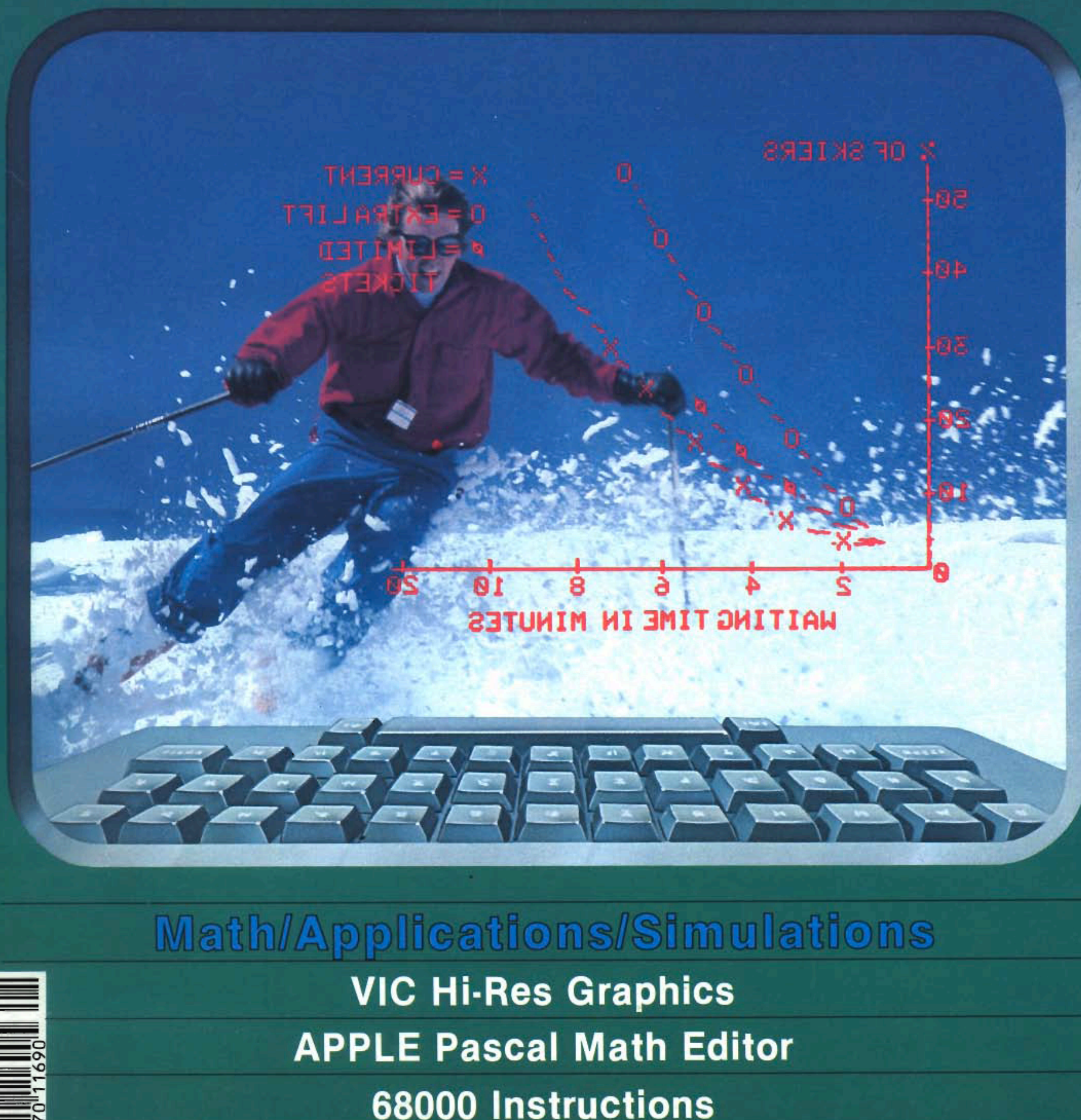
NO. 56 JANUARY 1983

U.S./Canada Edition: $2.50
International Edition: $2.95
United Kingdom Edition: £2.00

MICRO™

Advancing Computer Knowledge



Math/Applications/Simulations

VIC Hi-Res Graphics

APPLE Pascal Math Editor

68000 Instructions

ARTSCI HAS
NEW PRODUCTS, NEW PACKAGING, NEW OFFICES,

[illegible] Favorite. New features include 80-column board and hard-disk compatibility, 70-column upper and lower case display 160-character line length, multiple-copy print feature, and DOS housekeeping commands within the program. $149.95

[illegible] disk compatibility, individual column widths, invisible columns for confidential data and full compatibility with VisiCalc.™ Expansion memory boards may be used to expand the worksheet size. $149.95

[illegible]

ARTSCI — a new look, with the same old reliability at an affordable price

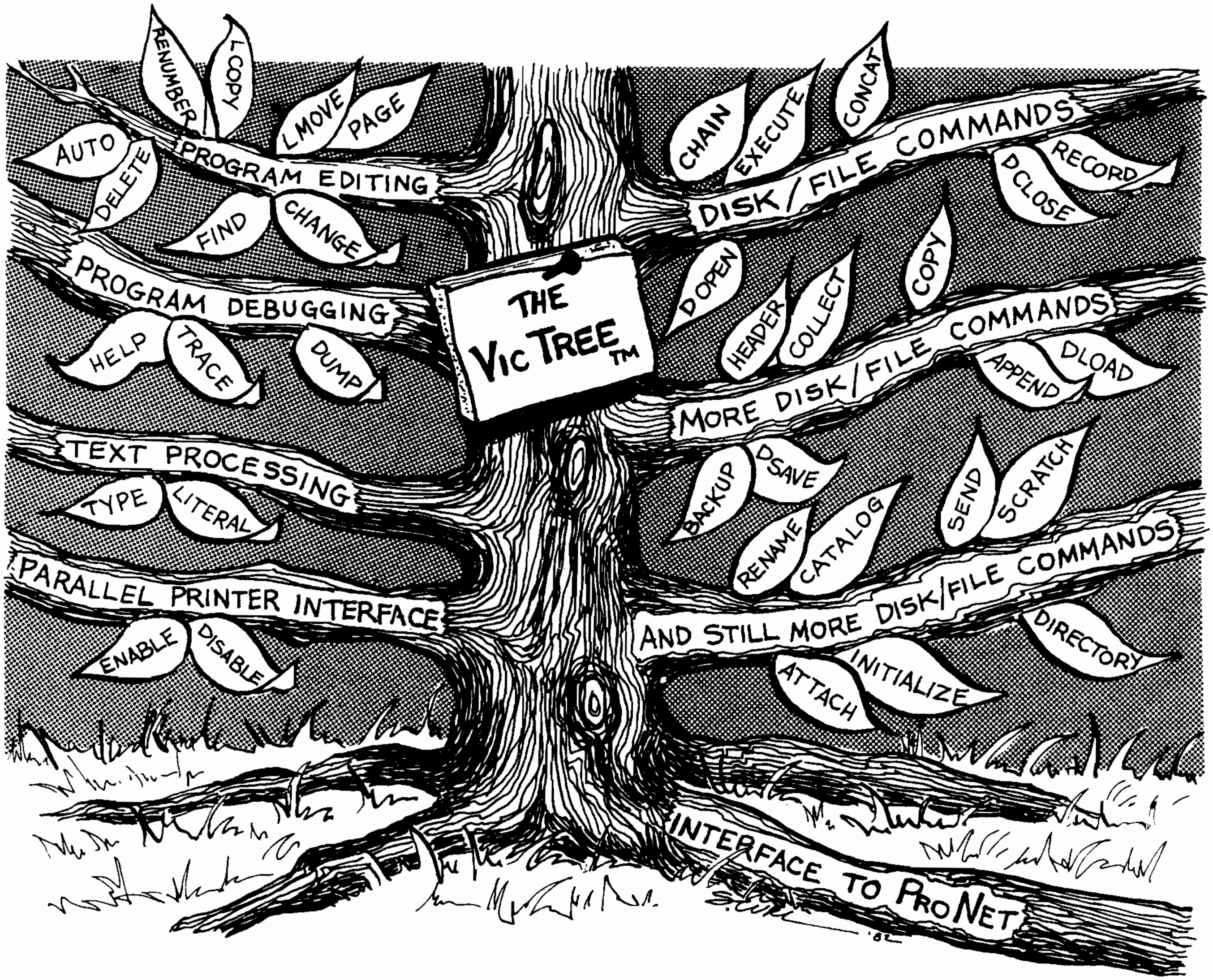
5547 satsuma avenue • north hollywood, california 91601 • 213/985-2922

artsc

where art and science work fo

VisiCalc is a registered trademark of VisiCorp. MAGICALC is a trademark of Artsci, Inc.

Skyles Electric Works Presents



The VicTree™

...Leaves your new Vic (or CBM 64) with 35 additional commands.
...Branches out to most BASIC 4.0 programs.
...Roots into most printers.

New from Skyles: the VicTree, a coordinated hardware and software package that allows your Vic to branch out in unbelievable directions and makes it easier than ever to do BASIC programming, debugging and to access your disk. And the new VicTree provides routines to interface the Vic to the powerful ProNet local network. 8kb of ROM — 4kb for the BASIC commands, 4kb for disk commands and interfacing to ProNet — plus 4kb of RAM for miscellaneous storage. Perfect not only for the new Vic but also for the Commodore 64. Unbelievably simple to use and to install, the VicTree gives you all the additional BASIC 4.0 commands to allow most BASIC 4.0 programs to work on your new Vic or CBM 64.

Now only $89.95...or **$99.95** complete with Centronics standard printer cable. (Cable alone **$19.95.**) Available now from your local dealer or order through your Visa or MasterCard toll free: (800) 227-9998 (California, Canada, Alaska, Hawaii: (415) 965-1735) or send check or money order directly to:

Skyles Electric Works

231E South Whisman Road
Mountain View, CA 94041
(415) 965-1735

MICRO™

January Highlights

Due to the nature of our features on simulations, applications, and math in this issue, many articles here will interest users of a wide range of systems. Even the programs written for a particular machine usually can be adapted to another. Apple, Commodore, Atari, VIC, OSI, and TSC XBASIC users will all find material of interest.

Simulations and Applications

Simulations save time and money in business, education, and research. For instance, a flight simulator program, available commercially for the Apple and other microcomputers, allows the user to control a plane through keyboard commands. Bigger computers, coupled with replicas of actual airplane control panels, allow student pilots to log a considerable amount of flying time without renting a plane or jeopardizing lives. Simulator programs are used in scientific research for testing mathematical models (e.g., of a predator-prey relationship) and in industry for determining how products will stand up to various kinds of stress.

"Discrete Event Simulation" by Anita and Bill Walker (p. 21) discusses techniques that can be applied to simulation programming with a microcomputer. The Walkers use as an example a program, written for the Apple II, that simulates the flow of customers in a bank line. "Rocket I," a program by David Eagle (p. 31), predicts the performance of a model rocket engine, given its specifications. "Sun and Moon" by Svend Ostrup (p. 35) is a high-resolution simulation of the apparent orbits of the sun and moon with respect to the earth. Phases of the moon are simulated, along with lunar and solar eclipses. In this month's editorial, Editor-in-Chief Bob Tripp describes his experience with simulations.

Accompanying our simulation feature are applications — a more familiar use of the computer. The computer's use in a non-computer activity may be as little as performing calculations or as much as actually operating a scientific experiment.

The second part of Jim Strasma's series on package programming using the CBM disk operating system ("It's All Relative, Part 2," page 52) will be of particular interest to the business user. Engineering applications are included in Andrew Cornwall's "Microcomputer Design of Transistor Amplifiers" (p. 59), and "Microcomputers in a College Teaching Laboratory" by Thor Olsen, et. al. (p. 38). In "Measurement of a 35mm Focal Plane Shutter" (p. 45), Mike Dougherty describes simple hardware and Atari software to test the accuracy and reliability of the shutter found in most single-lens reflex cameras. "Doing Time" by Jim Schreier (p. 28) shows how to do calculations involving time in TSC BASIC on 6809-based computers.

Mathematics Articles

Timothy Stryker's "Signed Binary Multiplication" (p. 76), Charles Muhleman's "Numerical Rounding" (p. 89), and P.P. Ong's "Methods to Evaluate Complex Roots" (p. 71), will be of interest to everyone who uses the computer to solve mathematical problems. "Apple Math Editor" by Robert Walker (p. 78) is a sophisticated program written in Apple Pascal that provides convenient display, editing, and printing of mathematical formulas. "Using Long Integers" by David Oshel (p. 86) describes the implementation of a bullet-proof string conversion for Pascal 1.1 long integers with implied decimal points.

Color Computer Data Sheet

Color Computer programmers will want to keep the data sheet (p. 109) handy. Here, on one easy-to-read sheet, is essential information on character codes, memory locations, and hardware interfacing.

Columns

System-specific information rounds out this month's magazine. Paul Swanson's "From Here to Atari" (p. 19) discusses reference books every Atari programmer will want to keep on hand. Loren Wright's "PET Vet" (p. 69) offers more observations on the new Commodore 64 and some how-to information on transferring programs from one Commodore machine to another. "CoCo Bits" by John Steiner (p. 92) provides news relating to the Color Computer, lists several programming books, and discusses the set-up for a high-resolution graphics display. Tim Osborn, in "Apple Slices" (p. 64), presents a program, ALTERNATE INDEX, that expands the capabilities of BINARY-SEARCH, a program discussed in his previous column.

MICRO™

NO. 56 JANUARY 1983

MICRO™

Advancing Computer Knowledge

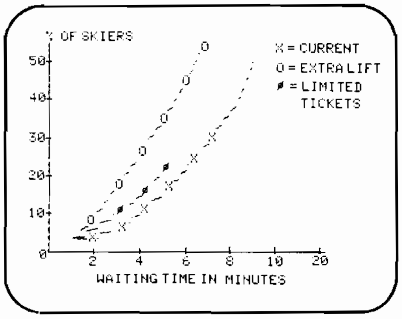
SIMULATIONS/APPLICATIONS

TUTORIALS

MATH

DEPARTMENTS

About the Cover



The skier on our cover this month flies gracefully down a snowy Waterville Valley slope. His face reflects the exhilaration every skier feels while out in the sun and crisp air.

See our editorial for a discussion on queuing — something many downhillers experience before they hit the slopes.

Cover photo by:

Joan Eaton
Waterville Valley Photo
Waterville Valley, NH

MICRO is published monthly by:
MICRO INK, Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional mailing offices
USPS Publication Number: 483470
ISSN: 0271-9002

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to
MICRO INK
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

Subscription Rates	*Per Year*
U.S.	$24.00
	2 yr. / $42.00
Foreign surface mail	$27.00
Air mail:	
Europe	$42.00
Mexico, Central America, Middle East, North Africa, Central Africa	$48.00
South America, South Africa, Far East, Australasia, New Zealand	$72.00

MICRO™ Editorial

A GASP, a Wheeze, and a 'Gotcha'

The typical MICRO reader owns a system and uses it primarily for serious work and program development. This issue focuses on ways to use your computer in real applications, mathematical problems, and in discrete event simulation. While the value and use of the real applications and the mathematical material should be obvious, the computer simulation will be a new topic for many readers.

"Discrete Event Simulation" (see the Walker article starting on page 21) is an exciting, broad area of computer application that often disguises itself as a rather dull, limited technique. This is due, I believe, to the examples presented: average waiting time in a bank queue, average waiting time in a doctor's office, and so forth. Don't let these particular examples mislead you. Computer simulation can be fun!

Years ago, I took a course in Discrete Computer Simulation. The basis of the course was a computer simulation package called "General Activity Simulation Program" (GASP). Written as a series of FORTRAN subroutines, this was configured to run on a PDP-10. The user would write a program that set up the operating environment parameters and called various support subroutines as required.

While many classmates simulated traffic lights and cafeterias as term projects, I chose to simulate the Waterville Valley Ski Area of New Hampshire. This month's cover symbolizes this study. Whenever I think of downhill skiing, two images come immediately to mind. First, there is the image of racing down the clean white slopes, passing through the picturesque trails, breathing the fresh air. Second, there is the image of the lift line, with the long wait, the dreary dirty snow underfoot, the cold of just standing and waiting. My simulation addressed methods of reducing the lift line wait by limiting the number of tickets sold each day, developing additional long trails, and adding another lift. Since the lift manager in those days was my cousin, I was able to get real information about the length of the lift ride, average time down the slope, number of customers, and so forth.

My first 'real' simulation was of a microprocessor. We needed to know if the processor could successfully handle eight operators simultaneously typing on individual keyboards. Unfortunately, the PDP-10 was not available. I located a PDP-9 and converted 'GASP' from the PDP-10 to 'Wheeze' on the PDP-9. The conversion was not difficult, and I think it could be easily converted to run in BASIC on almost any of the current micros.

Converting the program wasn't a problem, but running it became a nightmare. A simulation of this nature, where the event is the keystroke of one of eight operators, will be necessarily slow. The actual event might average one occurrence every 10 milliseconds or so (eight operators typing at twelve characters per second each), while the simulation processing might take one to three seconds per event, creating a 100- to 300-fold time expansion. A simulation of five minutes of typing could take between 500 and 1500 minutes to run! Since the PDP-9 was not being used for anything else, that should not have been a problem - but it was.

Everytime the program was run, it would work for a while, but would crash before completion. I noticed that the crashes seemed to occur at about 11:30 AM and 3:30 PM. A little investigation revealed that the machinists in the shop on the floor above quit for lunch at 11:30 and quit for the day at 3:30. That was the 'Gotcha' - a power surge from the machines being turned off. The simulation program worked perfectly - but only at night.

The results of the keyboard simulation showed that not only could the microprocessor keep up with the eight operators, it would be idle almost 80 percent of the time!

There are many interesting events that may be simulated. With your dedicated equipment, you can do significant simulations.

Robert M. Tripp

Robert M. Tripp
President/Editor-in-Chief

MICRO™

Updates and Microbes

Homespun Revision

John Beckett of Collegedale, TN, sent in this revision to "A Homespun 32K Color Computer" (53:91).

Do solder the chips together, rather than expecting hand-bent pins to make good contact. Model I users will be happy to comment in favor of *anything* to improve the communication between your CPU and memory.

It is best to put a ferrite bead around the wire connected to the 6883 chip, just before it reaches the 6883. Failing that, use a 33-ohm resistor. This is done in Tandy's 32K version, and is recommended by Motorola in their 6883 data sheet.

Later models of the PC board have a place on the PC board where you can connect the lead from the extra "bunk" of chips. It would be best to connect to that place, so as to avoid soldering directly to the 6883.

What's Where in the Apple Atlas Updates

The following subroutines have been relocated in the new (Autostart) ROMS:

Subroutine	Old Monitor Applesoft	New Autostart Applesoft
HGR2	F3D4	F3D8
HGR	F3DE	F3E2
HCLR	F3EE	F3F2
BKGND	F3F2	F3F4
HPOSN	F40D	F411
HPLOT	F453	F457
HLIN	F530	F53A

Microbes

The following change should be made in the review of Light-Pen in Reviews in Brief (53:97).

Under the minuses, the first sentence should read "The programs require a machine-language routine..." rather than "The programs use a machine-language routine...."

Let us know if you've updated an article or discovered a bug. Send a note to: Updates/Microbes, MICRO, P.O. Box 6502, Chelmsford, MA 01824.

MICRO

VIC Hi-Res Graphics Explained

by Nicholas J. Vrtis

This article demonstrates the use of VIC's 160-by-176 dot, high-resolution graphics. A sample BASIC program illustrates the necessary set-up.

VIC Graphics Demo
requires:

VIC-20 with 3K extra memory (may be modified for unexpanded VICs or for more memory)

The VIC manuals refer to the capability of high-resolution graphics. There is even a section in the *VIC-20 Programmer's Reference Guide* that shows how to do 64 by 64 bit graphics. Unfortunately, it is not obvious how it all works. The purpose of this article is to help shed some light on the subject of VIC graphics.

To understand high-resolution graphics you have to understand how programmable characters work. The VIC doesn't really have a "graphics" mode, but it does have two features that allow for graphics displays. The first and most important is that the contents of the pointer that normally points to the character ROM can be changed to point to RAM. The other is that the character size can be changed.

Before explaining how these combine to get graphics, I need to review quickly how characters are normally displayed on the screen. A more detailed explanation can be found in the *Programmer's Reference Guide* and in a number of articles on special characters for the VIC. Each byte in screen memory is used as an index into the character memory. It is actually character memory that tells the VIC which dots to turn on or off in the display. In normal mode a character is 8 rows of 8 dots per row. In expanded character size, a character is 16 rows of 8 dots per row. By telling the VIC chip that character memory is located in RAM, which dots are turned on or off can be controlled from a program. This is how special characters are created.

So how does this lead to graphics you ask? Good question! If I were to POKE the values 0 through 255 in the first 256 screen memory locations, all the possible characters would be displayed in order. Now let's take a moment to look at where the bits for each character come from. The first 8 by 8 square of dots (an @) comes from the first 8 bytes of character memory at the rate of 8 bits per byte. The second 8

Listing 1

```
100 PRINT"VIC GRAPHICS DEMO"
110 PRINT"BY: NICK VRTIS"
120 POKE1,PEEK(55)          :REM SAVE CURRENT END
130 POKE2,PEEK(56)
140 POKE55,0                :REM SET NEW END TO 4096
150 POKE56,4096/256
160 CLR                     :REM CLR BECAUSE END WAS CHANGED
170 CM=4096
180 RC=10
190 CR=22
200 NR=16
210 SF=RC*NR/2-1            :REM SET GRAPH SCALE FACTOR
220 REM ZERO FUTURE CHARACTER MEMORY
230 FOR X=CM TO CM+RC*CR*NR-1
240 POKE X,0 :NEXT
250 REM ALL 1'S TO UNUSED TO MAKE A BORDER
260 FOR X=CM+RC*CR*NR TO 7679
270 POKE X,255:NEXT
280 REM SET TO 8X16 CHARACTER SIZE
290 POKE 36867,PEEK(36867) OR 1
300 BY=PEEK(36879) AND 7   :REM GET CURRENT BACKGROUND
310 FOR X=0 TO RC*CR-1     :REM INDEXES TO SCREEN MEMORY
320 POKE 7680+X,X
330 POKE 38400+X,BY
340 NEXT
350 FOR X=RC*CR TO 511     :REM FILL REST WITH UNUSEL CHARACTER
360 POKE7680+X,RC*CR
370 POKE38400+X,BY
380 NEXT
390 REM TELL VIC NUMBER OF CHARACTERS/ROW
400 POKE 36866,(PEEK(36866) AND 128) + CR
410 REM CHANGE ADDRESS OF CHARACTER MEMORY
420 POKE 36869,(PEEK(36869) AND 240) + CM/1024 + 8
500 FOR X=0 TO CR*8-1
510 Y=INT(SF+SF*SIN(X/10)+1)
520 GOSUB 1000
530 NEXT
600 GETA$                   :REM WAIT FOR ANY KEY
610 IF A$="" THEN 600
620 POKE55,PEEK(1)          :REM RESTORE OLD EOM
630 POKE56,PEEK(2)
640 SYS(59829)              :REM RESET VIC CHIP
650 END
1000 YR=Y/NR
1010 CH=INT(X/8)+INT(YR)*CR
1020 RW=(YR-INT(YR))*NR
1030 BY=CM+CH*NR+RW
1040 BI=7-(X-INT(X/8)*8)
1050 POKEBY,PEEK(BY) OR (2^BI)
1060 RETURN
```

by 8 square of dots comes from the second 8 bytes of character memory, etc. What is being displayed on the screen is not the 256 displayable VIC characters, but all the bits that are set in the 2048 bytes starting at the address defined as character memory! (256 indexes * 8 rows * 8 dots = 16384 dots = 2048 bytes * 8 bits).

With this knowledge I can calculate which bit in which byte is to be set to a one in order to turn on a selected dot. The following formulas are adapted from the *VIC Programmers Reference Guide*.

```
CHAR = INT(X/8) + INT(Y/NR) * CR
ROW = (Y/NR - INT(Y/NR)) * NR
BYTE = SM + CHAR * NR + ROW
BIT = 7 - (X - (INT(X/8) *8)
```

For these formulas, X represents ascending values to the right, and Y represents ascending values from the top down. CR is the number of characters per row, which we will discuss later, and NR is the number of rows of dots per character (8 for normal size characters).

There is only one more major observation to make. As everyone knows, the VIC screen is 22 characters by 23 rows, for a total of 506 characters being displayed at one time. How can I fill a screen of 506 characters with only 256 unique combinations? The trick is double-height characters. The double-height characters don't change the dot size displayed on the screen, so each "character" covers twice as much screen area.

To put things a little differently, the VIC screen is 176 dots wide by 184 dots high, for a total of 32384 dots (4048 bytes). The double-high characters provide for 32768 dots (256 indexes * 16 rows * 8 dots per), so obviously all the problems are taken care of, right? Wrong. The problem is the memory the VIC chip itself can address. As stated in the expansion modules, the VIC chip (as opposed to the VIC computer), can only address memory from 4096 to 8191 (hex $1000 to $1FFF). While this 4096 bytes is sufficient to hold a full set of double-high character memory, we still need to take the 512 bytes of screen memory from this same area.

We've discussed most of the information you use to do graphics on the VIC. There are a few minor technical details left and compromises concerning the amount of graphics and memory needed for BASIC. Character memory can start at one of four RAM locations: 4096, 5120, 6144, or 7168 (with a 12, 13, 14, or 15 in the last four bits of location 36869). Screen memory can be at any of eight RAM locations: 4096, 4608, 5120, 5632, 6144, 6656, 7168, or 7680 (bits 4-7 of location 36869 control which 1024 boundary, and bit 7 of location 36866 controls whether it is an even 1024 or 512 boundary). Character and screen memory are set independently, and can even occupy the same locations. In fact, for the maximum resolution graphics, they have to overlap some. If character memory is set to 4096, and screen memory to start at 7680 by:

```
POKE 36869,(PEEK(36869) AND
240) + 12
```

there are 3584 bytes available for

graphics. This allows for 22 columns by 10 rows, or 176 dots across by 160 dots down (each row is 16 dots high with double-height characters). It doesn't allow for room for a BASIC program on an upexpanded VIC. If you want room for a 1K BASIC program, move character memory to 5120, and keep screen memory at 7680 with the following:

```
POKE 36869,(PEEK(36869) AND 240)
 + 13
```

This allows for 2560 bytes of graphics data, and a default grid of 176 dots (22 characters) by 112 dots down (7 double-high characters) using 2464 bytes. That is not a very square area to graph in, so use the following to change the number of characters per row from 22 to 17:

```
POKE 36866,(PEEK(36866) AND 128)
 + 17
```

This gives 136 dots across by 144 dots down.

Note that all this discussion applies to unexpanded VICs, or VICs with only the 3K expander. There is one further complication for VICs with more than 8K. For these systems, the screen memory defaults to location 4096, and the BASIC program starts at 4608. In order to use graphics with these systems, the start of the BASIC program must be moved above the area used for the screen and character memory (i.e., above 8191). Under the right conditions, it can be done by the BASIC program that is running, but it is much simpler to do before loading the program.

The program included with this article is a sample of how to use high-resolution graphics with the VIC. If you look at it, you will find that most of the program is involved in setting things up, and that lines 500 to 530 are the ones that create the actual graph (a simple sine curve). The program as shown is for a VIC with the 3K expander. If you have an unexpanded VIC, change the following lines and remove all the REMs. This will give a 136 by 144 dot field.

```
150 POKE56,5120/256
170 CM = 5120
180 RC = 7
190 CR = 17
```

If you have the 8K expander you should remove statements 150 and 160, since the end of memory is above screen memory. You will also have to enter the following statements in direct mode before loading the BASIC program. These statements move screen memory to where it is on the standard VIC, and also set the start of the BASIC program to just above screen memory. This lets us use memory from 4096-7680 as character memory.

```
POKE 36866,150:POKE 36869,240:
  POKE 648,30
POKE 43,1: POKE 44,32
POKE 8192,0
CLR:NEW
```

You should press the CLR/HOME key to clear the screen after typing in the first line. This will tell BASIC you changed the screen location.

Lines 120-160 establish a new top of memory, which is below where the new character memory will be. The CLR makes sure BASIC doesn't use any of that memory. Lines 170-210 set up constants used later.

CM is the location of character memory
RC is the number of row characters
CR is the number of characters per row
NR is the number of dots per row character
SF is a scale factor to center the sine curve

Note that RC*NR is the number of dot rows, and CR*8 is the number of dots wide.

The loop at 230 initializes the character memory we will use for the graph to zeros, while the one starting at 260 initializes the rest of character memory to ones. Note that these loops initialize a lot of memory, so they take a few seconds to run.

Lines 310-340 POKE the numbers 0-219 (153 for unexpanded VICs) to screen memory, while lines 350-380 POKE an unused character into the rest of screen memory. Since the location of character memory has not been changed yet, you will get a demonstration of the VIC character set.

Lines 400 and 420 change the characters per row (only necessary for the unexpanded version), and move character memory to the RAM area we have set up previously. Since a one bit on displays the border color, and we have initialized all the unused character memory to ones, the screen will appear to shrink at this time.

Lines 500-530 plot the sine curve by calling the plot routine at lines 1000-1060. This routine was described earlier.

Lines 600 and 610 allow you to admire your work by waiting for a key to be pressed. Then lines 620 and 630 reset the top of memory back to their original values. The SYS(59829) resets the VIC chip to its normal default values.

Nick Vrtis is the Manager of Technical Support at Amway Corporation. You may contact him at 5863 Pinetree S.E., Kentwood, MI 49508.

MICRO

68000 Shift, Rotate, and Bit Manipulation Instructions

by Joe Hootman

Our series on 68000 instructions continues. Previous detailed tables appeared in September, November, and December.

