 GOTO 10
```

Run the program, press any key, then press **ENTER**. The computer will set the string variable `X$` equal to the key you type, print the key and its ASCII number, and go back to the beginning for another input. Try inputting letters to see the range of numbers used for the alphabet. Depending on the letters you pick, your screen could look like this:

```
? A
A 65
? B
B 66
? C
C 67
```

Lesson 18: Hangperson (continued)

The ASCII numbers 65, 66, and 67 are used to represent the letters A, B, and C. The complete alphabet is represented by the ASCII numbers 65 (A) through 90 (Z).

Punctuation, numbers, and symbols also have ASCII numbers. Try typing the hyphen `-` and see that its ASCII number is 45. The colon and the comma are used by the computer as control words.

Experiment 2: LEN and MID Functions

You can use the LEN function to find the number of characters in a string. In Hangperson, this function is used to find out how many letters are in the code word. Stop the program with `BREAK` and change Line 20 by entering this new instruction:

```
20 PRINT LEN(X$)
```

Now run the program. Enter any word and the computer will print the number of characters in the word, like this:

```
? A
1
? ALL
3
? ALPHABET
8
?
```

The MID function can also be used to print any specific character in a string or word. Stop the program and change Line 20 to this new instruction:

```
20 PRINT MID$(X$,3,1)
```

The number 3 tells the computer to skip to the third character, and the number 1 is the number of characters after that to be printed. In this example, the function prints the third letter in the word you enter.

Run the program and enter a word. When you press `ENTER`, the computer will print the third letter or character in the word, like this:

```
? TEST
S
? COMPUTER
M
? 45678
6
?
```

In each case, the computer printed the third character. You can also use the MID function to print any other portion of a string by changing the numbers 3 and 1 in the instruction. Later in the program you will use the MID function to check each letter in the code word.

Experiment 3: Input Word and Load Array

The first step in designing Hangperson is putting the code word into the program and creating an array to store the characters that have been guessed correctly. The program begins by setting the variable W\$ equal to the code word. Then the array S is dimensioned to hold a number for each character in the code word, W\$. Each of these positions in the array is set equal to the number 45.

Later you will use this array to store the ASCII numbers for the letters that have been guessed correctly. For now, each position in the array holds the number 45 — the ASCII number for a hyphen or dash.

Begin building Hangperson by clearing the computer with **NEM** and entering these instructions:

```
10 INPUT W$
20 DIM S(LEN(W$))
30 FOR L=1 TO LEN(W$)
40 S(L)=45
50 NEXT L
```

Experiment 4: Print the Array

This next section in the program prints the characters whose ASCII numbers are stored in the array. If the array contained the numbers 65, 66, and 67, for example, these instructions would print the characters A, B, and C on the screen. Now add this section of the program that begins on Line 300 with a remark and prints the characters whose ASCII numbers are stored in the array.

```
300 REM ... PRINT ARRAY ...
310 FOR L=1 TO LEN(W$)
320 PRINT CHR$(S(L));
330 NEXT L
```

Lesson 18: Hangperson (continued)

Type **LIST** and press **ENTER** to see the complete program. Compare your listing to the one below and correct any errors. (Blank lines are added to separate this listing into the three sections shown in the flowchart.)

```
10 INPUT W$

20 DIM S(LEN(W$))
30 FOR L=1 TO LEN(W$)
40 S(L)=45
50 NEXT L

300 REM ... PRINT ARRAY ...
310 FOR L=1 TO LEN(W$)
320 PRINT CHR$(S(L));
330 NEXT L
```

A flowchart for this program is shown in Figure 1 on page 151. As you can see, the computer will create an array with a position for each letter in the word. Then the computer will fill the array by storing the number 45 in each position. Finally, the numbers stored in the array are converted to characters and printed.

Run the program and enter a code word. The computer will print a dash for each letter in the word, like this:

```
? TESTING
-----
OK
```

Experiment 5: Input and Score Guess

With these additions, you will be able to play Hangman. Add these instructions to input a letter and score the results:

```
100 CLS
110 INPUT G$
200 REM ... SCORE ...
210 FOR L=1 TO LEN(W$)
220 IF G$=MID$(W$,L,1) GOTO 240
230 GOTO 250
240 S(L)=ASC(G$)
250 NEXT L
```

After clearing the screen in Line 100, the computer will set G\$ equal to a letter typed on the keyboard. Then the letter is compared with each character in the code word. If they are equal, the computer sets the matching position in the array equal to the ASCII number of the letter. Finally, the numbers stored in the array are converted to characters and printed.

Lesson 18: Hangperson (continued)

Run the program, enter **TESTING** as the code word, and then enter the letter T. Your screen should look like this:

```
? T
T__T____
OK
```

As you can see, the code word has seven letters. The first and the fourth letter are T.

One more change will complete the program. Add this instruction and cause the program to loop back for another guess after printing the array:

```
400 GOTO 110
```

Run the program again, enter **TESTING** for the code word and then try these letters: T, A, S, I, G. Your screen will show:

```
? T
T__T____? A
T__T____? S
T-ST____? I
T-STI____? G
T-STI-G?
```

Each letter you input that matches a letter in the code word is added to the printout. Continue entering letters to complete spelling the code word. Notice that the computer keeps track of the letters that match the code word to show your progress.

Hangperson is now complete and you can play the game with a friend. You can run the program, enter a code word (without showing your friend), and then have your friend try to guess the word by entering letters. After the word has been guessed, press **BREAK** to stop the program. To go again, run the program and input a new code word.

The flowchart for the program is shown in Figure 2 on page 152 and the complete listing is shown below.

```
10 INPUT W$
20 DIM S(LEN(W$))
30 FOR L=1 TO LEN(W$)
40 S(L)=45
50 NEXT L
100 CLS
110 INPUT G$
```

Lesson 18: Hangperson (continued)

```
200 REM ... SCORE ...
210 FOR L=1 TO LEN(W$)
220 IF G$=MID$(W$.L,1) GOTO 240
230 GOTO 250
240 S(L)=ASC(G$)
250 NEXT L

300 REM ... PRINT ARRAY ...
310 FOR L=1 TO LEN(W$)
320 PRINT CHR$(S(L));
330 NEXT L

400 GOTO 110
```

Experiment 6: Improve Inputs

Now that the basic game is running, you can begin adding improvements to the program. These next experiments add details that make the program much more interesting.

Begin by changing Line 10 so that it clears the screen and prints the message CODE WORD in addition to setting the word you type equal to W\$.

```
10 CLS:INPUT "CODE WORD";W$
```

The next improvement in the program will eliminate pressing the ENTER key after each letter you guess. Replace the standard input instruction in Line 110 with these lines:

```
110 G$=INKEY$
120 IF G$="" GOTO 110
130 PRINT G$
```

The difference between using INPUT and INKEY\$ is that INPUT requires the **ENTER** key after each letter. These instructions loop (repeat) until a key is pressed, then G\$ is set equal to the character and printed automatically. Run the program again and see how these changes improve the game.

Experiment 7: Add Sound Effects

These instructions add sound effects whenever you guess a letter correctly, and play random sounds to match the printout:

```
245 SOUND 100,2

325 SOUND S(L),1
```

Run the program, enter a fairly long word, adjust the volume on your TV, and try this addition to the program.

Experiment 8: Allow Only Eight Guesses

The classic Hangperson game only allows eight wrong answers. Add this feature by using the variable `W` to store the number of guesses that don't match a letter in the code word. If `W` is less than 8, the game continues. If `W` equals 8, the person is hanged and the game is over.

Begin by listing the scoring section of the program by pressing **CLEAR** and entering **LIST 200-250**. Your screen will show:

```
200 REM ... SCORE ...
210 FOR L=1 TO LEN(W$)
220 IF G$=MID$(W$,L,1) GOTO 240
230 GOTO 250
240 S(L)=ASC(G$)
245 SOUND 100,2
250 NEXT L
```

Add this instruction to the beginning of the program. This sets `W`, the number of wrong answers, to 0 when the program starts.

```
60 W=0
```

A wrong answer is scored only if the letter picked `G$` does not match any letter in the code word `W$`. To keep track of any match, add these two instructions. Line 205 sets `M`, the number of matches, to 0 when the scoring begins. Line 247 sets `M` equal to 1 if `G$` matches a letter in the code. With these changes, `M` will be 0 if `G$` does not match any letter in the code word.

```
205 M=0
```

```
247 M=1
```

Now you can test to see if `M` is zero, and increase `W` if it is not, with this instruction:

```
260 IF M=0 THEN W=W+1
```

The final step will change the program so that it only allows eight wrong guesses. This new instruction replaces Line 400 and sends the computer back for another input only if the number of wrong guesses is less than 8:

```
400 IF W<8 GOTO 110
```

As a final touch, add this instruction to print the correct answer if the number of wrong answers exceeds the limit (and the game is lost).

Lesson 18: Hangperson (continued)

```
410 PRINT "SORRY"  
420 PRINT "THE ANSWER WAS"  
430 PRINT W$
```

With these changes, the program repeats if the number of wrong answers is less than 8, and prints the answer if there are 8 mistakes.

Experiment 9: Test for Correct Answer

With these three instructions your program will check to see if the code word has been broken. If all the letters in the code have been guessed, the program will print a winning message and stop. An easy way to test for the correct answer is to see if there are any dashes left in the array. If there are no dashes, then all letters have been guessed correctly.

Set the variable `D` to 0 before printing the array. As the array is printed, check each position in the array and set `D` equal to 1 if any dashes are present. Remember that the array will contain the number 45 if a dash is printed on the screen. If there are no dashes `D=0`, print the message in Line 340 and end the program.

```
305 D=0  
322 IF S(L)=45 THEN D=1  
340 IF D=0 THEN PRINT:PRINT " YOU WIN!":END
```

Now run this version of the program and see if the improvements make the game easier and more fun to play.

Experiment 10: Compare Designs

Compare the present design, shown in Figure 3, with the previous design in Figure 2. Except for the ending sequence that limits the game to eight wrong answers, and the winning announcement, the two designs are very much alike.

A good programming technique is to build a simple version of a design; then add improvements and expansions. You can create longer or more complex programs by working in small steps and testing the results as you go along.

A listing of the complete program is shown on the next page. As before, blank lines have been added to separate the listing into sections that match the flowchart.

The program for the next experiment is recorded on the Lesson 18 cassette. This version of Hangperson uses a slightly different screen format and includes a picture that “grows” with every guess that doesn’t match the code word.

```
10 CLS:INPUT "CODE WORD";W$
20 DIM S(LEN(W$))
30 FOR L=1 TO LEN(W$)
40 S(L)=45
50 NEXT L
60 W=0

100 CLS
110 G$=INKEY$
120 IF G$="" GOTO 110
130 PRINT G$

200 REM ... SCORE ...
205 M=0
210 FOR L=1 TO LEN(W$)
220 IF G$=MID$(W$,L,1) GOTO 240
230 GOTO 250
240 S(L)=ASC(G$)
245 SOUND 100,3
247 M=1
250 NEXT L
260 IF M=0 THEN W=W+1

300 REM ... PRINT ARRAY ...
305 D=0
310 FOR L=1 TO LEN(W$)
320 PRINT CHR$(S(L));
322 IF S(L)=45 THEN D=1
325 SOUND S(L),1
330 NEXT L

340 IF D=0 THEN PRINT:PRINT "YOU
WIN!":END

400 IF W<8 GOTO 110
410 PRINT "SORRY"
420 PRINT "THE ANSWER WAS"
430 PRINT W$
```

Experiment 11: Hangperson

This word guessing game is usually played by two people. One person programs a secret code word and the other tries to guess what it is. Letters are guessed one at a time. If the letter guessed is contained in the word, the letter is shown in its correct position. Guess a letter that is not in the word and the Hangperson picture grows. If you miss eight letters, the picture is complete, and the person is hanged.

Lesson 18: Hangperson (continued)

Begin by loading the Lesson 18 cassette with **CLOAD**, then run the program. When CODE WORD? appears, type any word and press **ENTER**. Then a second person can type letters one at a time and try to guess the secret word.

Play the game several times and see what the program does when right and wrong guesses are made, then try playing this game with a friend. The flowchart in Figure 4 on page 154 shows how the program is designed. Notice that the computer can take one of several paths, depending on the status of the game. For example, if the last letter guessed was not correct, the computer will respond YES to the question NO MATCH? The next step will be to print the picture on the screen. If there are less than eight wrong answers, the computer will go back to Line 100 to input another guess. If this is guess number eight, the computer will print the correct answer and ask if you wish to try again.

Play the game several times, using the flowchart to see what the computer is doing as you run the program.

How It Works

If you haven't done the experiments in this lesson, you could see how it works by starting with Experiment 1 and building this program step by step. You will quickly see how an easier version can be written with only a few instructions. Then follow the experiments as each section in the program is expanded. The picture is created with graphics characters, as explained in Lesson 21: Graphics.

The following listing shows the complete program as recorded on the cassette. Spaces have been added to this listing to match the flowchart in Figure 4.

```
1 REM ... HANGPERSON ...
10 CLS:INPUT " CODE WORD":W$
20 DIM S(LEN(W$))
30 FOR L=1 TO LEN(W$)
40 S(L)=45
50 NEXT L
60 W=0

100 CLS
105 PRINT " ... HANGPERSON
... "
107 PRINT " GUESS A LETTE
R"
110 G$=INKEY$
120 IF G$="" GOTO 110
130 PRINT @72.G$:" ; ";
```

Lesson 18: Hangperson (continued)

```
200 REM ... SCORE ...
205 M=0
210 FOR L=1 TO LEN(W$)
220 IF G$=MID$(W$,L,1) GOTO 240
230 GOTO 250
240 S(L)=ASC(G$)
245 SOUND 200.6
247 M=1
250 NEXT L
260 IF M=0 THEN W=W+1

300 REM ... PRINT ARRAY ...
305 D=0
310 FOR L=1 TO LEN(W$)
320 PRINT CHR$(S(L));
322 IF S(L)=45 THEN D=1
325 SOUND S(L),1
330 NEXT L

340 IF D=0 THEN PRINT @448, "YOU
WIN!":GOTO 420

350 IF M=0 THEN GOSUB 500

400 IF W<8 GOTO 110
405 PRINT
410 PRINT @448," SORRY, THE WORD
WAS: ";W$

420 PRINT "TRY AGAIN (Y,N)?";
430 X$=INKEY$:IF X$="" GOTO 430
440 IF X$="Y" THEN RUN ELSE END

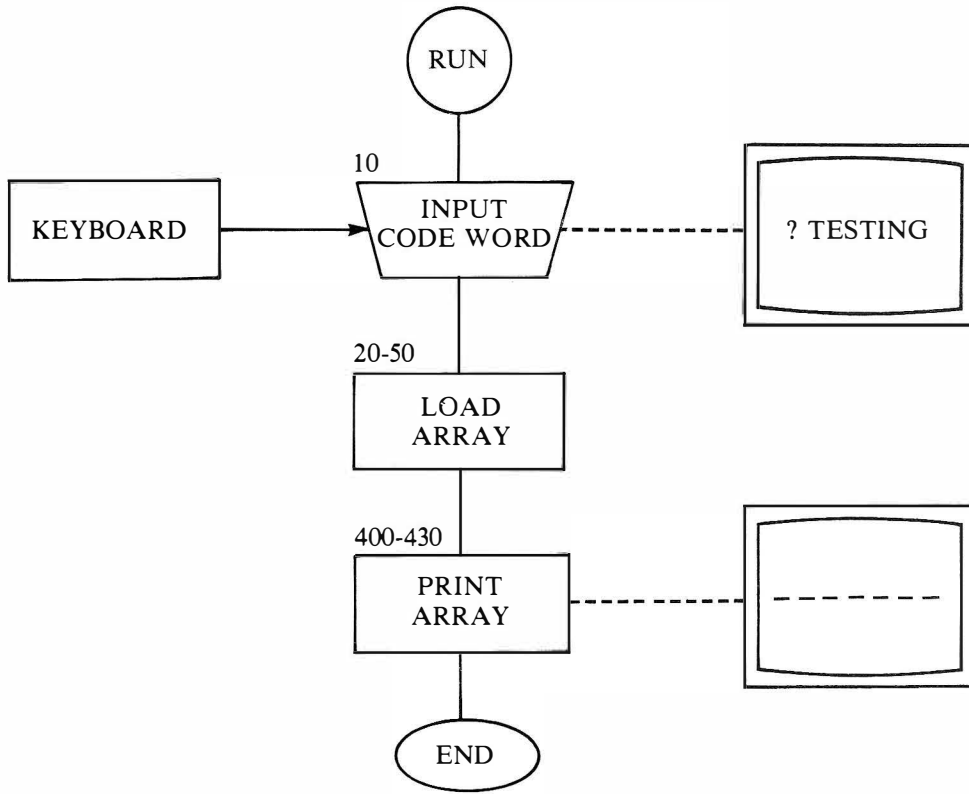
500 REM ... DRAW PICTURES ...
508 RESTORE
510 FOR Y=5 TO 5+W
520 FOR X=11 TO 17
525 READ C
527 SOUND C,1
528 PRINT @32*Y+X,"+";
530 PRINT @32*Y+X,CHR$(C);
540 NEXT X:NEXT Y
560 RETURN

