14278

COLLECTING SHORTWAVE FOLK MUSIC

Popular Electronics®

WORLD'S LARGEST-SELLING ELECTRONICS MAGAZINE

MAY 1976/$1

Build the Audio "Detective"

MEASURES PHONO CARTRIDGE AND MIKE OUTPUTS

How to Improve Equipment Grounds

Severe-Weather Alert Project

ACTIVATES EMERGENCY WEATHER BROADCASTS

TV Table Tennis II

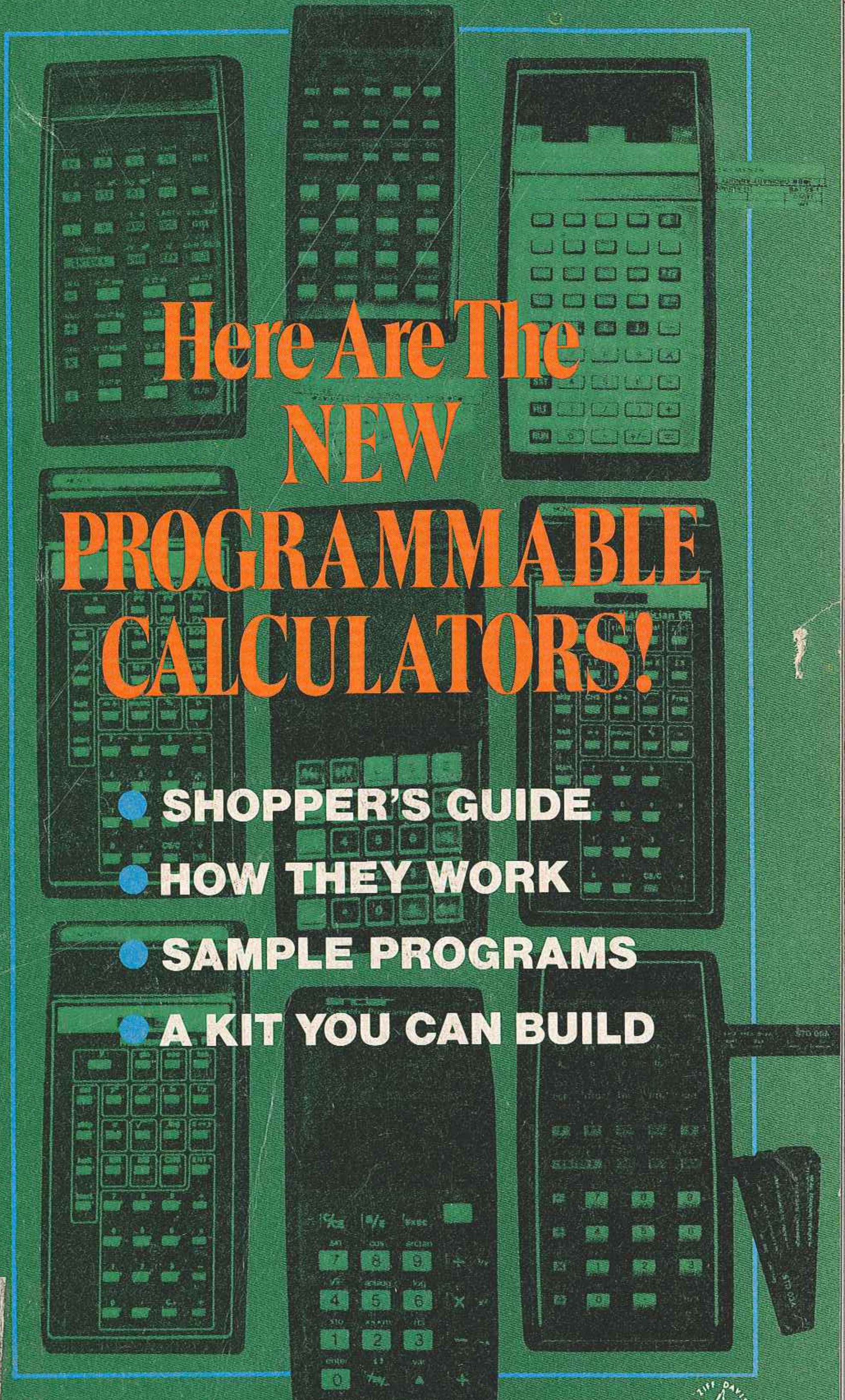
ADDS SCORING & SOUND

Test Reports

STAX SR-5 ELECTROSTATIC HEADPHONES

EPICURE MODEL 4 STEREO PREAMPLIFIER

KRIS VICTOR II MOBILE



Here Are The NEW PROGRAMMABLE CALCULATORS!