The Shift and Rotate Instructions

The shift and rotate operations implemented in the 68000 are delineated in table 1. The distinction between shift and rotate is that shift does not preserve the bits as they leave the register except in the carry bit. Rotate, on the other hand, cycles the bits around the register to the most significant bit position or to the least significant bit position, depending on whether rotate is a rotate right or a rotate left.

Another interesting point is that registers can be shifted/rotated any number of bits by denoting the bit count in a preassigned data register. Memory can be shifted/rotated only one bit at a time. This suggests there might be a time savings if the data in the memory were brought from memory to a register location before shifting. This is true; and if three or more shifts are to be done on data in memory, it should be put into a register for shifting.

Table of Definitions of Opword Formats for Shift and Rotate

i/r = 0 Immediate shift count. The shift count is specified by this to range between 1 and 8 shifts. Zero in the count register results in a shift of 8. The rest of the bits denote a shift of 1 to 7.

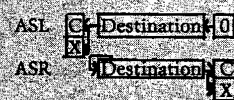
i/r = 1 Register shift count. The shift count is contained in the data register denoted.

dr = 0 Shift Right

dr = 1 Shift Left

Table 1: **Shift and Rotate Instructions**

Mnemonic	Data Size/CCR	Name	Comments
ASL ASR	8, 16, 32 CCR X N Z V C * * * * *	Arithmetic Shift, Left and Right	The arithmetic shift will shift the bits in the destination by a predetermined number of times and the carry bit receives the last bit shifted out. The shift count can be specified either by immediate data or by the contents of a data register.



Opword Format

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Count/ Register	dr	Size	i/r	0	0	Register

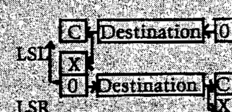
Register Shifts

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
1	1	1	0	0	0	0	dr	1	1	Effective Address Mode	Register

Memory Shifts

The following effective addressing modes cannot be used in the memory rotate: 1, 2, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Name	Comments
LSL LSR	8, 16, 32 CCR X N Z V C * * * 0 *	Logical Shift Left and Right	The logical shift will shift the bits in the destination a predetermined number of times and the carry bit receives the last bit shifted out. The shift count can be specified either by the immediate data or the contents of a data register.



Opword Format

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Count/ Register	dr	Size	i/r	0	1	Register

Register Shifts

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
1	1	1	0	0	0	1	dr	1	1	Effective Address Mode	Register

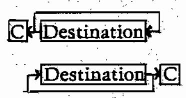
Memory Shifts

The following effective addressing modes cannot be used in the memory rotate: 1, 2, 11, 12, 13, 14.*

(continued)

Table 1 *(continued)*

Mnemonic	Data Size/CCR	Name	Comments
ROL ROR	8, 16, 32 CCR X N Z V C - * * 0 *	Rotate without extenstion	The destination is rotated as indicated below. The extension bit is not included in the rotation. The number of times the rotate is performed can be specified immediately or by data in a register.



Opword Format

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Count/ Register	dr	Size	i/r	1	1	Register

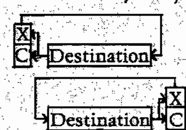
Register Rotate

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
1	1	1	0	0	1	1	dr	1	1	Effective Address Mode	Register

Memory Rotate

The following effective addressing modes cannot be used in the memory rotate: 1, 2, 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Name	Comments
ROXL ROXR	8, 16, 32 CCR X N Z V C * * * 0 *	Rotate with extension	The bits in the destination will be rotated as specified below and the extended bit is included in the rotation. The number of times the rotation is to be performed is specified immediately or by data in a register.



Opword Format

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Count/ Register	dr	Size	i/r	1	0	Register

Register Rotate

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
1	1	1	0	0	1	0	dr	1	1	Effective Address Mode	Register

Memory Rotate

The following effective addressing modes cannot be used in the memory rotate: 1, 2, 11, 12, 13, 14.*

***Table 2:* Bit Manipulation Instructions**

Mnemonic	Data Size/CCR	Name	Comments
BCHG	8, 32 CCR X N Z V C - - * - -	Test a Bit and Change	A bit in a particular bit position can be tested and its state reflected in the Z bit of the CCR. The state of the bit is changed in the destination.

Opword Format

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	Register	1	0	1	Effective Address Mode	Register

Register

The bit number that is to be tested and changed is contained in a data register defined by a register number in the register field. The effective address specifies the destination.

The following effective addressing modes cannot be used: 2, 10, 11, 12, 13, 14.*

(continued)

Size field

- 00 - Byte operation
- 01 - Word operation
- 10 - Long word operation

Register field — Specifies data register to be shifted.

Bit Manipulation Instructions

Table 2 describes the bit testing and manipulation instructions which exist in the 68000. Bit manipulation instructions are used to test, test and set, bit test and change, or test and reset a bit. The result of a test is found in the Z bit of the CCR. The bit to be tested is specified by a bit number in a specified data register or by a bit number in the extension word. Notice that BCHG, BCLR, and BSET all test bits and then may change the state of the bit. These instructions do not apply directly to the address register.

Contact Professor Hootman at the University of North Dakota, Dept. of Electrical Engineering, University Station, Grand Forks, ND 58202.

Table 2 *(continued)*

Mnemonic	Data Size/CCR	Name	Comments

Opword Format

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
0	0	0	0	1	0	0	0	0	1	Effective Address Mode	Register
0	0	0	0	0	0	0	0	Bit number			

Immediate

The bit number that is to be tested and changed is contained in the immediate word following the opword. The effective address specifies the destination location. The following effective address modes cannot be used: 2, 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Name	Comments
BCLR	8, 32 CCR X N Z V C - - * - -	Test a Bit and Clear	The state of a particular bit in the destination is tested and its state reflected in the Z bit of the CCR. The particular bit is cleared in the destination.

Opword Format

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	Register	1	1	0	Effective Address Mode	Register

Register

The bit number that is contained in the data register defines the bit to be tested and cleared. The effective address specifies the destination. The following effective address modes cannot be used: 2, 10, 11, 12, 13, 14.*

Opword Format

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
0	0	0	0	1	0	0	0	1	0	Effective Address Mode	Register
Bit number											

Immediate

The effective address specifies the destination location. The following effective addresses cannot be used: 2, 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Name	Comments
BSET	8, 32 CCR X N Z V C - - * - -	Test a Bit and Set	The bit in the destination is tested and the state of the bit is reflected in the Z bit of the CCR. The specified bit is set in the destination.

Opword Format

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	Register	1	1	1	Effective Address Mode	Register

Register

The bit number contained in the data register specified by the register field is the bit to be tested.

The effective address specifies the destination location. The following effective address modes cannot be used: 2, 10, 11, 12, 13, 14.*

Opword Format

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
0	0	0	0	1	0	0	0	1	1	Effective Address Mode	Register
Bit number											

The effective address specifies the destination location and the bit number specifies the bit to be tested. The following effective address modes cannot be used: 2, 10, 11, 12, 13, 14.*

(continued)

Table 2 *(continued)*

Mnemonic	Data Size/CCR	Function	Comments
BTST	8, 32 CCR X N Z V C - - * - -	Test a Bit	The state of a bit in the destination is tested and the state of the bit is reflected in the Z bit.

Opword Format

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	Register	1	0	0	Effective Address Mode	Register

Register

The bit number is specified in the data register specified by the register field. The effective address specifies the destination location. The following effective address modes cannot be used: 2, 12, 13, 14.*

Opword Format

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
0	0	0	0	1	0	0	0	0	0	Effective Address Mode	Register
Bit number											

The effective address specifies the destination location and the bit number specifies the bit location. The following effective address modes cannot be used: 2, 12, 13, 14.*

*The addressing modes will be covered in future issues.

MICRO™

MICRO™

From Here To Atari

By Paul S. Swanson

This month's column covers technical literature available for Atari computers. The term technical, of course, means different things depending on your programming level of expertise.

For non-programmers who want to learn, there is one general book on the market that provides a good introduction to programming. This book, *Karel the Robot* by Richard E. Pattis (Wiley, 1981), was intended as an introduction to Pascal, but is well written as an introduction to almost any computer language.

For those who already know something about programming and own an Atari computer and a BASIC Language Cartridge, there are two good sources. One is *Atari BASIC* by Albrecht, Finkel, and Brown (Wiley, 1979), which is written to teach you how to program in BASIC. The *BASIC Reference Manual* from Atari outlines the available BASIC commands and has some handy reference tables. One table, labeled "Memory Locations," provides vectors, shadow locations, and hardware locations that you can PEEK or POKE for special actions. These two books come with the BASIC cartridge in a programmer's kit from Atari.

Your next step in acquiring literature from Atari is a reference book called *De Re Atari*, which was written by several Atari staff members and is available at most computer stores that carry the Atari. In addition to the features I listed above, this book also explains how Atari BASIC uses memory, then does the same for the resident operating system and disk operating system. Other topics include vertical blank interrupts, cassette operations, television artifacts, and the GTIA chip (if you aren't familiar with this chip you are in for a pleasant surprise).

In the middle of digesting *De Re Atari*, you will probably become interested in machine language. I know of no machine-language book available from Atari, but almost any book on the 6502 should work. I use *Programming the 6502* by Rodnay Zaks (Sybex, 1978). Another is Lance Leventhal's *6502 Assembly Language Programming*, (Osborne/McGraw-Hill, 1979).

There are other books available for Atari computers at the level of *De Re Atari*. *Your Atari Computer* by Poole, McNiff, and Cook (Osborne/McGraw-Hill, 1982), is a good example. It covers certain features of the Atari and its peripherals in more depth and is therefore a good supplement to *De Re Atari*.

For even more advanced programmers, Atari publishes the *Technical User Notes*, a combination of the *Operating System Manual* and the *Hardware Manual*. These are strictly reference books — don't look for long explanations. They are concise descriptions of all the different system features. BASIC, for example, is not even mentioned. The few examples are in machine language.

I have all the above-mentioned books within arm's reach of my Atari computer, as well as a few reference books concerning integrated circuits (I also experiment with my own electronic circuitry). The *Hardware Manual* contains all the wiring diagrams of the Atari computer (both the 400 and the 800), invaluable for interfacing.

Talking to Other Computers

One question from a reader reminded me of a recent project I embarked upon. The question concerned moving data from an Apple to an Atari. I recently set up communication between my Atari and a 6502-based system I built from scratch. This allowed me to develop the 6502's operating system using an assembler on the Atari. I communicated to and from the Atari through game controller ports 3 and 4. Using one plug connected at game controller 4, I set up a serial communication through half of one of the two PIA bytes. The PIA can be directly accessed and programmed through hardware registers. A register named PBCTL (for Port B control) at location $D303 (decimal 54019) allows you to set up game controller ports 3 and 4 as either input, output, or any combination on the eight joystick pins. From BASIC, POKE 54019,56, then POKE 54017 with a bit map of which pins you want as input and which you want as output. For input, use a zero bit; for output use a one. Next, POKE 54019,60. The joystick pins on game controllers 3 and 4 are now set up the way the bit map specified.

The eight joystick pins are the top pins on each game controller jack excluding the far right pin on each. The Port B byte includes the eight pins on jacks 3 and 4. The lowest order bit is the top leftmost pin on jack 3; the highest order bit is the fourth pin from the left on the top row of jack 4.

If you are working on transmitting data from the Apple to the Atari, I have another suggestion that will help things run faster. The Apple clock runs at 1 MHz, but the Atari clock runs at about 1.79 MHz; therefore, the Atari can process information about 75% faster than the Apple. If you have conversions, use the Atari. To get the full advantage of the Atari's faster clock, write a zero to location $D40E and another zero to location $D400. Location $D400 enables and disables the different types of direct memory access available. Location $D40E enables and disables the non-maskable interrupts (except SYSTEM RESET). You will have no screen display after that. Write to $D40E first because $D400 is shadowed during the vertical blank interrupt. The zero in $D40E will stop the shadowing and allow access directly to the hardware register. It also allows an easier method for undoing all that disabling. When you have written those two zeros out, run the conversion routine. When the conversion is done, just write a $40 to location $D40E to re-enable the vertical blank interrupt. The shadowing will re-enable the DMA by rewriting the original contents of location $D400.

In Conclusion

Future columns will be based on letters from readers. If you have any suggested topics or questions concerning the Atari, write me at 97 Jackson Street, Cambridge, MA 02140. MICRO

Discrete Event Simulation in Pascal

by Anita and Bill Walker

This article explains some of the techniques used in simulating real-world situations on the computer. An example program involving a queue is presented.

Program Bank
requires:
Pascal

Introduction

What is computer simulation? Intuitively, we suggest that it is the act of causing a computer to imitate a real-world situation so that you can analyze the effects of changing portions of the environment in that situation. Ideally this process will be sufficiently accurate to allow you to make management decisions without performing experiments to test the idea. The Apollo moon-landing trips were extensively simulated before the first mission, providing valuable insight into possible difficulties without risking loss of hardware or personnel.

One method of providing answers to hypothetical questions in a simulation is to observe the situation in question for a specific interval and take notes. A less time-consuming method is to program the computer to emulate the situation and answer the questions for you. Although this process rarely gives exact answers, it is possible to use the computer to gain valuable insight. This tutorial discusses some of the techniques used in discrete event simulation. (Do not expect the results to be the gospel truth.) It also suggests a few tools that might be useful to the simulator and provides an example.

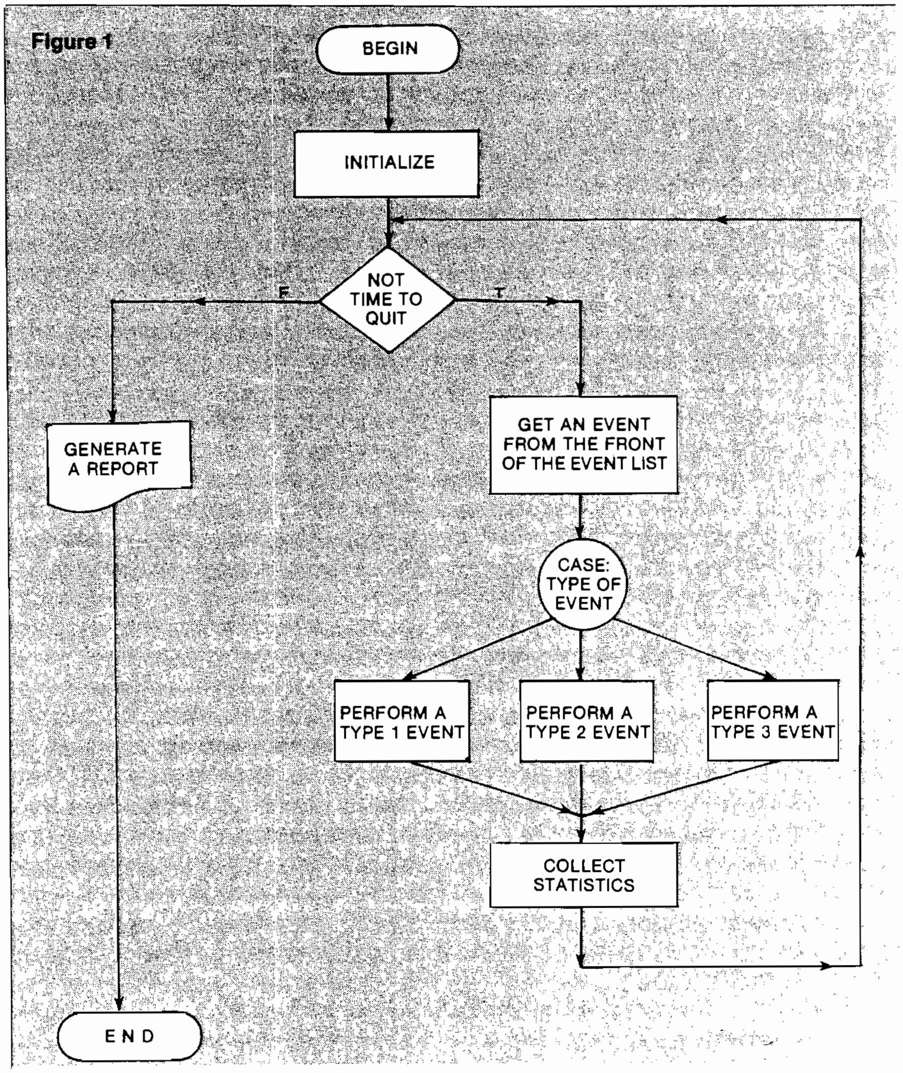
How Do Discrete Simulations Work?

A simulation that emulates a physical system can be programmed for the computer. Within this system are various events that occur at particular points in time and affect the physical system in predictable ways, often generating additional events. To emulate a physical system, you must first specify a list of possible events. This is a crucial part of the process and will be discussed later in more detail.

As the events are generated, they are placed in a line called the event list, which is maintained sequentially. Think of the event list as being sorted according to increasing values of time. Suppose the list contains three different types of events. It is ordered according to time, not type, so it may be possible for several events of one type to occur before an event of another type occurs. Figure 1 is a flowchart of a typical control program and, in effect, oversees the simulation process.

After initialization, select an event from the event list and perform the ac-

Figure 1



tions dictated by the type of event until the simulation is over. Keep notes on the effect of the actions. The actions you perform often generate more events that are added to the event list by order of increasing time. After the run is completed print the statistical results and conclude the program run.

We discuss each portion of this flowchart in detail and develop utility procedures that allow you to actually write a program based on figure 1.

Tools for Simulation

To formulate a sample program you must first develop several tools to use in the discrete simulation program. This section explains two such tool packages: 1. managing the event list, keeping the events in increasing order of time, getting the next event from the list, and properly adding new events in the list; and 2. discussing various random (actually pseudo-random) number generators, emphasizing the generation of numbers on 8-bit machines (with the possibility of expanding the generators to run on other machines).

The Event-List Manager

The event-list manager consists of several procedures designed to handle the event list. Remember, the event list consists of a series of events kept in chronological order (for this example). If a new event is created with a scheduled time of occurrence, it must be placed properly among the other events already in the list. The important thing to remember is that when the new event is inserted into the list, the list still must be in chronological order.

You may insert events anywhere in the event list but, typically, events are removed from the list one at a time and from one end of the list only. In other words, when you reach the point for an event to occur, that event is removed from the front of the list and a procedure is executed to carry out the action dictated by that event.

In this discussion, we maintain a linked list, using pointers, which contains the events of a simulation in chronological order. Two procedures are necessary: SCHEDULE, which inserts an event into its proper place in the event list, and GETEVENT, which removes the next (front) event from the event list. To facilitate writing the procedures, we define an event as a Pascal record structure with three fields. One of the fields is linked to the next event, and the other two fields contain the time the event occurred and the type of event. Although it is not necessary to do so, we use several global variables to implement our event-list manager. One of these is HEAD, which is a pointer that returns NIL if the event list contains no events.

The procedure SCHEDULE (see listing 1) schedules events properly into the event list. Procedure GETEVENT is a procedure that returns the type of event and its scheduled time of occurrence at the front of the list as well as deletes that event from the event list. We make use of the event type in the control program. It is possible to rewrite these procedures as functions, but we prefer the method chosen.

Random Number Generators

The function RND (see listing 1) generates a sequence of pseudo-random numbers on most 8-bit machines that is distributed almost uniformly between the values of 0 and 1. Pseudo random means the numbers are not truly random numbers, but depend in some manner on each other. If you start with the same value for SEED each time, you will get the same sequence of numbers. However, *distribution* of the numbers is more important than true randomness. Uniform distribution means that a number is likely to fall equally anywhere between 0 and 1. The function (unfortunately) produces a numerical sequence that repeats itself every 64 numbers. If you are using a longer word length for your machine, you can arrange the function to produce longer sequences before it repeats itself.

The function RNEXP is used to generate numbers with an exponential distribution whose average is "U". This distribution is often more useful in simulation than the uniform distribution of RND. Since the function RNEXP depends on the function RND, the exponential distribution generator will also repeat after a short sequence. If you have a random-number generator available for your machine, the function RNEXP can be used directly.

Other Tools

Other necessary tools include procedures to accomplish an action demanded by an event to keep statistics on the state of the system after each event, and to report the final results of the simulation. These tools are exceedingly sensitive to the nature of the actual system being simulated and, as a rule, cannot be generalized.

How to Write a Simulation Program

Although each simulation program is different, it is possible to make a few general statements. First, an event is something that causes the state of a system (a set of data) to change. Note that we are talking about discrete simulations only. For example, consider the case of a line of people waiting for service at a bank teller's window. At any given point in time the system state is completely described by counting the number of people in the line. The state changes when someone joins the line or when someone departs. You might describe the state of the system by saying "There are ... people in the line." Proper events cause the state to change. In this case those events are identified as ARRIVE and DEPART.

It is important to specify the proper events for a simulation when writing a useful program. If an event is hidden or overlooked, you may get meaningless results. If too many events are specified, programming may be awkward or even impossible.

After you have chosen the events for a simulation you must identify the characteristics (parameters) you want to measure. Parameters are part of the state of the system and should be things that are affected by the events. A typical parameter for the bank window example would be a measurement of the average length of the customer line. This length is affected by two events only — ARRIVE and DEPART — and the length does not change until one of these events takes place.

In the flowchart of figure 1 the procedure COLLECT STATISTICS keeps running totals of the state variables (parameters) measured. Consequently, design of this procedure depends upon what those state variables are.

Another portion of the program called an event procedure accomplishes the action(s) demanded by a particular event. In the above example you might use such a procedure to add a person to the end of a line (corresponding to the event ARRIVE) or remove a person from the front of the line (corresponding to the event DEPART).

Finally, the report section should present a summary of the statistics that you collect with the procedure COLLECT STATISTICS.

(Text continued on page 25)

Program Bank

```
program bank (input,output);
uses transcend,applestuff;
 { this is an APPLE statement.  Other computers
    can probably omit this statement entirely }
const starttime = 0;
      endtime = 14400; { 4 hours in seconds }
      uariv = 50.0; { average interarrival time }
      userv = 40.0; { average service time }
type ptr = ↑event;
     event = record
              eventtype : char;
              eventtime : integer;
              link : ptr;
             end;
var  kindofevent : char;
     departcount,arrivecount,queuelength,
      maxqueuelength, time, oldtime,
      oldqueuelength,eventcounter     : integer;
      timequeuelength : real;
     head,p,oldptr,q : ptr;
function rnd (list: integer): real;
 var x : real;
 begin
   { This function should be replaced by
      an appropriate function for your system.
      It's purpose is to generate uniformly
      distributed random numbers between 0 and 1 }
   x := random;
   { if we want the first list, access the generator
      a second time to try to remove some bias }
      if list = 1 then x := random;
   { be sure not to generate 0 as a number, since
      the function RNEXP below would blow up }
   if x = 0 then x := x + 1;
   rnd := x / 32767;
 end;
function rnexp(list:integer; u : real) : real;
 begin
  { This is a pseudo-random number generator for
     generating exponentially distributed pseudo-
     random numbers with an average value of u.
     It depends greatly on function RND above, and
     if RND repeats its sequence of numbers fairly
     often, so will this function }
    { this function selects random numbers from
       two different lists which are generated by
       RND above }
  rnexp := (-u)×ln(rnd(list)));
 end;
procedure getevent (var typeofevent : char; var newtime : integer);
 { this procedure gets the next event from the
    event list }
 begin
  if head <> nil then
   begin
    typeofevent := head↑.eventtype;
    newtime := head↑.eventtime;
    head := head↑.link;
   end;
 end;
procedure schedule (typeofevent : char; newtime : integer);
 { this procedure enters a new event into the event list }
 var quit : boolean;
 begin
  { first we create the new event and initialize it }
  new(q);
  q↑.link := nil;
  q↑.eventtime := newtime;
  q↑.eventtype := typeofevent;
  { now we place the new event in its proper place
     in the event list }
  if head = nil then head := q
  else
   begin
    if (newtime < head↑.eventtime) then
     begin
      q↑.link := head;
      head := q;
     end
```

Program Bank *(continued)*

```
     else
      begin
       p := head;
       quit := false;
       while ((p↑.eventtime <= newtime) and
          (quit = false)) do
         begin
          if p↑.link = nil then
           begin
            p↑.link := q;
            quit := true;
           end
          else
           begin
            oldptr := p;
            p := p↑.link;
           end;
         end; { of the while }
       if quit <> true then
        begin
         oldptr↑.link := q;
         q↑.link := p;
        end;
      end;
    end;
  end;{ of schedule }
procedure initialize;
 var newtime : integer;
 begin
  departcount := 0;
  arrivecount := 0;
  queuelength := 0;
  maxqueuelength := 0;
  time := 0;
  oldtime := 0;
  oldqueuelength := 0;
  eventcounter := 0;
  time := 0;
  timequeuelength := 0.0;
  head := nil;
  p := nil;
  oldptr := nil;
  q := nil;
  { schedule the initial event }
  newtime := time + round (rnexp(1,uariv));
  schedule ('a',newtime);
  { randomize the random number generator--
     this is how to do it on the APPLE }
  randomize;
 end;
procedure statistics;
 { this collects the statistics }
 begin
  { if you want LOTS of output, you can
      remove the comment symbols around the
      following: }
   { if kindofevent = 'a' then write ('arrival ')
  else write ('departure ');
  writeln ( ' at ',time, ' seconds'); }
  { update the event counters }
  eventcounter := eventcounter + 1;
  if kindofevent = 'a' then arrivecount := arrivecount + 1
  else departcount := departcount + 1;
   { update the queuelength }
  if maxqueuelength < queuelength then
     maxqueuelength := queuelength;
   { update the time averaged queuelength }
  timequeuelength := timequeuelength +
       (time - oldtime ) × oldqueuelength;
   { update the accumulation stuff }
  oldqueuelength := queuelength;
  oldtime := time;
 end; { of statistics }
procedure makereport;
 { this procedure reports all of the results }
```

(Continued on next page)

Program Bank *(continued)*

```
begin
 writeln (chr(7),chr(7));
 writeln ;
 writeln (' the simulation was run for ',
   (endtime-starttime)/60 :10:2, ' minutes ');
 writeln;
 writeln ('there were ',eventcounter,' events with ');
 writeln ('  ',arrivecount,' arrivals, and ');
 writeln ('  ',departcount, ' departures.');
 writeln;
 writeln (' the maximum length of the queue was ');
 writeln ('      ',maxqueuelength,' people, and the');
 writeln (' time-averaged queue length was ');
 writeln ('     ',timequeuelength/(endtime-starttime):10:2,' people');
 writeln;
end;
  { in a more elaborate program, the following two
     procedures may actually handle a queue, instead
     of simply updating a counter }
procedure addtoqueue;
 { this adds people to the waiting line }
 begin
  queuelength := queuelength + 1;
 end;
procedure popqueue;
 { this deletes people from the waiting line }
 begin
  if queuelength > 0 then queuelength := queuelength - 1;
 end;
{ the following procedures are the 'event procedures ' }
procedure service;
 { this procedure, while not properly an 'event', provides
    service to a customer if it is needed.  It is called
    by the events ARRIVE and DEPART }
 var newtime : integer;
```

Program Bank *(continued)*

```
 begin
  if queuelength <> 0 then
   begin
    schedule ('d',newtime);
   end;
 end;
procedure arrive;
 var newtime : integer;
 begin
  addtoqueue;
  newtime := round(rnexp(1,uariv)) + time;
  schedule ('a',newtime);
  if queuelength = 1 then service;
 end;
procedure depart;
 begin
  popqueue;
  service;
 end;
begin { main program }
 initialize;
 while time <= endtime do
  begin
   getevent (kindofevent,time);
   case kindofevent of
    'a' : arrive;
    'd' : depart;
   end;
   statistics;
  end;
 makereport;
end.
```

In the following section we simulate the classic bank line problem. The example is instructive and provides an opportunity to apply several of the concepts we have discussed.

The Example

In this example we make several assumptions: 1. the waiting line is a queue (no one butts in, no one leaves early), service takes place at the front of the line only, and new arrivals join the end; 2. arrivals occur with an exponential distribution interarrival time of 50 seconds; and 3. the time it takes for the teller to serve a customer is also exponentially distributed with an average time of 40 seconds.

The next step is to measure the state variables (parameters); in this case, the average length of the line and the longest length of the line.

Listing 1 provides a simulation of the bank-line situation. The program can be adapted to most single-server queue systems, although it probably will be necessary to change the characteristics of the pseudo-random number generators to suit other physical situations. The exponential distributions used here are not unrealistic for this situation. The listing is written in UCSD Pascal on an Apple II. (*Note:* there are many languages available for simulation programs. We chose Pascal as the most commonly available language suitable to the hobbyist.) With other versions of Pascal you could take advantage of the dispose function of standard Pascal. The program runs to completion in about three minutes for a four-hour simulation, with a typical event count exceeding 500.