600 DATA 208,211,215,211,219,211
,208
610 DATA 145,223,254,223,253,223
,146
620 DATA 208,220,223,223,223,220
,208
```

Lesson 18: Hangperson (continued)

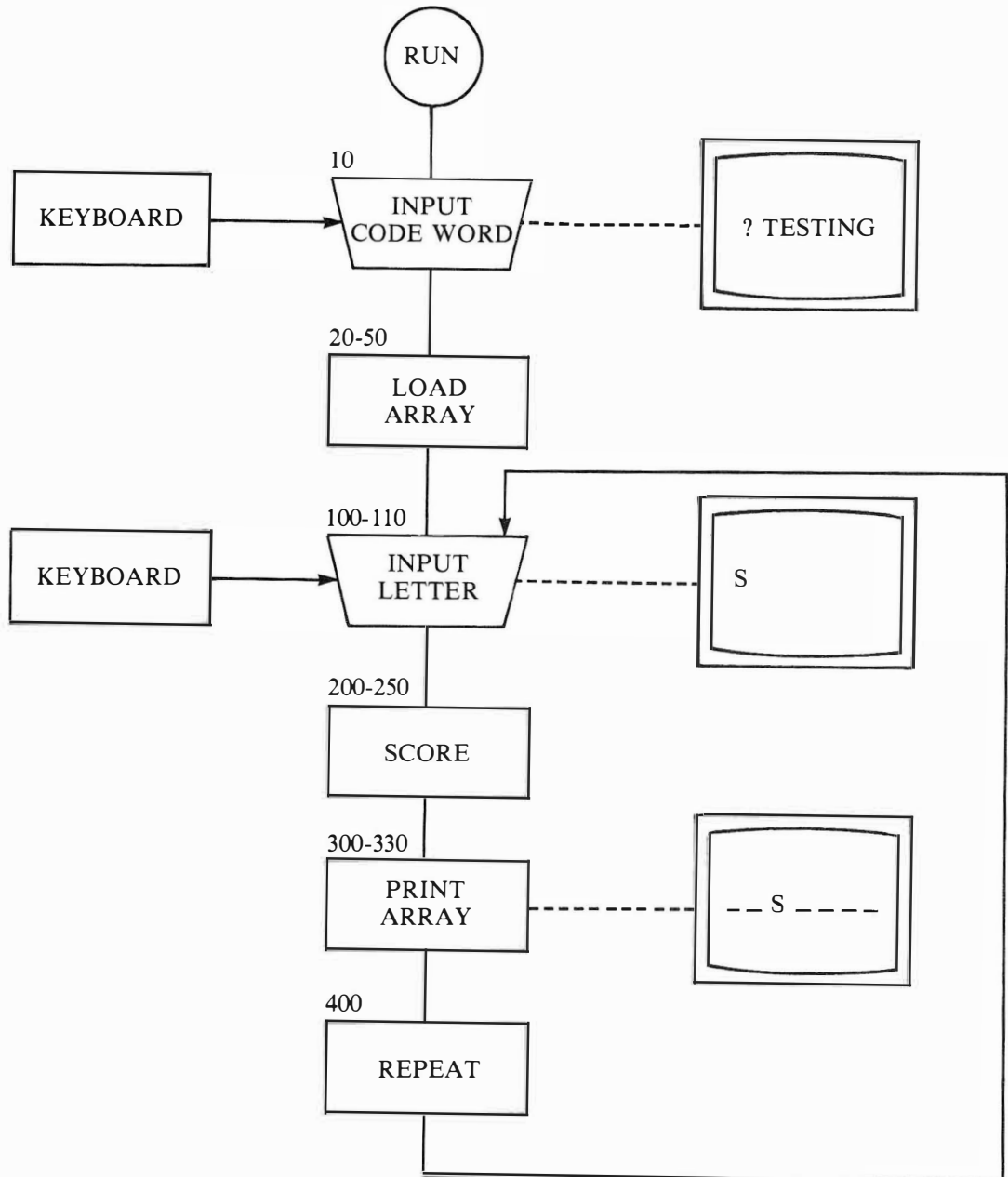
630 DATA 211,211,211,223,211,211
.211
640 DATA 223,208,223,223,223,208
.223
650 DATA 223,208,223,223,223,208
.223
660 DATA 254,208,223,220,223,208
.253
670 DATA 208,208,223,208,223,208
.208
680 DATA 208,243,251,208,247,243
.208

HANG PERSON
FIG. 1

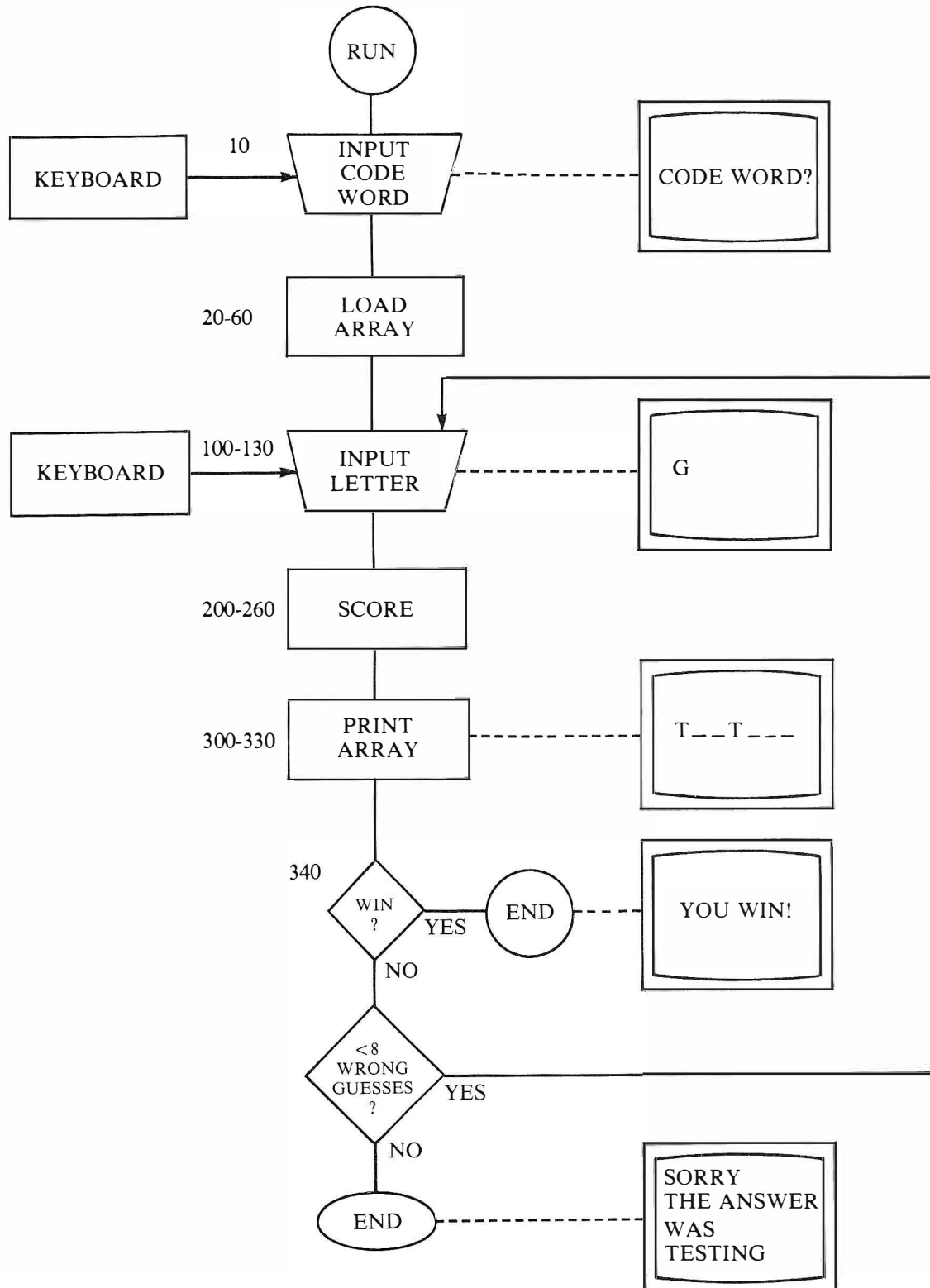


Lesson 18: Hangperson (continued)

HANG PERSON
FIG. 2

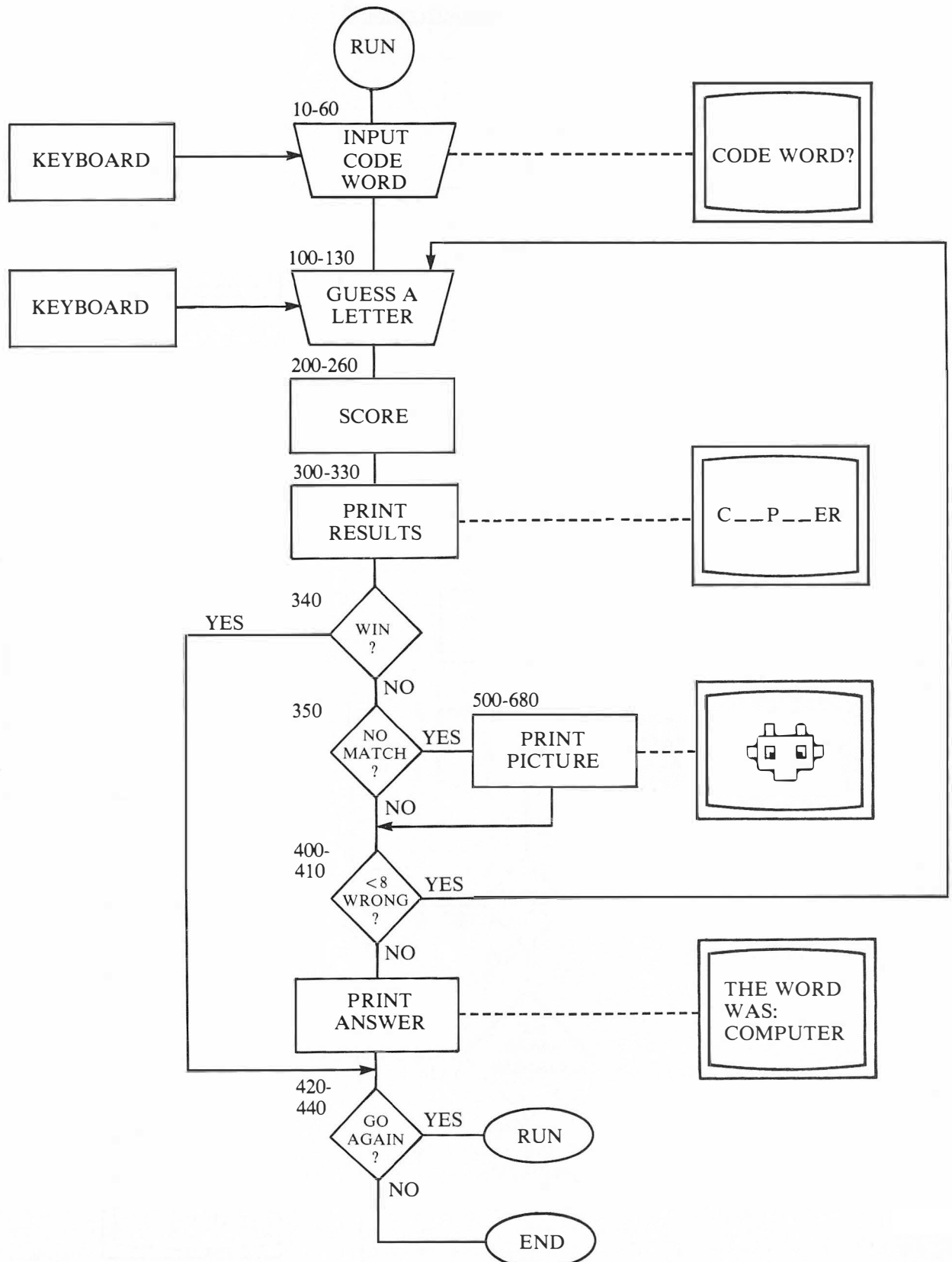


HANG PERSON
FIG. 3



Lesson 18: Hangperson (continued)

HANG PERSON
FIG. 4



Lesson 19: Music Teacher

Music Instruction and Game Design

There are several computer games on the market that create random tunes and test your ability to repeat each note correctly. Music Teacher is a software version of these games and uses the computer keyboard to input the notes.

This lesson begins with a review of the SOUND command. Then you will create an instrument, using the computer keyboard and the numbers 1 through 8 to play one octave in the key of C.

Program modules are used to input keys from the keyboard and to play music notes. The final version of Music Teacher can be loaded from the Lesson 19 cassette or created by following these experiments.

Experiment 1: Sound F,D

Begin by typing **NEW** to clear any old programs, then enter these instructions:

```
10 INPUT F,D
```

```
20 SOUND F,D
```

```
30 GOTO 10
```

Run this program, adjust the volume on your TV, and enter numbers between 1 and 255 for the frequency **F** and the duration **D** of a musical tone. You will have to press **ENTER** after each number. The first number you enter determines the pitch, with lower numbers producing a lower pitch. The second number sets the duration of the tone, with 1 being a short note and 255 being a very long note. To play a music scale, enter the numbers below.

When you run the program, the computer will print a question mark. Type the first number to select the frequency. Then the computer will type a double question mark. Type the second number to select the duration. With both numbers entered, the computer will play the note.

```
89, 2
```

```
108, 2
```

```
125, 2
```

```
133, 2
```

```
147, 2
```

```
159, 2
```

```
170, 2
```

```
176, 2
```

Experiment 2: Do-Re-Mi

Type **NEW** and press **ENTER** to clear the previous program, then enter this new program that plays a musical scale when you input the numbers 1 through 8.

Line 200 sets the variable `N` equal to the number you input. The instructions beginning at Line 600 convert `N` to a frequency number that matches a note in the musical scale. The sound is created with frequency 7 and duration 2.

```
200 INPUT N
600 REM ... PRINT & PLAY ...
610 PRINT N
620 IF N=1 THEN F=89
630 IF N=2 THEN F=108
640 IF N=3 THEN F=125
650 IF N=4 THEN F=133
660 IF N=5 THEN F=147
670 IF N=6 THEN F=159
680 IF N=7 THEN F=170
690 IF N=8 THEN F=176
700 SOUND F,2
710 GOTO 200
```

Run this program and enter numbers from 1 to 8 to play a musical scale. As before, you will have to press the **ENTER** key after each number. Lines 600 to 690 are a look-up table that converts these numbers into frequency numbers that match the scale in the key of C. To play a tune, try these numbers: 3, 2, 1, 2, 3, 3, 3, 2, 2, 2, 3, 5, 5.

Experiment 3: Input Note

You can eliminate using the **ENTER** key after each note with this next change. The `INKEY$` command reads the ASCII value of any key that is pressed, and does not wait for the **ENTER** key. Line 220 converts this ASCII value into a number between 1 and 8.

```
200 REM ... INPUT NOTE ...  
210 N$=INKEY$:IF N$="" GOTO 210  
220 N=ASC(N$)-48
```

Run this version of the program and see that the computer keyboard is much more like a piano or organ with this change.

The flowchart in Figure 1 on page 168 shows what the program is now doing. The INPUT NOTE section takes a key from the keyboard and sets N equal to the number typed. Then the PRINT & PLAY section converts the number into a frequency and plays the note with a duration of 2.

Here is a complete listing of the program in Figure 1:

```
200 REM ... INPUT NOTE ...  
210 N$=INKEY$:IF N$="" GOTO 210  
220 N=ASC(N$)-48  
  
600 REM ... PRINT & PLAY ...  
610 PRINT N  
620 IF N=1 THEN F=89  
630 IF N=2 THEN F=108  
640 IF N=3 THEN F=125  
650 IF N=4 THEN F=133  
660 IF N=5 THEN F=147  
670 IF N=6 THEN F=159  
680 IF N=7 THEN F=170  
690 IF N=8 THEN F=176  
700 SOUND F,2  
710 GOTO 200
```

Experiment 4: Add Subroutine

Lines 600-710 are a set of instructions that can be used in any program to convert a series of numbers into notes. You will use these instructions in two places in the final program. Whenever you have a set of instructions like these that are used more than once in a program, it can be helpful to create a subroutine.

The instruction GOSUB 600 tells the computer to transfer the program to Line 600. When the computer reads the word RETURN, it transfers the program back to the line following the GOSUB. Add these lines to your program to create the subroutine:

```
230 GOSUB 600  
240 GOTO 200  
  
710 RETURN
```

Lesson 19: Music Teacher (continued)

If you run the program you will see that it works exactly the same as before. The flowchart in Figure 2 on page 169 shows how the flow of the program has changed, however, with the PRINT & PLAY section acting as a subroutine, rather than part of the main program. This will be an advantage later, when you will use the subroutine again.

Experiment 5: Create a Game

Look ahead to Figure 3 on page 170 and see how the program becomes a game by adding four short program sections. Enter these next lines to form the START section of the program. This section will create an array S(15) to hold numbers representing notes in a tune, set the total number of notes in the tune T to 0, clear the screen, and print the title.

```
10 REM ... MUSIC TEACHER ...
20 DIM S(15)
30 T=0
40 CLS
50 PRINT "      ... MUSIC TEACHER
   .."
```

The ADD A NEW NOTE section adds a random note to the song by picking a random number from 1 to 8 and storing it in the array at position S(T).

```
100 REM ... ADD A NEW NOTE ...
110 CLS
120 T=T+1
130 S(T)=RND(8)
```

These instructions cycle once for each note in the song. They send the computer to the PRINT & PLAY subroutine to play the note and print its number on the screen. After the song is played, the screen is cleared.

```
150 REM ... PLAY SONG ...
160 FOR L=1 TO T
170 N=S(L):GOSUB 600
180 NEXT L
190 CLS
```

These next few instructions create a loop so that the INPUT NOTE section cycles once for each note in the song. The PRINT & PLAY subroutine is used to play the note typed on the keyboard. If any note doesn't match, the computer starts over with a new tune. If there are no mistakes, the computer goes to Line 100 to add another note to the tune.

```
205 FOR L=1 TO T
230 GOSUB 600
240 IF N<>S(L) GOTO 30
250 NEXT L
260 GOTO 100
```

Now run the game. The computer will pick a random note, play it, and print its number. Match the note on the keyboard and the computer will play a two-note song for you to copy. Try to copy as many notes as you can. If you make a mistake, the computer will start over with a new tune.

In the next few experiments, you will add several features including a display that shows you the music being played and some sound effects. You can continue to expand this software or load the final program from the Lesson 19 cassette.

How It Works

Here is the complete program listing. This program is shown in Figure 3. The subroutine beginning in Line 600 is used two places in the program. In the PLAY SONG section, the variable N is set equal to each number in the array. The subroutine converts N into a frequency and plays the note. The INPUT NOTE section also uses the subroutine to play the note. Here, N is set equal to the key typed on the keyboard before the subroutine is used. In both cases, the variable N stored the number of the note being played.

```
10 REM ... MUSIC TEACHER ...
20 DIM S(15)
30 T=0
40 CLS
50 PRINT " ... MUSIC TEACHER
... "

100 REM ... ADD A NEW NOTE ...
110 CLS
120 T=T+1
130 S(T)=RND(8)
150 REM ... PLAY SONG ...
```

Lesson 19: Music Teacher (continued)

```
160 FOR L=1 TO T
170 N=S(L):GOSUB 600
180 NEXT L
190 CLS

200 REM ... INPUT NOTE ...
205 FOR L=1 TO T
210 N$=INKEY$:IF N$="" GOTO 210
220 N=ASC(N$)-48
230 GOSUB 600
240 IF N<>S(L) GOTO 30
250 NEXT L
260 GOTO 100

600 REM ... PRINT & PLAY ...
610 PRINT N
620 IF N=1 THEN F=89
630 IF N=2 THEN F=108
640 IF N=3 THEN F=125
650 IF N=4 THEN F=133
660 IF N=5 THEN F=147
670 IF N=6 THEN F=159
680 IF N=7 THEN F=170
690 IF N=8 THEN F=176
700 SOUND F,2
710 RETURN
```

Experiment 6: Add a Delay

There are two places where a slight time delay in the program makes it easier to use. To create a delay, make a one-line program loop, like this:

```
FOR X=1 TO 300:NEXT X
```

When you type this instruction and press the **ENTER** key, the computer counts to 300 before returning the cursor and the OK to the screen. These two instructions add a similar pause before the computer plays the song and erases the notes from the screen.

Add these instructions, play the game, and see if you prefer these slight delays. You can use a similar instruction to add a pause to any program. For longer or shorter times, change the number 300.

```
155 FOR DLY=1 TO 300:NEXT DLY
```

```
185 FOR DLY=1 TO 300:NEXT DLY
```

The variable `DLY` was used instead of a single letter variable such as `X` to make delay lines like these easy to see in a program listing and to avoid conflict with other variables used in the program.

Experiment 7: OOPS!

This set of instructions adds two new features: sound effects and automatic repeats for up to three mistakes. When you play a wrong note, the computer will tell you about it with sound effects. If you've made less than three errors, the program plays the tune again and gives you another chance to guess it correctly.

First change Line 240 so that the computer goes to this section if the note you input does not match the next note in the song. Then add the section at Line 300 to create the sound.

The variable *M* is used to keep track of the number of mistakes. Start with *M*=0. For each error, *M* increases by 1. If you get the song right, *M* is set equal to zero in Line 255 and the computer adds a new note.

Add these instructions to change the program as shown in Figure 4. The improved graphics and the YOU WIN section are added in the next experiments.