- SHOPPER'S GUIDE
- HOW THEY WORK
- SAMPLE PROGRAMS
- A KIT YOU CAN BUILD

A ZIFF-DAVIS PUBLICATION

Electronics Library

ENCYCLOPEDIA OF ELECTRONIC CIRCUITS
by Leo Sands and Donald Mackenroth

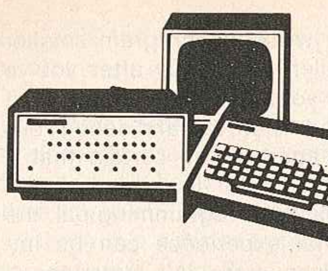
Many different representative circuits are illustrated and analyzed, including a-f and r-f amplifiers, audio control circuits, biasing, control, and coupling circuits, frequency converters and multipliers, instrumentation and logic circuits, modulators and demodulators, power supplies, signal processors, and signal generators. Tube, transistor, and IC configurations are covered. Capsule descriptions for each entry appear in paragraph form.
Published by Parker Publishing Co., West Nyack, NY 10994. 288 pages, hard cover.

SEMICONDUCTOR FUNDAMENTALS
by R.F. Coughlin & F.F. Driscoll

As the title implies, this is a very basic book designed primarily for the beginning student in technical schools, junior colleges, or high schools. The authors have written under the assumption that all this material is new to the student and have treated each topic in depth. The text material is progressive, with the first eight chapters laying the groundwork for actual work with semiconductor devices as covered by later chapters. Diagrams, charts, and sketches are used lavishly as are examples, summaries, and self-check problems. For the reader with a tenuous grasp on semiconductor theory, this book should help strengthen his understanding.
Published by Prentice-Hall, Inc., Englewood Cliffs, NJ 07632. 308 pages. $14.95 hard cover.

REFERENCE DATA FOR RADIO ENGINEERS, Sixth Edition

The new edition of this engineer's "bible" has been completely updated, revised, and enlarged to encompass material on the latest techniques in electrical and electronic engineering. Much of the material is too new to be found in many reference libraries. Subject areas that have been given major additions and/or revisions include: semiconductors and transistors, microminiature electronics, digital computers, electroacoustics, optical communications, navigation, transistor circuits, radio noise & interference, quantum electronics, optoelectronics, and filters. The material is broken up into 48 in-depth chapters and includes hundreds of charts, nomographs, diagrams, curves, tables, and illustrations.
Published by Howard W. Sams & Co., Inc., 4300 W. 62 St., Indianapolis, IN 46206. Hard cover. More than 1300 pages. $30.00.



Computer Bits

By Jerry Ogdin

GAMES FOR LEARNING

LEARNING to program isn't difficult, but it does take experience. One of the biggest headaches in learning how to program is sample problem selection. Most text books illustrate problems that are either too simple, too complex, or too far afield for good learning experience.

We don't believe that the experience in converting traditional games to computer form is very enlightening. Some of the worst examples of coding are to be seen in some published game programs, often because they have been written by novices. One should wait to write programs like blackjack, poker, NIM and tic-tac-toe until he has more experience. The programming games I'm talking about are "games" for writing programs.

Reading Programs. Authors learn to write by reading good works. Composers who haven't heard good music can't compose very well. Programmers think that all they need to do is read the reference manual and they'll have everything they need. Well, it isn't so. You should read programs, too. The best way to glean experience from somebody else's hours of labor is to read their codes. Unfortunately, you'll have to read an awful lot before you can decide which is good and which is bad. Ideally, you should only read the good stuff.

The best way to attack a program reading exercise is to get a pad and pencil, and start breaking the program into small "modules." You can usually discern a pattern of subroutines. Find all the instructions that refer to subroutines, and then locate all of the subroutines. Determine what each subroutine does and jot it down, you can make up a table of subroutine names, their intended functions and necessary data conditions for proper execution. After all the subroutines are defined, you can trace through the main program.

Sometimes you will run into a stone wall. If you can't figure out how something works, go back to the reference manual for the computer and work it through in detail. Manually, act like the computer, recording the register contents on paper so you can be certain you're following the operation in detail. Eventually, you'll discover how it works, and you'll have learned a new programming technique.

Steps in Programming. Programming is a mental activity. You have to think about it before you approach the computer. You should work out your understanding of the problem, and reduce it to some kind of written description. If you want to write a program that will help check-out the new terminal you've just attached to your computer, your stated goal might be:

Level 1: Write the letter "A" out to the terminal indefinitely.

Now, in successive stages, break this instruction into more detailed steps until you arrive at a point where you think you can write code. In this simple example, you might write:

Level 2: a. Put ASCII "A" in a register.
b. Send the register out to the terminal.
c. Go back to step a.

Now, translate these statements into a detailed form that takes cognizance of your particular computer and its environment:

Level 3: a. Put 0100 0001 (letter A in ASCII) into A-register.
b. CALL the TYPE subroutine.
c. JUMP back to step a.

Finally, you can write this in code for your particular computer. Novice programmers try to do all these steps in their heads; semi-experienced programmers start writing at Level 3. But, experienced programmers always try to start at Level 2, where their thinking processes are not limited by the idiosyncracies of the computer.

Once you have a clear, albeit informal, description of what you want the computer to do (Level 3), you can code it in assembly language or actual binary (or octal or hex), depending upon what form you feel most comfortable with. At long last, your program will be in memory and ready for execution.

It is not uncommon to spend ¾ of your time planning before you write your code.