Figure 2

Trial Number	Average Queue Length	Maximum Queue Length	Number of Arrivals	Number of Departures
1	2.78938	13	281	280
2	4.14340	17	301	301
3	2.81646	15	273	271
4	2.34562	10	277	271
5	3.67563	13	259	258
6	1.74604	8	264	261
7	3.17368	15	282	276
8	2.41681	12	259	258
9	6.13910	22	314	308
10	3.58667	19	261	260
11	2.83958	10	279	275
12	1.71257	7	261	260
13	4.14527	15	294	293
14	3.32611	14	285	282
15	7.24937	19	331	315
16	1.93847	9	264	262
17	3.52042	12	312	309
18	7.25556	22	305	304
19	3.92014	10	304	304
20	4.29167	16	297	296
21	3.45194	12	283	277
22	4.97257	16	302	299
23	2.65333	10	282	279
24	3.04583	10	302	299
25	7.77750	20	322	316
26	8.19340	25	303	293
27	2.57618	12	286	285
28	3.48049	15	293	291
29	3.67924	14	292	284
30	12.89966	26	333	322
31	4.99736	21	303	299
32	2.65465	9	264	263
33	2.34090	8	301	296
34	2.53410	10	274	270
35	3.77437	13	293	288
36	2.14736	13	278	277
37	2.81681	12	292	290
38	7.78674	31	317	294
39	2.68153	11	273	268
40	3.36868	14	278	277
Total Average	**4.02**	**31**	**Average Maximum Length**	**14.50**

How to Make Use of the Simulation Program

Run the program many times so the random-number generators provide different sequences of events each time. (This is usually accomplished simply by changing the SEED of the function.) Each run of the program provides a number that represents the maximum length of the queue during that run. If you run the program ten times, you have ten different numbers. An average of these numbers gives you meaningful data about what to expect from the actual physical situation. The results of a single simulation run, however, are unlikely to provide much information. A powerful theorem in mathematics, Central Limit Theorem, allows you to draw some meaningful conclusions by examining the averages of several program runs. The usual procedure is to form a confidence interval for the parameter that you choose to measure. We have presented a summary for the example problem in figure 2.

Mathematical Analysis

The programmer should be aware that the results obtained from the discrete simulation process are at most good approximations to the results obtained in the real situation itself. It is gratifying, however, to solve the simulation problem using analytic methods and to discover just how accurate these approximations are. Analytic solutions are not always obtainable and hence the need for simulations.

In the following pages we use mathematics to investigate the bank-line simulation. You should become familiar with the notations and terminology used. *Queuing time* is the total time that a single customer is in the system. This time begins when the customer arrives at the end of the line and stops when he leaves the line after being served. *Waiting time* is the time between arrival and service. Use the following notations:

T_a = average lapse of time between the arrivals of two consecutive customers

λ = average arrival rate of the customers, given by the formula
$\lambda = 1 / T_a$

T_s = average time needed to serve one customer

μ = average service rate for each customer, given by
$\mu = 1 / T_s$

I = intensity of customer traffic, given by any of the following:
$I = T_s / T_a = \lambda T_s = \lambda / \mu$

ϱ = the amount of time a single bank teller needs to serve a customer (usually a decimal or a percentage)

The following averages are useful when certain distributions and probabilities are difficult to obtain:

L_q = the average number of customers in the system (length of the queue)

L_w = the average number of customers in the waiting line

T_q = the average queuing time

T_w = the average waiting time

In this example the values which determine T_a and T_s are *exponentially distributed*. (Consider the exponential curve $y = e^x$. Each service time (t) achieved in the problem lies on the exponential curve. Hence, every t is shown as $t = e^x$ for some number x (x real). The typical time needed to serve one customer (T_s) is obtained by averaging a large number of individual service times of less than 40 seconds with a small number of service times of more than 40 seconds. Thus, the average service time (T_s = 40 seconds) is represented by the horizontal line t = 40. The average time between the arrival of two consecutive customers (T_a = 50 seconds) is represented by the line t = 50.

Given all the above, you can evaluate the desired quantities and compare them to your computer results. You can see immediately that T_a = 50 seconds and T_s = 40 seconds (given quantities) lead to the results:

$\lambda = 1/50$, which means that on the average one person arrives every 50 seconds, and

$\mu = 1/40$, which means that on the average one person is served every 40 seconds, and

I = 40/50.

If I is less than 1, that indicates the bank teller is serving faster than the customers are arriving. A traffic intensity greater than 1 indicates the teller is serving slower than the customers are arriving.

We define ϱ in the following way: a long period of time is represented by T_L, the number of customers arriving at the system by $n = T_L / T_a$, and the total service time by nT_s. Therefore, the time that the bank teller is busy is
$\varrho = nT_s / T_L = nT_s / nT_a = T_s / T_a$
Here $\varrho = 40/50 = 4/5$. The teller is busy 4/5 of the time (T_L).

The formula for the quantity L_q is
$L_q = (\lambda^2 b_2) / 2(1-\varrho)) + \varrho$ where
$b_n = n!\, T_s^n$

If you evaluate this expression you learn that $L_q = 4.0$ for this simulation. You also have $L_w = L_q - \varrho$, which calculates as $L_w = 3.2$. Similarly, $T_q = [\lambda b_2 / 2(1-\varrho)] + b_1$ evaluates to T_q = 200 seconds. T_w is given by $T_w = \lambda b_2 / 2(1-\varrho)$, and evaluates to 160 seconds. Note that the average queuing time is equal to the average waiting time plus the average time needed to serve one customer.

Figure 3 compares the results of the analytic investigation with the numbers obtained from the computer simulation. They seem to agree with each other in a reasonable fashion.

***Figure 3:* Analytic versus Computed Results**

AVERAGE QUEUE LENGTH INDICATED BY LISTING ONE	4.02
AVERAGE QUEUE LENGTH COMPUTED ANALYTICALLY	4.00
PERCENTAGE DIFFERENCE	½%

Conclusions

Although the science of simulation is rather complicated, we are able to draw some meaningful results from discrete-event simulation techniques. Hopefully, you will study these techniques further. If you do not, perhaps this article will serve to give you a speaking acquaintance with some of the procedures involved.

Suggested Reading

1. Fishman, G.S., "Concepts and Methods in Discrete Event Digital Simulation," John Wiley & Sons, New York, NY, 1973.
2. Gorney, Len, "Queuing Theory," *BYTE*, Vol. 4, #4 & #5, 1979.
3. Jensen, K. and Wirth, N., *Pascal User Manual and Report*, Springer-Verlag, New York, NY, 1974.
4. Kiviat, P.J., Villaneuva, R., and H. Markowitz, "The SIMSCRIPT II Programming Language," Prentice Hall, Englewood Cliffs, NJ, 1969.
5. Knuth, D.E., "The Art of Computer Programming, Vol 2: Semi-Numerical Algorithms," Addison-Wesley, Reading, MA, 1969.
6. Lewis, T.G. and Smith, M.Z., *Applying Data Structures*, Houghton Mifflin Co., Boston, MA, 1976.
7. Payne, James A., "Introduction to Simulation: Programming Techniques and Methods of Analysis," unpublished notes, copyright by J.A. Payne, 1979.
8. Schriber, T.J., "Simulation Using GPSS," John Wiley & Sons, New York, NY, 1974.
9. Wirth, Niklaus, *Algorithms + Data Structures = Programs*, Prentice-hall, Englewood Cliffs, NJ, 1976.

Bill Walker is an assistant professor of electrical engineering and computer science at the University of Oklahoma. He has a B.S. from West Texas State University and an M.S. and Ph.D. in mathematics from Texas Tech University. Anita is a teaching assistant and Karcher Fellow at the University of Oklahoma. She has a B.A. in German and a B.S. in mathematics from SMU, and an M.A. in mathematics from the University of Oklahoma. She is currently pursuing her Ph.D. in mathematics. You may contact the Walkers at Box 2806, Norman, OK 73070.

MICRO

Doing Time on the 6809

by Jim Schreier

Calculating time is simple, but requires special attention when manipulated by a BASIC program. Here are two ways to add time using TSC's XBASIC.

Doing Time
requires:
BASIC with string functions

Pennies automatically add up to dollars, but seconds refuse to add up to minutes. If Thomas Jefferson had planned our way of telling time, the following programs would have been unnecessary. Telling time is confused just enough to need special handling in your BASIC programs.

Adding seconds, minutes, and hours may be done with string manipulations (see program A), or by using a simple formula (program B). The formula approach is faster and applies to almost any BASIC. The string approach uses TSC XBASIC's INSTR command, which searches for a substring within the main string. As such, program A would be limited to more advanced BASICs.

The object of each approach is to add similar time units, subtract the next higher full unit, leave the remainder, and increment the next highest full unit. So 91 seconds would be reported as 1 minute and 31 seconds.

Each approach is presented as a usable program. You may adapt the program to work as a subroutine, or keep it as a handy time adder. I have found the programs useful in adding the lengths of video disk movies and multi-record stereo sets. (If *The Godfather* runs 171 minutes and *The Godfather II* runs 200 mintues, dare I try to watch both in one evening?)

Program A

Although manipulating strings to add time may be the long way home, it does demonstrate the "scenic route." The idea is to locate the decimal point once the total number of seconds have been divided by the constant 60 (line 120). If no decimal point occurs (tested in line 130), the program prints out the results and concludes. Line 140 uses the INSTR (IN STRing) command to locate the position of the decimal point, allowing the necessary string maneuvering (lines 150-160). The results are printed as hours and minutes and the program concludes.

Program B

The formula approach is less complex. Hours, minutes, and seconds must be entered in strict order. To enter 91 seconds, use "0,0,91". This program is more extensive than the one used for program A. It reports the total entered times as seconds, minutes, hours, and days.

Line 250 is a representative example for the calculations. The total seconds, when divided by the constant 60, gives the number of minutes. When the number of minutes are multiplied by 60 and subtracted from the total seconds, the remaining seconds become available. The newly calculated minutes are then added to the total minutes and the process is repeated to calculate hours and days.

Each program used control "L" — CHR$(12) — to clear the CRT and home up the cursor. This should be adjusted to meet your requirements. Both programs set some variables and constants to integer by adding a percent sign (A%). If your BASIC does not support integers, leave the percent signs out of the listings.

Program A

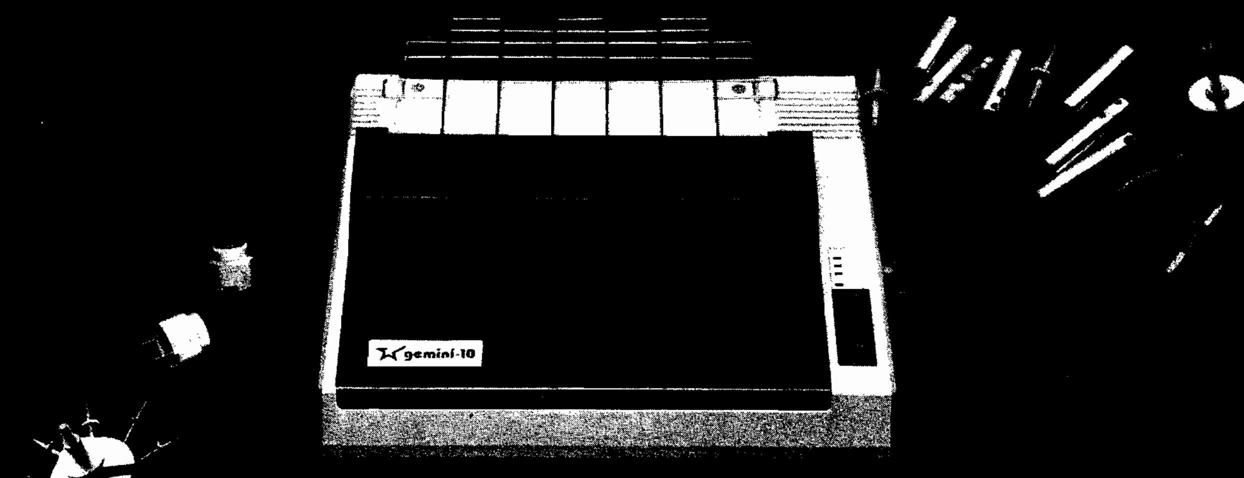
```
10 REM TIMESTR.BAS (Time String)
20 PRINT CHR$(12):PRINT
30 H%=60:W% =1:P$=''.''
40 REM Obtain input
50 INPUT ''How many items to add'',A%
60 FOR X%=1 TO A%
70 PRINT ''Enter item'';X%;
80 INPUT B
90 T=T+B
100 NEXT X%
110 REM Caculate number of hours and minutes
120 H$=STR$(T/H%)
130 IF T/H%=INT(T/H%) THEN 180
140 I%=INSTR(W%,H$,P$)
150 MN$=RIGHT$(H$,(LEN(H$)-I%)+W%)
160 H$=LEFT$(H$,I%-W%)
170 MN=INT(VAL(MN$)*H%+.5)
180 REM Print out results
190 IF T < H% THEN H$=''Zero''
200 PRINT:PRINT
210 PRINT ''Total Time: '';H$;
'' Hours and'';MN;''Minutes''
220 END
```

Jim Schreier has been a computer enthusiast since 1977. His articles have appeared in a number of magazines, and he has lectured about computers throughout the western United States. Contact Mr. Schreier in Phoenix, AZ 85040.

Program B

```
10 REM TIMEFORM.BAS (Time Formula)
20 REM Copyright (c) 1982 by Jim Schreier
30 REM This Basic program caculates time from hours, minutes and seconds
40 REM Clear screen and home up cursor is Control L. Set to your terminal.
50 CL$=CHR$(12)
60 REM Set program constants
70 ME%=100:C%=60:C1%=24
80 PRINT CL$
90 PRINT TAB(26);''TIME CACULATIONS''
100 PRINT:PRINT
110 INPUT ''Please enter the number of items'',NI%
120 IF NI% < 1 OR NI% > ME% THEN 130 ELSE 170
130 PRINT
140 IF NI% < 1 THEN PRINT '' > Entry out of range. Lower limit is 1 ...'':GOTO 160
150 IF NI% > ME% THEN PRINT '' > Entry out of range. Upper limit is'';ME%;''...''
160 PRINT:GOTO 110
170 REM Obtain input
180 PRINT:PRINT
190 FOR A%=1 TO NI%
200 INPUT ''Enter Hours, Minutes and Seconds, (H,M,S)'',H%,M%,S%
210 TH%=TH%+H%:TM%=TM%+M%:TS%=TS%+S%
220 NEXT A%
230 REM Caculate seconds into minutes and seconds
240 IF TS%=0 OR TS% < C%-1 THEN 260
250 B1%=TS%/C%:TS%=TS%-(B1%*C%):TM%=TM%+B1%
260 REM Caculate minutes into hours and minutes
270 IF TM%=0 OR TM% < C%-1 THEN 290
280 B2%=TM%/C%:TM%=TM%-(B2%*C%):TH%=TH%+B2%
290 REM Caculate hours into days and hours
300 IF TH%=0 OR TH% < C1%-1 THEN 320
310 B3%=TH%/C1%:TH%=TH%-(B3%*C1%):TH%=TH%+B3%
320 REM Report time as Days, Hours, Minutes and Seconds
330 PRINT CL$:PRINT
340 PRINT TAB(5);''DAYS'';TAB(25);''HOURS'';TAB(45);''MINUTES'';TAB(65);''SECONDS''
350 FOR X%=1 TO 67:PRINT TAB(5);''-'';:NEXT X%
360 PRINT
370 PRINT TAB(5);B3%;TAB(25);TH%;TAB(45);TM%;TAB(65);TS%
380 END
```

MICRO



GEMINI– FOR PRINTER VALUE THAT'S OUT OF THIS WORLD

Over thirty years of down-to-earth experience as a precision parts manufacturer has enabled Star to produce the Gemini series of dot matrix printers—a stellar combination of printer quality, flexibility, and reliability. And for a list price of nearly 25% less than the best selling competitor.

The Gemini 10 has a 10″ carriage and the Gemini 15 a 15½″ carriage. Plus, the Gemini 15 has the added capability of a bottom paper feed. In both models, Gemini quality means a print speed of 100 cps, high-resolution bit image and block graphics, and extra fast forms feed.

Gemini's flexibility is embodied in its diverse specialized printing capabilities such as super/sub script, underlining, back-spacing, double strike mode and emphasized print mode. Another extraordinary standard feature is a 2.3K buffer. An additional 4K is optional. That's twice the memory of leading, comparable printers. And Gemini is compatible with most software packages that support the leading printers.

Gemini reliability is more than just a promise. It's as concrete as a 180 day warranty (90 days for ribbon and print head), a mean time between failure rate of 5 million lines, a print head life of over 100 million characters, and a 100% duty cycle that allows the Gemini to print continuously. Plus, prompt, nationwide service is readily available.

So if you're looking for an incredibly high-quality, low-cost printer that's out of this world, look to the manufacturer with its feet on the ground—Star and the Gemini 10, Gemini 15 dot matrix printers.

star®
micronics • inc

MAKING A NAME FOR OURSELVES

1120 Empire Central Place, Suite 216, Dallas, TX 75247
For more information, please call Bob Hazzard, Vice President, at (214) 631-8560.

Listing 1

```
10 REM * Program ''ROCKET 1''
20 REM * Copyright (C) 1982
30 REM *
40 REM * Determines flight performance of model rockets
50 REM *
60 REM * Altitude at burnout in meters
70 REM * Velocity at burnout in meters/second
80 REM * Coast time and Total flight time in seconds
90 REM * Maximum altitude in meters
100 REM *
110 G0 = 9.80665 : R0 = 1.22557 : LN = 100
120 DEF FNA(X) =  (1-2.2556913 E-5×X) ± 4.256116
130 DEF FNB(X) =  .5×(SQR(ABS(X))+X/SQR(ABS(X)))
140 REM *
150 CL$ =  CHR$(11) + CHR$(24) : REM Clear Screen
155 PRINT CL$ : PRINT TAB(5); ''Program Rocket 1'': PRINT
160 PRINT : INPUT ''Launch site altitude (Meters) '';H1
170 PRINT : INPUT ''Launch site temperature (Deg F) '';K1
190 PRINT : INPUT ''Thrust duration (Seconds) '';T1
200 PRINT : INPUT ''Total impulse (Newton-seconds) '';I1
210 PRINT : INPUT ''Initial mass (Grams) '';M1
220 PRINT : INPUT ''Propellant mass (Grams) '';M2
230 PRINT : INPUT ''Frontal diameter (mm) '';G1
240 PRINT : INPUT ''Drag coefficient '';G2
250 REM *
260 REM * Convert mass to kilograms and diameter to square meters
270 M1 = .001 × M1 : M2 = .001 × M2 : G1 = PI × G1 × G1 / 4E6
280 REM *
290 REM * Compensate for launch site altitude and temperature
300 R1 = R0 × FNA(H1) / (1 + (K1 - 59) / 518.67)
310 REM *
320 REM * Determine analytic solution
330 F1 = I1 / T1 : M3 = (M1 - M2 / 2) : K2 = .5 × R1 × G1 × G2
340 A = M3 × G0 : B = T1 × FNB(K2 × (F1 - A)) / M3 : C = EXP(B)
350 D = EXP(-B) : E = .5 × (C+D) : F = (C-D) / (C+D)
360 X1 = (M3 / K2) × LOG(E) : V1 = F × FNB((F1-A) / K2) : M3 = M1 - M2
370 A = M3 × G0 : T2 = FNB(M3 / (K2 × G0)) × ATN(V1 × FNB(K2 / A))
380 X2 = (M3 / (2 × K2)) × LOG(K2 × V1 × V1 / A + 1)
390 T3 = T1 + T2 : X3 = X1 + X2
400 REM *
410 REM * Print results
420 PRINT CL$ : PRINT : PRINT TAB(5); ''Burnout altitude (Meters) '';TAB(50);    X1
430 PRINT : PRINT TAB(5); ''Burnout velocity (Meters/second) '';TAB(50); V1
440 PRINT : PRINT TAB(5); ''Coast time (Seconds) '';TAB(50); T2
450 PRINT : PRINT TAB(5); ''Total flight time (Seconds) '';TAB(50); T3
460 PRINT : PRINT TAB(5); ''Maximum altitude (Meters) '';TAB(50); X3
470 REM *
480 REM * Request another selection
490 PRINT : INPUT ''Another selection (Y/N) '';A$ : IF A$ = ''N''THEN 530
500 PRINT : INPUT ''Another launch site (Y/N)'';A$ : IF A$ = ''Y''THEN 160
510 PRINT : INPUT ''Another rocket engine (Y/N) '';A$ : IF A$ = ''Y''THEN 190
520 PRINT : INPUT ''Different mass or drag (Y/N) '';A$ : IF A$ = ''Y''THEN 210
530 PRINT CL$ : END
```

The user responds with "Y" to compute the flight performance of a model rocket with different mass or drag characteristics.

If another selection is made, the program will again execute the prompts necessary for the new selection. If the user answers "YES" to the prompt "ANOTHER SELECTION" but does not actually make a different selection, the program will stop after cycling through all the selection questions.

Program Output

ROCKET1 outputs the model rocket altitude performance in units of the metric system. The burnout altitude and the maximum altitude are printed in meters and the burnout velocity is printed in meters per second. Coast time and total flight time are printed in seconds.

For users who want to see other variables used in the software, "R1" is the launch site density in kilograms per cubic meters. The variable "X2" is the coast altitude increment in meters and "K2" is the variable ½ϱCdA in the units of kilograms per meter. "F1" is the average thrust of the model rocket engine in newtons.

Technical Discussion

ROCKET1 first converts the lift-off and propellant masses to kilograms and determines the cross-sectional area of the model rocket in square meters. The atmospheric density at the launch site is then computed as a function of the launch site altitude and temperature.

The burnout altitude and velocity are computed with the following equations:

$$Xbo = (m/k) \ln[\cosh(td/m \sqrt{k(F-mg)})]$$

$$Vbo = \sqrt{(F-mg)/k}\, \tanh[td/m \sqrt{k(F-mg)}]$$

where,

- m = average mass = lift-off mass — (propellant mass/2)
- k = ½ϱCdA
- ϱ = atmospheric density
- Cd = drag coefficient
- A = cross-sectional area
- F = average thrust = total impluse/thrust duration
- td = thrust duration

The altitude gained during the coast flight and the coast time are determined using the next set of equations:

$$Xc = (m/2k) \ln(kVbo^2/mg + 1)$$

$$tc = (m/kg)\, \text{atan}(Vbo \sqrt{k/mg})$$

where,

- m = burnout mass = lift-off mass — propellant mass
- g = acceleration of gravity

The maximum altitude and total flight time are given by these equations:

$$X = Xbo + Xc$$

$$T = td + tc$$

References

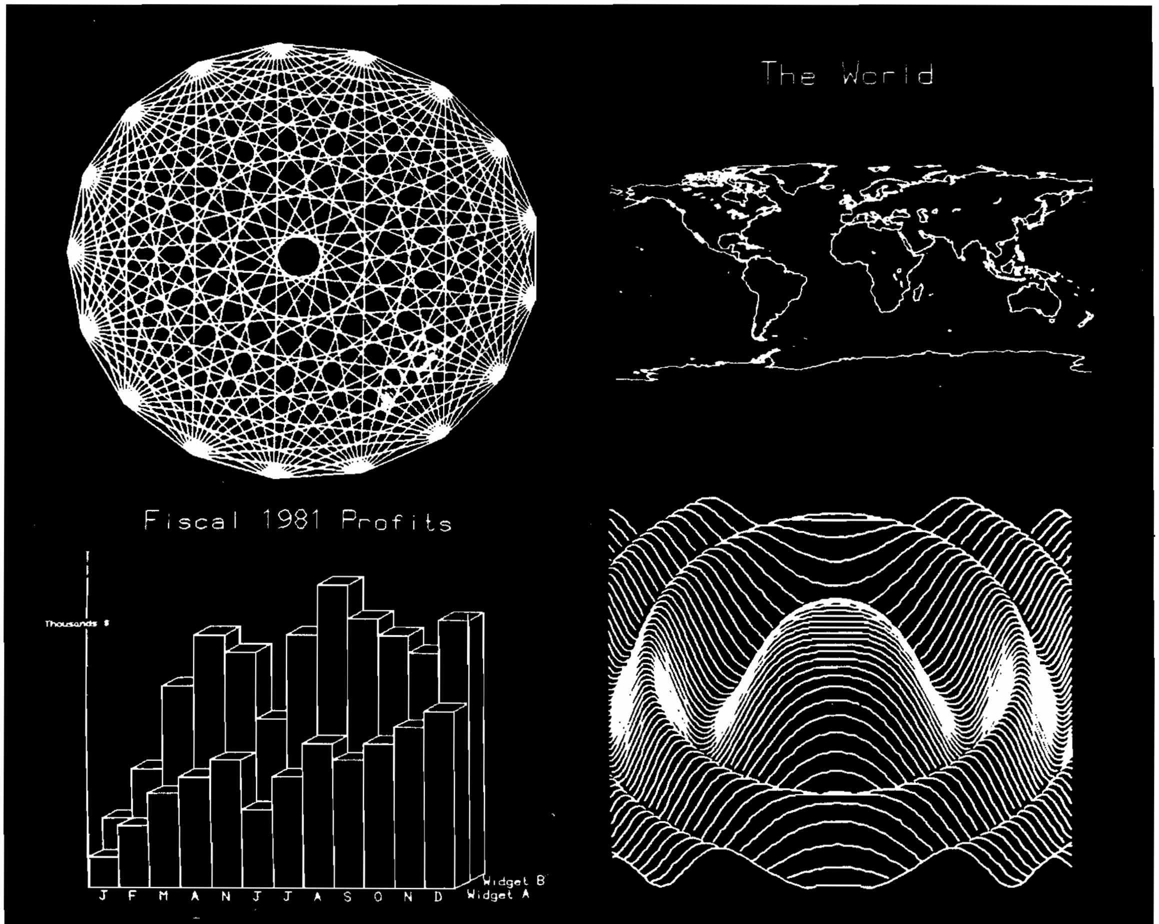
1. G.K. Mandell, G.J. Caporaso, and W.P. Bengen, "Topics in Advanced Model Rocketry," MIT Press, 1971.
2. "Altitude Prediction Charts," Estes Industries Technical Report TR-10, 1971.
3. "Aerodynamic Drag of Model Rockets," Estes Industries Technical Report TR-11, 1970.
4. D. Malewicki, "Model Rocket Altitude Performance," Centuri Engineering Company Technical Information Report TIR-100, 1968.

David Eagle is an aerospace engineer with an undergraduate and graduate degree from the University of Michigan. He presently works at Lear-Siegler, Inc., in Grand Rapids, MI, on projects which involve the most fuel-efficient way to fly airplanes. You may contact Mr. Eagle at 3759 76th St. SW, Byron Center, MI 49315

MICRO

ANNOUNCING ElectroScreen™ the Superior Alternative to the Traditional Alphanumeric Terminals

only $595



The ElectroScreen™ Intelligent Graphics Board Features:

Graphics
- 512 x 480 resolution bit-mapped display
- Interleaved memory access — fast, snow-free updates

Intelligence
- 6809 on-board mpu
- 6K on-board firmware
- STD syntax high level graphics command set
- Removes host graphics software burden
- Flexible text and graphics integration
- Multiple character sizes
- User programs can be run on-board

Terminal
- Terminal emulation on power-up
- 83 characters by 48 lines display
- Easy switching among user-defined character sets
- Fast hardware scrolling

Additional Features
- SS-50C and SS-64 compatible board
- Board communicates with host through parallel latches
- Composite and TTL level video output
- 8 channel 8 bit A/D converter
- Board occupies 4 address bytes

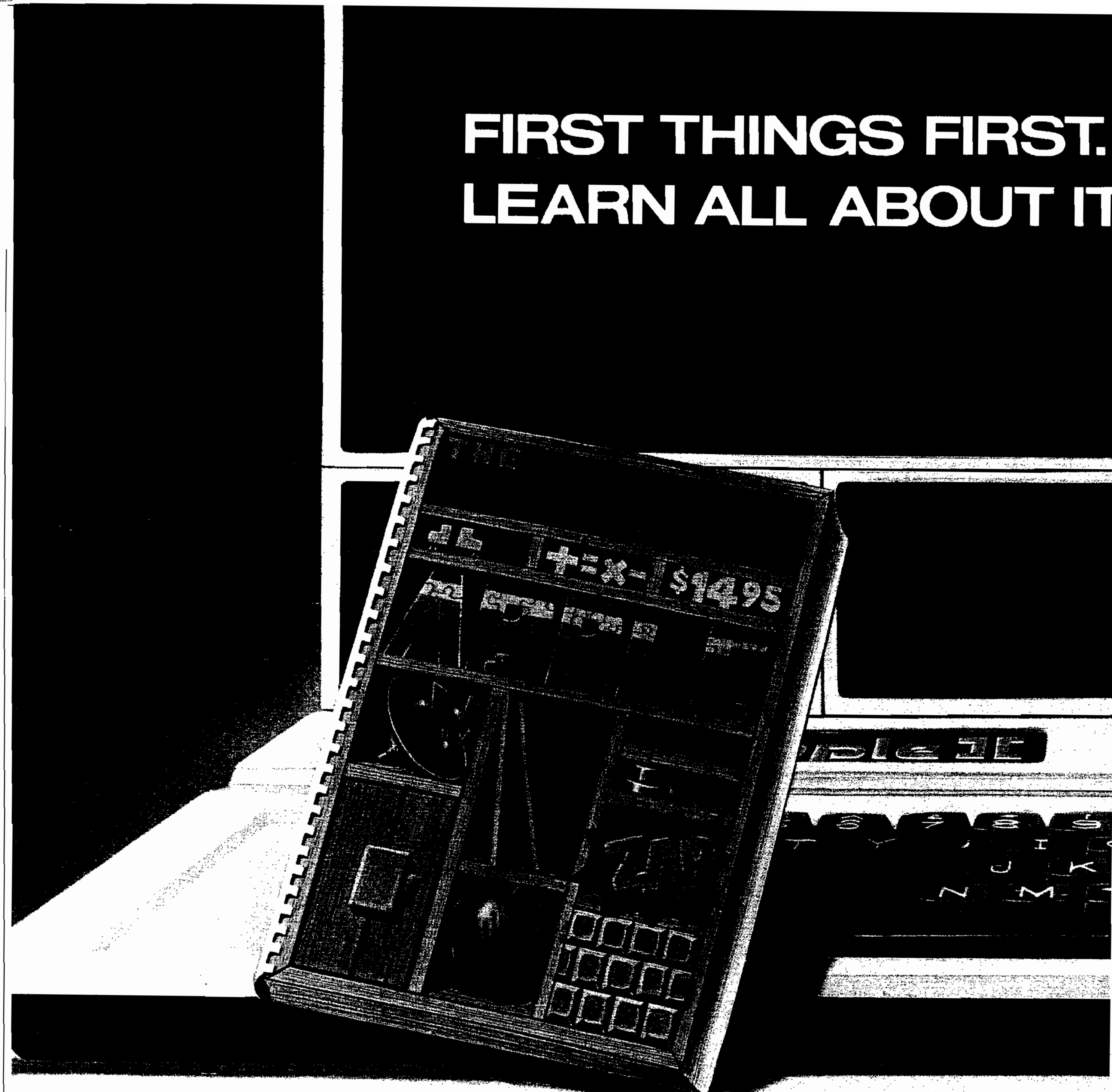
See your dealer today!