```
240 IF N<>S(L) GOTO 300
300 REM ... OOPS! ...
310 FOR L=F-8 TO 1 STEP -8
320 SOUND L,1
330 NEXT L
340 M=M+1
350 IF M<3 GOTO 150 ELSE 30
30 T=0:M=0
255 M=0
```

Experiment 8: Music Tutor

It is much easier to guess the notes if you can see them printed in standard music notation. While this addition to the program only prints the numbers, not pictures of the notes, on the music staff, the positions may help you guess the tune.

A new subroutine will be added to print a musical staff on the screen. The scale is in the key of C and looks like this:

Lesson 19: Music Teacher (continued)

```
-----  
-----  
C                               8      do  
B -----7-----ti-----  
A                               6      la  
G -----5-----so-----  
F          4      fa  
E -----3-----mi-----  
D          2      re  
C - 1 - do
```

First use GOSUB instructions to call the subroutine from two places in the program. Lines 151 and 204 each use the subroutine to print an empty staff on the screen and clear all the notes. This is done before the computer plays the song and again before the notes are copied.

Line 830 prints a dotted line and then a full line of blank spaces. When this line is printed in a program loop, it erases all the notes on the screen, leaving the five lines and five spaces. Copy this line exactly. There are 32 dashes followed by 31 spaces inside the quotation marks. Notice on your screen that the dashes exactly fill the screen and that the quotation marks are lined up vertically.

Now enter the two instructions that call the subroutine, and the subroutine:

```
151 GOSUB 800  
  
204 GOSUB 800  
  
800 REM ... ERASE NOTES ...  
  
810 PRINT @64  
  
820 FOR L=1 TO 5  
  
830 PRINT "-----  
-----  
"  
  
840 NEXT L  
  
850 PRINT  
  
860 RETURN
```

These next two instructions that look like comic book swear words are used to print a number on the screen for each note that is played. The

math is used to position the number up and down to match the music scale, and move it left and right to match its place in the song.

Lines 110 and 190 are removed because it is no longer necessary to clear the screen.

```
610 IF N<>1 THEN PRINT @448-N*32
    +L*2.CHR$(N+48);
```

```
615 IF N=1 THEN PRINT @447-N*32+
    L*2, "-";CHR$(N+48);"-";
```

```
110
```

```
190
```

How It Works

LINES 610 and 615 position a number on the screen. If the number is 1, the note is printed with a dash on both sides, like this:

```
-1-
```

The variable *N* is a number from 1 to 8 that matches a note in the musical scale. Line 610 calculates the position in terms of location 448, near the lower-left corner of the screen. It moves up (negative direction) one row times the value of *N*. If *N*=3 for example, it moves up three rows on the screen. Then it moves right a distance $2*L$, or two positions on the screen for each position in the song.

When the final position is calculated, the computer prints the number, using its ASCII value. This extra step is done instead of just using PRINT *N* because it does not eliminate the space before and after numbers that are printed on the screen.

LINE 615 is used if *N*=1. It calculates from location 447, one position to the left of Line 610, and prints a dash before and after the number.

Experiment 9: You Win!

This final addition adds "positive reinforcement" for playing the game by playing a special tune if you get all 15 notes right.

In order to change the sound, it is necessary to change Line 700 where the sound is created. This instruction now says SOUND F,2. Instead of using a duration of 2, this instruction makes the duration the variable *D*:

```
700 SOUND F.D
```

Lesson 19: Music Teacher (continued)

Set D equal to 3 at the beginning with this change:

```
30 T=0:M=0:D=3
```

The next instruction tests to see if all 15 notes have been guessed. The Figure 4 flowchart on page 171 shows how this line changes the program:

```
260 IF T=15 GOTO 900 ELSE 100
```

This section sets D=1 and plays the tune three times, as fast as possible. The program pauses with a request to try again.

```
900 REM ... YOU WIN! ...  
910 FOR A=1 TO 3  
920 FOR B=1 TO 15  
930 N=S(B):D=1  
940 GOSUB 620  
950 NEXT B:NEXT A  
960 PRINT @452, "TRY AGAIN (Y,N)?"  
"  
970 X$=INKEY$:IF X$="" GOTO 970  
980 IF X$="Y" GOTO 30 ELSE CLS
```

Experiment 10: Music Teacher

You can build this program by following the experiments in this lesson or by loading the Lesson 19 tape with **C L O A D**.

When you run the program you will see the five bars of a musical scale. The computer will pick a note and briefly flash a number on the screen. Try to see the number and repeat it by pressing one of the keys.

Guess correctly and the computer will add another note to the tune and play it again. Make a mistake and the tune repeats for another try. Three mistakes on the same note and a new, one-note tune begins.

If you can get the complete sequence of 15 notes right, the computer will play it back three times at triple speed.

How It Works

The Figure 4 flowchart shows what the computer does while running Music Teacher's Tune. The complete listing follows these descriptions.

LINES 10-50 dimension an array `S(15)` to hold numbers for each note. The total number of notes in the song and number of missed notes are set equal to 0. The note duration `D` is set to 3.

LINES 100-130 add a new note to the program by increasing `T` and setting the next location in the array equal to a random number between 1 and 8.

LINES 150-185 play the song with a loop that cycles once for each note. On each cycle, `N` is set equal to a number in the array. Then the subroutine at Line 600 converts `N` to a frequency number and plays the note. Short delays are used before starting the song and before clearing the screen.

LINES 200-260 input the note typed on the keyboard. The `INKEY$` function reads the key directly, without use of the `ENTER` key. The variable `N` is set equal to the value of the key typed. If this value does not match the next note in the song, the program goes to `OOPS!`

LINES 300-350 create a falling tone if the guess was wrong. The `STEP` function in the loop allows it to count backwards, decreasing `L` and the frequency. If this mistake is not the third, the program goes to Line 150 to play the song again. Miss three times in a row and the program starts over — with a one-note song to copy.

LINES 600-710 print and play the note. It is printed at the correct location by either Line 610 or Line 615. The look-up table then converts `N` into a number corresponding to the correct pitch. The `SOUND` command converts the number into a tone whose duration is set by `B`.

LINES 800-860 erase the notes on the screen by printing a set of lines and spaces. This subroutine is called by the `PLAY SONG` and `INPUT NOTE` sections of the program.

LINES 900-980 are a prize for getting all 15 notes correct. The double loop plays all 15 notes 3 times, as fast as possible. The speed increase is caused by changing `D`, the duration of the note played in Line 700. the ending sequence is the standard module, described in Section 3, page 215.

Here is the complete listing:

```
10 REM ... MUSIC TEACHER ...
20 DIM S(15)
30 T=0:M=0:D=3
```

Lesson 19: Music Teacher (continued)

```
40 CLS
50 PRINT "          ... MUSIC TEACHER
   ..."

100 REM ... ADD A NEW NOTE ...
120 T=T+1
130 S(T)=RND(8)

150 REM ... PLAY SONG ...
151 GOSUB 800
155 FOR DLY=1 TO 300:NEXT DLY
160 FOR L=1 TO T
170 N=S(L):GOSUB 600
180 NEXT L
185 FOR DLY=1 TO 300:NEXT DLY

200 REM ... INPUT NOTE ...
204 GOSUB 800
205 FOR L=1 TO T
210 N$=INKEY$:IF N$=" " GOTO 210
220 N=ASC(N$)-48
230 GOSUB 600
240 IF N<>S(L) GOTO 300
250 NEXT L
255 M=0
260 IF T=15 GOTO 900 ELSE 100

300 REM ... OOPS! ...
310 FOR L=F-8 TO 1 STEP -8
320 SOUND L,1
330 NEXT L
340 M=M+1
350 IF M<3 GOTO 150 ELSE 30

600 REM ... PRINT & PLAY ...
610 IF N<>1 THEN PRINT @448-N*32
   +L*2,CHR$(N+48);
615 IF N=1 THEN PRINT @447-N*32+
   L*2,"-";CHR$(N+48);"-";
620 IF N=1 THEN F=89
630 IF N=2 THEN F=108
640 IF N=3 THEN F=125
650 IF N=4 THEN F=133
660 IF N=5 THEN F=147
670 IF N=6 THEN F=159
680 IF N=7 THEN F=170
690 IF N=8 THEN F=176
700 SOUND F,D
710 RETURN
```

Lesson 19: Music Teacher (continued)

```
800 REM ... ERASE NOTES ...
810 PRINT @64
820 FOR L=1 TO 5
830 PRINT "-----"
-----
      "

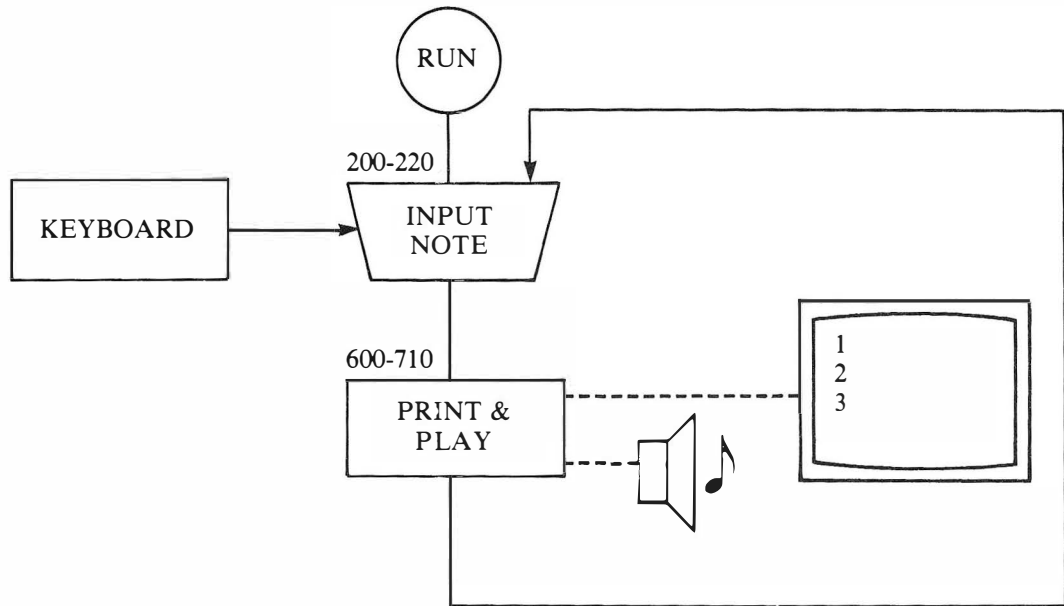
840 NEXT L
850 PRINT
860 RETURN

900 REM ... YOU WIN! ...
910 FOR A=1 TO 3
920 FOR B=1 TO 15
930 N=S(B):D=1
940 GOSUB 620
950 NEXT B:NEXT A
960 PRINT @452,"TRY AGAIN (Y,N)?
"

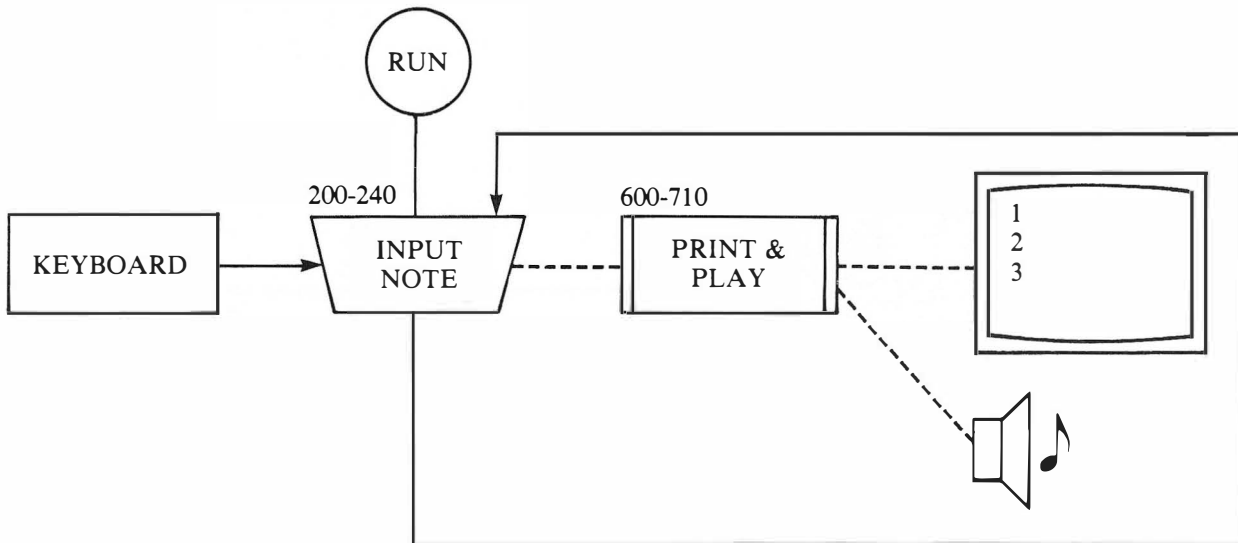
970 X$=INKEY$:IF X$="" GOTO 970
980 IF X$="Y" GOTO 30 ELSE CLS
```

Lesson 19: Music Teacher (continued)

MUSIC
TEACHER
FIG. 1

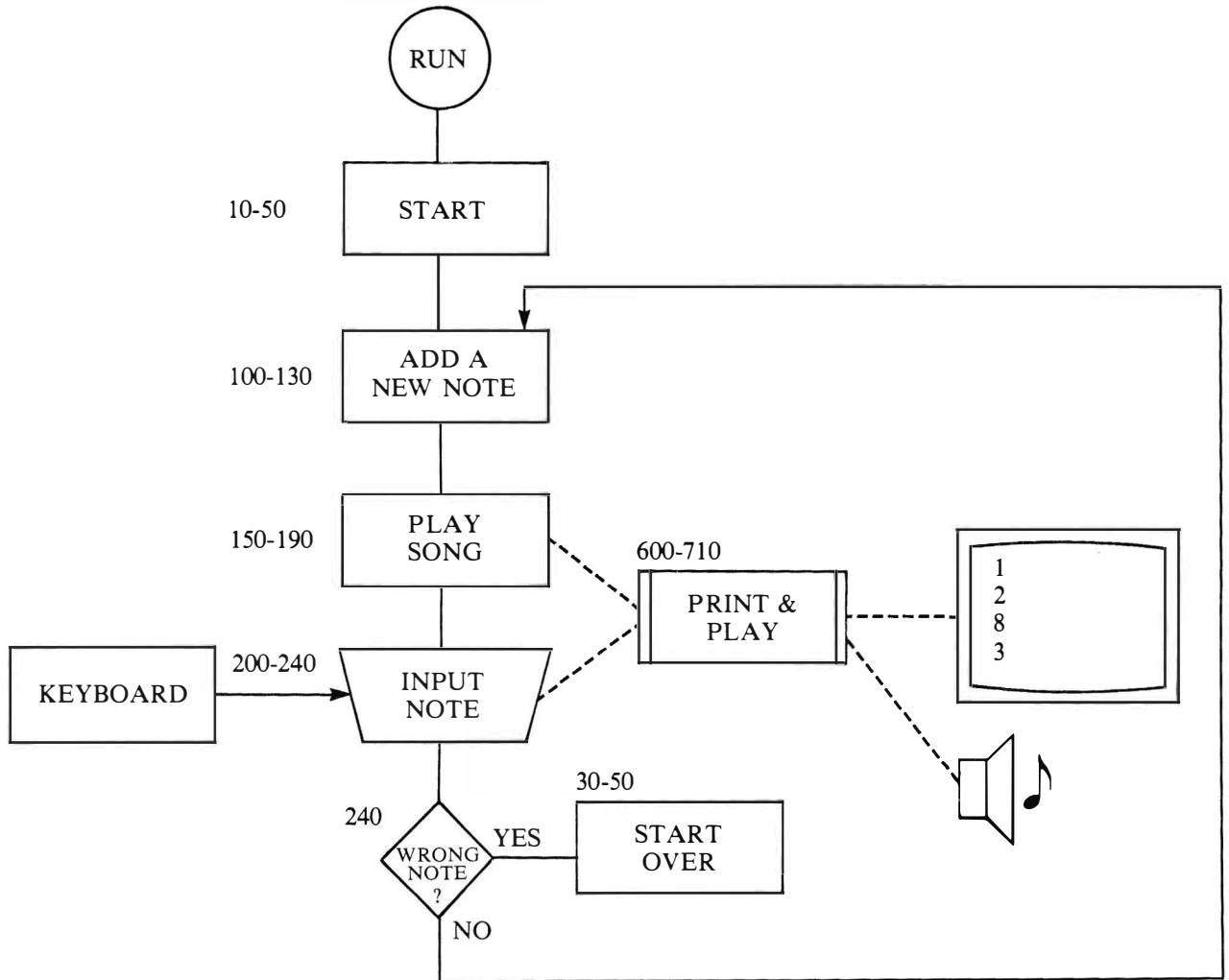


MUSIC
TEACHER
FIG. 2



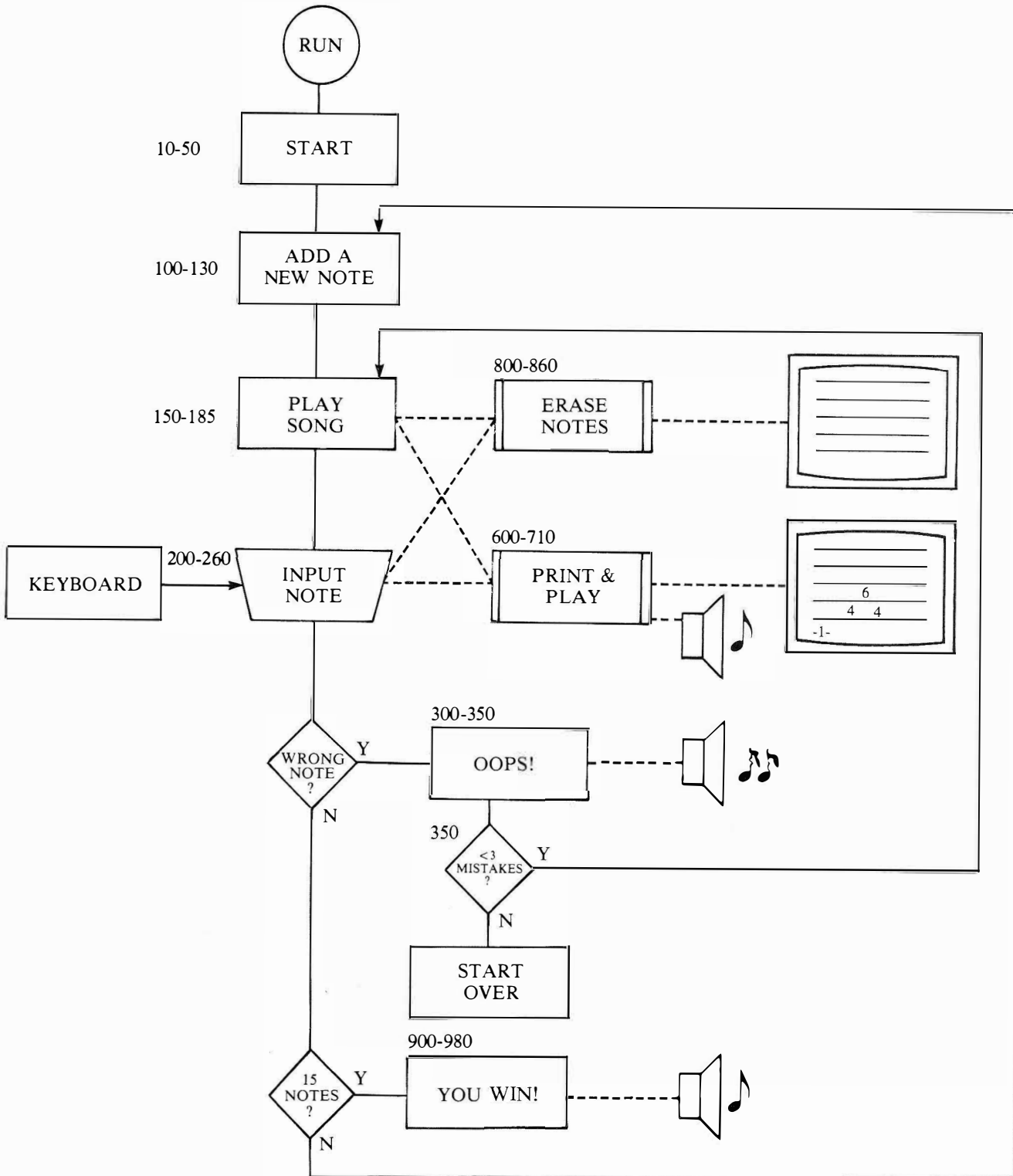
Lesson 19: Music Teacher (continued)

MUSIC
TEACHER
FIG. 3



Lesson 19: Music Teacher (continued)

MUSIC
TEACHER
FIG. 4



Lesson 20: Car Calculator

Branching and Special Calculators

This lesson's software creates a special calculator for solving problems about distance, rate, time, and gas mileage. While you probably wouldn't have any trouble finding these answers with a simple calculator, this lesson will show how you can write a program to create any special calculator you wish. If you know the formulas for solving problems, you can use a program like this one to input the data and create a printout of the results.

Experiment 1: Distance Calculator

This short program calculates distance traveled when the time and rate of speed are known. Type **NEW** to erase any program in memory and enter these instructions:

```
300 REM ... DISTANCE ...  
305 PRINT  
310 INPUT " TIME IN HOURS";T  
320 INPUT " RATE IN MILES PER HO  
UR";R  
330 D=R*T  
335 PRINT  
340 PRINT " AT";R;"MILES PER HOU  
R"  
350 PRINT " FOR";T;"HOURS"  
360 PRINT " YOU WILL TRAVEL"  
370 PRINT D;"MILES"
```

Run the program and enter numbers for T and R. The formula in Line 330 calculates the answer. A complete printout is created with Lines 335-370.

If you enter **1.5** for the time and **55** for the rate, the screen will show:

```
TIME IN HOURS? 1.5  
RATE IN MILES PER HOUR? 55
```

Lesson 20: Car Calculator (continued)

```
AT 55 MILES PER HOUR
FOR 1.5 HOURS
YOU WILL TRAVEL
82.5 MILES
OK
```

Experiment 2: Time Calculator

Add this section and calculate the time required when the distance and rate are known. These instructions work exactly the same way as before to input values for the variables, calculate the answer with a formula, and print the results.

```
400 REM ... TIME ...
405 PRINT
410 INPUT " DISTANCE IN MILES";D
420 INPUT " RATE IN MILES PER HO
UR";R
430 T=D/R
435 PRINT
440 PRINT " TRAVELING";D; "MILES"
450 PRINT " AT";R; "MILES PER HO
UR"
460 PRINT " WILL TAKE YOU"
470 PRINT T;"HOURS"
```

Instead of running the program, type **GOTO 400**. The computer will run the second section, starting at Line 400. Input 313 miles for the distance and 55 MPH for the rate. The printout will show:

```
DISTANCE IN MILES? 313
RATE IN MILES PER HOUR? 55

TRAVELING 313 MILES
AT 55 MILES PER HOUR
WILL TAKE YOU
5.69090909 HOURS
OK
```

In Experiment 4 you will see how to round off this answer automatically to 5.7 hours.

Experiment 3: Speed Calculator

There is nothing new or different here. With new variables to input and a new formula, this is the same program as before. Add these lines to calculate average speed:

```
500 REM ... SPEED ...  
505 PRINT  
510 INPUT " DISTANCE IN MILES";D  
520 INPUT " TIME IN HOURS";T  
530 R=D/T  
535 PRINT  
540 PRINT " TRAVELING";D; "MILES"  
550 PRINT " IN";T;"HOURS"  
560 PRINT " YOU WILL AVERAGE"  
570 PRINT R; "MILES PER HOUR"
```

To test this segment of the program, enter **GOTO 500**. Find out what your average speed would have to be to travel from here to the sun in one day with these inputs:

```
DISTANCE IN MILES? 93000000  
TIME IN HOURS? 24  
  
TRAVELING 93000000 MILES  
IN 24 HOURS  
YOU WILL AVERAGE  
3875000 MILES PER HOUR  
OK
```

Experiment 4: Rounding Off Answers

To avoid answers such as 5.69090909 HOURS, round off the results of any division problems with this programming module. These two lines will round off the time T and rate R to the nearest tenth — or to one decimal place:

```
432 T=INT(T*10+.5)/10  
532 R=INT(R*10+.5)/10
```

Lesson 20: Car Calculator (continued)

After adding these lines, test the rounding off. Enter `GOTO 400` and then enter `313` for the miles, `55` for the rate. This time, instead of printing 5.69898989, the computer will round off and print 5.7 hours as the answer.

Experiment 5: Variable Names

In most of these programs single-letter variables have been used. It is convenient to use D for distance, R for rate, T for time, and similar letters to name variables in a program. You can also use complete names, starting with a letter, such as: DISTANCE, RATE, TIME, or even UNCLEDICK. While the names you use may be of any length, the computer reads only the first two letters of a variable name.

In this final section of the program, the variables are named: MILES, GAL, and MPG. Notice that well-chosen variable names can make a program easier to read.

To see that long variable names really work, do this:

```
MYSTERYNUMBER=5
OK
PRINT MY
5
OK
```

Now add this ecology-minded section to your program and compute gas mileage easily:

```
600 REM ... MPG ...
605 PRINT
610 INPUT " DISTANCE IN MILES";M
ILES
620 INPUT " FUEL IN GALLONS";GAL
630 MPG=MILES/GAL
640 MPG=INT(MPG*100+.5)/100
645 PRINT
650 PRINT " TRAVELING";MILES;"MI
LES"
660 PRINT " ON";GAL; "GALLONS OF
FUEL"
```

```
670 PRINT " YOU WILL AVERAGE"
```

```
680 PRINT MPG;"MILES PER GALLON"
```

Test this section with `GOTO 600` and see that it works the same way as the others.

Experiment 6: Menu

There are two things you need to do to combine these sections and create a useful program. This menu section will start the computer in the right place, and the next experiment will provide an ending for the program.

```
100 REM ... MENU ...  
110 CLS:PRINT:PRINT  
120 PRINT " ... CAR CALCULATO  
R ..."  
130 PRINT:PRINT  
140 PRINT " 1. DISTANCE TRAVEL  
ED"  
150 PRINT " 2. TIME REQUIRED"  
160 PRINT " 3. AVERAGE SPEED"  
170 PRINT " 4. GAS MILEAGE"  
180 PRINT:PRINT  
190 PRINT " SELECT (1-4)"  
200 K$=INKEY$:IFK$="" GOTO 200  
210 SEL=ASC(K$)-48  
220 CLS  
230 ON SEL GOTO 300,400,500,600  
240 GOTO 100
```

Experiment 7: More?

The menu will direct the computer to the appropriate section. With the following instructions, the computer will ask if you want another calculation. Depending on your response, the program will go back to the menu, or stop.

```
380 GOTO 700

480 GOTO 700

580 GOTO 700

700 REM ... MORE? ...

705 PRINT

710 PRINT "    ANOTHER CALCULATIO
N (Y,N)?"

720 K$=INKEY$:IF K$=" " GOTO 720

730 IF K$="Y" GOTO 100 ELSE CLS
```

Now your program is complete. Run it and see that the menu directs the computer to the right section for inputting data and printing results.

Experiment 8: Car Calculator

This program is recorded on the Lesson 20 cassette. You can build the program from scratch, following the experiments in this lesson, or you can load the cassette and run the final program.

The standard program menu with four choices is displayed. Press one of these keys and the computer requests the first input. Type a number and press **ENTER**. Answer the second question by typing a number and pressing **ENTER** again. The final printout will repeat the data and state the result.

While these calculations are trivial, this program shows several important techniques for building a special calculator by using a menu and a series of formulas. With only a few changes in the wording and in the formulas, you could create your own software for real estate calculations, loans and interest tables, English and metric conversion, cost or material estimates, and many other calculations.

How It Works

The flowchart on page 181 shows how this program is organized, with a menu, four sections for inputting data and printing the results,

Lesson 20: Car Calculator (continued)

and a final section that returns to the menu if another calculation is requested.

Here is the complete listing:

```
10 REM ... CAR CALCULATOR ...
100 REM ... MENU ...
110 CLS:PRINT:PRINT
120 PRINT "      ... CAR CALCULATO
R ..."
130 PRINT:PRINT
140 PRINT "    1. DISTANCE TRAVEL
ED"
150 PRINT "    2. TIME REQUIRED"
160 PRINT "    3. AVERAGE SPEED"
170 PRINT "    4. GAS MILEAGE"
180 PRINT:PRINT
190 PRINT "          SELECT (1-4)"
200 K$=INKEY$:IFK$="" GOTO 200
210 SEL=ASC(K$)-48
220 CLS
230 ON SEL GOTO 300,400,500,600
240 GOTO 100

300 REM ... DISTANCE ...
305 PRINT
310 INPUT " TIME IN HOURS";T
320 INPUT " RATE IN MILES PER HO
UR";R
330 D=R*T
335 PRINT
340 PRINT " AT";R"MILES PER HOU
R"
350 PRINT " FOR";T;"HOURS"
360 PRINT " YOU WILL TRAVEL"
370 PRINT D; "MILES"
380 GOTO 700

400 REM ... TIME ...
405 PRINT
410 INPUT " DISTANCE IN MILES";D
420 INPUT " RATE IN MILES PER HO
UR";R
430 T=D/R
432 T=INT(T*10+.5)/10
435 PRINT
440 PRINT " TRAVELING";D;"MILES"
450 PRINT " AT";R;"MILES PER HOU
R"
460 PRINT " WILL TAKE YOU"
470 PRINT T;"HOURS"
480 GOTO 700
```

Lesson 20: Car Calculator (continued)

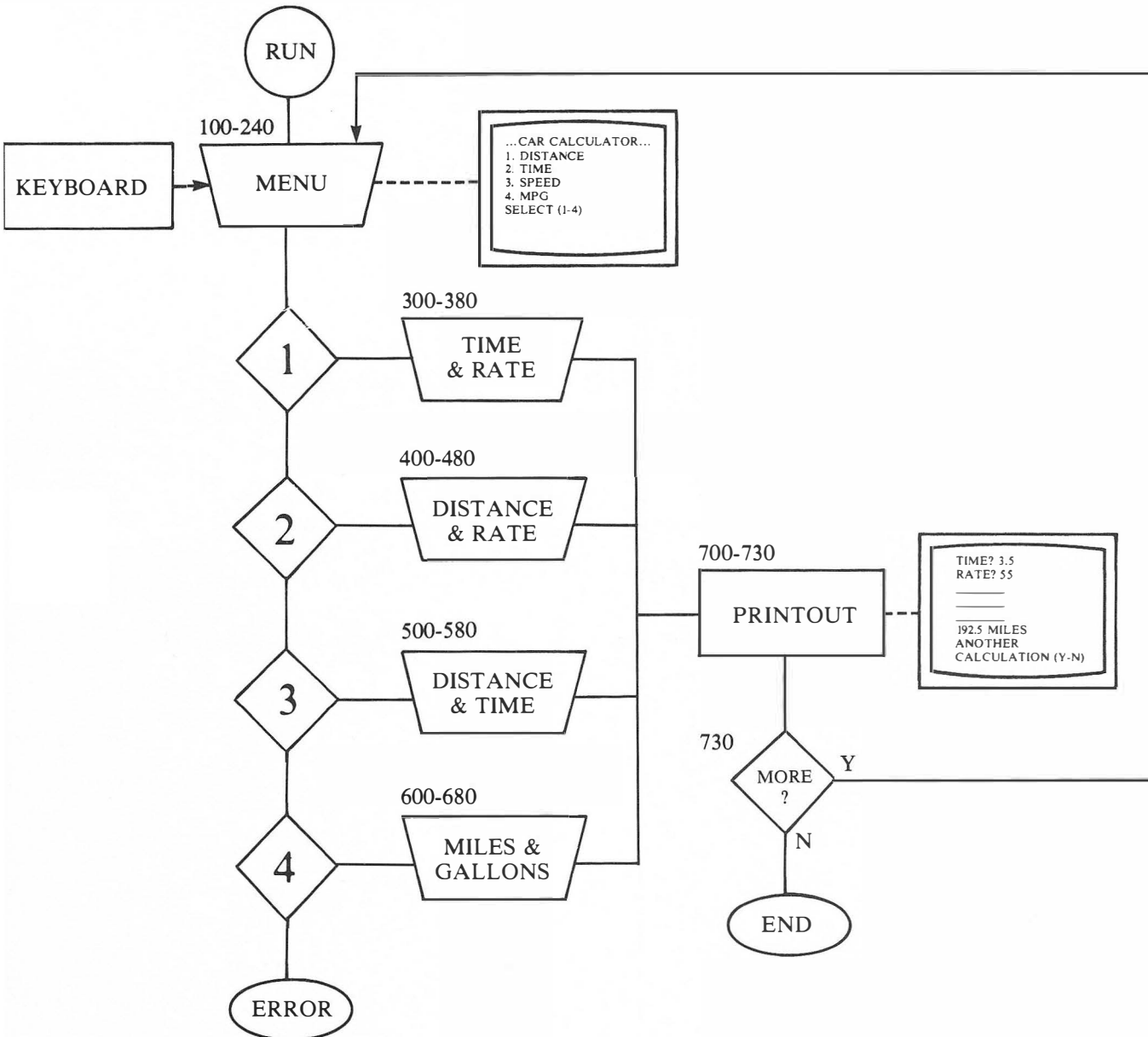
```
500 REM ... SPEED ...
505 PRINT
510 INPUT " DISTANCE IN MILES";D
520 INPUT " TIME IN HOURS";T
530 R=D/T
532 R=INT(R*10)+.5)/10
535 PRINT
540 PRINT " TRAVELING";D;"MILES"
550 PRINT " IN";T;"HOURS"
560 PRINT " YOU WILL AVERAGE"
570 PRINT R;"MILES PER HOUR"
580 GOTO 700

600 REM ... MPG ...
605 PRINT
610 INPUT " DISTANCE IN MILES";M
ILES
620 INPUT " FUEL IN GALLONS";GAL
630 MPG=MILES/GAL
640 MPG=INT(MPG*100+.5)/100
645 PRINT
650 PRINT " TRAVELING";MILES;"MI
LES"
660 PRINT " ON";GAL;"GALLONS OF
FUEL"
680 PRINT MPG;"MILES PER GALLON"

700 REM ... MORE? ...
705 PRINT
710 PRINT " ANOTHER CALCULATIO
N (Y,N)?"
720 K$=INKEY$:IF K$="" GOTO 720
730 IF K$="Y" GOTO 100 ELSE CLS
```

Lesson 20: Car Calculator (continued)

**CAR
CALCULATOR
FIG. 1**



Lesson 21: Graphics

Video Art and Graphics Characters

There are many ways you can use the graphics capabilities of your Color Computer. For example, you can create a message center, an electronic billboard, charts and graphs, video art, games, and even cartoons on your TV.

This lesson contains special software that helps you include color drawings, pictures, and art in your computer programs. You can use this programming tool to make full color signs with large and small letters, charts, or even pictures.

Experiment 1: Graphics

Begin by loading the Lesson 21 cassette program. When you run the program you will see this menu:

```
... GRAPHICS ...
```

1. GRAPHICS CHARACTERS
2. BILLBOARDS & SIGNS
3. ART & ANIMATION

```
SELECT (1-3)
```

Graphics Characters, the first selection, is a programmer's tool. This program makes it easier for you to add pictures, graphics, and cartoons to your programs. You can load and run this program each time you wish to convert a drawing on graph paper to an image on the screen. This software makes programming pictures much easier because it shows you exactly what the screen will display for each square on the graph paper.

Billboards & Signs, the second selection, is a simple example of a screen with large and small letters. By following this example, you can easily add full screen messages to your programs.

Art & Animation, the last selection, displays a more detailed drawing of a space person with a computer voice. With careful work, you could use this technique to create a short cartoon or "film strip." Before you imagine anything like the effects in "Star Wars", "Snow White", or the latest computer graphics TV commercial, remember that your Color Computer can only draw colored dots — much like needlepoint or paint-by-numbers sets.

To see all three selections on the menu, press the number and read the description below. If you are looking at Section 1 and wish to see

another selection, press **BREAK** to stop the program, then type **RUN** and press **ENTER** to start with a new menu. If you are looking at Sections 2 or 3 and wish to make another selection, just press any key and the program will return to the menu automatically.

Experiment 2: Graphics Characters

Select **1** to see Graphics Characters, the program that helps you add pictures to programs. The screen will show 16 graphics shapes, with their numbers. This collection of shapes is printed in orange, with a black background. Now press number **2** and the screen will show these same shapes as they appear in pink, with a black background. Notice that shapes 157 and 158 look like eyes.

Press each number from 2 to 8 on the keyboard and see how these 16 shapes can be printed in each of the eight colors. For each shape, in each color, there is a specific number you will use. With this program running in your computer, it is easy to copy a drawing on graph paper to an image on the screen. With these 128 graphics symbols (16 shapes in 8 colors) you can print a wide variety of pictures.

Experiment 3: Character Numbers

The numbers next to each graphics character are used to print the character on the screen. The statement at the bottom of the screen is an automatic converter to show you what numbers are associated with each of the keys on the keyboard. Press the letter **A** and see that its character number is 65. Letter **Z** is the number 90, the up arrow is 94, and the question mark is number 63. This feature is useful whenever you wish to convert a letter, number, or punctuation mark to its character number. These numbers are called ASCII numbers and are used by most computers to represent keyboard and graphics characters.

Experiment 4: Billboards & Signs

For a simple example of a picture printed with these graphics shapes, press **BREAK** to stop the program. Then run the program again. When the menu appears, select number **2**.

Your screen will show a large HI! with . . . GRAPHICS . . . in smaller letters. This is a simple example to show how you can use large and small letters on the screen at the same time. You may have seen the in-store demonstration of the Color Computer that uses similar techniques to create a series of messages on the TV.

Figure 1 on page 190 shows how these displays are created. The TV screen is divided into 32 columns (0 to 31) and 16 rows (0 to 15). The characters you saw in the first experiment can be placed in any of

these squares. For this example, a section of the screen from column 11 to column 20 and from row 4 to row 8 is used to print the word HI!.

Each block inside this section is assigned a number. Compare the picture on your screen with Figure 1 and see which numbers are used for each color. Each number represents a solid square of color. Squares with number 143 are used for the background. Squares numbered 239 form the H, number 207 squares make the I, and number 159 squares are used for the exclamation point.

The first step in creating a picture like this is to sketch the design on graph paper, as shown in Figure 1. Then the colors are selected and the numbers for each block are filled in. It will probably help to use a work sheet to show what numbers are used in each block, like this:

Work Sheet for "HI!"

	11	12	13	14	15	16	17	18	19	20
4	239	143	143	239	143	143	207	143	143	159
5	239	143	143	239	143	143	207	143	143	159
6	239	239	239	239	143	143	207	143	143	159
7	239	143	143	239	143	143	207	143	143	143
8	239	143	143	239	143	143	207	143	143	159

If you look closely at the work sheet you can see the letters H and I and the exclamation point outlined in the numbers 239, 207, and 159.

It is much easier to fill in this work sheet if you use the Graphics Characters design aid, selection **1** on the menu, for selecting graphics characters. Return to the menu by pressing any key, then select option **1** to return to Graphics Characters. You will see the 16 shapes in orange. This drawing is made up of solid color blocks, as shown in the lower-right hand corner. Press **7** now and see that shape number 239 is a solid square in light blue. Press **3** and see that number 175 is a dark green square. Press **2** and see that 159 is solid pink. (The colors will vary, depending on your TV and your perception.)

The final step, transferring the numbers on your work sheet to the program, is done in a series of data statements. To reproduce your own pattern, copy these instructions and substitute your numbers for the ones shown. The instructions for writing HI! look like this in the program: (The data instructions shown below are not wrapped around

Lesson 21: Graphics (continued)

or continued on the next line, as they actually appear when you list the program on your screen.)