Programming "Games." The easiest kind of "game," and one that programmers engage in all the time, is to write some specific program in a small number of instructions, regardless of execution time. Pick some function—say, editing a number from BCD into proper money notation with commas, decimal point and dollar sign. Or, try to write a program that will move any contiguous group of bytes to anywhere else in memory. Note that these programs have three important features: they are relatively small, easy to understand, and potentially useful to you in the future.

Now, write the program any way you can and demonstrate that it works. Here comes the challenge. Write it again, but this time set a goal of reducing the size of the program by some specific number of bytes. Also, set a time limit so you don't go on forever. If you have written a program of about 100 instructions or bytes of memory, you can set the goal of reducing the number of bytes by 10 per cent within two hours. With your goal—and the original program—in hand, go back to the programming reference manual for your computer and try to find some unique set of instructions that will help you write the program smaller and smaller. Stop only after you've achieved your goal or the time limit has passed. Then, test and debug your program again to be certain that it works.

If you have a programming pal, the educational experience can be improved even more. All experienced programmers have witnessed somebody demolish their well-polished code with a clever trick. Usually, you can do it right back! It is not uncommon for a small subroutine to be cut to a fourth of its original size by two people working back and forth in this challenging fashion. You'll certainly learn a lot about programming this way.

Next, write the same program, but make it your goal to write a program that executes faster than the smallest program you've written. You'll find that you generally can't optimize both size and execution time simultaneously, so your faster program may be larger than your smallest program. Again, challenge another programmer to beat your time—you may be surprised at the results. But, you can then go back and beat his time—and he'll be surprised.

From this set of exercises, you'll quickly learn that a program that works may not be enough. You may also have to rewrite and rewrite to make it fit within the resources you have available. You will also learn more about the computer and its instructions and how to exploit them. Finally, you should probably learn that no program is "best". Some programs are shorter than others, some are faster, and some offer more features or fewer restrictions.

After you've finished your programs, don't discard them. No matter how trivial the program may seem, keep it on hand for the day when you need it again. Get a strong three-ring notebook and file all your programs there. If you can't think up a good classification scheme, store them away in chronological order. You'll be amazed at how frequently you go back to crib something from earlier work.

Another challenging program is one that will completely clear all of storage to zero. This program must not only zero out all bytes of storage, but it must also destroy itself completely. On many computers, it can't be done (so far as anybody knows, anyway). However, you should see just how few bytes you have to leave in memory; like golf, the goal is to minimize the score.

This is a particularly nasty programming problem. Since the program wipes itself out of memory, you have to put it back into memory every time you use it. That is a nuisance, and so there is a definite incentive to do the program right the first time. Experienced competent programmers spend a lot of time "desk checking" their programs. They read and re-read the program until they're absolutely certain that it is completely correct. Only then do they submit the program to the computer. It is better than putting just any old program in memory to see if it'll work. This self-clearing program will show you just how important that desk-checking is, because if you don't do it right, you'll have to put the program back into memory time and time again.

By the way, it isn't considered fair to put the program in read-only memory and then clear only the read-write memory—the program must be in read-write storage, too. Send me your examples in source code form for your computer, and we'll publish the best ones for each popular model in a forthcoming issue. ◈

(Editor's Note: Someone once told us of a two-step program that will zero all memory except for addresses 0 and 1 on an Altair-8800. We have tried it and it works. After turning the main power on, resetting and stopping the computer, and before loading any bootstrap or the BASIC tape, deposit 063 (INX SP) at address zero, followed by Deposit Next 307 (RSTO). Re-examine address zero (063), then hit RUN. Allow the machine to run until the address LED's settle down, then hit STOP. Re-examine address zero and 1. There should be only one data LED on at each address, all other addresses should be dark. You may have to repeat the 063-307 program again. If you need to clear the two addresses, simply Deposit zero in each.)

More Hobbyist Clubs.

Florida
South Florida Computer Group (305) 752-8395, (305) 271-2805
Illinois
Cache, Bill Precht, P.O. Box 36, Vernon Hills, IL 60061
Chicago Area Computer Hobbyists' Exchange (312) 620-1671
Louisiana
Emile Alline, 1119 Penn Ave., Slidell, LA 70458. (504) 641-2360
Massachusetts
The Alcove Computer Club, (617) 664-4271
Minnesota
Cliff Tanner, 121 Avenue C, Cloquet, MN 55720. (218) 879-4581
Mississippi
Gulfport Computer Group (601) 896-7193
New Jersey
The Amateur Computer Club of New Jersey, Sol Libes, (201) 889-2000
New York
Ithaca Computer Group (607) 272-2339
New York City Micro Hobbyists Group, Robert Schwartz, 376 Riverside Dr., Apt. 1E, New York, NY 10025
Texas
Computer Hobbyist Group of North Texas, Bill Fuller, 2377 Dalworth 157, Grand Prairie, TX 75050
NASA/JSC Club, Marlowe D. Cassetti, 1011 Devon Port Lane, Seabrook, TX 53085
Washington
Steve Herber, 2362 Franklin East, No. 402, Seattle, WA 98102
Wisconsin
Don Stevens, P.O. Box 159, Sheboygan Falls, WI 53085