The ElectroScreen manual is available for $10, credited toward purchase of the board.

The ElectroScreen has a 90 day warranty from purchase date.

Dealers, please contact us for our special introductory package.

Privac Inc (703)671-3900
3711 S. George Mason Dr., Falls Church, Va. 22041



When you don't know the first thing about your new Apple II* you need a friendly, cheerful, easy going teacher at your side. And the ELEMENTARY APPLE is just that kind of book.

It sweeps away the confusion—explains your Apple in everyday language—shows you how to hook it up, how to use the keyboard and work on the screen.

Gently and carefully it gives you an understanding of all the things your Apple can do. And then, it even shows how easy it is for anyone to write a simple program—provides common sense answers about graphics, utility programs, and the how and why of word processors, business programs and hardware like printers.

Yes, there's a lot of information. But, not one chapter one word is dull or difficult to follow or complicated. Prove yourself. Visit your computer store. Open the ELEMENTAF APPLE. Read a page of the introduction, then flip it open anywhere and read a paragraph or so. You'll find it's as understandable, as helpful and as marvelous as we say.

If you, or a member of your family, is an Apple begin this is the book you need. It'll teach you everything you wa know, in the way you want to learn.

Only $14.95. At computer and book stores, or:

DATAMOST INC

9748 Cozycroft Ave., Chatsworth, CA 91311. (213) 709-120

VISA/MASTERCARD accepted. $2.00 shipping/handling charge.
(California residents add 6½% sales tax)
*Apple II is a trademark of Apple Computers, Inc.

Sun and Moon on the APPLE

by Svend Ostrup

This Applesoft program produces a high-resolution graphic simulation of the apparent orbiting of the sun and moon around Earth as well as of the phases of the moon. It also predicts solar and lunar eclipses.

Sun and Moon
requires:
48K Apple

The astronomy program listed simulates the apparent rotation of the sun and moon around Earth, as well as the phases of the moon day by day, beginning at a starting date chosen by the user. The locations of the ascending and descending nodes of the moon and of the moon's perigee are also shown. High-resolution page 1 shows all the above simultaneously with the current date and the moon's elongation. The program also predicts eclipses or the possibility of eclipses.

The material in this program is based on information provided by my son, Gert Ostrup, an amateur astronomer, and is published as an example of collaboration between novices in different fields — in this case astronomy and programming.

The program is straightforward and presents no difficulties. Some explanations, however, might be of value for the user to get full benefit from the program. Let us start looking at the firmament.

Type in the program and RUN. You will be informed that you can: 1. stop running the program at any date by pressing S (stop); 2. re-start by pressing SPACE; 3. get a prompt for a new starting date by pressing M; and 4. exit the program by pressing ESC. You should be aware that nothing will happen until the program has finished drawing the phase of the moon for the day in question, so some patience is required.

You will now be prompted to input a starting date (note the sequence: day, month, year). Try 28,12,1981. The program then draws a reference circle of dots spaced 10 degrees apart, marks the center (which is the location of Earth) and the 3 and 9 o'clock locations. The starting date soon appears and the sun and moon and three other objects (see below) are drawn inside the reference circle. Next the moon is drawn in its correct phase to the right of the reference circle. Meanwhile, the elongation of the moon (angular longitudinal distance between sun and moon in degrees) is printed. After a pause, the program goes on to the next day.

Let the program run briefly. When you reach 03-01 1982 press S and wait for the program to stop so you can take a closer look at the various features. The 3 o'clock position of the reference circle is the equinox (the point of intersection between the orbit of the sun — the ecliptica — and a plane through the equator of Earth). Thus, when the sun passes this point (21 March), the northern hemisphere enters the summer season, which will last until the sun passes the 9 o'clock position. The sun and moon both move counter-clockwise, the moon at about 13 degrees/day and the sun at about 1 degree/day. The cross you find between 12 and 1 o'clock is the perigee of the moon, which is the point closest to Earth in the moon's orbit. The perigee moves in the same direction as the sun and moon, but more slowly.

The shapes you see opposite each other near 11 and 5 o'clock are the nodes; i.e., the points of intersection between the orbit of the moon and the ecliptica. The ascending node is marked with a half cross that lacks the lower bar, while the descending node is marked with a cross that lacks the upper bar. The nodes move in opposite directions from the sun and moon at very slow speeds.

Now continue the program by pressing SPACE. When you reach 08-01 1982 and the drawing of the (full) moon is finished, the program stops, sounds the bell twice, and in flashing letters informs you of a lunar eclipse!

While a program that simulates the movements of the planets around the sun by using Keppler's equation might be quite accurate, this is not the case when you simulate the moon orbiting Earth. The reason is that the actual deviations from the Kepplerian method are not always negligible and might vary a few degrees. The user should be aware of this inherent inaccuracy that has an impact on the prediction of eclipses. Thus, when an eclipse warning (like the one you have just seen) is given, the actual eclipse might, in rare cases, take place the day prior to or the day after the date foreseen by the program.

I am aware that you could include the official predictions of eclipses, say for the past and next ten years, as a look-up table in the program. However, I have found it more interesting to relate the warnings to the locations of the sun, moon, and the nodes, as calculated and drawn by the program.

Eclipses can occur only when the sun and moon overlap (conjunction) as seen from Earth, or when they are exactly opposite from each other (opposition) as seen from Earth. Therefore, a prerequisite for the occurrence of a solar eclipse is that the longitude of the moon is equal to the longitude of the sun. A prerequisite for the occurrence of a lunar eclipse is that the difference between the solar and the lunar longitudes equals 180 degrees. In other words, the moon is overtaking the sun (or its opposition) on the day of an eclipse. This condition is investigated in line 2010 by looking at the sign of the sinus of the said difference. A sign change is required.

The said condition, however, is not sufficient for an eclipse to occur. (If it were we would have an eclipse every fortnight!) In the case of a lunar eclipse, the moon must pass through the shadow of Earth (not above or below it). Earth and moon must thus be in line, within certain limits. This happens only when the sun (and thus also the moon) are sufficiently close to one of the nodes. Therefore, conditions of eclipses are studied by investigating whether or not the sun is sufficiently close to one of the nodes at the moment when the sun and moon are in conjunction or opposition.

The location of the perigee is of interest when judging the extent of central solar eclipses. (Will they be total or annular?) Remember that the perigee is the point in the orbit of the moon closest to Earth. When the moon is close to the perigee its apparent size, as seen from Earth, is bigger than that of the sun, a prerequisite for a total lunar eclipse.

The date change takes place at midnight Greenwich mean time. To use local time, the following simple program change is required: if your time is behind Greenwich mean time (which is the case in the U.S.A.) by six hours, convert the hour difference to a decimal day difference (6/24 = 0.25 in this case) and add the figure to the constant 715953.5 in line 6930; i.e., change 715953.5 to 715953.75.

You may contact Mr. Ostrup at Lindevangsvej 12, DK 3460 Birkerod, Denmark.

Sun and Moon Listing

```
10 REM *   SUN & MOON          *
20 REM *   BY SVEND ØSTRUP  *
30 REM *   LINDEVANGSVEJ 12     *
40 REM *   3460 BIRKERØD   *
50 REM *   DENMARK            *
100 REM SET LOMEM AND/OR HIMEM IF NEEDED
105 REM Arrays X,Y and A contain plotting coordinates
110 DIM X(4,2): DIM Y(4,2): DIM A(4,2): DIM S(12)
120 GOTO 6000
199 REM Calculate andgle VV from coordinates XX,YY by ATN
200 IF XX = Ø AND YY > Ø THEN VV = PI : RETURN
210 IF XX = Ø AND YY < Ø THEN VV = 3 × PI / 2:RETURN
220 VV = ATN (YY / XX)
230 IF XX < Ø THEN VV = VV + PI
240 RETURN
290 REM Calculate Coordinates and Angle
300 V = V + Z × N:O = O + Z × H
310 MA = V - O:E1 = MA
320 EA = MA + E × SIN (E1)
330 IF ABS (E1 - EA) > .ØØØ5 THEN E1 = EA: GOTO 320
340 YY = SQR (1 - E × E) × SIN (EA)
350 XX = COS (EA) - E 360 RA = SQR (XX × XX + YY × YY) 370 GOSUB 200
380 A = VV + O
390 X = HU + K × RA × COS (A)
400 Y = ET - K × RA × SIN (A)
410 RETURN
499 REM Calculate plotting coordinates and longitude
500 O = O + Z × H
510 XØ = K × COS (O):YØ = K × SIN (O)
520 RETURN
598 REM Calculates and prints elongation, and draws
599 REM picture of moon, showing phase at current date
600 DA = A(1,1) - A(Ø,1)
605 IF DA > 2 × PI THEN DA = DA - 2 × PI: GOTO 605
610 IF DA < Ø THEN DA = DA + 2 × PI: GOTO 610
615 EL = DA
620 IF DA > PI THEN DA = DA - PI:W = 1: GOTO 640
630 W = Ø
640 VTAB 24: HTAB 29: PRINT " ELONG.:"
650 SV$ = RIGHT$ ( STR$ ( INT (EL × F + .5) + 1ØØØ),3)
660 HTAB 36: PRINT SV$;: HTAB 1
670 FOR I = S1 TO - S1 STEP - 1
680 RR = SQR (S2 - I × I)
690 IF W = Ø THEN HCOLOR= 3: GOTO 710
700 HCOLOR= Ø
710 HPLOT TF + RR × COS (DA),ET - I TO TF + RR,ET - I
720 IF W = Ø THEN HCOLOR= Ø: GOTO 740
730 HCOLOR= 3
740 HPLOT TF - RR,ET - I TO TF + RR × COS (DA),ET - I
750 NEXT
760 RETURN
950 HTAB 2Ø - LEN (Q$) / 2: PRINT Q$: PRINT : RETURN
999 REM Increment day number
1000 Z = Z + 1
1200 REM Calculate longitude and plotting coordinates
1201 REM for sun and moon, using subroutine 300
1202 REM SUN
1210 V = VS:N = NS:E = ES:K = KS:O = OS:H = HS
1220 GOSUB 300
1230 X(Ø,2) = X:Y(Ø,2) = Y:A(Ø,2) = A
1300 REM MOON
1310 V = VM:N = NM:E = EM:K = KM:O = OM:H = HM
1320 GOSUB 300
```

Sun and Moon Listing *(continued)*

```
1330 X(1,2) = X:Y(1,2) = Y:A(1,2) = A
1398 REM Calculate longitude and plotting coordinates
1399 REM for nodes and perigee using subroutine 300
1400 REM NODES
1410 O = VN:H = NN:K = KN
1420 GOSUB 500
1430 X(2,2) = HU + XØ:Y(2,2) = ET - YØ:A(2,2) = O
1440 X(3,2) = HU - XØ:Y(3,2) = ET + YØ:A(3,2) = O + PI
1500 REM PERIHELION
1510 O = OM:H = HM:K = KP
1520 GOSUB 500
1530 X(4,2) = HU + XØ:Y(4,2) = ET - YØ:A(4,2) = O
1599 REM Extinguish previous day's Sun,Moon,Nodes and Perigee
1600 XDRAW 1 AT X(Ø,Ø),Y(Ø,Ø)
1610 XDRAW 2 AT X(1,Ø),Y(1,Ø)
1620 XDRAW 3 AT X(2,Ø),Y(2,Ø)
1630 XDRAW 4 AT X(3,Ø),Y(3,Ø)
1640 XDRAW 5 AT X(4,Ø),Y(4,Ø)
1698 REM Print current date, taking change of month,years
1699 REM and leap years into account
1700 IF INT (D) > S(M) THEN D = D - S(M):M = M + 1
1710 IF M > 12 THEN M = M - 12:AA = AA + 1:T = T + 1
1720 IF T = 4 THEN T = Ø
1730 IF T = Ø THEN S(2) = 29: GOTO 1750
1740 S(2) = 28
1750 HOME : VTAB 22: HTAB 29
1760 PRINT RIGHT$ (( STR$ ( INT (D) + 1ØØ)),2);" -"
1765 RIGHT$ (( STR$ (M + 1ØØ)),2);"  " AA
1799 REM Plot current day
1800 DRAW 1 AT X(Ø,1),Y(Ø,1)
1810 DRAW 2 AT X(1,1),Y(1,1)
1820 DRAW 3 AT X(2,1),Y(2,1)
1830 DRAW 4 AT X(3,1),Y(3,1)
1840 DRAW 5 AT X(4,1),Y(4,1)
1899 REM Draw current day's Moon
1900 GOSUB 600
1950 Q = FRE (Ø)
2000 D1 = A(Ø,1) - A(1,1):D2 = A(Ø,2) - A(1,2)
2010 IF SGN ( SIN (D1)) = SGN ( SIN (D2)) THEN 5000
2020 C = SIN (D1) / ( SIN (D1) - SIN (D2))
2030 SK = A(Ø,1) - A(2,1) + C × (A(Ø,2) - A(2,2) - A(Ø,1) + A(2,1))
2040 LI = ABS ( COS (SK))
2099 REM Check Eclipse
2100 IF COS (D1) > Ø AND LI > Ø.96639 THEN Q$ =
     " SOLAR ECLIPSE"  :GOTO 2200
2110 IF COS (D1) > Ø AND LI > Ø.946Ø4 THEN Q$ =
     " POSSIBLE SOLAR ECLIPSE" : GOTO 2200
2120 IF COS (D1) < Ø AND LI > Ø.98723 THEN Q$ =
     " LUNAR ECLIPSE"  :GOTO 2200
2130 IF COS (D1) < Ø AND LI > Ø.97698 THEN Q$ =
     " POSSIBLE LUNAR ECLIPSE" : GOTO 2200
2140 GOTO 5100
2200 VTAB 22 : HTAB 1 : FLASH : PRINT Q$
2210 NORMAL : PRINT : CALL BE: CALL BE
2220 PRINT " RE-START:<SPACE> " : GET SV$
2230 GOTO 5200
5000 FOR I = Ø TO 8ØØ: NEXT
5100 PE = PEEK (49152)
5110 IF PE = 155 THEN SV = PEEK (49168): TEXT : HOME : END
5120 IF PE = 2Ø5 THEN SV = PEEK (49168): GOTO 6700
5130 IF PE = 211 THEN SV = PEEK (49168): GET SV$
5199 REM Change current to previous,next to current
5200 FOR I = Ø TO 4
```

Sun and Moon Listing *(continued)*

```
5210 X(I,Ø) = X(I,1):X(I,1) = X(I,2)
5220 Y(I,Ø) = Y(I,1):Y(I,1) = Y(I,2)
5230 A(I,Ø) = A(I,1):A(I,1) = A(I,2)
5240  NEXT
5250 D = D + 1
5260  HCOLOR= 3
5270  GOTO 1000
6000  TEXT : HOME : VTAB 6
6010 Q$ = " APPARENT MOVEMENTS" :GOSUB 950
6020 Q$ = " OF" :GOSUB 950
6030 Q$ = " SUN AND MOON" :GOSUB 950
6040 Q$ = " AROUND THE EARTH" :GOSUB 950
6050 Q$ = " BY" :GOSUB 950
6060 Q$ = " SVEND ØSTRUP" :GOSUB 950
6070 Q$ = " FEBRUARY 1982." :GOSUB 950
6100 PI = 3.14159265:HU = 1ØØ:ET = 8Ø:F = 18Ø / PI
6110 S1 = 3Ø:S2 = 9ØØ:TF = 24Ø
6120 BE = 64477: REM BELL
6200  REM SUN
6210 VS = 4.88968:NS = .Ø172Ø279:ES = .Ø167259:KS = 72
6220 OS = 4.92624:HS = .ØØØØØØ82
6300  REM MOON
6310 VM = 5.43Ø83:NM = .2299715:EM = .Ø549ØØ5:KM = 72
6320 OM = 4.46361:HM = .ØØ1944368
6400  REM ASCENDING NODE
6420 VN = 3.1188827:NN = - Ø.ØØØ924219:KN = 63
6500  REM PERIHELION OF MOON
6510 KP = 59
6599  REM Load shape table at $300
6600  RESTORE : FOR I = Ø TO 66
6610  READ Q: POKE 768 + I,Q: NEXT
6620  POKE 232,ØØ: POKE 233,Ø3
6650 S(1) = 31:S(2) = 28:S(3) = 31:S(4) = 3Ø:S(5) = 31:S(6) = 3Ø
6655 S(7) = 31:S(8) = 31:S(9) = 3Ø:S(1Ø) = 31:S(11) = 3Ø:S(12) = 31
6700  TEXT : HOME : VTAB 6
6710  PRINT : PRINT " ONCE CELESTIAL BODIES MOVE YOU CAN: "
     :PRINT
6720  PRINT "     STOP MOVEMENTS  PRESS <S>"
6730  PRINT "     RE-START MOVM.  PRESS <SPACE>"
6740  PRINT "     NEW START DATE  PRESS <M>"
6750  PRINT "     EXIT PROGRAM    PRESS <ESC>"
6760  PRINT : PRINT : PRINT
6800  INPUT " STARTING DATE (DD,MM,YYYY) " D,M,AA
6810  IF M < 1 OR M > 12 THEN 6700
6820  IF M = 2 AND D < 3Ø THEN 6900
6830  IF D > S(M) THEN 6700
6899  REM Caculate day number Z,Ø is 12:ØØM,1/1/6Ø GMT
6900 T =  INT ((AA / 4 -  INT (AA / 4)) × 4 + .Ø5)
6910  IF M < 3 THEN AØ = AA - 1:MØ = M + 13: GOTO 6930
6920 AØ = AA:MØ = M + 1
6930 Z =  INT (365.25 × AØ) +  INT (3Ø.6ØØ1 × MØ) +
     D - 715953.5
6950  FOR I = Ø TO 4:X(I,Ø) = Ø:Y(I,Ø) = 18Ø:A(I,Ø) =
     Ø : NEXT
7200  REM  CALCULATE START SUN
7210 V = VS:N = NS:E = ES:K = KS:O = OS:H = HS
7220  GOSUB 300
7230 X(Ø,1) = X:Y(Ø,1) = Y:A(Ø,1) = A
7300  REM  CALCULATE MOON
7310 V = VM:N = NM:E = EM:K = KM:O = OM:H = HM
7320  GOSUB 300
7330 X(1,1) = X:Y(1,1) = Y:A(1,1) = A
7400  REM  CALC. NODES
7410 O = VN:H = NN:K = KN
7420  GOSUB 500
7430 X(2,1) = HU + XØ:Y(2,1) = ET - YØ:A(2,1) = O
7440 X(3,1) = HU - XØ:Y(3,1) = ET + YØ:A(3,1) = O + PI
7500  REM  CALC.PERIHELION
7510 O = OM:H = HM:K = KP
7520  GOSUB 500
7530 X(4,1) = HU + XØ:Y(4,1) = ET - YØ:A(4,1) = O
8000  HGR : HCOLOR= 3: SCALE= 1: ROT= Ø
8010  FOR I = Ø TO 2 × PI STEP PI / 18
8020  HPLOT HU + ET × COS (I),ET - ET × SIN (I)
8030  NEXT
8040  HPLOT HU,7Ø TO HU,9Ø
8050  HPLOT 9Ø,ET TO 11Ø,ET
8060  HPLOT 16,ET TO 2Ø,ET
8070  HPLOT 18Ø,ET TO 184,ET
8080  GOTO 1000
9000  DATA 5,Ø,12,Ø,41,Ø,54,Ø,58,Ø,62,Ø,37,63,54,45,37,
        228,63,23,54
9010  DATA 14,45,213,19,246,24,24,192,24,4Ø,5,64,72,32,76,137,146,18
9020  DATA 45,Ø,37,63,54,45,37,228,63,23,54,14,45,5,Ø,1Ø3,21,6
9025  DATA Ø,245,7,32,Ø,172,3Ø,7,32,Ø
9030  END
```

MICRO

Microcomputers in a College Teaching Laboratory, Part 3

by Thor Olsen, Howard Saltsburg, Richard H. Heist

Process control is illustrated using two simple experiments — an air bath and a simulated chemical reaction in an industrial-type chemical reactor. Circuits presented include an LED training device, AC power controller, and PET parallel port multiplexer.

Part I of this series (MICRO 53:53) provided an overview of the undergraduate Chemical Engineering laboratory program at the University of Rochester. Part II (MICRO 55:59) focused on the use of computers for data acquisition in a laboratory environment. This article emphasizes the output of signals from the computer that, together with data acquisition, enables you to "close the loop" so a process or an instrument can be controlled with a microcomputer.

Closing the Loop: Process Control

During the sophomore laboratory course, students learn to generate and control digital output signals from the microcomputer. A light-emitting diode (LED) module that attaches to the parallel port of the PET computer and maps the data bus to eight LEDs is the primary tool. The module is battery operated and completely self-contained (see figures 1 and 2). Although the module is simple, its effectiveness in visualizing operator control of the output port is remarkable. With this device, it is easy to demonstrate that the computer can be used to control any external device that requires simple on/off operation.

The use of the LED mapping illustrates a primitive form of control in which the eye acts as a sensor, and the operator can respond manually to an error by making a change to correct the situation. This is, in fact, a form of open loop control. For laboratory and engineering purposes, however, the implementation of automatic, so called closed loop, control is of more interest. In closed loop control the operator defines a quantity called the process variable, such as the temperature, and selects a desired value (the set point) at which this variable should be maintained. The difference between the process variable and the set point (i.e., the error) is used to determine how the device that influences the process variable should respond to correct the error.

The home thermostat is a simple form of control; it controls the room temperature (the process variable) simply by turning the heater or air conditioner on and off. As is commonly experienced with this type of control, the room temperature will vary automatically and continuously about the set point. Without intelligent devices, it is difficult (and expensive) to utilize more sophisticated control strategies, which would give less variation about the set point. The microcomputer, however, is an ideal device for such tasks as it can be used in complex decision-making modes. In contrast to conventional analog control devices where the control strategy often is implemented by mechanical means, the microcomputer allows strategy to be easily changed as needed. All that is required is modification of the software.

Since the students have had experience in reading temperature with the thermistor/555 timer circuit and have learned to send digital information to an external device, such as the LED module, it is a relatively simple matter to combine the two functions in a process control experiment. Although the theory of process control is not usually taught until the senior year, we have found that in the laboratory an introduction to the topic can be given to the sophomores.

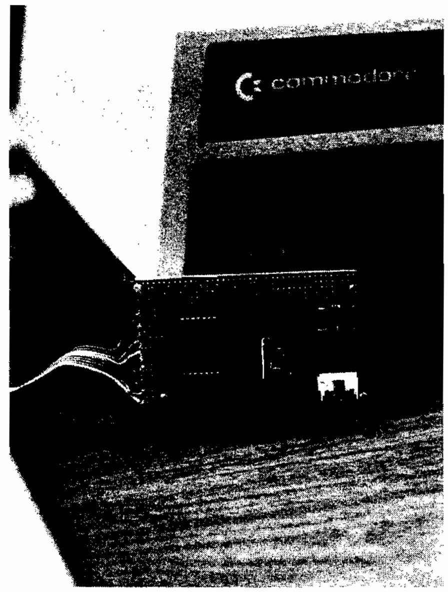


Figure 1: **Photograph of the battery-operated module used to map the PET data bus to eight LEDs.**

A simple recirculating air heater, or air bath, provides a practical application of the theory. The students are required to write a program in Structured BASIC to effect the desired temperature control of the air bath with on/off and proportional control strategies. The success of a strategy is illustrated by running the program with the air bath interfaced to the microcomputer. Also, the air bath allows operator control of recirculated air *versus* intake of (cold) room air so that sudden changes can be made in the heat requirement of the system (load changes) to further test the control strategy. Because the air bath is simple and inexpensive, each computer can be equipped with its own system. Thus, each student has easy access to an experimental station where he can develop and test his program.

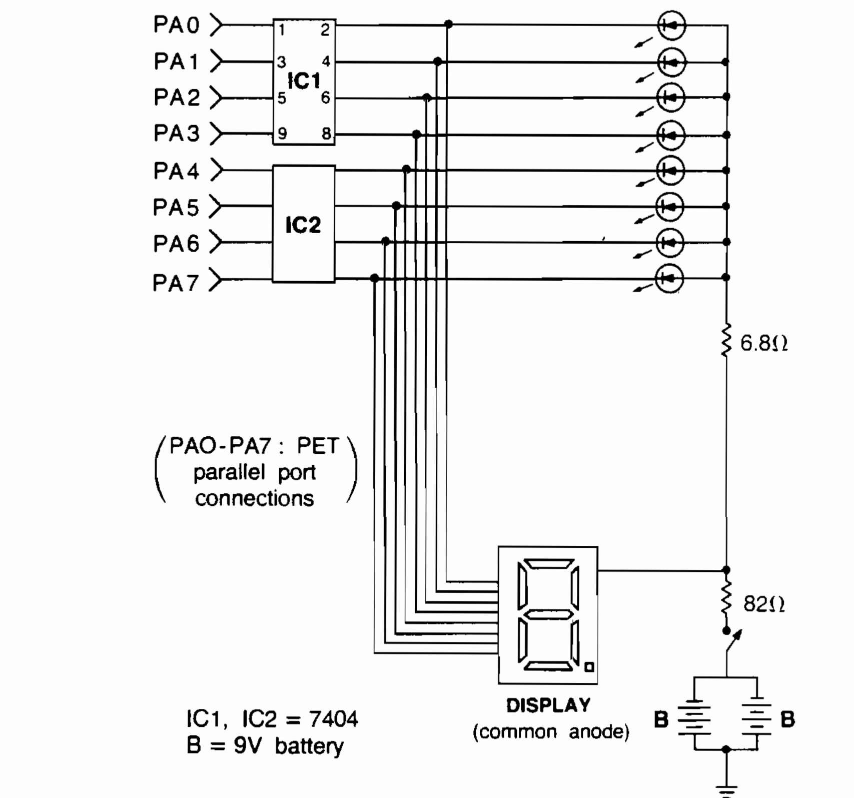
The Air Bath

The air bath consists of a box, 12 × 9 × 4 inches. The front cover is plexiglass; all other sides are made of wood, covered on the inside with aluminum foil. The box has a vertical partition, open at the top and bottom to allow circulation of air. Mounted inside the box is a light bulb, which is painted black (heater), and a fan to circulate the air. There are ventilation holes at the bottom on each side of the box and a sliding damper, which in one extreme position blocks the air exhaust vents, and in the other, the recirculation opening of the center partition. By moving the damper, the operator can impose a load change on the operating conditions of the box. A thermistor, located near the exhaust vents, is used with the 555 timer circuit to monitor the air temperature of the bath.

The AC power control circuit for the heater is shown in figure 3. The operation of the circuit can be described as follows: When the output line from the PET is high (logic 1), the 2N2222 transistor (Q1) is turned on, allowing current to flow through the LED of an optoisolator (IC1). When the LED is emitting, the photoconductor element of IC1, a TRIAC, will allow control current to flow to the power-controlling TRIAC (Q2), and the heater is turned on. This circuit permits only two operating states: power on and power off. Most control strategies, however, call for the use of fractions of full power. Fortunately, such fractional-power operation can be simulated by dividing the operating time into short "control intervals;" e.g., one second each, and turning the power on for that fraction of each control interval that corresponds to the fraction of full power called for by the control algorithm. Thus, a variety of more complex control strategies can be implemented, even with this simple type of hardware.

The air bath is also small enough so that the response time for a load change is only a minute or so, and the time required to reach a desired set point temperature of 50 degrees Celsius, starting from room temperature, is about five minutes. Although this response time is much shorter than that of most chemical process equipment, it is useful in the laboratory. The fast response provides a highly interactive situation, promotes independent efforts in development of the required control strategies, and permits exploration of alternative control modes.

Figure 2: Schematic of the LED module circuit.



With typical operating temperatures from 45 to 60 degrees Celsius, typical responses of the bath temperature to load changes are shown in figures 4 and 5. In figure 4 the PET is programmed to display data (including the measured air bath temperature) on the left side of the screen and plot the temperature on the right side. The set point temperature is represented by the straight vertical line in the center of the plot. The time between successive screen display updates is about six seconds, although the temperature is measured and the heater power updated several times between consecutive screen updates. The peak in the temperature profile shown on the screen display reflects a momentary increase in temperature when the damper was closed. The control algorithm (power to heater proportional to the error —

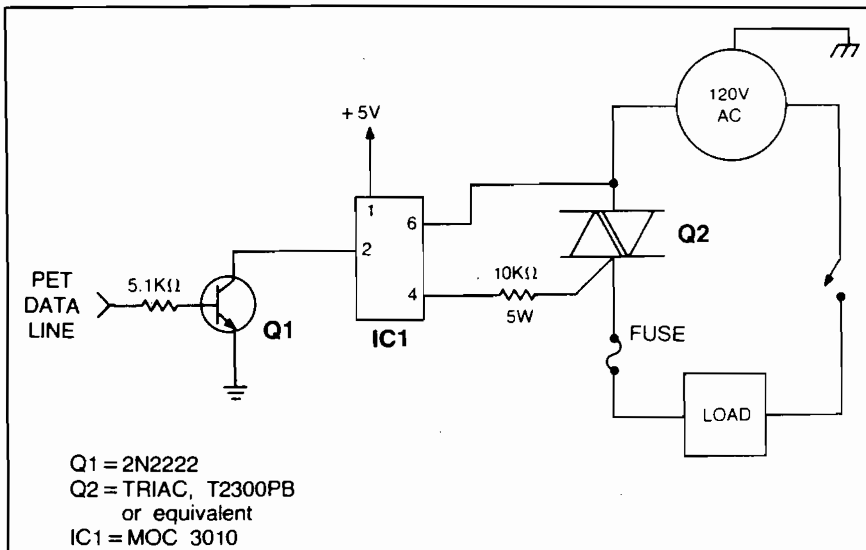


Figure 3: Schematic for AC power controller on the air bath unit.