```
2000 REM .. HI! ...
2010 CLS
2020 FOR Y=4 TO 8
2030 FOR X=11 TO 20
2040 READ C
2050 PRINT @32*Y+X,CHR$(C);
2060 NEXT X: NEXT Y
```

```
2104 DATA 239,143,143,239,143,143,207,143,143,159
2105 DATA 239,143,143,239,143,143,207,143,143,159
2106 DATA 239,239,239,239,143,143,207,143,143,159
2107 DATA 239,143,143,239,143,143,207,143,143,143
2108 DATA 239,143,143,239,143,143,207,143,143,159
```

The numbers 4 and 8 in Line 2020 tell the computer which columns to use and the numbers 11 and 20 in Line 2030 define the rows. Line 2040 sets the variable C equal to the numbers in the data statements. Then these numbers are plotted on the screen to cover a square area and spell HI!. The data statements are printed in five separate lines so that it is easier to edit the program to make changes or correct any errors in the final picture.

The added message in small letters is a standard print statement that begins in row 11, column 8. This is printed with the following instruction:

```
2070 PRINT @32*11+8, "... GRAPHICS ..."
```

The instructions on Lines 2080 and 2090 hold the computer at this point until you press a key. Then the program runs from the beginning again, printing the menu as before.

```
2080 R$=INKEY$:IF R$="" GOTO 2080
2090 RUN
```

Experiment 5: Art & Animation

Figure 2 on page 191 shows how graphics shapes in addition to solid blocks of color can be used in creating a more detailed drawing. The principle is exactly the same as in printing HI!, except that several kinds of graphics shapes in addition to solid blocks of color are used.

If you are watching HI!, press any key to return to the main menu. If you are watching Graphics Characters, press **BREAK** to stop the program and type **RUN**.

Lesson 21: Graphics (continued)

From the menu, select Art & Animation to draw a Space Person. Adjust the volume control on your TV to hear a lecture about computer graphics. (Unfortunately, the lecture is in Martian.)

To see how this figure is constructed, return to the Graphics Characters section of the program. Press any key to return to the menu, then select number **1**. Now press number **2** and display the characters from 144, to 159. Two of these characters, numbers 157 and 158, are used for the eyes of the space person in the drawing. On the Space Person work sheet below, locate the numbers 158 and 157 in row number 4 (the second row on the work sheet). These characters correspond to the position of the eyes in the drawing.

Continue comparing the drawing in Figure 2 with the characters on the screen. The work sheet will help you identify each block by number. The display on the screen will show you what each number looks like. Remember to press any number from 1 to 8 to see the complete selection of graphics characters.

Work Sheet for Space Person

	12	13	14	15	16	17	18
3	208	211	215	211	219	211	208
4	225	223	158	223	157	223	226
5	208	220	223	223	223	220	208
6	179	179	179	191	179	179	179
7	191	176	191	191	191	176	191
8	191	176	239	239	239	176	191
9	142	128	255	252	255	128	141
10	240	240	255	240	255	240	240
11	240	240	255	240	255	240	240
12	128	131	139	128	135	131	128

The following portion of the program prints the graphics characters whose numbers are shown in the Space Person work sheet.

```
3000 REM ... SPACE PERSON ...
3010 CLS
```

Lesson 21: Graphics (continued)

```
3020 FOR Y=3 TO 12
3030 FOR X=12 TO 18
3040 READ C
3050 PRINT @32*Y+X,CHR$(C);
3060 NEXT X: NEXT Y

3070 DATA 208,211,215,211,219,211,208
3080 DATA 225,223,158,223,157,223,226
3090 DATA 208,220,223,223,223,220,208
3100 DATA 179,179,179,191,179,179,179
3110 DATA 191,176,191,191,191,176,191
3120 DATA 191,176,239,239,239,176,191
3130 DATA 142,128,255,252,255,128,141
3140 DATA 240,240,255,240,255,240,240
3150 DATA 240,240,255,240,255,240,240
3160 DATA 128,131,139,128,135,131,128
```

Notice that the program listed above is very similar to the program used for printing HI!. The numbers used in Lines 3020 and 3030 determine the overall size of the figure drawn on the screen. In this case, the drawing goes from row 3 to row 12 (Line 3020) and from column 12 to column 18 (Line 3030). There are more numbers in the data statements because Space Person is larger than HI!, with more squares to fill in. Also notice that there is more variation in the numbers because more kinds of graphics shapes are being used.

After Space Person has been drawn, the computer goes to another section of the program to create the "lecture." Line 3065 does the routing:

```
3065 GOTO 3200
```

Experiment 6: Speak & Spiel

This section of the program is a continuing loop that opens and closes Space Person's mouth and creates the speech-like sounds. The mouth animation is done by alternating the equal sign (=) and the dash (-).

The program waits a random length of time, then prints =, makes a sound, prints -, and waits again. The sound is a random frequency between SOUND 231 and SOUND 240. The duration is also random and ranges from 1 to 3. When any key is pressed, the program is RUN again. This restores the data statements and displays the main menu.

Here is a listing of this section of the program. To duplicate Space Person's speech in any program, just copy this idea:

```
3200 REM ... SPEAK ...
3210 T=RND(300)
3220 FOR DLY=1 TO T:NEXT DLY
```

```
3230 PRINT @5*32+15,"=";  
3240 SOUND RND(10)+230,RND(3)  
3250 PRINT @5*32+15,"-";  
3260 R$=INKEY$:IF R$="" GOTO 320  
0  
3270 RUN
```

Experiment 7: On Your Own

Creating your own graphics designs is tedious, but not difficult. With careful planning you can design longer programs with many pictures or messages. Begin with a simple drawing that doesn't contain too many squares. Copy the drawing on graph paper, using the shapes in Graphics Characters as a guide.

As a start, try a drawing that is the same size as HI! You can then use the program listed in Experiment 4 to draw the figure on the screen, and change the numbers in the data statements to print the design you have drawn.

Once you have a design that prints correctly, you can move it to any location on the screen by changing the row and column numbers. Be careful to keep the total width and height the same, or you will get very strange results.

If you wish to print messages or characters from the keyboard, use PRINT @, as in Lines 3230 and 3250. The background color for all characters on the keyboard is green.

Time delays are necessary if you wish to pause between pictures or messages. You can set T equal to the delay you wish and use the time delay module in Line 3220, or use an input from the keyboard to continue the program, as in Line 2080. You could also control the program by using the joystick button to advance to the next picture, similar to a slide projector.

Many kinds of lettering can be printed with graphics characters. The examples in Figures 3 and 4 show some of the lettering styles used in professional computer displays.

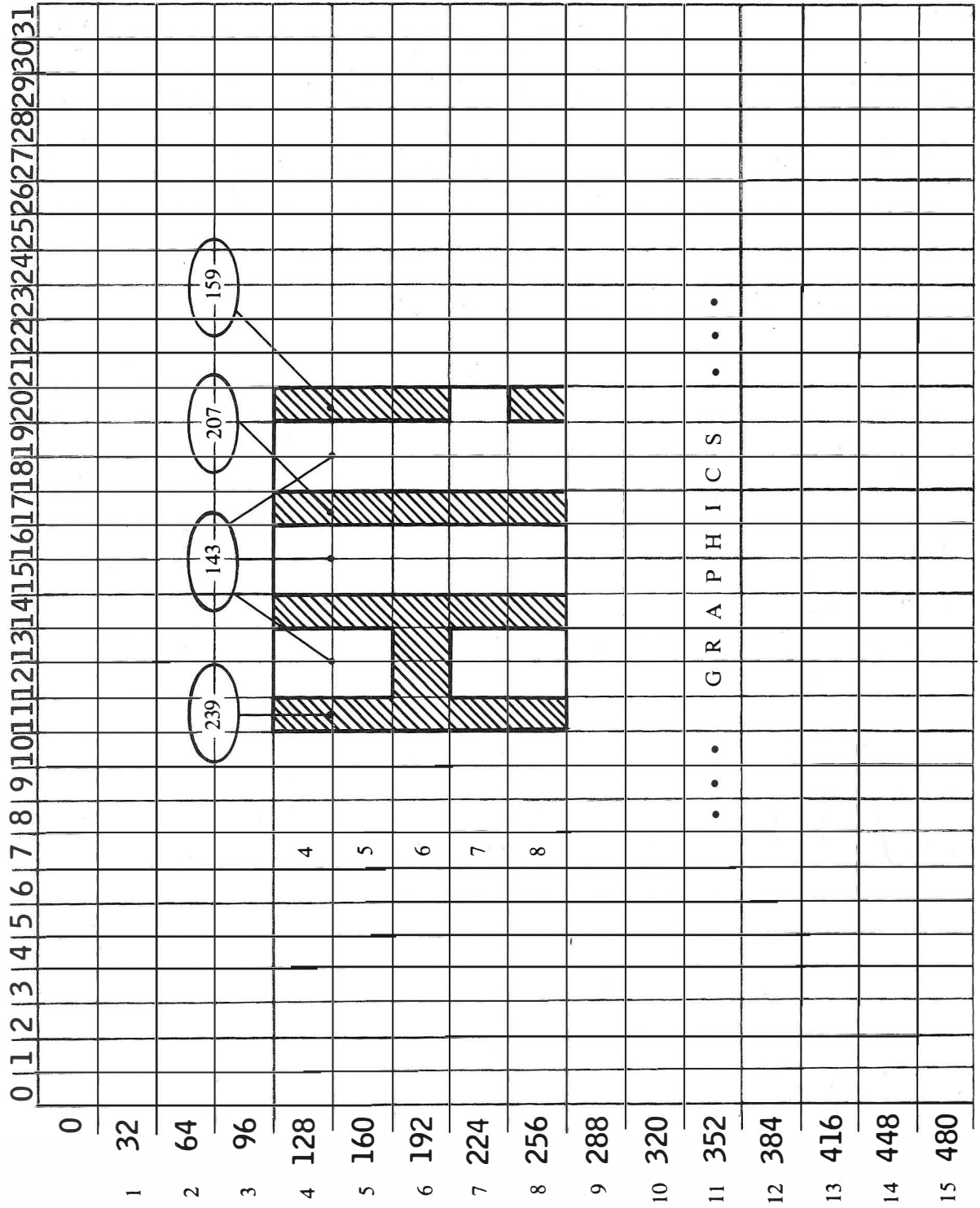


FIGURE 1

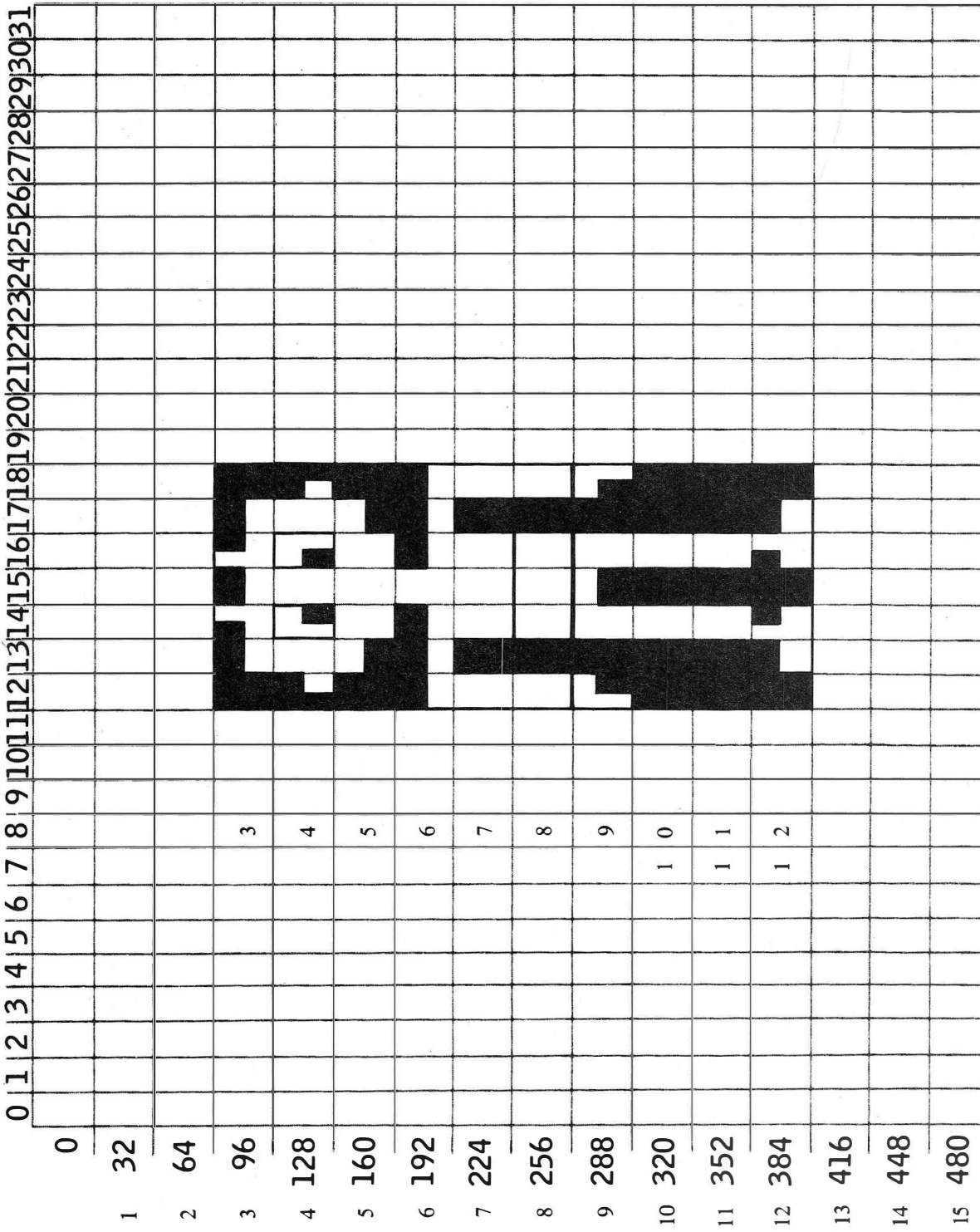


FIGURE 2

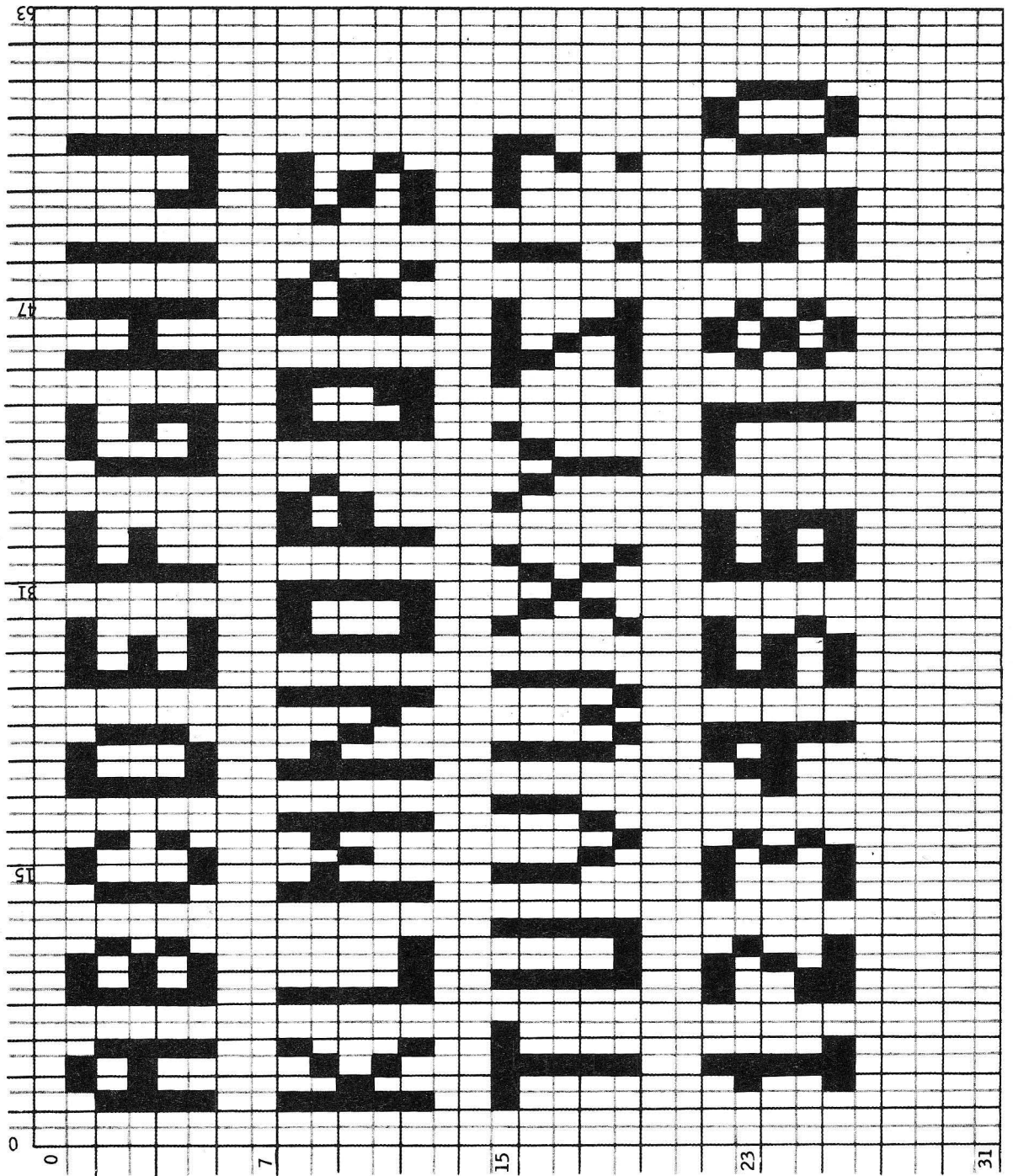


FIGURE 3

Lesson 22: Player Piano

Music Instruments

This program creates an instrument you can play like a piano. You type in tunes and the computer plays them back. This hunt-and-peck method works well on a computer because you can back up and correct any wrong notes.

If you have tried to add music or even short tunes to your programs, you know how difficult it is to convert numbers from a frequency table into notes that sound the way you want them to. This special software package turns your computer keyboard into a music keyboard, making it much easier to program music or add songs to your software.

The response time is not as fast as an electric piano, organ, or synthesizer, but you will be able to replay your music as many times as you like by pressing a key. When you are satisfied with the results, this program will print out the frequency table so that you can add the music to any program.

Experiment 1: Player Piano

Begin by loading the program from the Lesson 22 cassette. When you run the program you will see this message on the screen:

```
... PLAYER PIANO ...
```

Adjust the volume on your TV and press any of the keys in the row that is closest to you. These keys correspond to the white keys on a piano, with the letter C and middle C aligned, like this:

Computer keys	Z	X	C	V	B	N	M	<	>	?
Piano keys	A	B	C	D	E	F	G	A	B	C

Pick out a tune and notice the printing on the screen. If you can't think of a tune, play these letters: **C B M C B M N B V C**. The numbers on your screen are those used with the SOUND command to make the sounds you hear. The note C, for example, is played when your computer sees this instruction:

```
SOUND 176.2
```

Experiment 2: Playback

Now press **P** and get a playback and a copy of the music you just entered. To play at full speed, without the notes on the screen, press **@**.

Lesson 22: Player Piano (continued)

When in mode P, the computer adds a short time to each note because it takes time to print the numbers on the screen. Music you copy and use in a program will be played at the @ speed.

Experiment 3: Clear

To clear the notes from the system means to erase them forever and make room for something new. Press the **CLEAR** key and see your composition vanish.

Experiment 4: ← Octave Shift →

The left and right arrows move the keyboard one octave. Press the left arrow and try entering a short tune. Notice that the keyboard is shifted one octave down. To shift back again, press the right arrow. Press it again to play the highest octave.

The keyboard stays where you leave it. You can jump from one end of the range to the other by pressing the arrow twice.

If you have a critical ear, you will notice that the highest notes in the top octave are not quite on pitch. You can change the pitch, or make up any scale you like, by modifying the frequency look-up tables in the program.

Experiment 5: Electrical Eraser

The up-arrow key on the left has a very useful function in this program. If you play a wrong note, just press this "back up" arrow and play the note again. The information on the screen will not change, but the wrong note will be erased from the memory. Test this yourself by entering a song you like and using the eraser to back up and remove mistakes.

Experiment 6: Programming Holds

The zero (0) key is used to hold the note one more beat. After pressing any of the note keys, press **0** once for each beat you wish to extend the note. Try this example and see the effect of the zeros after each note. Press **CLEAR** and **2** before you begin. Press **0**, not the letter (O), when you type these characters on the keyboard:

```
C00C0CC00  
G0VV0CC0XC000
```

Your screen should look like the following:

PLAYER PIANO

```
89 6
89 4
89 2
89 6
117 4
108 2
108 4
89 2
89 4
78 2
89 8
```

Now play the tune with both **P** and **@**. The timing will sound very similar because the slight additional time required to print the note on the screen in **P** mode is not significant with the longer notes.

Experiment 7: Programming Notes

The **SOUND** command is used to play notes in the TV speaker. The first column on the screen is the frequency number and the second column is the duration number. To reproduce this song in a program, just add these instructions:

```
SOUND 89,6
SOUND 89,4
SOUND 89,2
SOUND 89,6
SOUND 117,4
SOUND 108,2
SOUND 108,4
SOUND 89,2
SOUND 89,4
SOUND 78,2
SOUND 89,8
```

Use a line number with each instruction, and place this series of 11 lines in your program at the spot where someone loses the game, gets gunned down at the pass, goes bankrupt, has a negative balance, or whatever.

Experiment 8: Programming Time

The numbers 1-8 control time. To see the effect, just press **1** and type a few notes. Now press any other number, play a few more notes, and notice the timing difference. This adjustment sets the beat of the music, as well as the feel of the keyboard. After trying a few notes at different times, press **@** again and see how the computer sounds with changes in the timing notation.

Lesson 22: Player Piano (continued)

For a quick example of high speed playing, select the top octave with the right arrow, clear Player Piano with **CLEAR**, and type these keys:

```
2 C B M / 1
/ / / / 2 M
1 M M M M 2
B M B C 0 0
```

The screen will show these notes, with the top two notes scrolled off the top:

```
176 2
193 2
204 2
218 2
218 1
218 1
218 1
218 1
204 2
204 1
204 1
204 1
204 1
193 2
204 2
193 2
176 8
```

If you press a wrong key, back up with the up-arrow. You can back up several notes, if necessary, but you cannot erase notes in the middle of a song. You can also **CLEAR** and start again.

After your input matches the music, press **@** and hear it at full speed. This particular tune could be added to a program to announce the winner of a game or signal the start of a contest.

Experiment 9: Adding Music to a Program

There are two ways you can add your music to a computer program. Any song with only a few notes can easily be played in a program by using the **SOUND** command for each note. The first column on the screen shows the frequency number and the second column shows the duration. To add the song you just played, enter the following instructions. The line numbers you would add to each instruction would depend on the location you wanted this tune to have in your program.

```
SOUND 176.2
SOUND 193.2
SOUND 204.2
SOUND 218.2
SOUND 218.1
```

```
SOUND 218,1
SOUND 218,1
SOUND 218,1
SOUND 204,2
SOUND 204,1
SOUND 204,1
SOUND 204,1
SOUND 204,1
SOUND 204,1
SOUND 193,2
SOUND 204,2
SOUND 193,2
SOUND 176,2
```

If you have a large number of notes to play, it would be easier to use a program loop and a data statement, like this:

```
FOR L=1 TO 17
READ F,D
SOUND F,D
NEXT L
DATA 176,2,193,2,204,2,218,2,218,
1,218,1,218,1,218,1,204,2,204,1,2
04,1,204,1,204,1,193,2,204,2,193,
2,176,2
```

Either method will create the same sound when added to a program. For short songs, a separate instruction and SOUND command for every note is easier. Longer songs can be handled more easily if you use the second method and program a loop with DATA statements. With either method, each instruction would have a line number matched to its position in the program.

If you wish to play the same song more than once in a program, you need to restore the data with a RESTORE instruction.

More Notes With More Memory

Both the program and the notes you have played are stored in the computer's memory. If your Color Computer has more than 4K memory, you can have more than 74 notes stored in your song. If your computer has the 16K memory expansion, change Line 20 in the program to allow more storage space for the music.

```
20 DIM S(1200)
```

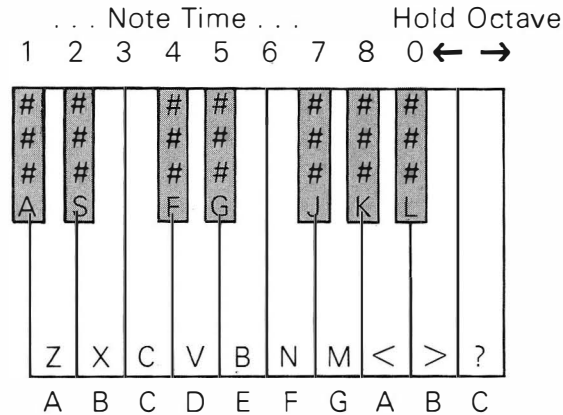
This change allows 1200 numbers to be stored by the program. Since one number is required for the frequency and one more for the duration, this means that you can store and play back a song up to 600 notes long.

Lesson 22: Player Piano (continued)

If you run over the note storage limit with either size memory, the computer will print BS ERROR IN 190. To restore the program, type **G O T O [] 3 0**. This will allow you to hear and edit the song you have entered.

Keyboard Summary

Here is a complete diagram of the keyboard and its functions:



- CLEAR** = Erase Song
P = Play **@** Play Fast
↑ = Back Up

Frequency Look-Up Table

This table shows the musical note in the key of C, the typewriter key that corresponds to that note, and the frequency numbers for all three octaves. The middle octave is set initially.

KEY	K\$	OCT1	OCT2	OCT3
F#	A	45	153	207
A	Z	58	159	210
A#	S	69	165	213
B	X	78	170	216
C	C	89	176	218
C#	F	99	180	221
D	V	108	185	223
D#	G	117	189	225
E	B	125	193	227
F	N	133	197	229
F#	J	140	200	231
G	M	147	204	232
G#	K	153	207	234
A	,	159	210	236
A#	L	165	213	237
B	.	170	216	238
C	/	176	218	239

Lesson 23: Menus

Program Titles and Starting Screens

Your software can begin with a menu, listing a selection of possible things to do. Examples include the menus in Temperature Converter, Lesson 15, and in Car Calculator, Lesson 20. At the beginning of each of these programs, the computer prints a menu and lists several alternatives. When a selection is made by typing a number, the computer branches to the correct section of the program.

These menus each serve the same function. The difference is in the words printed on the screen and in the particular line numbers that are selected.

Simple Menu Module: Temperature Converter

The menu module used in Temperature Converter looks like this on your screen:

```
TEMPERATURE CONVERTER

1. FAHRENHEIT TO CELSIUS
2. CELSIUS TO FAHRENHEIT

SELECT (1,2)?
```

If you don't remember what this menu does, load the Lesson 15 cassette and run the program again. When you type **1** and press **ENTER**, the program branches to the section that converts fahrenheit temperatures to celsius. If you select **2** and press **ENTER**, the program branches to a different section and converts temperatures from celsius to fahrenheit. Notice that the **ENTER** key must be pressed after the number is entered.

Here is how the menu section of this program is written:

```
10 CLS:PRINT
20 PRINT "      TEMPERATURE CONVERTER"
30 PRINT
40 PRINT "      1. FAHRENHEIT TO CELSIUS"
50 PRINT "      2. CELSIUS TO FAHRENHEIT"
60 PRINT
70 INPUT "      SELECT (1,2)";S
80 CLS:PRINT
90 ON S GOTO 100,200
```

Where the word PRINT appears, the computer prints any message in quotes and spaces down one line on the screen. In Lines 10, 30, and 80 the PRINT instruction is used to skip a line. Notice how this creates

Lesson 23: Menus (continued)

a space after the title, before the SELECT (1,2) line, and at the beginning of the next frame in the program. These spaces are added for appearance and don't actually affect the running of the program. Details like this can help your programs look better and be easier to read.

After the menu title and a list of the selections (Lines 10-60) are printed on the screen, the INPUT instruction is used to print the select line and input the number from the keyboard. When the INPUT instruction is used, the **ENTER** key must be pressed after the number is typed.

You can easily change the information on the screen to match the actual selections in your program. Additional selections may be added by adding more lines. As an example, your menu could look like this:

```
* COMPUTER CHESS *  
  
BY RICK PROGRAMMER  
  
1. ONE-PLAYER GAME  
2. TWO-PLAYER GAME  
3. CHESS DEMONSTRATION  
4. CHESS TEACHER  
  
SELECT (1-4)
```

The number of selections, as well as printing on the screen, can be adjusted easily. After a number has been entered, the program must react to the user's selection.

One way to do this is to use the ON GOTO instruction to send the computer to a specific line in the program. In Temperature Converter, this instruction in Line 90 sends the computer to Line 100 if the input is a **1** and to Line 200 if the input is a **2**. The sections of the program beginning at Lines 100 and 200 input the temperature, calculate the answer, and print the results.

Advanced Menu Module: Car Calculator

The more advanced menu used in Car Calculator has two important advantages over the Temperature Converter menu. It does not require the use of the ENTER key after the selection has been typed and it automatically reprints the menu if a wrong key is pressed. If you load the Lesson 20 cassette and run the program you will see this menu on the screen:

```
... CAR CALCULATOR ...
```

1. DISTANCE TRAVELED
2. TIME REQUIRED
3. AVERAGE SPEED
4. GAS MILEAGE

```
SELECT (1-4)
```

Notice that there is no question mark after the SELECT (1-4) line in the menu. If you press any key from 1 to 4, the program will branch to the appropriate section. This action is immediate and the ENTER key is not required. Press any other number or any letter on the keyboard and the menu will blink, showing that the input does not work.

This menu is much easier to use than the menu in Temperature Converter. Pressing ENTER after each number is not difficult, but it is an extra step that you can eliminate with more careful programming. With new computer users, forgetting to press ENTER after a selection can cause confusion and make computers seem "hard to understand." In a similar way, having the computer blink the menu if they press a wrong key is a gentle reminder that helps make computers easy to use.

Compare this listing from Car Calculator with Temperature Converter and see how these two important features have been added. As before, you can change the words and the number of selections in your menu to match any program you write.

```
100 REM ... MENU ...
110 CLS:PRINT:PRINT
120 PRINT "      ... CAR CALCULATOR ..."
130 PRINT:PRINT
140 PRINT "      1. DISTANCE TRAVELED"
150 PRINT "      2. TIME REQUIRED"
160 PRINT "      3. AVERAGE SPEED"
170 PRINT "      4. GAS MILEAGE"
180 PRINT:PRINT
190 PRINT "      SELECT (1-4)
200 K$=INKEY$:IF K$="" GOTO 200
210 SEL=ASC(K$)-48
220 CLS
230 ON SEL GOTO 300,400,500,600
240 GOTO 100
```

LINES 110-190 print the menu on the screen.

LINE 200 sets the string variable K\$ equal to the key pressed. If no key is pressed, K\$="" and Line 200 repeats. As soon as any number or letter is typed, K\$ is set equal to the key.

Lesson 23: Menus (continued)

LINE 210 sets the variable SEL equal to the value of the key. If **1** is pressed, SEL=1; if **3** is pressed, SEL=3.

LINE 230 sends the program to the designated lines if SEL is equal to 1, 2, 3, or 4. If any other number or letter is pressed, the program goes to Line 100 to clear the screen and print a new menu.

Advanced Menu Module: Graphics

Graphics, Lesson 21, shows another example of this technique. Here, the program design is the same and only the words on the screen and the specific line numbers for the sections that follow have been changed.

Here is the menu as it appears on the screen, followed by the actual instructions used in the program:

```
    ... GRAPHICS ...

    1. GRAPHICS CHARACTERS
    2. BILLBOARDS & SIGNS
    3. ART & ANIMATION

    SELECT (1-3)

100 REM ... GRAPHICS ...
20 CLS:PRINT:PRINT
300 PRINT " ... GRAPHICS ...
40 PRINT
50 PRINT "    1. GRAPHICS CHARACTERS"
60 PRINT "    2. BILLBOARDS & SIGNS"
70 PRINT "    3. ART & ANIMATION"
100 PRINT
110 PRINT "        SELECT (1-3)"
120 K$=INKEY$:IF K$="" GOTO 120
130 K=ASC(K$)-48
140 ON K GOTO 1030,2000,3000
150 GOTO 10
```

On Your Own

Use these two menus as examples and pick either the simple version or the more advanced version and simply copy the instructions. Naturally, you will use text and line numbers that match your program design. A menu can begin your program, or it can be used anywhere within. With commercial programs, several menus are often used to direct people through many possible alternatives and selections. In some programs, a menu can be called from anywhere in a program to select a new set of options.

Lesson 24: Program Restarts

Program Endings with Automatic Restart

Many programs are designed with an automatic restart feature. This option can be added to any program with a few simple instructions.

Go Again (Y,N)?

The first lesson in this book, Mathematician, has a program restart option. After the program has calculated the answers, the computer prints GO AGAIN (Y,N)? and waits for an input. Here is the complete program:

```
10 INPUT "A";A
20 INPUT "B";B
30 PRINT "A + B=";A + B
40 PRINT "A - B=";A - B
50 PRINT "A * B=";A * B
60 PRINT "A / B=";A / B
70 INPUT "GO AGAIN (Y,N)?" ;K$
80 IF K$="Y" GOTO 10
```

The lines that create the restart are Lines 70 and 80. Line 70 prints the message on the screen and sets the string variable K\$ equal to the key you press. All INPUT instructions require the **ENTER** key, and K\$ is set equal to the key you type only after the **ENTER** key is pressed. After this input, the computer compares K\$ with the letter Y. If K\$="Y" the computer goes to Line 10 and the program begins again. If any other key is pressed, the computer goes to the next line in the program. Since Line 80 is the last line, the program stops if the key pressed is not Y.

In Guessing Game, Lesson 3, the autostart feature is added in Lines 80 and 90. These additions to the program work exactly the same way as the previous example, except that the text is changed to match the new program. With these lines added, the computer prints TRY AGAIN (Y,N)? and picks a new number for you to guess if your answer is Y. Again, you must press the **ENTER** key after selecting your answer. Here are the actual instructions used for this program restart:

```
80 INPUT "TRY AGAIN (Y,N)";A$
90 IF A$="Y" GOTO 10
```

These same instructions, with different line numbers, are used in Cipher and Music Teacher. The programs, Area Calculator and Coloring Box, simply print MORE (Y,N)? and restart if the input is a **Y**. This technique is also used in Temperature Conversion to ask ANOTHER CONVERSION (Y,N)? You can, of course, change the message within

Lesson 24: Program Restarts (continued)

the quotes to anything you like, such as: CARE TO TRY ONE MORE TIME (Y,N)? or even PLAY IT AGAIN, SAM (Y,N)?

Restart Without the ENTER Key

In Hangperson, Lesson 11, the restart feature uses the INKEY\$ instruction instead of INPUT, as with previous programs. This change makes the program easier to use because the ENTER key is not required. Another change is necessary because of the DATA statements that follow the main program. Here is the section of Hangperson that creates the program restart:

```
420 PRINT "TRY AGAIN (Y,N)?"
430 X$=INKEY$:IF X$=" " GOTO 430
440 IF X$="Y" THEN RUN ELSE END
```

The first change, the elimination of the ENTER key, is accomplished by switching from INPUT to INKEY\$ as a method for inputting the key from the keyboard. In this version above, the computer prints the question (with the question mark) and waits for a key to be pressed. Immediately after a key has been pressed, the computer runs the program from the beginning. In this program, RUN is used because the command GOTO 10 would cause an error. The DIMENSION statement in Line 20 is only used once in a program and cannot be repeated.

The second change from previous restart instructions is the END command. This provides an alternative to restarting the program with RUN. If the program were to continue after Line 440, the computer would read the subroutine starting at Line 500 and create another error. With the restart instructions as shown, the computer will run the program if the key pressed is **Y** and end the program if any other letter or number is pressed.

In Car Calculator a similar set of instructions is used to restart the program. Again, the ENTER key is not required and the program repeats automatically if **Y** is pressed. With these instructions, the computer clears the screen if you do not wish to continue with the program:

```
700 REM ... MORE? ...
705 PRINT
710 PRINT " ANOTHER CALCULATION (Y,N)?"
720 K$=INKEY$:IF K$=" " GOTO 720
730 IF K$="Y" GOTO 100 ELSE CLS
```

The choice between INPUT and INKEY\$ as a method for creating the program restart depends on whether you want the person using your program to have to press the **ENTER** key or not. INKEY\$ reacts immediately to any key while INPUT waits for the enter key to be pressed.

Lesson 24: Program Restarts (continued)

In a program like Cipher, the response to the keyboard during the game is immediate and INKEY\$ is used for all inputs to the program. In programs where the user is pressing **ENTER** throughout, INPUT can be used for program restarts because pressing **ENTER** after the Y,N question would seem like a natural response.

INKEY\$ can be used when only one key is to be entered or when you know exactly how many keys will be typed. If more than one key is to be entered at one time and the length is uncertain, as with a name, INPUT is required.

Lesson 25: Time Delays

Controlling Time Intervals with Counters and Sound

With many program designs, speed is an important consideration. Having the software run as fast as possible is usually the goal. If you are moving objects in a game, for example, you usually want them to move quickly. With a few programs, however, a time delay or pause is essential to their operating at the right pace or speed.

With your Color Computer, you create time delays whenever the SOUND instruction is used. When you program the computer to create a sound, the other activities in the program automatically stop until the sound has finished, then the computer continues with the next instruction. You can also create silent time delays by using a short program loop.

Time Delay: Decision Maker

In Decision Maker, Lesson 8, a time delay is used to program a hold that displays the image on the screen for a specific length of time. Here is the single instruction that creates the delay:

```
90 FOR DLY=1 TO 500:NEXT DLY
```

After printing the answer on the screen, the computer completes this instruction. If you look closely, you will see that it does nothing but wait as the computer counts from 1 to 500. After this counting is complete, the computer goes on to the next instruction in the program and clears the screen.

This pause is added to the program so that the answer will remain on the screen for a short time, and then be erased by the next few instructions. The variable DLY keeps track of the count. The length of time the program waits at Line 90 is set by the number in the instruction. If a number larger than 500 is used, the delay is increased.

The SOUND instruction can also be used to create program delays because the Color Computer does not run any other instructions while the sound is being generated. This instruction creates a similar delay by playing a random note for a duration of 25:

```
90 SOUND RND(100),25
```

In practice, either the program loop or the tone can be used with identical results as far as the timing is concerned. With either method, trimming, or slight adjustment of the timing interval, is done by adjusting the number shown. For the program loop, changing the number of times the loop cycles will change the time delay. For the tone, adjusting the second number (25 in the example above) will change the duration of the sound and the time delay in the program.

Time Delay: Time Machine

In Time Machine, Lesson 12, a single instruction is used to slow the program a specific amount so that the total program cycles in 0.1 second. In this example, the variable used is T and the complete instruction looks like this:

```
90 FOR T=1 TO 19:NEXT T
```

The number 19 was chosen so that the complete program would cycle at the correct rate to create a clock. This delay is, of course, much shorter than the delay in Decision Maker where the program cycles 500 times before continuing.

The experiments in Lesson 12 show how a ticking sound can be added to the clock. This addition slows the program down and requires an adjustment in the time delay to keep the total program cycle time at 0.1 second. In fact, any change in a program will affect its overall speed, including the addition of new instructions or extensions and additions to existing lines.

Lesson 26: Inputs

Using the Keyboard to Control Programs

The person using one of your programs will usually not be aware of the instructions, the GOSUBs, and the details of your work. They will be very aware of the input sections of your program, however, because these sections determine how they interact with the computer. If you handle the inputs well, your programs will be easy to follow. Inputs that are awkward or unclear will immediately make your program, and the computer, hard to use.

Keyboard inputs used with many programs in this book contain messages or prompts that tell the user what to do next. Wherever possible, the ENTER key is not required and the computer responds immediately, like a piano keyboard.

Simple Inputs: Mathematician

In the first lesson, two variables are used in the program to represent the two numbers to be added, subtracted, multiplied, and divided. The program steps for setting these variables look like this:

```
10 INPUT "A";A
20 INPUT "B";B
```

When these instructions are read by the computer, the letter inside the quotation marks is printed, followed by a question mark. When a number is typed and the **ENTER** key pressed, the computer prints the input on the screen and sets the variable equal to the number.

Any message may be written inside the quotation marks and printed on the screen. These instructions would be even more clear if they were written like this:

```
10 INPUT "NUMBER A=";A
20 INPUT "NUMBER B=";B
```

You may want to go even further in telling the user how to interact with your program and print complete instructions, like this:

```
10 INPUT "TYPE A NUMBER FOR A, T
HEN PRESS THE ENTER KEY";A
15 PRINT
20 INPUT "NOW INPUT A NUMBER FOR
B AND PRESS ENTER AGAIN";B
```

When these instructions are run, the screen will show:

Lesson 26: Inputs (continued)

```
TYPE A NUMBER FOR A, THEN PRESS  
THE ENTER KEY? 5
```

```
NOW INPUT A NUMBER FOR B AND  
PRESS ENTER AGAIN? 6
```

Notice that the extra spaces after the word `AND` in Line 20 are used to space the words correctly on the screen and that the computer automatically adds a question mark after the message. While this amount of on-screen instruction would be ideal for someone using a computer for the first time, this could be too much for an experienced user and might get boring if you repeated this amount of information for many inputs in a program.

Input Single Letter: Go Again?

The easiest way to program the computer to print a message and get a letter from the keyboard is with this instruction, as used in Mathematician:

```
70 INPUT "GO AGAIN (Y,N)";K$
```

The computer will print the message inside the quotes and wait for you to type one or more keys and press **ENTER**. The string variable `K$` will contain the key or keys you type.

You can eliminate the `ENTER` key by using `INKEY$`. First print a message telling the user what to do, then get an input from the keyboard with `INKEY$`. Since `INKEY$` reads the keyboard immediately, you will need a program loop to scan the keyboard until a key is pressed. These instructions are used with many programs to get a single letter:

```
10 PRINT "GO AGAIN? (Y,N)"  
20 K$=INKEY$:IF K$="" GOTO 20
```

Notice that Line 10 prints the message and that Line 20 cycles or repeats until a key is pressed. If no key has been pressed, `K$` is a blank (" ") and Line 20 repeats over and over again.

The next line in your program will tell the computer what to do with the input. In the above example, `Y` could send the computer back to the beginning and any other letter could end the program. While this example is often used to create program restarts, this technique can also be used for many other applications.

The dollar signs after `INKEY$` and `K$` tell the computer that the variable can be any key on the keyboard — a letter, number, or punctuation mark.

Input Single Digit Numbers

If your program requires a numerical input, and you know the total number of digits, you can write instructions using `INKEY$` and eliminate the `ENTER` key. This simplifies the input for the user and makes interaction with the program faster and easier.

In using `INKEY$` to get a key from the keyboard, you must convert the string variable to a number. These instructions input a single key as a string variable and convert the input to a number. If the key pressed is `1`, for example, the string variable `K$` will be set to `"1"` and the number variable `A` will be set to the number `1`:

```
10 PRINT "NUMBER A="
20 K$=INKEY$:IF K$="" GOTO 20
30 A=ASC(K$)-48
```

This short program inputs a single key and sets `A` equal to the ASCII value of the key minus 48. Since the ASCII value of the numbers is their number plus 48, this program automatically sets the variable `A` equal to the number typed.

The reason for this complicated dance with `INKEY$` and ASCII values is to get numbers from the keyboard without having to press `ENTER`. To convince yourself that this works, run the program, type any number on the keyboard, and print the value of `A`.

If you would like the computer to echo or verify the number you type, add a semicolon to Line 10 so that the printing will continue on that line, then add a print instruction:

```
10 PRINT "NUMBER A=";
20 K$=INKEY$:IF K$="" GOTO 20
30 A=ASC(K$)-48
40 PRINT A
```

Here is `Mathematician`, rewritten so that the response is immediate and the `ENTER` key is not used. As you can see, the programming is much more complex than in the earlier version. Enter this program and see that it is much faster and easier to use, then you can decide if this programming technique is worth the additional effort.

```
10 PRINT "NUMBER A=";
12 K$=INKEY$:IF K$="" GOTO 12
14 A=ASC(K$)-48
16 PRINT A
20 PRINT "NUMBER B=";
22 K$=INKEY$:IF K$="" GOTO 22
24 B=ASC(K$)-48
26 PRINT B
30 PRINT "A+B=";A+B
```

Lesson 26: Inputs (continued)

```
40 PRINT "A - B=";A - B
50 PRINT "A * B=";A * B
60 PRINT "A / B=";A / B
70 PRINT "MORE? (Y,N)"
72 K$=INKEY$:IF K$=" " GOTO 72
80 IF K$="Y" THEN CLS:GOTO 10
```

Notice that this program only allows single digit numbers for A and B. If you wish to input numbers larger than 9, another step must be added to the program.

Input Multi-Digit Numbers: Math Teacher

The easiest way to input numbers such as 123 and 45654 is with the INPUT instruction, like this:

```
10 INPUT "NUMBER PLEASE";N
```

The variable N will be set to whatever number you type (up to nine digits long), after the **ENTER** key is pressed. In Math Teacher and similar programs, pressing **ENTER** after each answer simply does not work because students will soon get tired of it.

These instructions in Math Teacher compare the number of digits typed on the keyboard with the number of digits in the answer. When they are equal, the input is complete and the computer checks to see if the answer is correct. If the correct answer is 6, for example, the program will compare the input as soon as any key is pressed. If 23 is the correct answer, the computer will wait until two keys have been pressed before comparing the input with the correct answer.

```
400 REM ... INPUT MODULE ...
410 B$=""
420 A$=INKEY$
430 IF A$="" GOTO 420
440 PRINT A$;
450 B$=B$+A$
460 IF X>9 AND LEN(B$)<2 THEN 420
470 IF X>99 AND LEN(B$)<3 THEN 420
```

In this example, the variable X is equal to the correct answer. The numbers typed on the keyboard are combined until the total number of digits is the same as the answer. With this approach to program design, the student can enter answers quickly, with the computer responding immediately to the input.

Input Word: Hangperson

Whenever you wish to input a word, such as the code word in Hangperson or the player's name in a game program, the user will have to use **ENTER** because the computer will have no way of knowing when the input is complete. If you add a reminder to press **ENTER**, there will be no possibility that the user will type a word and then wait, wondering why the computer isn't doing anything.

Here is how this reminder is added to the cassette version of Hangperson. You can use this approach whenever your programs require that the **ENTER** key be used:

```
10 CLS:PRINT "TYPE CODE WORD, PR  
ESS ENTER KEY"  
12 INPUT W$
```

On Your Own

The methods you use in creating inputs for your programs will influence how people feel about your software. If the inputs make sense, are easy to use, and operate immediately whenever possible, your programs will help people use computers in a friendly, secure way. The reverse, inputs that are confusing and slow, will make your software seem badly designed, even if the concepts are brilliant.

An important question to ask yourself in evaluating your own work is whether the choices for action at each point in your program are clear or not. It is perfectly fine to create situations wherein the choices are hard to make, as in a strategy game, but there should be no difficulty at all in understanding what the choices are.

Another point to consider is the consequence of an incorrect entry at any point in your program. In the Advanced Menu Module, for example, the screen blinks if you enter any letter or number that doesn't match the choices shown. This non-destructive interaction when someone types a wrong key is a gentle acknowledgement of a mistake. This is far better than doing nothing with a wrong entry and having the user sit, wondering what happened.

Lesson 27: Music and Sound Effects

From Bach to BANG!

Music, beeps, and other sound effects are used in many programs in this book. Except for Music Teacher and Player Piano, these sounds are not required and are added just for effect. This simple addition of a few beeps here and there in your programs can make a big difference. People have been hearing computers make beeps and clicks in science fiction movies and TV space operas for many years. So don't overlook the possibility of sound effects, or even music, when you are adding the finishing touches to any program.

Music is discussed in Lesson 22, where you can use the Player Piano model to create and play tunes. Adding your compositions to any program is simply a matter of copying the data, as described in the lesson. The other sounds produced by the Color Computer are "music" in that they are made up of single-frequency tones. Adding randomness to the pitch and the duration, however, can produce a wide range of "computer" sounds. Whistles and shrieks result when you change the frequency in a program loop.

Music: Cipher

If you have played the Cipher program and broken the code word, you have already heard the victory tune at the end. This tune was first developed on Player Piano. After playing a tune on the piano and editing it until it is correct, you can transfer the tune to any program by copying the numbers on the screen.

This process is described in Lesson 23. In Cipher, the tune looks like this in the program listing:

```
600 REM . . . SCORE BOARD . . .
610 PRINT
620 IF R<P GOTO 300
640 SOUND 126,6
642 SOUND 148,4
648 SOUND 126,2
650 SOUND 132,2
652 SOUND 148,2
```

Feedback: Sorting

In Sorting, Lesson 14, sounds are used to help you see what is happening as the program runs. A short beep is played each time the computer sorts one of the items on the screen. As the process continues, less and less time is required to sort each item and the beeps get faster.

Lesson 27: Music and Sound Effects (continued)

Audio feedback like this can help you follow the progress of any software if you place instructions for different sounds at different places in the program. To find out how many times a subroutine is being called, for example, you can add a beep before the RETURN and just listen. In large programs with many branches, different notes in each section can be a diagnostic tool.

Music to Think By: Decision Maker

You can use random numbers to create a wide variety of "computer sounds." In Lesson 8, the Music to Think By experiment adds a computer sound effect with these lines:

```
35 FOR S=1 TO 10
36 SOUND RND(100)+150,RND(6)
37 NEXT S
```

The random number generator in your Color Computer is used twice in Line 36. The number controlling the frequency or pitch of the sound is equal to $RND(100)+150$. This is a random number from 1 to 100, added to 150. The net result is a random frequency number from 150 to 250. The range of frequencies and the variation in timing were selected because the effect sounds, to the designer, like a computer thinking. Changing any of the numbers in Line 36 will create a different effect. The right modification would probably create something you would prefer.

If you would like to experiment with computer sounds, try this short program that lets you change the numbers in Line 36 and hear the results.

```
10 REM ... COMPUTER SOUNDS ...
20 CLS
30 INPUT A,B,C,D
40 FOR L=1 TO D
50 SOUND RND(A)+B,RND(C)
60 NEXT L
70 PRINT A;B;C;D
80 GOTO 30
```

When you run this program, enter these numbers: 100, 150, 6, 10. This will duplicate the effect used in Decision Maker. To copy the frequency range used for the robot in Graphics, Lesson 7, input the numbers 10, 230, 3, 10. The robot speech uses an additional delay between the sounds.

After the sound stops, enter another set of numbers and hear the change. If your choices result in numbers that are greater than 255, the program will stop and print: FC ERROR IN 50.

Dribble: Music Teacher

In Music Teacher, a special sound effect was programmed to signal a mistake. It is easy to program the computer to make a loud buzz, of course, but the sound effect was created for a different result. The design goal was to show that the particular note that was played does not work, rather than simply saying: "You blew it!"

The instructions below are used in Music Teacher to rapidly lower the frequency of the note that is not correct. This shows the note, not just the player, to be a wrong choice at this time. This dribble effect will sound slightly different for each note on the scale.

```
300 REM ... OOPS! ...
310 FOR L=F-8 TO 1 STEP -8
320 SOUND L,1
330 NEXT L
```

To hear this effect, load the Lesson 7 cassette and make a mistake, or copy the program above. Simulate the wrong note by inputting a value for F, like this:

```
10 INPUT "FREQUENCY NUMBER (1-255)";F
```

Animated Speech: Graphics

Creating a computer voice for our robot in Graphics, Lesson 7, required a continuous loop to create the sound as well as the opening and closing of the robot's mouth. Here is the complete section that produces the effect:

```
3200 REM ... SPEAK ...
3210 T=RND(300)
3220 FOR DLY=1 TO T:NEXT DLY
3230 PRINT @5*32+15,"=";
3240 SOUND RND(10)+230,RND(3)
3250 PRINT @5*32+15,"-";
3260 GOTO 3200
```

If you clear the screen and run this program you will see the robot's mouth near the center of your screen and hear the speech sounds. It is easier to load the Lesson 7 cassette, select **3** from the menu, and see the complete picture.

On Your Own

In helping you create music and sound effects, the computer can be a valuable instrument. A key in learning to control the wide range of possibilities is to write short programs that let you experiment with ideas and hear the results right away. With this feedback, you can explore ideas or compositions and use the computer to play them.

Computer music itself is a field that may interest you. While this book has used music as an add-on to programs, a complete composition played by a computer is well within the capabilities of your Color Computer. Many modern compositional forms may be written in software and played by computer. Randomness can be used in programs to create many kinds of variations in pitch and tempo. Or you may be more interested in building an instrument you could play, similar to Player Piano.

Lesson 28: Rounding Off Numbers

Lesson 28: Rounding Off Numbers

Making the Digits Fit and Printing Dollars and Sense

The computer often creates answers such as 21.1111111 or 266.666667 when dividing two numbers. The computer automatically rounds off these answers to nine digits. With programs like Temperature Converter and Interest Calculator, it is much better to round off answers like these. When printing dollar amounts it is also a good idea to round off to the nearest penny, rather than show a balance of \$13.33333.

The Integer Function

When the integer function is used, the computer retains only the integer or whole number and discards any fractional part.

Number	Integer
1	1
1.123	1
11.4	11
21.899	21
.955	0

The integer of any number is created with this instruction, where N is the number:

$$N = \text{INT}(N)$$

It is more common to consider a fractional part of a number and to round off the answer. That is, if the fraction is .5 or more, the next highest number is used. To duplicate this with the integer function, just add 0.5 to the number before the integer is calculated.

$$N = \text{INT}(N + .5)$$

With this change, the numbers from the previous table look like this:

N	INT(N)	INT(N+.5)
1	1	1
1.123	1	1
11.4	11	11
21.899	21	22
.955	0	1

Lesson 28: Rounding Off Numbers (continued)

Nearest Degree: Temperature Converter

In Temperature Converter, the answers are rounded off to the nearest degree. This is done by adding .5 to the answer and creating the integer of the result. Both conversion sections of this program use the same design, like this:

```
100 INPUT "HOW MANY DEGREES F";F
110 C=5/9*(F-32)
115 C=INT(C+.5)
120 PRINT F:"DEGREES F =";C;"DEG
REES C"
```

In Line 110 the variable `C` is set equal to the temperature in celsius. The next line rounds this value off to the nearest degree. It would be possible to round off the value to the nearest tenth of a degree by substituting this instruction:

```
115 C=INT(C*10+.5)/10
```

Now the value is multiplied by 10, rounded off to the nearest integer, and divided by 10 again. The effect will be to round off all values of `C` to the nearest .1 degree, as shown below:

N	INT(N)	INT(N+.5)	INT(N*10+.5)/10
1	1	1	1.1
1.123	1	1	1.0
11.4	11	11	11.4
21.899	21	22	21.9
.955	0	1	1

The number 10 in Line 115 creates a final value that is rounded off to one decimal place. Replacing 10 with 100 creates a value that is rounded off to two decimal places. Similarly, you can increase the number of zeros in the number used in Line 115 and increase the number of decimal places in the final answer.

Nearest Penny: Interest Calculator

A common requirement for business programs is to calculate and print results to the nearest penny. Since this represents a number of dollars, accurate to two decimal places, the number 100 is used to round off the answer. This instruction from Interest Calculator can be used to round off the number `N` to two decimal places.