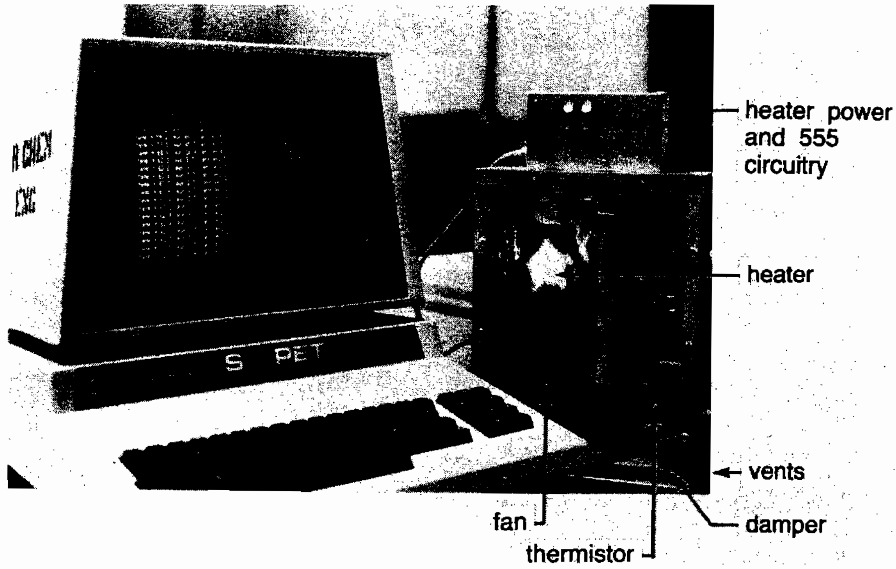


***Figure 4:* Photograph of the microcomputer/air bath combination used for process control experiments by our students. See text for description.**

proportional control) gradually reduced the temperature toward the set point. A graph of the response of the bath temperature to a load change is shown in figure 5. Again, the center line indicates the set-point temperature. Note that the control algorithm used to generate the data in figure 5 involves corrections that are the sum of terms involving proportionality, time integral and time derivative of the error (PID), although the hardware is unchanged. The additional computations, relative to proportional control, present no problem for the microcomputer, even when controlling a device with as short response times as the air bath.

The response to this project has been gratifying. The students apply the material they have learned during the semester and acquire confidence in the use of the computer in a laboratory environment. The concept of the computer being a tool is re-emphasized by its use in the solution of a realistic engineering problem.

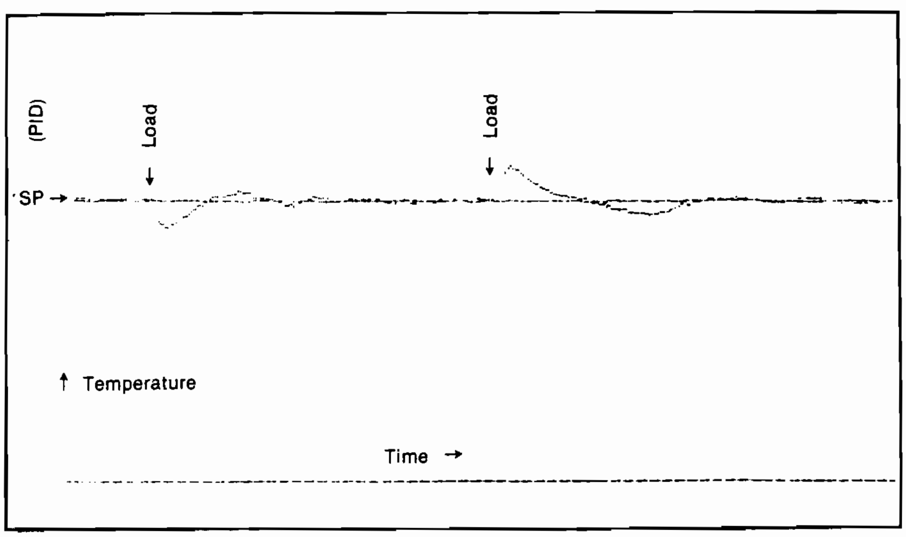


***Figure 5:* High resolution, dot matrix printer output from an air bath experiment. The straight, center line represents the set point temperature, the trace is the actual temperature in the bath and the "load" markings indicate when the damper was opened and closed, respectively (see text). The time between the two load markings is approximately 12 minutes. The time for the bath to respond to a load change is typically around 5 - 10 seconds. The type of control procedure used in this experiment was PID (proportional-integral-derivative). See reference 3 for details concerning control strategies.**

The Continuous Stirred Tank Reactor

The air bath just described plays an important role in the students' laboratory experience. It is their first "real" chemical engineering experiment and clearly illustrates how the computer can be used to control a process device. The concepts used with the air bath are directly applicable to more realistic problems, but, unfortunately, the dynamic behavior of real process devices cannot be determined adequately from the study of such small-scale laboratory equipment. Therefore, it is important to deal with real industrial devices. Toward this end, a pilot-plant scale chemical reactor has been interfaced to a microcomputer. The reactor, shown in figure 6, is simply an oval tank surrounded by a water-cooled jacket and equipped with a stirrer.

The reactor is operated as a continuous stirred tank reactor (CSTR); i.e., reactants are continuously fed to the reactor, and a mixture of reactants and product is continuously withdrawn. Again, the problem is to control temperature, but the heat source is now, in principle, a chemical reaction. Rather than work with an actual chemical reaction, however, an exothermic (heat generating) reaction is simulated by feeding water to the CSTR (instead of reactants) and bleeding steam into the flow to heat the reactor contents. By controlling the rate of steam addition, the heat released by an actual chemical reaction can be simulated safely and inexpensively.

The process hardware allows three stream temperatures to be measured; the reactor output (product stream), and the cooling jacket input and output. Two variables can be controlled by the computer: 1. the flow rate of cooling water through the jacket (which controls the "reaction"); and, 2. the flow rate of steam into the reactor (which effects the simulation). The temperature measurements are made using thermistors and the 555 timer A/D[2]. The flow rates of cooling water and steam are regulated by two commercial flow controllers. An analog signal of 4 - 20 mA is required for each. The controller design further requires that once a current is set at a certain level, it must remain at that level until a change in controller setting is desired.

Since the design requires two output ports for the two flow controllers

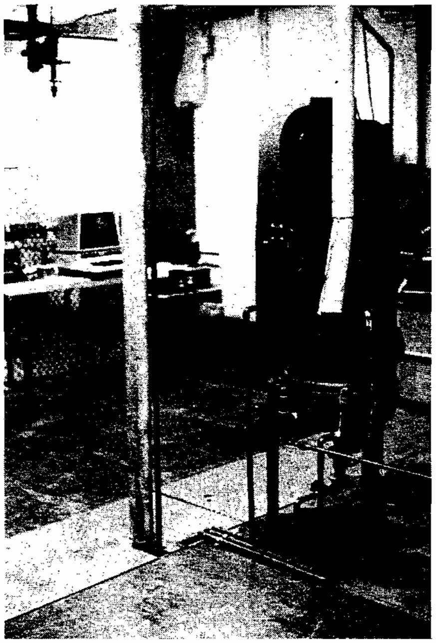


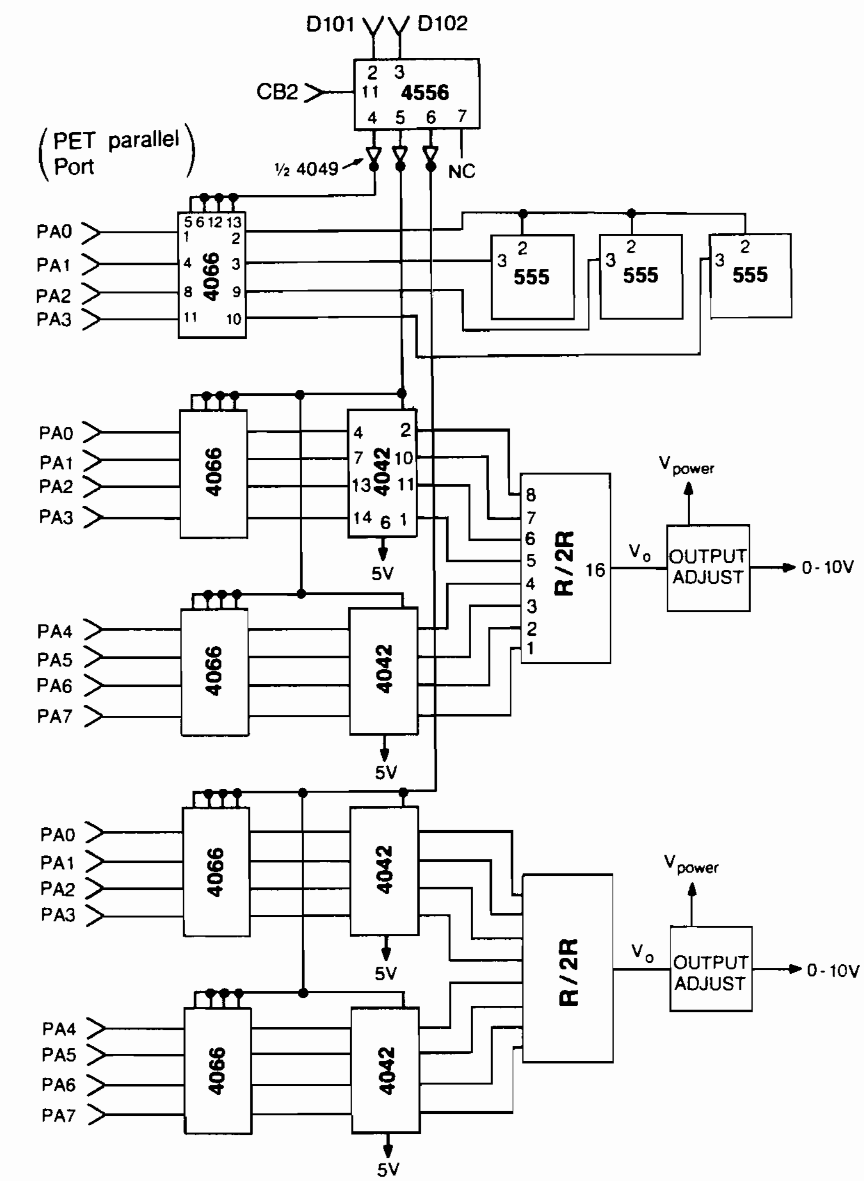
Figure 6: **Photograph of the continuous, stirred tank reactor (CSTR). The actual reactor tank and cooling jacket comprise the oval portion of the device. (The bulky portion above the reactor contains the motor and variable-speed transmission for the stirrer.)**

and one input port for the three thermistors, it was necessary to multiplex the PET parallel port. The circuit used is shown in figure 7; other versions have been described in the literature noted in reference 4. Details concerning the individual integrated circuits can be found elsewhere[5], but a qualitative description of their circuit function may be useful. The 4066 integrated circuits are CMOS analog switches. Each chip contains four separate switches which, upon command from the computer, can be individually opened or closed. If a switch is closed, its internal resistance is only a few hundred ohms; if the switch is opened, its resistance increases by about ten orders of magnitude. The net effect of opening a switch is that a device connected to the data bus through this high resistance is effectively isolated from the bus. For instance, if the switches in the top 4066 chip in figure 7 are closed and all the switches in the remaining four 4066 chips are opened, the three 555 timers will be connected to the computer while the remaining elements will not affect the data bus. Thus, by selectively controlling the individual 4066 chips, the single parallel port can be multiplexed quite easily.

The selective control is provided by a 4556 CMOS binary to 1-of-4 decoder integrated circuit (figure 7). By placing the binary representation of the numbers 0 to 3 on the input connections of the 4556 (labeled D101 and D102 in figure 7) any one of the four output connections can be activated (actually, deactivated since the selected output is brought to ground potential). Therefore, by using the computer to pass the numbers 0 to 3 to the 4556, any one of the three separate devices illustrated in figure 7 can be accessed. The fourth possibility is used to isolate all three devices from the data bus. Since the parallel port is in use, the 4556 decoder is connected to the IEEE-488 port, which is also available on the back of all PET computers. This port can be used as a data port in much the same manner as the parallel port[6]. The only difficulty arises when the IEEE-488 port is to be used to communicate with another device, such as a printer or disk drive. Since the 4556 is also connected to the IEEE-488 bus, the different devices attached to the 4556 would be accessed whenever the state of the two data lines, D101 and D102, changed.

This problem is circumvented by using the enable command on the 4556 decoder. If the enable command is not activated the chip automatically ignores all input. Thus, irrespective of the contents of the IEEE-488 data bus, the devices multiplexed to the parallel port will not be disturbed if the enable command is not activated. The CB2 control line, available at the parallel port and accessible to the computer program, is used to control the 4556

Figure 7: **Schematic of the PET parallel port multiplexing circuit described in the text.**



enable command. Details concerning the PET input/output ports are available from a variety of sources (references 4, 6-8) and will not be discussed here.

To carry out the control function, the microcomputer must generate an analog signal that must be passed to the flow controllers and maintained, even after the flow controller interface is removed from the data bus. The method employed for digital to analog (D/A) conversion utilizes an arrangement of precision resistors called an R/2R ladder network. Details of this method can be found in the literature in reference 9.

Essentially, the device produces a voltage output proportional to the value of the binary number applied to the network input. The output from the R/2R network should be buffered[10]. What is done with the output depends upon the specific application at hand. For example, if a range of voltage is required, the buffered output can be used with a Darlington network. If a current range is required, the buffered output is used to drive a current source. In this particular example, the flow controllers for the cooling jacket and the steam line to the CSTR are current-to-pressure devices. This means that a range of input current (4 mA to 20 mA, in this case) is required to control the rate of fluid flow from no flow to full flow. The fraction of full flow is thus determined by the number (0 produces no flow, 255 produces full flow) placed on the PET parallel port by the computer program. The circuit currently in use with the CSTR that provides this range of current is shown in figure 8.

The binary number passed to the input of the R/2R network is maintained after the flow controller interface is isolated from the data bus by using CMOS 4042 latches (figure 7). The latch passes a binary number from the input to the output connections upon command, and then, on command, "latches" or holds that number on the output connections irrespective of what happens at the input. Thus, when one of the flow controller interfaces is selected, a number is placed on the I/O port reflecting a desired setting for the controller. This number is latched so

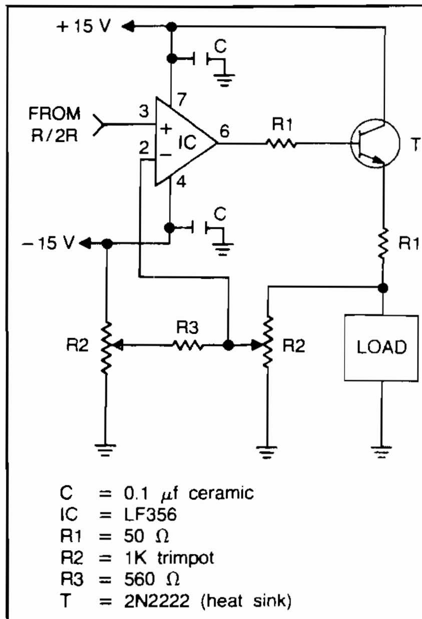


Figure 8: **Schematic of the current source used to actuate the flow controllers described in the text. The LOAD indicated in the schematic represents the flow controller. The circuit is designed to produce a linear variation in output current from 4 mA at zero volts (0 binary) to 20 mA maximum (255 binary). See text for additional details.**

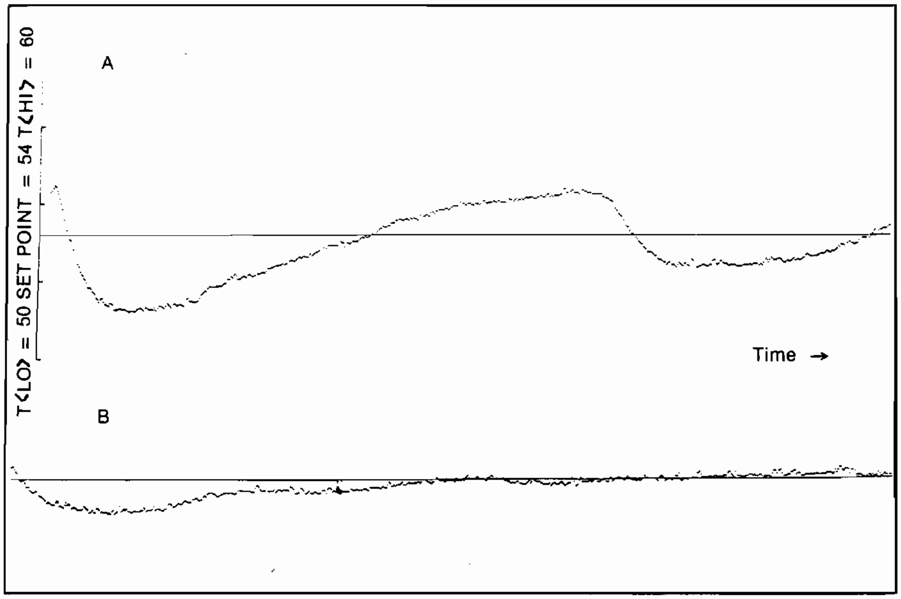


Figure 9: **Typical high resolution, dot matrix printer output from a control experiment using the CSTR. The curve marked "B" is a continuation of curve "A". The center line in each trace indicates the set-point temperature, and the trace is the output-stream temperature of the CSTR. The response time to a load change for the CSTR is typically 4 - 5 minutes. The time between the first trough and the second peak on plot "A" is approximately 40 minutes. The control strategy used in this experiment is called proportional-integral (PI).**

that it will remain as input to the R/2R network. The 4042 IC is then isolated from the I/O port, which becomes available for communication with another device.

The control of the multiplexed port, the temperature measurements, and the control of the flow controllers are accomplished entirely with software. A typical graph of the output temperature for the CSTR, as it responds to a load change, is shown in figure 9. Note in the figure caption the much longer time scales relative to the air bath experiment. With the CSTR the students receive first-hand experience with the problems associated with control of equipment — especially with the slow response time characteristic of many industrial devices.

Usually students don't become involved with the complicated CSTR until the senior laboratory course. The control experiments in the senior laboratory course primarily involve studies of the dynamic response of the CSTR to load changes when different control strategies (algorithms) are used.

In the final article of this series,we will discuss the interfacing of microcomputers to complex scientific instrumentation. Specific examples involving gas chromatography and converting a single beam spectrophotometer into an effective dual beam instrument will be presented.

REFERENCES

1. H. Saltsburg, R. H. Heist, and T. Olsen, "Microcomputers in a College Teaching Laboratory, Part 1," MICRO 53:53, October, 1982.
2. R. H. Heist, T. Olsen, and H. Saltsburg, "Microcomputers in a College Teaching Laboratory, Part 2," MICRO 55:59, December, 1982.
3. R. J. Bibbero, "Microprocessors in Instruments and Control," (John Wiley and Sons, New York, 1977), Chapter 2.
4. J. M. Downey and S. M. Rogers, "PET Interfacing," (Howard W. Sams & Co., Inc., Indianapolis, 1981).
5. See, for example, "Motorola CMOS Integrated Circuits," (Motorola Inc., 1978), 2nd ed.; D. Lancaster, "CMOS Cookbook," (Howard W. Sams and Co., Inc., Indianapolis, 1979).
6. E. Fisher and C. W. Jensen, "PET and the IEEE-488 Bus (GPIB)," (Osborne/McGraw-Hill, Berkeley, 1980).
7. See, for example, N. Hampshire, "The PET Revealed," (Computabits Ltd., Somerset, England, 1980); R. C. West, "Programming the PET/CBM, The Reference Encyclopedia for Commodore PET/CBM Users," (COMPUTE! Books, Greensboro, 1982).
8. See, for example, R. Zaks, "6502 Applications Book," (SYBEX Inc., 1979); M. L. DeJong, "Programming and Interfacing the 6502, with Experiments," (Howard W. Sams and Co. Inc., Indianapolis, 1980).
9. Z. H. Meiksin and P. C. Thackray, "Electronic Design with Off-The-Shelf Integrated Circuits," (Parker Publishing Co., Inc., West Nyack, NY, 1980), pgs. 307-310.
10. W. Jung, "IC op-amp Cookbook," (Howard W. Sams and Co., Inc., Indianapolis, 1979).

You may contact the authors at the Department of Chemical Engineering, University of Rochester, Rochester, NY 14627.

Measurement of a 35mm Focal Plane Shutter

by Mike Dougherty

The program SHUTTER uses inexpensive hardware to measure the accuracy and repeatability of the focal plane shutter found in most single-lens reflex cameras.

SHUTTER
requires:

Atari 800 (may be modified for others) and a few electronic components

Acting as the logic controller for hardware sensors, a computer can be used to measure events beyond human capabilities. One such event is the movement of a camera's focal plane shutter. A typical focal plane shutter is capable of exposing film from 1/1000th of a second to a full second. However, it is difficult to determine the shutter's accuracy. The following program, accompanied by simple hardware, gives the computer/photographer hobbiest a means to measure the accuracy and reliability of a focal plane shutter. Although SHUTTER was written for an Atari 800 personal computer, the basic concepts are transportable to other systems and the program can be converted.

Definitions

A focal plane shutter consists of two opaque curtains that move in front of the photographic film. Light comes through these curtains, which form a window or opening, and strikes the film. The amount of exposure is determined by the distance between the two curtains and the speed that the curtains move across the film. Looking from the back of my camera, the curtains move from right to left, exposing a vertical slice of film. This particular camera maintains a constant curtain speed while changing the distance between the curtains for different exposures. It is called a focal plane shutter because, for the best performance, the shutter must be placed as close to the plane of focus as possible.

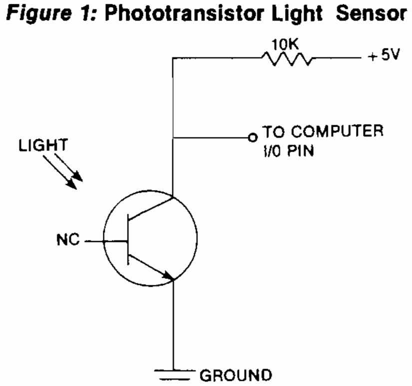
The photographer is concerned with two inaccuracies in this type of shutter — actual exposure and exposure consistency. Obviously, for good photographic results, the shutter should produce the desired exposure. However, any inaccuracy in the shutter may be corrected by changing the lens opening — as long as the shutter is consistent. Thus, in practice, consistency is usually more important than absolute accuracy.

Note that there is much practical latitude in the photographic process. Exposure errors as large as 33% (⅓ of an f/stop) may be unnoticed by some. Further, errors in exposure can often be corrected. However, each compensation or correction also compromises the final photographic result. To achieve the maximum photographic quality possible, every phase of the photographic process must be understood and utilized to its fullest degree. The first step, exposure of the film, is no exception.

Hardware

The light sensor used by SHUTTER consists of two elements: a phototransistor and a 10K resistor. The phototransistor (Radio Shack #276-130) does the actual work of detecting the light, while the resistor limits the current flow into the computer. The typical circuit for one light sensor is shown in figure 1.

Figure 1: **Phototransistor Light Sensor**



Listing 1

```
100 REM .....   S H U T T E R   ......
105 REM
110 REM ...   by Mike Dougherty
115 REM
120 REM A program to measure the shutter
125 REM speed of a focal plane shutter
130 REM using phototransistors.
135 REM
140 REM
145 REM .............................
150 REM
155 REM
1010 DIM SAMPLE(9),KEY$(1)
1020 GRAPHICS 0:POKE 752,1
1030 PRINT "Initializing USR functions"
1110 GOSUB 10000:REM INIT USR FUNCTIONS
1210 F=1.79*1000000:REM ATARI CLOCK IS 1.79 MHZ
1220 PORT=54017:REM JOYSTICK I/O PORT
1230 DELAY=0:REM USR DELAY CONSTANT
1240 ERROR=0:REM DETECTOR ERROR
1250 CDIST=24.64:REM DISTANCE BETWEEN VELOCITY SENSORS
1300 REM
1301 REM ... Main Program
1302 REM
1303 REM POLL USER FOR WHICH OF THE
1304 REM THREE FUNCTIONS TO EXECUTE.
```

While in the dark, the phototransistor is turned off and the current flows into an I/O pin of the computer with a voltage of +5V. When the light exceeds a specific threshold level, the phototransistor turns on and current flows to ground giving a zero voltage at the I/O pin. Thus, a computer I/O pin will be a logic 1 (+5V) in the dark and a logic 0 (ground) in sufficient light. For the Radio Shack phototransistor, sufficient light consists of a 50-watt reflector bulb (available from Sears) at a distance of 50 cm from the phototransistor. To measure the focal plane shutter exposure, the program simply measures the time that the I/O pin connected to the phototransistor remains low.

Unfortunately, there is a major source of error when measuring the highest shutter speeds. This error originates from the finite size of the light sensitive silicon in the phototransistor — 0.5 millimeters (mm) in my case. Since this is not infinitely small (or small enough to ignore), the shutter time measured will be longer than it should be. As the two curtains in my camera move toward the left, the I/O pin goes to zero when the left curtain uncovers the right edge of the light-sensitive silicon. (The response is virtually immediate since the source light is bright enough to drive the phototransistor into saturation with only a fraction of the silicon exposed to light.) The I/O pin will remain zero until the right curtain covers the left edge of the light-sensitive silicon. Instead of measuring the exposure time for an infinitely thin slice of film, a slice of 0.5 mm in width is measured. For my camera at 1/1000th of a second, this error becomes significant.

To eliminate the finite detector size error, the time required to cross the light sensitive silicon was determined and subtracted from the measured shutter time. To measure this time error, the velocity of the shutter curtains and the size of the light-sensitive area were measured. The light detector width, WIDTH, was measured by an accurate drafting scale and found to be 0.5 mm. (For the purposes of this article, I shall assume that this measurement contains no error!)