```
INT(N*100+.5)/100
```

While maintaining your accounts to the nearest penny may seem accurate enough, business programs often keep dollar amounts to the nearest mill, or one tenth of a cent.

Lesson 29: Scoreboards

Who Won?

A report card or progress report can be an important part of any educational program. Similarly, almost any game program, whether it is a strategy game or a high-speed action game, can benefit by the addition of a report or scoreboard at the end. The format for either of these reports is similar to the Menu Module in that a message is printed on the screen, showing the status of the program. The question, GO AGAIN (Y,N), can also be included in the display.

Report Card: Math Teacher

The scoreboard in Math Teacher is used to show the student's progress in two ways. After 20 problems have been answered, the report card shows the number of correct answers and the skill level. Here are the actual instructions used in Math Teacher:

```
700 REM ... REPORT CARD ...
710 CLS:PRINT
720 PRINT "          ... REPORT CAR
D ... ":PRINT
730 PRINT "      YOU GOT";20-G
740 PRINT "      OUT OF 20 CORRECT"
:PRINT
750 PRINT "      YOUR SKILL LEVEL I
S";S:PRINT
760 PRINT "      SAME PLAYER GO AGA
IN (Y,N)?"
770 Y$=INKEY$:IF Y$="" THEN 770
780 IF Y$="Y" THEN CLS:GOTO 20
```

This format is very similar to the Menu except that two variables are used to print the correct results. The variable `G` is used to store the number of wrong answers. The number `20-G` in the Report Card is the number of answers that were correct. The variable `S` stores the skill level. This number is dynamically adjusted by the program to match the difficulty of the problems to the student's ability.

If a student misses one problem and has a skill level of five, the scoreboard will show:

```
... REPORT CARD ...

YOU GOT 19
OUT OF 20 CORRECT

YOUR SKILL LEVEL IS 5

SAME PLAYER GO AGAIN (Y,N)?
```

Lesson 29: Scoreboards (continued)

With a similar design, you can add a scoreboard to show the results of any program where variables in the program change. Just use print instructions to write the messages on the screen, with variables representing the quantities that change. The program restart line can also be added to the report card, as shown in Lines 770 and 780 in this example.

Scoreboard: Multi-player game

A typical scoreboard for a two-player game could look like this after each round:

```
... SCOREBOARD ...

PLAYER      GAMES WON    TOP SCORE
-----
TOM         3            100
DICK        2            121
HARRY       1            85
```

SAME PLAYERS GO AGAIN? (Y,N)?

In this example, you would need an input section at the beginning of the program to input the players' names, and variables for storing the number of wins and highest score for each player. As with any program, the names you choose for the variables are up to you. Here are some suggestions:

```
First player's name ..... P1$
Second player's name ..... P2$
Third player's name ..... P3$

First player's wins ..... W1
Second player's wins ..... W2
Third player's wins ..... W3

First player's max. .... M1
Second player's max. .... M2
Third player's max. .... M3
```

You can easily create scoreboards that show the status of the players and any other information you would like to display. If you design the scoreboard early in the programming of your game idea, the variable names you select can be picked to make their function clear.

Lesson 30: Dynamic Debugger

Finding and Fixing Problems

If you try designing and writing your own program, you will find sooner or later that it is easy to forget how your own software works. A program that was perfectly clear to you when you wrote it can become a mystery when you look at it after a day or two. One way to find out how any program works is to run the program, look at all the variables, and see how they change and what they do.

This lesson shows you how to create and use the Dynamic Debugger, a powerful tool you can use to take the mystery out of almost any program. You can use this tool to understand and be more clear about your own work, or to figure out what someone else's software does.

Use Math Teacher as an example to see how the Debugger can help you understand how a program works. Begin by loading the program from the Lesson 17 cassette. Run Math Teacher and become familiar with what it does.

Mark the Variables

As you can see, this program uses many variables to keep track of the numbers used in the problems, the correct answers, the number of wrong answers, TRM time delays, and many other factors. Go through the listing and mark every variable used, including all the string variables with dollar signs.

Write a Custom Debugger

The Debugger is a short program you put in the computer in addition to the program you are trying to understand. The Debugger must begin with a line number that has a higher line number than any line in the main program. Since the last line in Math Teacher is Line 780, this Debugger can start at Line 1000.

To create a custom Debugger, just write an instruction to identify and print each variable in the program. Here is an example using Math Teacher:

```
1000 REM    . . .    DYNAMIC DEBUGGER
1010 PRINT "S";S
1020 PRINT "E";E
1030 PRINT "L";L
1040 PRINT "A";A
1050 PRINT "B";B
1060 PRINT "X";X
1070 PRINT "G";G
```

Lesson 30: Dynamic Debugger (continued)

```
1080 PRINT "T";T
1090 PRINT "VAL(B$)";VAL(B$)
1100 PRINT "LEN(B$)";LEN(B$)
```

To check this program and see what it does, just type **GOTO** **1000** and run the Debugger. Your screen will show a printout of all the variables and their current contents, like this:

```
GOTO 1000
S 5
E 0
L 1
A 2
B 1
X 2
G 3
T 114
VAL(B$) 0
LEN(B$) 0
OK
```

The values you will see will depend on the status of Math Teacher in your computer. In the example above, the skill level S is 5, the two numbers selected for the problem A and B are 2 and 1, the correct answer X is 2, and so on. At first, you may not know exactly what all the variables stand for. The point of the Debugger is that you can use the program itself to see how the variables change, and from this you will be able to actually see what they do.

Experiment 1: Problem and Answer Check

The program description and the flowchart with Lesson 17 point out that the variables A and P store the numbers in the problem, and that the variable X stores the answer. Begin using the Debugger to find out if this is actually happening or not.

1. Run the program as usual.
2. When the first problem appears on the screen, press the **BREAK** key to stop the program.
3. Type **GOTO** **1000** to run the Debugger program.

The printout on your screen will now show the first problem and the information in all the variables. This lets you see what is happening in the program at the point where the first problem is shown on the screen. You will see that the variable A contains the first number in the problem, variable B contains the second number, and variable X contains the answer.

The results you see will depend on the numbers chosen for the first problem. In the following example, the computer happened to pick $5 - 2 =$ as the first problem, with these results:

```
5 - 2 =  
BREAK IN 425  
OK  
GOTO 1000  
S 5  
E 0  
L 1  
A 5  
B 2  
X 3  
G 2  
T 242  
VAL(B$) 0  
LEN(B$) 0  
OK
```

In this example, $A=5$, $B=2$, and $X=3$. Your results will probably be different, but the variables A , B , and X should match the problem on the screen exactly.

When you are satisfied that the variables match, run the program again and use the Debugger to check A , B , and X with a different problem on the screen. Each time you run the program, the variables should match, as before. This shows you that these variables are doing what you expect them to and that the program is running correctly, up to this point.

Experiment 2: Problem Counter Check

Now check and verify that the program counter is working correctly. This counter is the variable L . In Line 20, L is used as the loop counter to keep track of the number of problems that have been printed on the screen. At the beginning, L should be equal to 1. As the number of problems increases, L should also increase to show the number of the next problem. Notice the value of L on the screen. In the first experiment, L is 1 because you stopped the program with the first problem on the screen. Now check L with this technique:

1. Run Math Teacher and answer the first 3 problems, leaving problem number 4 on the screen.
2. Stop Math Teacher with **BREAK**.
3. Run Dynamic Debugger with **GOTO** .

Lesson 30: Dynamic Debugger (continued)

Again, your screen will show different values for other variables, but L should be equal to 4, like this:

```
7 * 6 =  
BREAK IN 430  
OK  
GOTO 1000  
S 8  
E 0  
L 4  
A 7  
B 6  
X 12  
G 3  
T 143  
VAL(B$) 0  
LEN(B$) 0  
OK
```

As you can see from your printout, A, B, and X match the problem and the answer, as before, and L is equal to 4.

Experiment 3: TRM Counter Check

Variable T is used to keep track of the time required for each answer. Check the program and verify that T actually relates to the answer time.

1. Run Math Teacher.
As soon as the first problem is displayed, press **BREAK**.
2. Use Debugger to find the value of T and remember what the value is.
3. Run Math Teacher again.
This time, wait for a few seconds before pressing **BREAK**. You have now frozen the program in a different condition, with a longer time elapsed after the problem was first displayed.
4. Use Debugger again to see the value of T.
If the time delay before pressing **BREAK** was longer than before, the value of T should be higher.

5. Try this experiment again, waiting for about a minute before pressing **BREAK**, stopping the program and the timer.

The value of T should increase in proportion to the time elapsed.

You can repeat this experiment as many times as you like to check the operation of the TRM timer. The variable T is incremented in Line 425, which is part of the Input Module. As long as the program is waiting for an input, T will continue to increase. When any key is pressed for the answer, the value of T is saved and used later to adjust the skill level.

On Your Own

You have used Math Teacher to show how the Debugger can help you see what is happening in a program. Notice that you can stop at any point with **BREAK** and use Debugger to see exactly what is going on. With patience, this programming tool can be a valuable aid in understanding other people's programs, finding out what any variable in a program actually does, and even helping you recall and understand how your own programs work. In situations where any software just doesn't operate the way you expect it to, Debugger is your key to discovering the truth. After you are clear about what is happening, you can choose to make any changes or additions. After each modification, Debugger will show you the results.

This technique can be used without writing a complete program for printing the variables. If you only wish to check the operation of the timer, for example, you could use **BREAK** to stop the program at the point you wish to examine, and simply type: PRINT T . Where more than one variable is involved or when you want to understand all aspects of a program, writing a complete Dynamic Debugger is well worth the time required.

Buzzwords

Many conventional words have special meaning when computers are the subject being discussed or written about. This dictionary of computer terms can help you understand some of these words and help you use them effectively. Not all computer terms are included, but the common terms used in this book are described below.

ARRAY — a numbered sequence of variables that can hold integers.

BASIC — a computer language designed to be easy to use and understand. The beginner's all-purpose symbolic instruction code.

CHARACTER — anything that will print in a single space on the screen, including numbers, letters, punctuation, and graphics symbols.

COMMAND — a direction to the computer.

COMMAND MODE — a condition that exists when the computer is waiting for you to enter a command or a line number. The message OK appears on the screen whenever you are in command mode.

CONDITION — a comparison that can be true or false. Conditions are tested and evaluated in IF statements.

DIMENSIONING — automatically sets aside space and defines boundaries for an array.

GRAPHICS CHARACTER — a symbol used to create designs and patterns on the screen.

INSTRUCTION — a direction given to the computer. If a line number is used before an instruction, the instruction becomes part of a program when **ENTER** is pressed. If there is no line number, an instruction is carried out immediately when **ENTER** is pressed.

INTEGER — a whole number, without any fractional part, in the range of -32768 to 32767. **Note:** Do not use commas or periods with numbers.

LETTER — the characters from A to Z.

LINE — anything you type on the keyboard. Pressing **ENTER** ends a line.

LINE NUMBER — the number in front of an instruction which indicates its position in a program.

MACHINE CODE — a group of instructions, not in BASIC, which are read directly by the computer chip. To write machine code instructions, you need an instruction set for the computer chip and special programming techniques, using POKE.

Buzzwords (continued)

MEMORY — where the computer stores numbers, strings, arrays, instructions, complete programs, and whatever is currently being displayed on the TV screen. These all use space in memory. With more memory, your Color Computer will be able to hold and run longer programs.

MESSAGE — an informative note printed on the screen for the benefit of the user of a program.

NUMBER — the characters from 0 to 9, individually or combined.

PROGRAM — a set of lines with line numbers. A program can be entered from the keyboard or loaded from tape.

PUNCTUATION — the characters \$: ? () - + * / = > < ; . .

SUBROUTINE — a section of a program ending with RETURN and called by GOSUB.

VALUE — the result that can be printed. The value of 10 is 10. The value of 2+3 is 5. The value of A is the number stored in the variable A. The value of A\$ is the character or characters stored in A\$.

VARIABLE — a section in the computer's memory that holds information. All variables have names such as A, A\$, BOB, or X(1). Variables can easily be changed. Number variables each hold a number. String variables hold one or more characters. Arrays hold one or more number variables. Variable names may be of any length; however, only the first two characters are significant with this computer.

Index

Alarm clock, 76
Amling, Jim, ii
Area Calculator: Lesson 9, 55-59
Art and animation, 186
Arrays, 81, 89-90, 141-142
ASC(K\$), 34
ASCII numbers, 96-97, 117, 139-140, 156, 163, 184
Autostart, 19
Auto average, 26
Auto repeat, 47
Average Calculator: Lesson 4, 23-27
Billboards and signs, 184
Branching, 49-52, 126-127, 173-179
BREAK, 4
Buzzwords, 229
Calculators, 55-59
Car Calculator: Lesson 20, 173-179
CHR\$, 29, 32-34, 69-70
Cipher: Lesson 16, 113-122
CLS (clear screen), 19
Colon, 24, 73-74
COLOR, 67-69
Coloring Box: Lesson 11, 67-71
Command mode, 9, 122
Comparison (>, <, =), 17-18
Comparison (< >), 73-75
Compounding interest, 61-64
Coin Flipper: Lesson 2, 11-15
Counting Machine: Lesson 6, 37-43
Data Processing, 89-98
Decision Maker: Lesson 8, 49-52
Decisions (IF/THEN), 11, 12
Debugging programs, 223
Dice simulation, 80
DIM (dimension), 81, 89, 115, 197
Distance calculator, 173
Division by zero, 7
DLY (delay), 49, 51, 160, 207-208
Double loops, 67-69
Dynamic Debugger: Lesson 30, 223-227
Dynamic sorting, 94-95
ENTER, 1
Error message, 1, 7, 41
Expressway: Lesson 5, 29-34
FC ERROR, 41
Feedback, 215-216
Flash cards, 127
Flowchart, 10, 16, 21, 28, 35, 44, 48, 53, 60, 66, 72, 78, 88, 100-102, 111-112, 123, 135-137, 151-154, 168-171, 181

Index (continued)

Formulas, 23, 55-59, 173-175
FOR/FOR, NEXT/NEXT (double loops), 67-69
FOR/NEXT, 23-26
FOR/NEXT/STEP, 37-43
Game design, 113, 139, 146-147, 155-167
GOSUB, 157-167
GOTO, 2
Graph, 83-87
Graphics characters, 184
Graphics: Lesson 21, 183-192
Guessing Game: Lesson 3, 17-20
Hangperson: Lesson 18, 139-150
Help, i
Halt!, 15
How It Works, ii
IF/THEN, 7, 11-14, 17
INKEY\$, 34, 49-50, 77, 117, 156-157, 204-205
INPUT, 5-7, 19, 55-59, 144, 203-204
Inputs: Lesson 26, 209-213
Input module, 129-130
Input note, 156
Input word, 141
Instruction, 1
INT (integer), 41, 61-65, 219-220
Interest Calculator: Lesson 10, 61-65
Introduction to Computing, 1-4
JOYSTK(N), 29, 31, 34
Kaleidoscope: Lesson 7, 45-47
LEN (length of a string), 140
LET, ii
Line numbers, ii
LIST, 2, 3, 52
Loop, 13, 23-26, 37, 42
Mathematician: Lesson 1, 5-9
Math Teacher: Lesson 17, 125-134
Menus, 106, 177, 183
Menus: Lesson 23, 199-202
Messages, 7
MID (character in a string), 140
Multiple lines, 24, 73-74
Music and Sound Effects: Lesson 27, 215-218
Music notation, 161-162, 198
Music Teacher: Lesson 19, 155-167
Musical instruments, 193
Nelson, Ted, 43
NEW, 4
Not equal (< >), 73-75
O and 0, i
ON/GOTO, 49-50, 52
Parenthesis, 9
Pinball, ii
Player Piano, 193

POINT (B,V), 29, 31
 PRINT, 1-3, 20
 PRINT @, 14, 49, 70-71, 73-75, 82-86
 PRINT#-2, 65
 PRINT TAB, 29, 32, 33
 Printer option, 65
 Printing money (\$), 65
 Printing tables, 61-63
 Probability, 46
 Probability: Lesson 13, 79-87
 Program, 1-2
 Program Restarts: Lesson 24, 203-205
 Prompts, 104-105
 Random numbers, 11-15, 19-20, 49-51, 79-87
 READ/DATA, 148-150, 186, 188
 REDO, 8
 Report card, 133-134, 221-222
 RETURN, 157-167
 Reverse video, i
 Rounding off, 61-65, 103-104, 175-176
 Rounding Off Numbers: Lesson 28, 219-220
 RUN, 3
 Scoreboards: Lesson 29, 221-222
 Semicolon, 2, 14, 32
 SET(H,V,C), 45-47
 Scientific notation, 8
 SHAPE, 67-69
 Skill level adjustment, 130-131
 Sorting: Lesson 14, 89-99
 SOUND, 25-26, 29, 32, 40, 47, 51, 75, 87, 120, 144, 155-167, 188, 193, 195-197,
 207, 215-218
 Spaces between letters and numbers, i
 Space sounds, 42
 Speed calculator, 175
 STEP, 37
 String manipulation, 139
 Stop watch, 77
 Subroutines, 157-158
 Tables, 61-63
 Temperature Converter: Lesson 15, 103-110
 Time calculator, 174
 Time delays, 73-77, 160, 188
 Time Delays: Lesson 25, 207-208
 Time Response Monitoring, 132-134, 226
 Time limit, 32
 Time Machine: Lesson 12, 73-77
 Timer, 76
 Variables, 6, 12, 18, 223

IMPORTANT NOTICE

ALL RADIO SHACK COMPUTER PROGRAMS ARE LICENSED ON AN "AS IS" BASIS WITHOUT WARRANTY.

Radio Shack shall have no liability or responsibility to customer or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by computer equipment or programs sold by Radio Shack, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer or computer programs.

NOTE: Good data processing procedure dictates that the user test the program, run and test sample sets of data, and run the system in parallel with the system previously in use for a period of time adequate to insure that results of operation of the computer or program are satisfactory.

RADIO SHACK SOFTWARE LICENSE

A. Radio Shack grants to CUSTOMER a non-exclusive, paid up license to use on CUSTOMER'S computer the Radio Shack computer software received. Title to the media on which the software is recorded (cassette and/or disk) or stored (ROM) is transferred to the CUSTOMER, but not title to the software.

B. In consideration for this license, CUSTOMER shall not reproduce copies of Radio Shack software except to reproduce the number of copies required for use on CUSTOMER'S computer (if the software allows a backup copy to be made), and shall include Radio Shack's copyright notice on all copies of software reproduced in whole or in part.

C. CUSTOMER may resell Radio Shack's system and applications software (modified or not, in whole or in part), provided CUSTOMER has purchased one copy of the software for each one resold. The provisions of this software License (paragraphs A, B, and C) shall also be applicable to third parties purchasing such software from CUSTOMER.

RADIO SHACK  **A DIVISION OF TANDY CORPORATION**

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA

280-316 VICTORIA ROAD
RYDALMERE, N.S.W. 2116

BELGIUM

PARC INDUSTRIEL DE NANINNE
5140 NANINNE

U.K.

BILSTON ROAD WEDNESBURY
WEST MIDLANDS WS10 7JN