To measure the shutter velocity, only the velocity of the left curtain had to be measured. The right curtain must move at the same speed or one side of the film would receive more exposure

Listing 1 *(Continued)*

```
1305 REM
1310 GRAPHICS 0
1315 POSITION 5,5:PRINT "Select function"
1320 POSITION 7,7:PRINT " M - Monitor"
1322 POSITION 7,8:PRINT " V - Velocity"
1324 POSITION 7,9:PRINT " S - Shutter"
1330 POSITION 22,5:INPUT KEY$:REM INPUT USER CHOICE
1410 IF KEY$="S" THEN GOSUB 2000:GOTO 1310
1420 IF KEY$="V" THEN GOSUB 3000:GOTO 1310
1430 IF KEY$="M" THEN GOSUB 4000:GOTO 1310
1435 REM
1436 REM ... SOUND ERROR INPUT
1437 REM
1440 SOUND 0,150,10,8
1450 FOR WAIT=0 TO 50:NEXT WAIT
1460 SOUND 0,0,0,0
1470 GOTO 1320
2000 REM
2001 REM ... Shutter Function
2002 REM
2003 REM This function measures the
2004 REM actual exposure time of the
2005 REM shutter and computes the
2006 REM relative error.  The variable
2007 REM ERROR should be set via the
2008 REM Velocity function prior to
2009 REM running this function.
2010 REM
2110 GRAPHICS 0
2120 PRINT "Expected time in ms ";
2130 INPUT EXPECT
2140 ROUTINE=1536:REM 1ST USR FUNCTION
2150 MASK1=2:REM BIT #1 PHOTOTRANSISTOR
2160 MASK2=0:REM NOT USED
2170 GOSUB 9000
2180 PRINT "Shutter error:       ";(EXPECT-AVE)/EXPECT*100;"%"
2190 GOSUB 8000
2200 RETURN
3000 REM
3001 REM ... Velocity Function
3002 REM
3003 REM This function measures the
3004 REM leftward velocity of the
3005 REM shutter curtains.  This
3006 REM speed is used to compute the
3007 REM error due to a finite sensor
3008 REM size.  The variable CDIST,
3009 REM measured with the aid of the
3010 REM Monitor function, is combined
3011 REM with the detector width to
3012 REM compute ERROR.  This function
3013 REM should be run prior to the
3014 REM Shutter function for every
3015 REM shutter speed with a different
3016 REM shutter curtain velocity.
3017 REM
3110 GRAPHICS 0
3120 PRINT "Detector width in mm ";
3130 INPUT WIDTH
3140 ROUTINE=1664:REM 2ND USR FUNCTION
3150 MASK1=1:REM BIT #0 PHOTOTRANSISTOR
3160 MASK2=4:REM BIT #2 PHOTOTRANSISTOR
3170 ERROR=0:REM COMPUTING NEW ERROR
3180 GOSUB 9000
3190 ERROR=(WIDTH*AVE)/CDIST
3195 ERROR=INT(ERROR*100+0.5)/100
3200 PRINT "Detector error:      ";ERROR;" ms"
3210 GOSUB 8000
3220 RETURN
4000 REM
4001 REM ... Monitor Function
4002 REM
4003 REM This function is used to
4004 REM assist measuring the distance
4005 REM between the velocity sensors.
4006 REM This sensor distance, CDIST,
4007 REM should be edited into line
4008 REM 1250 of SHUTTER.  The Monitor
4009 REM function monitors all three
4010 REM phototransistor inputs.  Press
4011 REM any key to exit.
4012 REM
4110 GRAPHICS 0
4120 CHANNELS=PEEK(PORT):REM SAMPLE JOYSTICK #3 & #4
4130 FOR I=0 TO 2:REM ONLY FIRST 3 BITS
4140 CHAN=CHANNELS-INT(CHANNELS/2)*2:REM ISOLATE BIT
```

Listing 1 *(Continued)*

```
4150 PRINT "Bit #";I;": ";CHAN;"     ";
4160 CHANNELS=INT(CHANNELS/2)
4170 NEXT I
4175 PRINT
4180 IF PEEK(764)=255 THEN 4120
4190 POKE 764,255
4200 RETURN
8000 REM
8001 REM ... Pause subroutine
8002 REM
8003 REM Wait until a key is pressed
8004 REM with a flashing prompt.
8005 REM
8110 POKE 752,1
8120 POSITION 7,22:PRINT "Hit any KEY to continue";
8140 POSITION 7,22:PRINT "                        ";
8160 IF PEEK(764)=255 THEN 8120
8170 POKE 764,255
8180 RETURN
9000 REM
9001 REM ... Sample Phototransistors
9002 REM
9003 REM This subroutine samples ten
9004 REM shutter releases and computes
9005 REM the simple statistics of
9006 REM average, AVE, standard deviation,
9007 REM SD, and the normalized standard
9008 REM deviation (NSD).  This subroutine
9009 REM calls the USR function starting
9010 REM at the memory location ROUTINE.
9011 REM The bit masks, etc., are
9012 REM parameterized to allow this
9013 REM subroutine to be used by both
9014 REM the Velocity function and the
9015 REM Shutter function.
9016 REM
9100 PRINT
9110 FOR LOOP=0 TO 9 STEP 1
9120 IF PEEK(PORT)<>255 THEN 9120:REM WAIT UNTIL SHUTTER CLOSED
9130 CYCLES=USR(ROUTINE,PORT,MASK1,MASK2,DELAY)
9135 REM CONVERT MACHINE CYCLES TO MICROSECONDS, THEN MILLISECONDS
9140 TIME=((37+5*DELAY)*CYCLES)/F
9150 SAMPLE(LOOP)=INT(TIME*100000+0.5)/100-ERROR
9160 PRINT " TIME #";LOOP;" IN 1/1000 SEC: ";SAMPLE(LOOP)
9170 NEXT LOOP
9180 SOUND 0,100,10,15:FOR WAIT=0 TO 2:NEXT WAIT:SOUND 0,0,0,0
9185 REM COMPUTE THE STATISTICS
9190 AVE=0
9200 FOR LOOP=0 TO 9
9210 AVE=AVE+SAMPLE(LOOP)
9220 NEXT LOOP
9230 AVE=AVE/10
9240 VAR=0
9250 FOR LOOP=0 TO 9
9260 VAR=VAR+(SAMPLE(LOOP)-AVE)^2
9270 NEXT LOOP
9280 SD=SQR(VAR/9)
9290 PRINT :PRINT "AVERAGE TIME:          ";AVE
9300 PRINT "STANDARD DEVIATION:  ";SD
9310 PRINT "NORMALIZED SD (NSD): ";SD/AVE*100
9320 RETURN
10000 REM
10001 REM ... U S R    P O K E
10002 REM
10003 REM Poke the USR functions into
10004 REM reserved page 6 memory:
10005 REM $0600 - $06FF.
10006 REM
10010 DIM BYTE$(2)
10020 TRAP 10200:REM EXIT WHEN NO MORE DATA
10030 READ ADDRESS:REM USR STARTING ADDRESS
10100 READ BYTE$:REM READ MACHINE CODE BYTE
10110 IF BYTE$="**" THEN SOUND 0,0,0,0:GOTO 10030:REM
                                       END OF THIS USR FUNCTION
10120 GOSUB 10500:REM COMPUTE BYTE
10130 POKE ADDRESS,BYTE:REM PUT IN MEMORY
10140 ADDRESS=ADDRESS+1:REM NEXT MEMORY ADDRESS
10150 GOTO 10100
10200 RETURN
10500 REM
10501 REM ... BYTE$ --> BYTE
10502 REM
10510 BYTE=0
10520 V=ASC(BYTE$(1)):GOSUB 10600
10530 V=ASC(BYTE$(2)):GOSUB 10600
10540 RETURN
10600 REM
```

(continued)

than the other side. Two additional phototransistors were added to the sensor, one at each edge of the film opening. These phototransistors were used to measure the time from when the left curtain passed the right edge of the first sensor until the left curtain passed the right edge of the second sensor. The distance between the two phototransistors was determined by monitoring the two light sensor I/O pins *via* the SHUTTER Monitor function. On my hardware sensor, the distance measured was 24.64 mm. (Again, for this article, I will assume that this measurement contains no error.)

Once the focal plane curtain velocity, V, is known, the time error may be computed:

ERROR = WIDTH/V

My focal plane shutter curtain traveled the 24.64 mm of CDIST in 9.24 milliseconds (ms). This yields a velocity of

V = 24.64mm/9.24 ms = 2.67 mm/ms = 5.97 mph

For the 0.5 mm detector:

ERROR = 0.5 mm/2.67 mm/ms = 0.19 ms

With a 1/1000th of a second (1 ms) shutter speed, ERROR represents a 19% relative error.

The final hardware sensor consisted of three phototransistors mounted horizontally in a cardboard case — inexpensive but effective! This case was attached to the back of the camera (behind the focal plane shutter) with the phototransistors positioned in the film plane. A 50-watt reflector lamp was positioned 50 cm from the front of the camera directing light toward the phototransistors mounted on the other side of the shutter. The shutter measurement was performed without a lens mounted on the camera. The three sensor outputs were connected from right to left (looking from the back) to the least significant bits (bit #0, bit #1, and bit #2) of the Atari joystick #3 (STICK(2)) input port. This joystick is located at the hardware register address 54017, PORTB. All joystick ports are configured as input ports by the Atari operating system upon power up. Both the +5V and ground are obtained from the 9-pin joystick port. Here are the joystick pinouts:

Pin #1 Right velocity phototransistor (Bit #0)
Pin #2 Shutter phototransistor (Bit #1)
Pin #3 Left velocity phototransistor (Bit #2)
Pin #7 +5V power
Pin #8 Ground

Be sure to read the *Atari Hardware Manual* (available from Atari) before blindly wiring to the joystick connectors. Damage to the Atari may result from improper use. Therefore, most of my hardware projects use PORTB; in case of a faulty design, PORTA (joysticks #1 and #2) is still available for Star Raiders!

Software

SHUTTER is divided into three functions that allow the calibration of the shutter sensor hardware and measurement of the focal plane shutter. The BASIC listing of SHUTTER is given in listing 1.

The first function, Monitor, allows the user to measure the distance between the two velocity phototransistors, joystick I/O bits #0 and #2. With an accurate scale mounted to the sensor, a straight edge is manually moved in the same direction as the focal plane shutter. As the first phototransistor is uncovered, bit #0 goes low; call this location on the scale D1. Move the screen toward the left until the third phototransistor is uncovered and bit #2 goes low; call this location on the scale D2. The absolute value of D2-D1 is the calibration distance, CDIST, initialized in line number 1250 of SHUTTER. This value was measured as 24.64 mm on my specific sensor. Modify line 1250 as necessary for your own SHUTTER sensor hardware. The monitor function is exited by pressing any key except BREAK.

The other two functions utilize a statistical sample of ten shutter releases, performed by the subroutine in lines 9000 to 9320. This subroutine computes the average, the standard deviation, and the normalized standard deviation. The normalized standard deviation, NSD, is defined as

NSD = (SD/AVE)*100

The NSD is used to compare the consistency of one shutter speed to another. The sample subroutine rounds the measured times to the nearest hundredth of a millisecond. All times are

Listing 1 *(continued)*

```
10601 REM ... ADD HEX VALUE OF ASCII
10602 REM ... "V" TO BYTE.
10603 REM
10610 IF V<58 THEN BYTE=BYTE*16+V-48
10620 IF V>57 THEN BYTE=BYTE*16+V-55
10630 SOUND 0,BYTE,10,8
10640 RETURN
10700 REM
10701 REM --- U S R    D A T A
10702 REM
10705 DATA 1536
10710 DATA 68,68,85,CC,68,85,CB
10715 DATA 68,68,85,CD,68,68,68,68,85,CE
10720 DATA A9,00,8D,2F,02,A5,14
10725 DATA C5,14,F0,FC,A9,20
10730 DATA 8D,0E,D4,A9,00,85,D4
10735 DATA 85,D5,A0,00,B1,CB
10740 DATA 25,CD,D0,FA
10745 DATA 18,A5,D4,69,01,85,D4
10750 DATA A5,D5,69,00,85,D5,B0,0E
10755 DATA A6,CE,F0,04,CA,D0,FD
10760 DATA EA,B1,CB,25,CD,F0,E3
10765 DATA A9,60,8D,0E,D4
10770 DATA A9,22,8D,2F,02,60
10775 DATA **
10800 DATA 1664
10805 DATA 68,68,85,CC,68,85,CB
10810 DATA 68,68,85,CD,68,68,85,CF,68,68,85,CE
10815 DATA A9,00,8D,2F,02,A5,14
10820 DATA C5,14,F0,FC,A9,20
10825 DATA 8D,0E,D4,A9,00,85,D4
10830 DATA 85,D5,A0,00,B1,CB
10835 DATA 25,CD,D0,FA
10840 DATA 18,A5,D4,69,01,85,D4
10845 DATA A5,D5,69,00,85,D5,B0,0E
10850 DATA A6,CE,F0,04,CA,D0,FD
10855 DATA EA,B1,CB,25,CF,D0,E3
10860 DATA A9,60,8D,0E,D4
10865 DATA A9,22,8D,2F,02,60
10870 DATA **
```

Listing 2

```
0100 ; VELOCITY USR FUNCTION
0110 ; (LISTING #2)
0120 ;
0130 ;
0140 ;
0150 ; This function measures the time
0160 ; that a shutter takes to travel
0170 ; between the velocity phototransistors.
0180 ;
0190 ; Call the Velocity Function by:
0200 ;
0210 ; X=USR(1664,PORT,MASK1,MASK2,DELAY)
0220 ;
0230 ; where
0240 ;
0250 ;   PORT  - The I/O port address
0260 ;   MASK1 - "AND" mask to isolate
0270 ;           the first sensor
0280 ;   MASK2 - "AND" mask to isolate
0290 ;           the second sensor
0300 ;   DELAY - for the delay loop.
0310 ;
0315 ;
0320 ; Velocity performs the following:
0330 ;
0340 ; Initialize the variables
0350 ; Disable the Video DMA
0360 ; Disable the Real Time Clock interrupt
0370 ; Initialize the "timer", COUNT
0380 ; Wait until (PORT AND MASK1) = 0
0390 ; DOUNTIL (PORT AND MASK2) = 0
0400 ;   Increment COUNT by one
0410 ;   Wait in WAITLP DELAY times
0420 ; ENDDO
0430 ; Enable the Real Time Clock
0440 ; Enable the Video DMA
0450 ; Return the value COUNT
0460 ;
0470 ; The actual time of the DOUNTIL
0480 ; loop is  37+5*DELAY  machine
0490 ; cycles.  With DELAY=0 and the
0500 ; Atari 800 runing at 1.79 MHZ,
0510 ; each loop represents about
0520 ; 20.67 microseconds.
```

Listing 2 *(continued)*

```
             ;
             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
             ;
             ; DEFINE THE VARIABLES USED
             ;
00CB         PORT   =    $00CB      STORAGE FOR THE PORT ADDR
00CD         MASK1  =    $00CD      FOR FIRST SENSOR
00CF         MASK2  =    $00CF      FOR SECOND SENSOR
00CE         DELAY  =    $00CE      FOR DELAY FACTOR
00D4         COUNT  =    $00D4      USR RETURN VALUE
0014         RTC    =    $0014      REAL TIME CLOCK
022F         SDMACT =    $022F      DMA CONTROL SHADOW REGISTER
D40E         NMIEN  =    $D40E      NMI INTERRUPT ENABLE REGISTER
             ;
             ;
             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
             ;
             ;
0000                *=   $0680      DEFINE START IN FREE RAM
             ;
             ; INITIALIZE USR VARIABLES
             ;
0680 68      VEL    PLA             NUMBER OF ARGUMENTS
0681 68             PLA             FIRST USR ARGUMENT
0682 85CC           STA  PORT+1     MSB OF PORT ADDRESS
0684 68             PLA
0685 85CB           STA  PORT       LSB
0687 68             PLA             SECOND USR ARGUMENT
0688 68             PLA             ONE BYTE ONLY
0689 85CD           STA  MASK1      FOR FIRST SENSOR
068B 68             PLA             THIRD USR ARGUMENT
068C 68             PLA             ONE BYTE ONLY
068D 85CF           STA  MASK2      FOR SECOND SENSOR
068F 68             PLA             FOURTH USR ARGUMENT
0690 68             PLA             ONE BYTE ONLY
0691 85CE           STA  DELAY      DELAY LOOP CONSTANT
             ;
             ; DISABLE THE VIDEO DMA
             ;
0693 A900           LDA  #$00
0695 8D2F02         STA  SDMACT     CLEAR SHADOW REGISTER
0698 A514           LDA  RTC        WAIT UNTIL VBLANK SETS HARDWARE
069A C514    TICK   CMP  RTC        VBLANK UPDATES CLOCK
069C F0FC           BEQ  TICK       VBLANK HAS NOT YET OCCURRED
             ;
             ; DMA SHUT DOWN, TURN OFF VBLANK
             ; INTERRUPT.
             ;
069E A920           LDA  #$20
06A0 8D0ED4         STA  NMIEN      DISABLE NMI
             ;
             ; INITIALIZE THE LOOP COUNTER
             ;
06A3 A900           LDA  #$00
06A5 85D4           STA  COUNT      THIS IS THE USR LOCATION
06A7 85D5           STA  COUNT+1    TO RETURN A VALUE
             ;
             ; WAIT UNTIL FIRST SENSOR GOES LOW --
             ; I.E. THE CURTAIN UNCOVERS IT

             ;
06A9 A000           LDY  #$00       FOR INDIRECT INDEXED MODE
06AB B1CB    SENS1  LDA  (PORT),Y   GET I/O PORT
06AD 25CD           AND  MASK1      ISOLATE FIRST SENSOR
06AF D0FA           BNE  SENS1      NO LIGHT YET
             ;
             ; MAIN TIMING LOOP
             ;
06B1 18      TIME   CLC             COUNT THIS TIME IN THE LOOP
06B2 A5D4           LDA  COUNT      INCREMENT THE LOOP COUNTER
06B4 6901           ADC  #$01       LSB
06B6 85D4           STA  COUNT
06B8 A5D5           LDA  COUNT+1
06BA 6900           ADC  #$00       MSB
06BC 85D5           STA  COUNT+1
06BE B00E           BCS  RET        TOO LONG ERROR
06C0 A6CE           LDX  DELAY      DELAY IF NEEDED
06C2 F004           BEQ  SENS2      NO DELAY USED
06C4 CA      WAITLP DEX             WAIT A SMALL BIT
06C5 D0FD           BNE  WAITLP
06C7 EA             NOP             ADJUST TIMING
06C8 B1CB    SENS2  LDA  (PORT),Y   CHECK SECOND SENSOR
06CA 25CF           AND  MASK2      ISOLATE BIT
06CC D0E3           BNE  TIME       STILL IN THE DARK
             ;
             ; COUNT CONTAINS THE NUMBER OF TIMES
```

(continued)

corrected by the current value of the variable ERROR.

Both functions use an identical timing loop written at the assembler level and entered as machine code. This loop has a time resolution of 37 + 5*DELAY cycles. For the purposes of this article, the variable DELAY was set to zero. Thus, every unit of time counted by the USR functions represents 37 machine cycles. Since the Atari 800 runs with a clock frequency of 1.79 MHz this loop has a resolution of 20.67 microseconds. To accurately use the timing loop, all video Direct Memory Access (DMA) and Atari operating system interrupts must be disabled. For a complete explanation, refer to my previous article "A/D Conversion Using a 555 Timer IC," MICRO 52:14, as well as the manuals *Atari Operating System User's Manual* and *Atari Hardware Manual*.

The second function of SHUTTER, Velocity, allows the user to measure the velocity of the focal plane shutter curtain and compute the value of ERROR. The Velocity function uses the calibration distance, CDIST, which was measured with the aid of the Monitor function. Velocity prompts for the size of the light-sensitive area of the shutter phototransistor, bit #1 — in my case, 0.5 mm. After the ten shutter releases, the average time between the two velocity phototransistors, AVE, is used to compute ERROR. In my camera, the shutter curtain took 9.24 ms to travel the 24.64 mm of CDIST. With a detector WIDTH of 0.5 mm, the following proportion holds true:

$$\frac{9.24 \text{ ms}}{24.64 \text{ mm}} : \frac{\text{ERROR ms}}{0.5 \text{ mm}}$$

Solving for ERROR yields:

ERROR = ((9.24 ms)*(0.5 mm)) / 24.64 mm = 0.19 ms

The Velocity function uses the Atari USR function of listing 2.

The third function of SHUTTER, Shutter, allows the user to measure the shutter exposure in milliseconds and to

---

How to run a listing in MICRO's Software/Hardware Catalogs

The Software and Hardware Catalogs are provided as a service both to our readers and to the manufacturers. These entries are not MICRO reviews, but descriptions provided by the manufacturer.

To run a free listing in either catalog, a company fills out the appropriate form or merely mails in their material in the same format that appears in the magazine.

We try to limit entries to one company per month, on a first-come-first-serve basis.

If you sell products our readers should know about, write to Software/Hardware Catalog, MICRO, P.O. Box 6502, Chelmsford, MA 01824

Listing 2 *(continued)*

```
              1360 ; THAT THE TIME LOOP WAS EXECUTED.
              1370 ; THIS WILL BE RETURNED BY THE USR
              1380 ; FUNCTION
              1390 ;
              1400 ; ENABLE DMA AND INTERRUPTS
              1410 ;
06CE A960     1420 RET    LDA  #$60
06D0 8D0ED4   1430        STA  NMIEN      RESTORE REAL TIME CLOCK
06D3 A922     1440        LDA  #$22
06D5 8D2F02   1450        STA  SDMACT     RESTORE THE VIDEO SCREEN
06D8 60       1460        RTS             RETURN THE COUNT
06D9          1470        .END
```

Listing 3

```
              0100 ; SHUTTER USR FUNCTION
              0110 ; (LISTING #3)
              0120 ;
              0130 ;
              0140 ; This USR function measures the
              0150 ; number of SAMPLE loops that are
              0160 ; executed while the phototransistor
              0170 ; line is low.  Since each SAMPLE
              0180 ; loop takes 37+5*DELAY machine
              0190 ; cycles to execute, the USR routine
              0200 ; measures actual time.
              0210 ;
              0220 ; Call the USR function by:
              0230 ;
              0240 ; X=USR(1536,PORT,MASK1,MASK2,DELAY)
              0250 ;
              0260 ; where:
              0270 ;
              0280 ;   PORT  - The address of the I/O port
              0290 ;   MASK1 - To isolate the phototransistor
              0300 ;           sensor line
              0310 ;   MASK2 - Not used (Velocity uses
              0320 ;           this argument)
              0330 ;   DELAY - Delay loop variable
              0340 ;
              0350 ;
              0360 ;
              0370 ; The Shutter USR function performs
              0380 ; the following steps:
              0390 ;
              0400 ; Initialize the USR variables
              0410 ; Turn off the Video DMA
              0420 ; Turn off the Real Time Clock
              0430 ; Initialize the loop counter, COUNT
              0440 ; Wait for the sensor line to go low
              0450 ; DOUNTIL the sensor line goes high
              0460 ;   COUNT this time thru the SAMPLE loop
              0470 ;   Delay for DELAY number of WAITLPs
              0480 ; ENDDO
              0490 ; Enable the Real Time CLock
              0500 ; Enable the Video DMA
              0510 ; RETURN the COUNT
              0520 ;
              0530 ;
              0540 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
              0550 ;
              0560 ;
              0570 ; VARIABLE STORAGE LOCATIONS
              0580 ;
              0590 ;
00CB          0600 PORT   =    $00CB      FOR PORT ADDRESS
00CD          0610 MASK1  =    $00CD      TO ISOLATE SENSOR LINE
00CE          0620 DELAY  =    $00CE      DELAY VARIABLE
00D4          0630 COUNT  =    $00D4      LOOP COUNT
0014          0640 RTC    =    $0014      REAL TIME CLOCK (LSB)
022F          0650 SDMACT =    $022F      DMA SHADOW REGISTER
D40E          0660 NMIEN  =    $D40E      NMI ENABLE REGISTER
              0670 ;
              0680 ;
              0690 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
              0700 ;
              0710 ;
0000          0720        *=   $0600      FREE ATARI MEMORY
              0730 ;
              0740 ;
              0750 ; INITIALIZE THE VARIABLES
              0760 ;
              0770 SHUTTER
0600 68       0780        PLA             NUMBER OF USR ARGUMENTS
0601 68       0790        PLA             USR ARGUMENT # 1
0602 85CC     0800        STA  PORT+1     ADDRESS OF I/O PORT
0604 68       0810        PLA
0605 85CB     0820        STA  PORT
```

Listing 3 *(continued)*

```
0607 68                 PLA             USR ARGUMENT # 2
0608 68                 PLA             ONLY ONE BYTE
0609 85CD               STA   MASK1     "AND" MASK FOR SENSOR LINE
060B 68                 PLA             USR ARGUMENT # 3
060C 68                 PLA             -- NOT USED
060D 68                 PLA             USR ARGUMENT # 4
060E 68                 PLA             ONLY ONE BYTE
060F 85CE               STA   DELAY     DELAY LOOP VARIABLE
             ;
             ; DISABLE VIDEO DMA AND RTC
             ;
0611 A900               LDA   #$00
0613 8D2F02             STA   SDMACT    TURN OFF SHADOW REGISTER
0616 A514               LDA   RTC       VBLANK WILL SHUT OFF HARDWARE
0618 C514        TICK   CMP   RTC       WAIT TILL VBLANK EXECUTES
061A F0FC               BEQ   TICK      NOT YET
             ;
061C A920               LDA   #$20      SHUT OFF REAL TIME CLOCK
061E 8D0ED4             STA   NMIEN     (STOP VBLANK INTERRUPT)
             ;
             ; INITIALIZE LOOP COUNTER
             ;
             ; NOTE: THIS VARIABLE IS THE ADDRESS
             ; OF THE 16 BIT VALUE RETURNED BY
             ; THE USR FUNCTION.
             ;
0621 A900               LDA   #$00
0623 85D4               STA   COUNT     A VALUE OF ZERO => ERROR
0625 85D5               STA   COUNT+1
             ;
             ; WAIT UNTIL SHUTTER UNCOVERS THE
             ; PHOTOTRANSISTOR.
             ;
0627 A000               LDY   #$00      FOR INDIRECT INDEXED MODE
0629 B1CB        SENSOR LDA   (PORT),Y  GET I/O PORT
062B 25CD               AND   MASK1     ISOLATE THE SENSOR LINE
062D D0FA               BNE   SENSOR    STILL IN THE DARK
             ;
             ; MEASURE THE SHUTTER TIME
             ;
062F 18          SAMPLE CLC             COUNT THIS LOOP
0630 A5D4               LDA   COUNT
0632 6901               ADC   #$01
0634 85D4               STA   COUNT
0636 A5D5               LDA   COUNT+1
0638 6900               ADC   #$00
063A 85D5               STA   COUNT+1
063C B00E               BCS   RET       PULSE TOO LONG -- RETURN ZERO
063E A6CE               LDX   DELAY     GET DELAY VALUE
0640 F004               BEQ   GETIO     NO DELAY NEEDED
0642 CA          WAITLP DEX             DELAY FOR LONG PULSES
0643 D0FD               BNE   WAITLP    WAIT SOME MORE
0645 EA                 NOP             ADJUST TIMING
0646 B1CB        GETIO  LDA   (PORT),Y  GET I/O PORT
0648 25CD               AND   MASK1     ISOLATE SENSOR
064A F0E3               BEQ   SAMPLE    SHUTTER STILL OPEN
             ;
             ; RESET ATARI
             ;
064C A960        RET    LDA   #$60
064E 8D0ED4             STA   NMIEN     ENABLE REAL TIME CLOCK
0651 A922               LDA   #$22
0653 8D2F02             STA   SDMACT    ENABLE VIDEO DMA
0656 60                 RTS             RETURN COUNT
0657                    .END
```

***Table 1:* Results of SHUTTER**

Shutter Speed (Seconds)	Expected Time (ms)	Measured Time (ms)	Standard Deviation (ms)	Normalized Standard Deviation	Relative Error
1/1000	1.00	1.122	0.048	4.263	−12.2%
1/500	2.00	2.268	0.050	2.208	−13.4%
1/250	4.00	4.359	0.067	1.543	− 9.0%
1/125	8.00	8.516	0.037	0.429	− 6.5%
1/60	16.67	16.842	0.029	0.172	− 1.0%
1/30	33.33	33.276	0.106	0.319	0.2%
1/15	66.67	61.987	1.885	3.041	7.0%
1/8	125.00	126.595	2.652	2.095	− 1.3%
1/4	250.00	269.122	3.855	1.432	− 7.6%
1/2	500.00	534.875	4.295	0.803	− 7.0%
1	1000.00	1024.093	4.841	0.473	− 2.4%

compute the relative error. Since the Shutter function uses the current value of ERROR to correct for the sensor width, the Velocity function must be run prior to the first shutter measurement. (As my camera had a constant curtain speed, only one Velocity function needed to be performed.) The Shutter function prompts for the expected exposure time in milliseconds. After the ten shutter releases, the average value is used to compute the relative error from the expected value. This function utilizes the Atari USR function of listing 3.

Results

The results of SHUTTER as applied to my camera are presented in table 1. As expected, several shutter speeds have rather large relative errors. However, keep in mind that errors as large as 33% (⅓ of an f/stop) are often visually acceptable. Thus, with all shutter errors below 15%, this particular camera is reasonably accurate. Notice that the shutter yields excellent accuracy for 1/60th and 1/30th of a second, two commonly used shutter speeds.

The criteria of consistency is measured by the normalized standard deviation, NSD. All shutter speeds are within 5% consistency with individual bests again going to 1/60th and 1/30th of a second shutter speeds. Over one half of the shutter speeds are consistent to within a 2% NSD. The mechanical consistency of my camera shutter seems to be very good.

In conclusion, my camera passes the SHUTTER test in good condition. The highest shutter speeds and the seldom used ¼th and ½ of a second speeds slightly overexpose the film. Only the 1/15th of a second shutter speed, used for special effects with water subjects, underexposes the film. Since the consistency is excellent, these slight exposure errors may be easily corrected by adjusting the camera lens aperture.

Mike Dougherty works at Martin Marietta Aerospace in Denver, CO. His home-based system presently consists of an Atari 800 with 24K bytes of memory, the Atari 410 recorder, and the Atari 850 Interface Module for future communication with single-board computers. You may contact him at 7659 W. Fremont Ave., Littleton, CO 80123.

MICRO

It's All Relative — Using CBM's Relative Records Part 2

by Jim Strasma

In this second part of his series, Contributing Editor Jim Strasma discusses how to set up relative files and records.

The first article of this series (MICRO 55:37) explained a variety of techniques useful for setting up a program package that uses relative files. In this installment I create a relative file. If you have Chris Bennett's "Mail List 4040," you may want to have it handy as you read.

In part 1, I discussed how to set up most variables needed by the mail list and how to prepare to chain between the various program modules. After the setup module finishes, it chains to a short program called "4040 menu." This serves two functions: to separate the menu, ensuring that the setup module is run only once per use (all other modules chain to and from the menu), and to make sure the menu is short enough to load quickly, as it is called often.

Before you create a relative file, there are two features in the "4040 menu" that you may want to add to other programs. First there is a safety line:

```
1020 IF DI = 0 THEN END: REM Guard
     against cold start here
```

If you were to begin the program at the menu rather than at the setup module, the preset variables would be incorrect and the package would fail. Line 1020 ensures that the program will halt if the module begins with variables cleared.

The other menu feature is an undocumented command. When the menu appears no mention is made of a format module, even though it is callable from the menu. The format module was omitted to protect against accidentally erasing data.

To create a relative file, select the unwritten menu option 3. This loads the program "4040 format," which creates the files needed by the mail-list package. Because the user could get here by hitting the wrong key at the menu, this module begins with its own menu, limited to either returning to the main menu or formatting a new data disk. If the user elects to continue, he is asked to provide a name for the data diskette, which is then completely formatted. The format command in Commodore BASIC 4.0 is HEADER, as seen in line 1240:

```
1240 HEADER D(DD), (F$),IML ON
     U(UN)
```

The BASIC 2 equivalent, also used by VIC and the CBM 64, is:

```
1240 OPEN 15,UN,15
1241 PRINT#15,"N" + MID$(STR$
     (DD),2) + F$ + ",ML"
1242 CLOSE 15
```

Notice that the diskette ID number "ML" is not preset by the setup module. BASIC 4.0 does not allow the ID to be a variable. You may alter it, but only by changing line 1240.

In BASIC 2 you could predefine the ID as ID$ in the setup module. Then, instead of ending line 1241 with '+",ML" ', you would end it with '+"," + ID$'. While you are at it, preset the program and data drive numbers as string variables to avoid constantly referring to them as 'MID$(STR$(drive needed),2)'. I also suggest opening the disk-error channel in the setup module and leaving it open throughout the package since it is used often. To do this, leave out lines 1240 and 1242 above and change line 1241 to 1240.

Several chores within the mail list are handled in machine language. One of the most important of these is user input. Using the ordinary INPUT command of BASIC, it is fairly simple for a user to foul up data by hitting the wrong keys. The mail list prevents this by monitoring the keys as they are hit and excluding those which could cause trouble. The first time the user encounters this new input is in line 1180:

```
1180 SYS IN,64,16,1$
```

This line displays a row of 16 asterisks on the screen, with the leftmost one flashing rapidly. This area is the input field, and users are forced to remain within it until they hit [return]. The 64 is a mask to indicate which characters are allowed in the input. The response is placed in the variable L$. Parameters for the input follow the SYS command. If SYS were left out, the line would cause a SYNTAX ERROR. Fortunately, the machine-language module reads the extra characters and ups the BASIC text pointer so BASIC never sees the illegal syntax. In part 6 of this series, I will explain how this is done and suggest changes for those with BASIC 2.

Meanwhile, those without BASIC 4 may substitute an INPUT statement:

```
1180 INPUT 1$
1181 IF LEN(1$) > 16 THEN ?"[UP]";
     :GOTO 1180
```

The CURSOR UP returns overlong inputs to the original line to be redone.

As soon as the diskette is formatted (about a 20-second process on a 4040 drive) the relative file is created by line 1260:

```
1260 DOPEN#1,(F$),D(DD),L(RL)
     ON U(UN)
```

The L parameter tells DOS that this is a relative file. If the file already exists, it will be opened for use. However, if the file on the diskette has a different value for L than line 1260, a DOS error will result. To prevent any possibility of this, line 1260 is the only place in the mail list where L is set. Everywhere else the value stored by the disk is used. This shortens the program slightly but, more important, it eliminates a potential error.

The value RL contains the desired record length for the file, preset by the setup module. Currently, it is set at 150 characters, including all carriage-return characters. This is not enough to allow the user to fill every field in the record to its maximum length. However, this is rarely a problem since only the part of each field actually used is added to the record length, along with a leading quotation mark and a trailing carriage return. Most addresses can be written easily in under 150 characters, leaving room on the diskette for more records.

If the concept of files, records, and fields is new to you, think of a common 3" x 5" card-file box. The entire box of cards is the file, each card within the box is one record, and each line on a single card is a field. It works the same way on the computer, except that the cards and box are no longer visible.

For those with BASIC 2, the equivalent of line 1260 is:

```
1260 OPEN 1,UN,2,MID$(STR$(DD),2)
     + ":" + F$ + ",1" + CHR$(RL)
```

The secondary address, 2, is of no great significance here. Just be sure you don't assign the same secondary address to disk files that could be opened simultaneously. Note the use of CHR$ to send the record length.

In order for the relative file to work reliably, it is necessary to create the needed records in advance. DOS is able to append new records to the end of a relative file later, but initializing all that are likely to be needed at once avoids some errors, including the possibility of filling the disk. It also ensures that the file will use at least two sectors on the diskette, a necessity for updating file data properly.

Line 1280 defines the maximum record:

```
1280 RECORD#1,(NR)
```

Or, in BASIC 2.0,

```
1280 OPEN 15,UN,15
1281 B2 = INT(RL/256)
1282 BL = RL - 256*B2
1283 PRINT#15,"P" + CHR$(2) + CHR$
     (BL) + CHR$(BH) + CHR$(1)
```

Note that the 2 in CHR$(2) must be the same as the secondary address set in line 1260. B2 and B1 are temporary variables that contain the high- and low-byte values of the number in RL. CHR$(1) tells DOS to point at the first byte within record #RL. If line 1280 in BASIC 4 were changed to

```
1280 RECORD#1,(NR),8
```

then we would use CHR$(8) instead for BASIC 2. In that case, both BASICs would begin to access data at byte number 8 within the record. This option is rarely used.

If record #NR already exists, line 1280 takes only a moment. Here, however, record #NR first has to be created. DOS indicates this by sending a disk status of 50. This is not usually an error, but needs to be handled separately as most disk error-checking lines will consider it an error. Line 1290 does it this way:

```
1290 IF DS < >50 THEN 1510
```

If this is the first time record #NR is created, even a disk status of 0 would be an error for these purposes. If you are using BASIC 2, it is easier to put the disk status check into a single subroutine. Here is a simple example:

```
60000 REM Check disk status
60010 OPEN 15,UN,15
60020 INPUT#15,DS,DS$,ET,ES
60030 CLOSE 15
60040 RETURN
```

If you choose to use this subroutine add it to each module in the mail list. Then call it at the start of each disk error-check line in the package. Line 1290 in this module becomes:

```
1290 GOSUB 60000:IF DS < > 50
     THEN 1510
```

Note that this subroutine cannot be used by those with BASIC 4, as the variables DS and DS$ are reserved for this purpose by BASIC itself.

The next task is to write to the last record of the file. This forces BASIC to create all the records up to and including the last file. When records are newly created they are filled with CHR$(255), the Pi character. This character has a special advantage: when read from the disk it flashes the EOI line of the IEEE bus, signaling to BASIC that the entire record has now been sent from the disk. You can take advantage of this in line 1300 by writing the record with the same character:

```
1300 PRINT#1,PI$:REM Leaves null
```

Normally, writing a record takes only a moment. Here, however, DOS has to first create all the records up to and including number NR and fill them with dummy data. In the standard mail list this function takes about three minutes and creates 1000 records. After completion, the file may be closed:

```
1310 DCLOSE ON U(UN)
```

DCLOSE without a file number closes all disk files on the named unit (on unit 8 if no unit is given). In BASIC 2 the file must be closed by number with:

```
1310 CLOSE 1
```

This concludes the relative-file portion of "4040 header." There are still two other files to be created. These are sequential-index files, which help the mail list find records in the relative file by name rather than number. In BASIC 4 this is done with:

```
1320 DOPEN#1,D(DD),"INDEX",W
     ON U(UN)
```

In BASIC 2 it becomes:

```
1320 OPEN 1,UN,3,MID$(STR$
     (DD),2)+ ":INDEX,S,W"
```

with a similar change in line 1370. Remember also that in BASIC 2 you must end each PRINT# statement to the disk with CHR$(13) and a semicolon; otherwise a linefeed will also be sent to the disk, causing havoc when the file is read. C$ is preset to CHR$(13), so line 1300 becomes:

```
1300 PRINT#1,PI$C$;:REM Leaves null
```

Finally, SYS PA in line 1540 cleans up the stack pointer, erasing all open FOR...NEXT loops and active subroutines. Normally you would not want to do this. However, when chaining, the stack has less chance than usual to clean itself. Without SYS, repeated errors might cause a stack-related OUT OF MEMORY ERROR that would halt the program. If you are not using BASIC 4, choose the correct PA address below and substitute it in the setup module.

Panic Address on Commodore Machines

BASIC 4	$B612	46610
BASIC 2	$C597	50583
BASIC 1	$C588	50568
VIC	$C67E	50814
CBM 64	$A67E	42622

In the next installment of this series, I will look at the largest module in the mail list, the update module, which is responsible for maintaining all the files and data.

You may contact the author at 1280 Richland Ave., Lincoln, IL 62656.

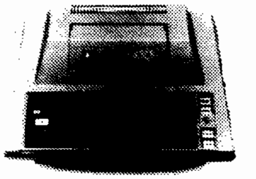
MICRO

NIBBLE *EXPRESS* Vol. I

TABLE OF CONTENTS

SECOND PRINTING! BRAND NEW PROGRAM LISTINGS!

computer mail order west

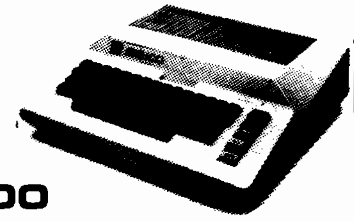


HOME COMPUTERS

ATARI

400

16K........$199
32K.(Non Atari Ram).......$274
48K.(Non Atari Ram).......$359



800

48K........$499

New low price effective January 1, 1983.

410 Recorder	$74.00
810 Disk Drive	$429.00
822 Printer	$269.00
825 Printer	$589.00
830 Modem	$159.00
820 Printer	$259.00
850 Interface	$169.00
CX40 Joysticks (pair)	$18.00
CX853 Atari 16K Ram	$77.95

Microtek 16K Ram	$74.95
Axlon Ramdisk (128K)	$429.95
Intec 48K Board	$159.00
Intek 32K Board	$74.00
One Year Extended Warranty	$70.00
CX481 Entertainer Package	$69.00
CX482 Educator Package	$130.00
CX 483 Programmer Package	$54.00
CX 484 Communicator Package	$344.00

PERCOM

DISK DRIVES FOR ATARI COMPUTERS

S1 Single Drive	$549.00
A1 Add-On Drive	$339.00
S2 Dual Drive	$879.00
Single Side Dual Head	$679.00
Dual Drive Dual Head	$1046.00

μ-SCI

MICRO-SCI
DISK DRIVES FO
FRANKLIN & APPL

A2	$299
A40	$369
A70	$499
C2 Controller	$79
C47 Controller	$89

SOFTWARE FOR ATARI

ATARI

Pac Man	$33	Missile Command	$29
Centipede	$33	Star Raiders	$35
Caverns of Mars	$32	Galaxian	$33
Asteroids	$29	Defender	$33

ON-LINE

Jawbreaker	$27	Mission Asteroid	$22
Softporn	$27	Mouskattack	$31
Wizard & Princess	$29	Frogger	$31
The Next Step	$34	Cross Fire (ROM)	$36

SYNAPSE

File Manager 800	$89	Shamus	$26
Chicken	$26	Protector	$26
Dodge Racer	$26	Nautilus	$26
Synassembler	$30	Slime	$26
Page 6	$19	Disk Manager	$24

DATASOFT

Pacific Highway	$25	Graphic Generator	$13
Canyon Climber	$25	Micro Painter	$25
Tumble Bugs	$25	Text Wizard	$79
Shooting Arcade	$25	Spell Wizard	$64
Clowns & Balloons	$25	Bishop's Square	$25
Graphic Master	$30	Sands of Egypt	$25

EPYX

Crush, Crumble	$24	Morloc's Tower	$16
Undead Crypt	$24	Rescue at Rigel	$24
Curse of Ra	$16	Ricochet	$16
Datestones	$16	Star Warrior	$29
Invasion Orion	$19	Temple Apshai	$29
Arthur's Heir	$24	Upper Reaches	$16

APX

Text Formatter	$18.50	Holy Grail	$24
Family Budget	$18.50	Player Piano	$18.50
Eastern Front	$24	Keyboard Piano	$18.50
Family Cash	$18.50	Number Blast	$13
Jukebox	$13	Frogmaster	$18.50
Downhill	$18.50	747 LandSimul.	$18.50
Outlaw	$18.50	Word Processor	$40

CBS

K-razy Shoot Out	$32	K-razy Antics	$32
K-razy Kritters	$32	K-star Patrol	$32

STICK STAND
$6.99

VISICORP

For Apple, IBM & Franklin

Visidex	$189.00
Visifile	$189.00
Visiplot	$159.00
Visiterm	$89.00
Visitrend/Plot	$229.00
VisiSchedule	$229.00
Desktop Plan	$189.00
VISICALC for Apple II plus, Atari, CBM & IBM	179.00

CONTINENTAL

The Home Acountant(Apple/Franklin)	$59.00
The Home Accountant (IBM)	$119.00
1st Class Mail (Apple/Franklin)	$59.00

SIRIUS

Free Fall	$24	Space Eggs	$24
Beer Run	$24	Sneakers	$24
Snake Byte	$24	Bandits	$28

BRODERBUND

Apple Panic	$23	Arcade Machine	$34
David's Magic	$27	Choplifter	$27
Star Blazer	$25	Serpitine	$27

INFOCOM

Deadline	$35	Zork I	$29
Star Cross	$29	Zork II or III	$29

MPC

Bubdisk (128K Ram)	$719.00

PRINTERS

Smith Corona

TP 1	$599.00

C.ITOH (TEC)

Starwriter (F10-40CPS)	$1399.00
Printmaster (F10-55CPS)	$1749.00
Prowriter 80 Col (P)	$499.00
Prowriter 80 Col (S)	$629.00
Prowriter 2 (132 Col)	$799.00

OKIDATA

82A	$429.00
83A	$659.00
84P	$1079.00
84S	$1199.00

IDS

MicroPrism	$649.00
132 (fully configured)	$1599.00
80 (fully configured)	$1399.00

Call for other configurations.

DAISYWRITER

Letter Quality	$1049.00

DIABLO

620	$1179.00
830	$1849.00

FLOPPY DISKS

MAXELL

MD I (Box of 10)	$32
MD II (Box of 10)	$44
MFD I (8")	$40
MFD II (8" DD)	$50

VERBATUM

5¼" SS DD	$2(
5¼" DS DD	$3(

ELEPHANT

5¼" SS DD	$19.9

MONITORS

AMDEK

300G	$169.0(
Color I	$339.0(
Color II	$699.0(
Color III	$429.0(

BMC

12" Green	$85.0(
13" Color 1400	$279.0(
13" Color 1401 (Mid Res)	$369.0(

ZENITH

ZVM121	$99.0(

SHARP

Sharp 13" Color TV	$275.0

PANASONIC

TR-120 MIP (High Res. Green)	$159.0
CT-160 Dual Mode Color	$299.0

MODEMS

HAYES

Smart	$239.0
Smart 1200 (1200 Baud)	$549.0
Chronograph	$199.0
Micromodem II (with Term)	$309.0
Microdem 100	$309.0

NOVATION

Cat	$144.0
D-Cat	$159.0
Auto Cat	$219.0
212 Auto Cat	$589.0
Apple Cat II	$279.0
212 Apple Cat II	$609.0

ANCHOR

Mark I (RS-232)	$79.0
Mark II (Atari)	$79.0
Mark III (TI-99)	$109.0
Mark IV (CBM/PET)	$125.0
Mark V (OSBORNE)	$95.0
Mark VI (IBM-PC)	$179.0
Mark VII (Auto Answer Call)	$119.0
TRS-80 Color Computer	$99.0
9 Volt Power Supply	$9.0

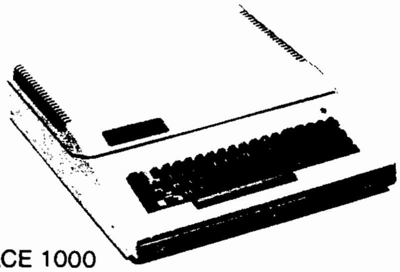
west **800-648-3311** wes

IN NV. CALL (702) 588-5654

P.O.BOX 6689 STATELINE, NV. 89449

INTERNATIONAL ORDERS: All shipments outside continental United States must be pre-paid by certified check only! Include 3% (minimun $3.00) shipping and handling. **EDUCATIONAL DISCOUNTS:** Additional discounts are available from both Computer Mail Order locations to qualified Educational Institutions.

FRANKLIN ACE 1000 SYSTEM



ACE 1000
ACE 10 with Controller Card
ACE Writer Word Processor
CALL...
FOR SYSTEM PRICE.

IBM PC

NEC 3550 Printer $2099

PERCOM DRIVES

5¼" 160K Disk Drive	$329
5¼" 320K Disk Drive	$449

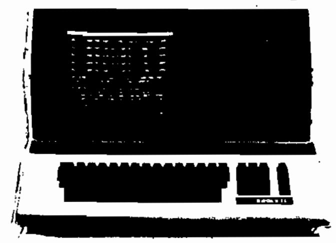
AMDEK

310A Amber Monitor	$179
Amdisk(3¼") Drive	$729
DXY Plotter	$759

SOFTWARE

I.U.S. Easywriter II	$249
I.U.S. Easyspeller	$129
Peachtree Peach Pak (GL/AP/AR)	$419
MPC Bubdisk	call

EAGLE

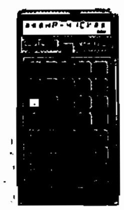


64K RAM
780 KB Disk Storage
Word Processing, Ultracalc CP/M
C-Basic Software
Smith Corona TP1
Letter Quality Printer
$2995.00
Retail Value $4895.00

RANA DISK DRIVES

Call for price and availability on the new Rana Disk Drives for the Apple and Franklin Computer Systems

HEWLETT PACKARD HP 41CV CALCULATOR $209



HP 41C	$149.00
HP 10C	$69.00
HP 11C	$79.00
HP 12C	$114.00
HP 15C	$109.00
NEW 16C	$114.00

HEWLETT PACKARD HP•85 $1969

HP-125	$1969.00
HP-85 16K Memory Module	$169.00
5¼"Dual Master Disk Drive	$1799.00
Hard Disk w/Floppy	$4349.00
Hard Disk	$3549.00
"Sweet Lips" Printer	$1199.00
80 Column Printer	$649.00

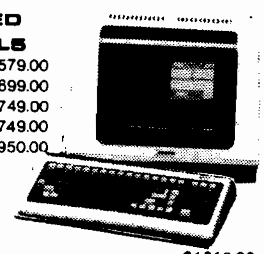
commodore

8032	$999.00
CBM 64	CALL
4032	$749.00
8096 Upgrade Kit	$369.00
Super Pet	$1599.00
2031	$369.00
8250 Double Sided Disk Drive	$1699.00
D9060 5 Megabyte Hard Disk	$2399.00
D9060 7.5 Megabyte Hard Disk	$2699.00
8050	$1299.00
4040	$969.00
8300 (Letter Quality)	$1549.00
8023	$599.00
4022	$399.00
New Z-Ram, Adds CP/M and 64K Ram	$549.00
The Manager	$209.00
Magis	CALL
Word Pro 5 plus	$319.00
Word Pro 4 plus	$299.00
Word Pro 3 plus	$199.00
The Administrator	$379.00
InfoPro Plus	$219.00
Power	$79.00
VIC 20 Dust Cover	$6.99
CBM 8032 Dust Cover	$14.99
CBM 8050/4040 Dust Cover	$10.99

SHARP PC-1500 POCKET COMPUTER $209

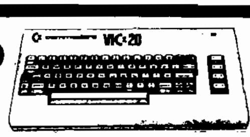
CE 150 Printer, Plotter and Cassette Interface Unit	$172.00
CE152 Cassette Recorder	$69.00
CE 155 8K Ram Expansion Module	$94.00

TELEVIDEO TERMINALS



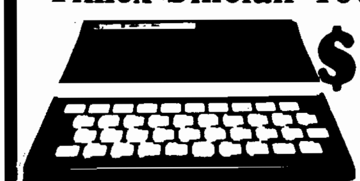
910	$579.00
912C	$699.00
920C	749.00
925C	$749.00
950	$950.00
800A	$1319.00
802	$2649.00
802H	$4695.00
806	$5495.00
816	$9495.00

VIC 20 $179



VIC 1530 Commodore Datassette	$69.00
VIC 1540 Disk Drive	$339.00
VIC 1541 (64 Disk Drive)	CALL
VIC 1525 Graphic Printer	$339.00
VIC 1210 3K Memory Expander	$32.00
VIC 1110 8K Memory Expander	$53.00
VIC 1111 16K Expansion	$94.00
VIC 1011 RS232C Terminal Interface	$43.00
VIC 1112 VIC IEEE-488 Interface	$86.00
VIC 1211 VIC 20 Super Expander	$53.00
VIC Mother Board	$99.00

Timex Sinclair 1000 $89



16K Memory Module	$44.95

Vu-Calc	$17.95	The Organizer	$14.95
Super Math	$12.95	The Budgeter	$13.95
Check Book Manager	$13.95	Stock Option	$14.95
Loan & Mortgage Amortizer	$12.95		

NEC

COMPUTERS

8001A	$729.00
8031	$729.00
8012	$549.00

PRINTERS

8023	$499.00
7710/7730	$2399.00
3510/3530	$1599.00

MONITORS

JB-1260	$129.00
JB-1201	$159.00
JC-1201	$319.00
JC-1203	$729.00

800-233-8950

east — east

IN PA. CALL (717) 327-9575

477 E. THIRD ST., WILLIAMSPORT, PA. 17701

In-stock items shipped same day you call. No risk, no deposit on C.O.D. orders. Pre-paid orders receive free shipping within the continental United States with no waiting period for certified checks or money orders. Add 3% (minimum $3.00) shipping and handling on all C.O.D. and Credit Card orders. NV and PA residents add sales tax. All items subject to availability and price change. **NOTE:** We stock manufacturer's and third-party software for most all computers on the market! Call today for our new catalogue.

computer mail order east

Microcomputer Design of Transistor Amplifiers

by Andy Cornwall

The class A transistor amplifier is the most common circuit in analog electronics. This article presents a BASIC program that takes the mystery and tedium out of designing practical small signal amplifier stages.

Amplifier Designer
requires:
BASIC
(written for PET, but easily convertible)

Computer hobbyists often have a wide interest in electronics. It is not unusual for a home microcomputer to be used for electronic circuit design. The program described here is for the design of a transistor, class A, small signal amplifier stage. Essentially, you tell the program what you want the amplifier to do, and component values and amplifier operating parameters (i.e., voltages, currents, and impedences) are calculated in return. At the least, the program removes the need for tedious calculations, and you need only a minimal knowledge of transistor amplifier theory. However, the program also provides simulation capability to determine quickly how changes in amplifier specifications alter component values and parameters.

Class A

The small signal, class A amplifier is the most common building block in analog electronics. It is basically a voltage amplifying device with a low voltage signal going in and a corresponding higher voltage signal coming out. If the gain from one amplifying stage is not enough, two or more can be chained together (or cascaded). The term "small signal" in this context means that the maximum output of the amplifer stage is usually less than ½ watt. High-power class A amplifier stages can be designed using this program, but more efficient, though more complex, circuits are available for high-power situations.

Generalized Model

Figure 1 is a schematic diagram of the amplifier. The circuit is intended to amplify AC inputs, such as audio, radio, or television signals. In addition to an NPN transistor, there are eight components: collector resistor (R_c), emitter resistor (R_e), voltage adjusting resistor (R_v), two base bias resistors (R_b and R_x), and three capacitors (input, output, and bypass). The values of these eight components are calculated by the program. Also included in the schematic is load impedence (R_l), the input impedence of the circuit or

Listing 1

```
80 REM VERSION EXP AMP18
90 REM** A PROGRAM FOR CLASS 'A'
100 REM** AMPLIFIER DESIGN.
110 REM**********************************
120 REM** BY ANDREW CORNWALL           **
130 REM**    66 LANDRACE CRES.         **
140 REM**    DARTMOUTH, NOVA SCOTIA    **
150 REM**    CANADA                    **
160 REM**********************************
170 K$="WITH LOAD"
180 FL$="*GAIN REDUCED BY "
190 SL$=" FOR HIGHER GAIN "
200 C$(1)="LOW TRANS.'B'."
210 D$(1)="RAISE TRANS.'B'."
220 C$(2)="TRANS.INTERN.RESIS."
230 D$(2)="LOWER OUT.IMPEDENCE"
240 C$(3)="LOW SUPPLY VOLTS."
250 D$(3)="RAISE SUP.VOLT."
260 NL$="*NON-OPERATING CONDITION.TO CORRECT"
270 N$(1)=" LOWER INPUT VOLTAGE/RAISE SUP. VOLT."
280 N$(2)=" LOWER VOLT.OUT/RAISE SUP.VOLT."
290 MV=.5:REM APPROX MIN VCE
300 J=10 :REM BIAS RES. CURRENT FACTOR
310 VP=.030 :REM EMIT. VOLT. FACTOR
320 PRINT"*** CLASS A AMPLIFIER DESIGN ***"
330 PRINT"ENTER AMPLIFIER SPECIFICATIONS:"
340 INPUT"SUPPLY VOLTAGE";VCC
350 INPUT"TRANSISTOR 'B'";B
360 INPUT"OUTPUT IMPEDENCE ('000 OHMS)";RC:RC=RC*1000
370 PRINT"LOAD IMPEDENCE ('000 OHMS) - ENTER"
380 INPUT"ZERO IF UNKNOWN";RL:RL=RL*1000
390 IF RL=0 THEN RL=10↑8:K$="WITHOUT LOAD"
400 RK= RC*RL/(RC+RL) :REM CALC.OUT.IMP.WITH LOAD.
410 INPUT"LOWEST SIGNAL FREQ (HERTZ)";F
420 INPUT"MAX. INPUT VOLTAGE SWING";VI
430 INPUT"MAX. OR SELECT GAIN (M/S)";G$
440 IF G$="M" THEN VO=B*VI: GOTO 480
450 IF G$<>"S"THEN 430
460 PRINT"MAX. OUTPUT VOLTAGE SWING ( WITH"
470 INPUT"LOAD IF GIVEN ABOVE)";VO
480 G=VO/VI :REM DESIRED GAIN
490 REM CHECK GAIN FOR'B',REDUCE GAIN AND VO IF'B'TOO LOW.
500 IF G>B THEN G=B:VO=B*VI:FLAG=1
510 REM PRELIMINARY SUPPLY VOLTAGE CONDITION CHECK.
520 IF VO>VCC-VI-MV THEN VO=VCC-VI-MV:G=VO/VI:FLAG=3
530 GO=G :REM LIMIT SET ON GAIN BY AMP. SPECS OR TRANSISTOR 'B'.
```

(continued)

device the amplifier stage will be feeding. Load impedence is a specification requested by the program.

Choice of Transistor

The transistor you choose must be NPN silicon (the most common type) and suitable for amplifying. The only transistor specification used in the program is static (or DC) current gain, referred to as either β (called beta) or hfe. However, if your amplifier will be handling high-frequency signals (over 1 MHz) connected to high supply voltage (over 25 V, or so), or dissipating high power (over about 100 mW, as calculated by the program), you should be sure the transistor you intend to use can handle these extremes. In general, the "five for a dollar," NPN, small signal, amplifiying transistors should be suitable for most applications.

Using the Program

To use the program you will have to input the amplifier specifications described below.

Supply Voltage (V_{cc})

Choice of supply voltage depends on the battery or other power source you want to use. Minimum practical supply voltage is about three volts.

Transistor β

Beta determines the maximum potential gain of the amplifer stage and influences the biasing characteristics. Unfortunately, transistor beta can be an enigma; even transistors with the same component number have different betas. Some transistor specifications mention minimum beta, others refer to 'typical' beta, and grab-bag transistors frequently come with no indication. When in doubt, you are probably safe to assume a value of 50 to 100.

Output Impedence

When you specify output impedence you are actually determining the value of R_c. Usually, output impedence is set equal to or less than load impedence. If in doubt, you might try setting the output impedence at 100 to 500 times the supply voltage.

Load Impedence

All else being equal, the effect of load impedence is to reduce voltage gain. If load impedence is entered into the program, the program compensates

Listing 1 *(continued)*

```
540 AL=B/(B+1):REM BETA FACTOR USED FOR CALC. RE.
550 REM TABLE HEADING FOR RUN MONITOR;TABLE VALUES PRINTED
    DURING CALCULATION.
560 PRINT"■VO          MV          VE          VV"
570 DEF FNM(X)=INT(X*100)/100
580 IE=0:RE=0
590 REM CHECK FOR NON-OPERATIONAL OUTPUT VOLTAGE CONDITION
600 IF VO<VI THEN NFLAG=1:GOTO1070
610 REM PRINT MONITOR VALUES.
620 PRINT FNM(VO),FNM(MV),FNM(VE),FNM(VV)
630 REM CALCULATE CURRENTS.
640 IQ=.5*(VO+2*MV)/RK
650 IB=IQ/B
660 IE=IQ+IB
670 REM CALC.TRANS.INTERNAL RESISTANCE RP AND RESISTOR RE.
680 RP=VP/IE
690 RE=AL*RK/G - RP
700 REM CHECK FOR TRANS.INTERNAL RESIS.COND.
710 REM IF RE IS NEG.,LOWER GAIN AND VO;REFIGURE.
720 IF RE<-.1 THEN G=AL*RK/RP:FLAG=2:VO=G*VI:GOTO 580
730 IF RE<0 THEN RE=0
740 REM CALC.VOLTAGES VTEQ,VK,AND VE.
750 VTEQ=IQ*RK+IE*RE
760 VK=VCC-IQ*RC
770 VE=VK-VTEQ-VP
780 REM CHECK TO SEE THAT VE ALLOWS INPUT VOLTAGE SWING.
790 REM IF NOT LOWER GAIN AND VO (INCREASING VE),REFIGURE.
800 IF VE>=VI/2 THEN 820
810 IF VE<VI/2 THEN VO=VO*.95:G=VO/VI:FLAG=3:GOTO 580
820 REM CALC. VOLTAGE VV.
830 VV=VE-IE*RE
840 REM CHECK FOR SUFFICIENT SUPPLY VOLTAGE.
850 REM IF VV IS NEG.,LOWER GAIN AND VO (INCREASING VV);REFIGURE.
860 IF VV<-.0001 THEN VO=VO+VV:G=VO/VI:FLAG=3:GOTO580
870 IF VV<0 THEN VV=0
880 REM IF GAIN REDUCED BY INTERNAL RESISTANCE,ATTEMPT TO RAISE
    GAIN BY
890 REM INCREASING MV (INCREASING IQ);REFIGURE.
900 IF FLAG=2 AND G>GT THEN MV=MV+VV/4:GT=G:G=GD:VO=G*VI:FLAG=0
    :GOTO580
910 REM CALC. RESISTOR RV.
920 RV=VV/IE
930 REM CALCULATE BIAS VOLTAGE RESISTORS.
940 VB=VE+.7
950 RB=(VCC-VB)/((J+1)*IB)
960 RX=VB/(J*IB)
970 REM CALCULATE INPUT IMPEDENCE
980 ZIN=1/(1/RX+1/RB+AL/(B*(RE+RP)))
990 REM CALCULATE CAPACITOR VALUES
1000 DEF FNM(X)=1/(2*π*F*X)
1010 CI=FNM(ZIN)
1020 IF RV<1 THEN CB=0:GOTO 1060
1030 RA=RE+RP+(RB*RX)/(B*(RB+RX))
1040 RR=RA*RV/(RA+RV)
1050 CB=FNM(RR)
1060 IF K$="WITH LOAD" THEN CL=FNM(RL)
1070 REM DISPLAY COMPONENT VALUES
1080 DEF FNO(X)=INT(X*100)/100000
1090 DEF FNR(X)=INT(X/10)/100
1100 DEF FNC(X)=INT(10↑8*X)/100
1110 DEF FNI(X)=INT(X*10↑6)/1000
1120 DEF FNV(X)=INT(1000*X)/1000
1130 PRINT"■■■■■COMPONENT VALUES■"
1140 PRINT"RESISTORS ('000 OHMS):
1150 PRINT"RC      RE      RV        RB      RX"
1160 PRINT FNR(RC);TAB(6)FNO(RE);TAB(14)FNO(RV);TAB(24)FNR(RB);
1170 PRINT TAB(32)FNR(RX);"■"
1180 PRINT"CAPACITORS (MICROFARADS):"
1190 PRINT"   INPUT      OUTPUT      BYPASS"
1200 PRINT TAB(2)FNC(CI);TAB(12)FNC(CL);TAB(23)FNC(CB)
1210 REM FLAG NON-OPERATIONAL CONDITION
1220 IF RV<O THEN NFLAG=2
1230 IF NFLAG>0 THEN PRINT"■"NL$:PRINT N$(NFLAG)
1240 PRINT TAB(7)"■■PRESS SPACE TO CONTINUE"
1250 GETA$:IFA$<>" "THEN 1250
1260 REM DISPLAY PARAMETERS
```

(Continued)

Listing 1 *(continued)*

```
1270 PRINT"■■OPERATING PARAMETERS"
1280 PRINT"■IMPEDENCES ('000 OHMS):"
1290 PRINT"INPUT    OUTPUT   OUTPUT (WITH LOAD)"
1300 PRINT FNR(ZIN);TAB(7)FNR(RC);
1310 IF K$="WITH LOAD" THEN PRINT TAB(16)FNR(RK)
1320 PRINT
1330 PRINT"■QUIESCENT CURRENT (MILLIAMPS):"
1340 PRINT"COLLECTOR     BIAS"
1350 PRINT FNI(IQ);TAB(13)FNI(IB)
1360 PRINT"■QUIESCENT VOLTAGES:"
1370 PRINT"SUPPLY   COLLECTOR   EMITTER   BASE"
1380 PRINT FNV(VCC);TAB(8)FNV(VK);TAB(20)FNV(VE);TAB(30)FNV(VB);"■"
1390 PRINT"SIGNAL VOLTAGES (MAX. SWING):"
1400 PRINT"INPUT    OUTPUT"
1410 PRINT VI;TAB(7)FNV(VI*G);"  ";K$;"■"
1420 PRINT TAB(7)"■■PRESS SPACE TO CONTINUE"
1430 GETA$:IFA$<>" "THEN 1430
1440 REM DISPLAY PARAMETERS
1450 PRINT"■■■GAIN: ";FNV(G);"   ";K$;"■"
1460 PRINT"MIN. DESIGN FREQ. (HERTZ) ";F
1470 PRINT"■MAX. TRANS. PWR. DIS. (MW'S)";FNI(VTEQ*IQ)
1480 IF NFLAG>0 THEN 1500
1490 IF FLAG>0 THEN PRINT:PRINT FL$;C$(FLAG):PRINT SL$;D$(FLAG)
1500 INPUT"■■ENTER 1 TO SEE COM. VALUES; 0 TO END"; P
1510 IF P=1 THEN 1070
1520 IF P<> 0 THEN 1500
1530 END
```

for potential loss of gain when calculating resistor values. Enter zero if you do not know load impedence; the default load resistance is 10K megohms, which is virtually no load at all.

Lowest Signal Frequency

This frequency helps determine capacitor values. The lowest hi-fidelity audio frequency is about 20 Hz, and lowest AM radio audio is about 100 to 200 Hz. Non-audio signals are considerably higher; for example, the low end of the AM broadcast band is 540,000 Hz.

Input Voltage Swing

Input voltage swing is the peak-to-peak value of input voltage. If, for instance, an input signal varies from +0.1 V to −0.1 V, the peak-to-peak voltage swing is 0.2 V.

Maximum or Select Gain

Choosing maximum gain automatically sets gain at its maximum value, given supply voltage, output impedence, and transistor characteristics (mainly beta, but also V_p described below).

Output Voltage Swing

If you choose "select gain" you will have to specify the peak-to-peak output. You'll get increased input impedence by reducing gain below maximum, but you may want to set output voltage somewhat higher than required to provide some design head room. In any case, output voltage swing must be somewhat less than the supply voltage. Also, output voltage swing cannot imply a gain higher than transistor beta. (The gain of the amplifier is defined as output voltage divided by input voltage.)

The Program Takes Over

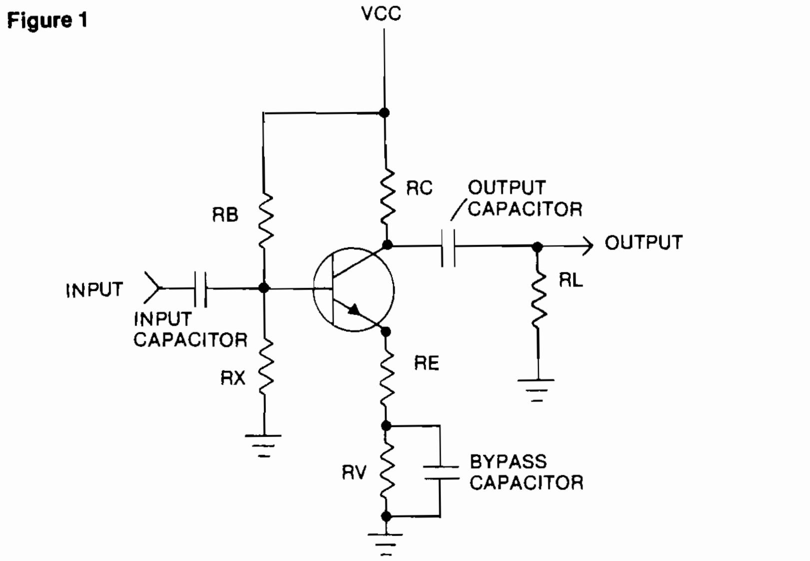
Despite your efforts to enter logical amplifier criteria, there may be inconsistencies that will lower your output voltage swing (or gain) specification. You can observe the iterative calculation process with the "run monitor" feature. If output voltage has been reduced, there will be a suggestion on how to recover gain at the end of the program output. When you indicate maximum gain, there will be a similar message on how to increase gain. There are inconsistencies that the program does not fix, however. These will make R_v negative, or drop gain below one. When either of these conditions occurs, you will receive a "non-operating condition" message along with a suggestion on how to overcome the problem.

Final Pointers

Now that you are ready to start designing amplifiers, here are a few tips.

- Try to have quiescent collector current of at least 1 mA. With some transistors, considerably higher collector current is desirable. To increase collector current, lower output impedence.
- If the design value of R_v happens to be near zero, this resistor and the bypass capacitor are not needed. Similarly, R_e is not needed if its value is near zero.
- When cascading amplifier stages, the output capacitor of one is the input capacitor of the other. Only one capacitor is needed to link the stages.
- You will find that standard values of commercially available resistors and

Figure 1



Sample Run

```
*** CLASS A AMPLIFIER DESIGN ***

ENTER AMPLIFIER SPECIFICATIONS:


SUPPLY VOLTAGE? 14

TRANSISTOR 'B'? 250

OUTPUT IMPEDENCE ('000 OHMS)? 2

LOAD IMPEDENCE ('000 OHMS) - ENTER
ZERO IF UNKNOWN? 30

LOWEST SIGNAL FREQ (HERTZ)? 20

MAX. INPUT VOLTAGE SWING? .5

MAX. OR SELECT GAIN (M/S)? S

MAX. OUTPUT VOLTAGE SWING ( WITH
LOAD IF GIVEN ABOVE)?'10


COMPONENT VALUES

RESISTORS ('000 OHMS):
RC      RE       RV       RB     RX
 2      .08318   .71758   84.77  26.06


CAPACITORS (MICROFARADS):
   INPUT      OUTPUT      BYPASS
   .73        .26         57.05

         PRESS SPACE TO CONTINUE


OPERATING PARAMETERS

IMPEDENCES ('000 OHMS):
INPUT    OUTPUT    OUTPUT (WITH LOAD)
 10.77   2         1.87


QUIESCENT CURRENT (MILLIAMPS):
COLLECTOR       BIAS
 2.933          .011


QUIESCENT VOLTAGES:
SUPPLY    COLLECTOR    EMITTER   BASE
 14       8.133        2.358     3.058


SIGNAL VOLTAGES (MAX. SWING):
INPUT    OUTPUT
 .5      10  WITH LOAD


         PRESS SPACE TO CONTINUE


GAIN:  20    WITH LOAD

MIN. DESIGN FREQ. (HERTZ)   20

MAX. TRANS. PWR. DIS. (MW'S) 16.852

ENTER 1 TO SEE COM. VALUES; 0 TO END? \
```

capacitors seldom match component design values. For resistors, use the closest available commercial value. In theory, the capacitors used should be equal to or larger than design values, but somewhat lower value capacitors should be acceptable for most purposes.

- It is likely that an actual amplifier stage will not have measured voltages, currents, and gain exactly as calculated by the program. Such deviation is to be expected considering components used will not exactly match design values. Also, actual transistor beta probably differs from that used in the program. However, unless there are large differences (greater than 20%, or so) between measured and design parameters, the amplifer should do the job you want.

Example

An example amplifier stage design is shown in the sample run. The objective of the design is to interface an AM/FM tuner module with a power amplifier. For full output the power amplifier requires asignal of 10 V peak-to-peak, but the tuner only provides output of ½ V. Input impedence of the power amplifier (which is the load impedence for the amplifier stage) is about 30,000 ohms. The power amplifier's own power supply can be tapped to obtain a V_{cc} of 14 V.

Try Experimenting

At the start of the program, variables MV, J, and VP are defined. The values in the program listing should be reliable in most situations, but you might want to experiment by changing the program values of these variables. The purpose of each value is mentioned in a REM statement. The value of MV should be no lower than the minimum (or saturation) collector-to-emitter voltage of your transistor. One-half volt will be sufficient for most transistors. J controls the sum of resistors R_b and R_x. A lower value of J increases R_b and R_x. This raises input impedence but makes proper transistor biasing more sensitive to the value of beta. If you are fairly certain about the beta of your transistor, you can lower J. Conversely, increasing J makes knowing beta less important. If you plan to run your transistor near its maximum output rating, beta will drop as the transistor gets hotter. In situations of variable or uncertain beta, use a higher value for J to increase bias stability.

Variable VP relates to emitter diode voltage drop of a transistor. Small signal transistors have an emitter voltage drop of .025 V, while power transistors may have a drop in the range of 0.5 V.

References

The following books are helpful if you want to learn about transistor amplifier theory.

1. Malvino, Albert Paul, *Electronics Principles*, McGraw-Hill, 1973.
2. Oleksey, Jerome E., *Practical Solid-State Circuit Design*, Howard W. Sams & Co., Inc., 1976.
3. Turino, Jon L, *Solid-State Circuits for Hobbyists and Experimenters*, Howard W. Sams & Co., Inc. 1975.

Andy Cornwall is an electronics hobbyist. After acquiring a Commodore PET he delved into computer design of electronic circuitry. You may contact him at 66 Landrace Cres., Dartmouth, Nova Scotia B2W 2P9.

MICRO

MICRO™

Apple Slices

By Tim Osborn

Last month's Apple Slices presented a fast method to find an element in an ascending ordered array using a binary search technique. This month's program, ALTERNATE INDEX, expands the capabilities of BINARY-SEARCH by creating ascending ordered arrays containing any substring of the base data array. This process allows you to declare any substring of the base array as a key, sort an array of these keys in ascending order, and search with BINARY-SEARCH on that key. You can then look back in the base array to find the full string value.

One possible application of this system would be to let a user quickly find information on a part, either by part number or part name. The base array would contain a list of parts and related information, including part number and part name. The array would be processed by ALTERNATE-INDEX twice — once for the part number key and once for the part name key. Once the key arrays are generated, the user would supply a part name or number. The system would look in the proper key array for a match, using BINARY-SEARCH. Once the item is found in the key array, the system can locate the element in the base array to display all the related data.

How ALTERNATE-INDEX Works

Because of the way Applesoft string arrays are stored in memory, it is possible to have two or more arrays that contain the same data without duplicating this data. Applesoft string arrays are actually a table of pointers to the related data, which is stored elsewhere in memory. Therefore, it is possible to build arrays with pointers to the same data contained in another array. With every pointer there is also a length field, which is the length of the string for that array element. By manipulating the length field and pointer, ALTERNATE-INDEX builds pointers to substrings of the base array, then sorts this key array in ascending order. The syntax is:

& S(XX$,YY$,B,E,ZZ%)

where

1. XX$ is the base array (any legal string array name).
2. YY$ is the key array (any legal string array name).
3. B is an Applesoft arithmetic expression that represents the beginning position of the key in the base array. This is relative to the beginning of the base string so that 0 is the first byte and (LEN(XX$(n) - 1) is the last byte of the base array element n. If B is greater than (LEN(XX$(n) - 1), then a null key element will result.
4. E is the ending position of the key. It can be replaced with any legal Applesoft arithmetic expression. If B is greater than E then a syntax error will result. If E is greater than (LEN(XX$(n) - 1) for any element n in the base array, then the value of (LEN(XX$(n) - 1) will be substituted for E.
5. The value contained in ZZ%(n) will be the element number in XX$ where the key in YY$(n) can be found as a substring. Any legal integer array name can be substituted for ZZ%.
6. The number of elements contained in each array must be the same or a syntax error will result.
7. All arrays must be one-dimensional or a syntax error will result.

ALTERNATE-INDEX is programmed so that you can use the & GET command of BINARY-SEARCH directly. If BINARY-SEARCH is not in memory upon encountering the & GET command, a syntax error message will be produced. To use & GET with ALTERNATE-INDEX you must BLOAD BINARY-SEARCH instead of BRUNing it.

If you will be using arrays that may contain duplicate keys, or wish to find the first key higher or lower than the search key, you can write your own serial search routine instead of using BINARY-SEARCH. The advantage of using ALTERNATE-INDEX in these cases is that you only go through the overhead of pulling out the key substring once instead of each time you search for a value.

To access an element in the base array, search the key array (either with BINARY-SEARCH or your own routine) for a desired value. Once the key element is found you then take the value found in the integer array element (ZZ%) that has the same element number as this key element. This is the number of the element in the base array that contains the desired data. For example, say we used the following statement to build the alternate index:

& S(QQ$,RR$,2,5,RR%)

and there is an element QQ$(n) that contains the following string:

"504134WIREHOUSINGS"

There would be an element RR$(i) that points at the substring "4134" and an element RR%(i) that contains the integer value n. We then use this integer to access the proper element in the base array.

Once the alternate index is built, elements in the base array can be found very quickly when BINARY-SEARCH is used to locate the keys in the key array. An even bigger plus to this system is that it allows you to access quickly the same data with more than one key substring without duplicating the data.

Subroutine Hints

ALTERNATE-INDEX can be used to sort the base array by specifying the base array as the key array and specifying a B value of 0 and an E value of 255. ALTERNATE-INDEX is set up to load at $90AF, so HIMEM should be set at 37038 or lower. ALTERNATE-INDEX, is designed to run on a 48K Apple II with MAXFILES set at three or less.

Because of space limitations I did not go into detail on how the subroutine works. If you have any questions please contact me at 62 Clement St., Manchester, NH 03102.

```
                 **********************************
                 *  APPLE SLICES BY TIM OSBORN   *
                 *A L T E R N A T E   I N D E X *
                 **********************************
                 ;
                 ;          <   EQUATES   >
                 ARRY1PTR EPZ $5Ø         ;WORK POINTER FOR SRC. ARRAY
                 PAIR1AD  EPZ ARRY1PTR    ;REUSE ARRY1PTR
                 ARRY2PTR EPZ $52         ;WORK POINTER FOR DEST. ARRAY
                 ARRY3PTR EPZ $54         ;WORK POINTER FOR INT. ARRAY
                 LOWTR    EPZ $9B         ;APPLESOFT WORK PTR
                 PAIR2AD  EPZ LOWTR       ;REUSE LOWTR FOR INTERNAL PURP
                 CHRGET   EPZ $B1         ;A-SOFT'S ROUTINE TO GET A BYTE
                 ;
                 AMPERV   EQU $3F5        ;AMPERSAND VECTOR LOCATED HERE
                 CHKOPN   EQU $DEBB       ;CHECK FOR OPEN QUOTE
                 GETARYPT EQU $F7D9       ;ROUTINE TO FIND ARRAY DESC
                 CHKCOM   EQU $DEBE       ;CHECK FOR COMMA
                 SYNERR   EQU $DEC9       ;DISPLAY SYNTAX ERROR
                 DATA     EQU $D995       ;ADV TXTPTR TO END OF STMT
                 FRMNUM   EQU $DD67       ;EVAL ARITH EXP.,PUTS IN FAC
                 CONINT   EQU $E6FB       ;CONVTS FAC TO INT,PUTS IN X
                 GET      EQU $941Ø       ;ENTRY TO BIN SEARCH ROUTINE
                 ;
                          ORG $9ØAF
                          OBJ $8ØØ        ;FOR LISA
90AF A9 4C       SETVEC   LDA #$4C        ;JUMP ABSOLUTE
90B1 8D F5 Ø3             STA AMPERV
90B4 A9 BF                LDA #ENTRY      ;LSB OF ENTRY ADDRESS
90B6 8D F6 Ø3             STA AMPERV+1
90B9 A9 9Ø                LDA /ENTRY      ;MSB OF ENTRY ADDRESS
90BB 8D F7 Ø3             STA AMPERV+2
90BE 6Ø                   RTS
90BF C9 53       ENTRY    CMP #$53        ;CHECK FOR SORT ''S''
90C1 FØ 1B                BEQ SRTENTRY
90C3 C9 BE                CMP #$BE        ;CHECK FOR GET COMMAND
90C5 DØ Ø9                BNE ENTRYERR
90C7 A9 2Ø                LDA #$2Ø        ;IF GET, MAKE SURE
90C9 A2 ØØ                LDX #ØØ
90CB DD 1Ø 94             CMP GET,X       ;BINARY SEARCH IS IN MEMORY
90CE FØ Ø3                BEQ ENTRY1
90D0 4C C9 DE    ENTRYERR JMP SYNERR      ;SYNTAX ERROR
90D3 A9 B1       ENTRY1   LDA #$B1        ;CHRGET ROUTINE
90D5 E8                   INX
90D6 DD 1Ø 94             CMP GET,X
90D9 DØ F5                BNE ENTRYERR
90DB 4C 1Ø 94             JMP GET
90DE A5 5Ø       SRTENTRY LDA ARRY1PTR    ;SAVE ZEROPAGE LOCATIONS
90E0 8D 64 93             STA ZEROSV
90E3 A5 51                LDA ARRY1PTR+1
90E5 8D 65 93             STA ZEROSV+1
90E8 A5 52                LDA ARRY2PTR
90EA 8D 66 93             STA ZEROSV+2
90ED A5 53                LDA ARRY2PTR+1
90EF 8D 67 93             STA ZEROSV+3
90F2 A5 54                LDA ARRY3PTR
90F4 8D 68 93             STA ZEROSV+4
90F7 A5 55                LDA ARRY3PTR+1
90F9 8D 69 93             STA ZEROSV+5
90FC 2Ø B1 ØØ             JSR CHRGET      ;GET NEXT CHARACTER
90FF 2Ø BB DE             JSR CHKOPN      ;SHOULD BE ''(''
9102 2Ø D9 F7             JSR GETARYPT    ;GET SOURCE ARRAY DESC.
9105 2Ø 6A 93             JSR CHKONE      ;SHOULD BE A 1-DIM ARRAY
9108 A5 9B                LDA LOWTR       ;SAVE ARRAY DESC. ADDRESS
910A 8D 54 93             STA SAVARRY1
910D A5 9C                LDA LOWTR+1
910F 8D 55 93             STA SAVARRY1+1
9112 2Ø BE DE             JSR CHKCOM      ;CHK FOR COMMA+LOAD NXT BYTE
9115 2Ø D9 F7             JSR GETARYPT    ;GET DEST. ARRAY DESC
9118 2Ø 6A 93             JSR CHKONE      ;SHOULD BE A 1-DIM ARRAY
911B A5 9B                LDA LOWTR       ;SAVE ARRAY DESC. ADDRESS
911D 8D 56 93             STA SAVARRY2
9120 A5 9C                LDA LOWTR+1
9122 8D 57 93             STA SAVARRY2+1
9125 2Ø BE DE             JSR CHKCOM
9128 2Ø 67 DD             JSR FRMNUM      ;EVAL STARTING POS EXPRESS
912B 2Ø FB E6             JSR CONINT      ;CONVERT TO INTEGER,PUT IN X
912E 8E 5B 93             STX STARTPOS    ;AND SAVE
9131 2Ø BE DE             JSR CHKCOM
9134 2Ø 67 DD             JSR FRMNUM      ;EVAL END POSITION EXPRESS
9137 2Ø FB E6             JSR CONINT      ;CONVERT TO INTEGER
913A 8E 5C 93             STX ENDPOS      ;AND SAVE
913D EC 5B 93             CPX STARTPOS    ;MAKE SURE ENDPOS>=START POS
9140 9Ø 8E                BCC ENTRYERR    ;NO GOOD
9142 2Ø BE DE             JSR CHKCOM
9145 2Ø D9 F7             JSR GETARYPT    ;GET INTEGER ARRAY POINTER
9148 2Ø 6A 93             JSR CHKONE      ;MAKE SURE 1-DIM. ARRAY
914B A5 9B                LDA LOWTR       ;SAVE DESC. ADRS.
914D 8D 58 93             STA SAVARRY3
9150 A5 9C                LDA LOWTR+1
9152 8D 59 93             STA SAVARRY3+1
9155 2Ø 76 93             JSR SETPTR1     ;ESTABLISH ARRY1 POINTER
9158 2Ø 81 93             JSR SETPTR2     ;ESTABLISH ARRY2 POINTER
915B 2Ø 8C 93             JSR SETPTR3     ;ESTABLISH ARRY3 POINTER
915E AØ Ø5                LDY #5
9160 B1 5Ø                LDA (ARRY1PTR),Y ;GET SIZE OF ARRAY
9162 8D 63 93             STA SIZE+1      ;MAKE LOW/HIGH
9165 D1 52                CMP (ARRY2PTR),Y ;MUST BE EQUAL SIZE ARRAYS
9167 FØ Ø3                BEQ SIZEEQ1     ;SIZES ARE EQUAL
9169 4C DØ 9Ø             JMP ENTRYERR    ;SIZES NOT EQUAL
916C D1 54       SIZEEQ1  CMP (ARRY3PTR),Y
916E FØ Ø3                BEQ SIZEEQ2
9170 4C DØ 9Ø             JMP ENTRYERR    ;SIZES NOT EQUAL
9173 C8          SIZEEQ2  INY
9174 B1 5Ø                LDA (ARRY1PTR),Y
9176 8D 62 93             STA SIZE
9179 D1 52                CMP (ARRY2PTR),Y
917B FØ Ø3                BEQ SIZEEQ3     ;SIZES ARE EQUAL
917D 4C DØ 9Ø             JMP ENTRYERR    ;SIZES NOT EQUAL
9180 D1 54       SIZEEQ3  CMP (ARRY3PTR),Y
9182 FØ Ø3                BEQ SIZEEQ4
9184 4C DØ 9Ø             JMP ENTRYERR    ;SIZES NOT EQUAL
9187 A9 Ø7       SIZEEQ4  LDA #Ø7
9189 2Ø 97 93             JSR ARY1PLUS    ;ARRY1PTR=ARRY1PTR+7
918C A9 Ø7                LDA #Ø7
918E 2Ø A3 93             JSR ARY2PLUS    ;ARRY2PTR=ARRY2PTR+7
9191 A9 Ø7                LDA #Ø7
9193 2Ø AF 93             JSR ARY3PLUS    ;ARRY3PTR=ARRY3PTR + 7
9196 2Ø CD 93             JSR INITINT     ;INITILIAZE INTEGER ARRAY
9199 AD 62 93    TRANSFER LDA SIZE        ;SEE IF MOVE-COUNT = Ø
919C ØD 63 93             ORA SIZE+1
919F DØ Ø3                BNE NOTDONE     ;MOVE IS NOT DONE
91A1 4C 27 92             JMP STRTSRT     ;DONE, NOW SORT
91A4 AØ ØØ       NOTDONE  LDY #ØØ
91A6 B1 5Ø                LDA (ARRY1PTR),Y ;GET LENGTH OF ELEMENT
91A8 8D 5A 93             STA ELMNTLEN    ;SAVE
91AB C9 ØØ                CMP #ØØ
91AD DØ ØE                BNE FOUNDEL
91AF A9 ØØ       NOELMNT  LDA #ØØ         ;NULL ELEMENT
91B1 8D 5F 93             STA NEWLEN      ;ZERO OUT LEN + ADDRESS
91B4 8D 5D 93             STA NEWAD
91B7 8D 5E 93             STA NEWAD+1
91BA 4C Ø4 92             JMP ESTDESC     ;GO ESTABLISH DESCRIPTOR
91BD C8          FOUNDEL  INY
91BE B1 5Ø                LDA (ARRY1PTR),Y ;GET ADDRESS OF ELEMENT
91C0 8D 6Ø 93             STA ELMNTPTR    ;AND SAVE
91C3 C8                   INY
91C4 B1 5Ø                LDA (ARRY1PTR),Y
91C6 8D 61 93             STA ELMNTPTR+1
91C9 AD 5A 93             LDA ELMNTLEN    ;IF ELMNTLEN < STARTPOS
91CC CD 5B 93             CMP STARTPOS    ;THEN ADDRESS + LENGTH = Ø
91CF 9Ø DE                BCC NOELMNT
91D1 18                   CLC
91D2 AD 6Ø 93             LDA ELMNTPTR    ;COMPUTE AD + LEN
91D5 6D 5B 93             ADC STARTPOS    ;AND SAVE
91D8 8D 5D 93             STA NEWAD
91DB AD 61 93             LDA ELMNTPTR+1
91DE 69 ØØ                ADC #ØØ
91E0 8D 5E 93             STA NEWAD+1
91E3 AD 5C 93             LDA ENDPOS      ;SEE IF ENDPOS > OR=ELMNTLEN
91E6 CD 5A 93             CMP ELMNTLEN
91E9 BØ 1Ø                BCS SHORTER     ;YES, SO USE ELMNTLEN
91EB 38                   SEC
91EC AD 5C 93             LDA ENDPOS      ;ELSE COMPUTE LEN,USE END POS
91EF ED 5B 93             SBC STARTPOS
91F2 8D 5F 93             STA NEWLEN      ;(NEWLEN=ENDPOS-STARTPOS+1)
91F5 EE 5F 93             INC NEWLEN
91F8 4C Ø4 92             JMP ESTDESC     ;GO ESTABLISH DESCRIPTOR
91FB AD 5A 93    SHORTER  LDA ELMNTLEN    ;(NEWLEN=ELMNTLEN-STARTPOS)
91FE ED 5B 93             SBC STARTPOS    ;(CARRY IS SET)
9201 8D 5F 93             STA NEWLEN
9204 AD 5F 93    ESTDESC  LDA NEWLEN      ;PUT LENGTH IN NEW
9207 AØ ØØ                LDY #ØØ         ;DSCRPTR END
9209 91 52                STA (ARRY2PTR),Y
920B C8                   INY
```

(continued)

```
920C AD 5D 93           LDA NEWAD           ;NO DO ADDRESS
920F 91 52              STA (ARRY2PTR),Y
9211 C8                 INY
9212 AD 5E 93           LDA NEWAD+1
9215 91 52              STA (ARRY2PTR),Y
9217 A9 Ø3              LDA #Ø3
9219 2Ø 97 93           JSR ARY1PLUS        ;ARRY1PTR=ARRY1PTR+3
921C A9 Ø3              LDA #Ø3
921E 2Ø A3 93           JSR ARY2PLUS        ;ARRY2PTR=ARRY2PTR+3
9221 2Ø BB 93           JSR DECSIZE         ;DECREMENT ELEMENT COUNT
9224 4C 99 91           JMP TRANSFER
9227 2Ø 81 93  STRTSRT  JSR SETPTR2         ;RESET ARRAY2 POINTERS
922A 2Ø 8C 93           JSR SETPTR3         ;RESET ARRAY3 POINTERS
922D AØ Ø5              LDY #Ø5
922F B1 52              LDA (ARRY2PTR),Y
9231 8D 63 93           STA SIZE+1          ;MAKE LOW/HIGH
9234 C8                 INY
9235 B1 52              LDA (ARRY2PTR),Y
9237 8D 62 93           STA SIZE
923A 2Ø BB 93  DOSORT   JSR DECSIZE         ;DEC SIZE (PAIRS=ELMNTS-1)
923D DØ Ø8              BNE DOSORT1         ;MORE PASSES TO MAKE
923F AD 62 93           LDA SIZE
9242 DØ Ø3              BNE DOSORT1         ;MORE PASSES TO MAKE
9244 4C DF 92           JMP DONEPASS        ;NO MORE SWAPS ARE POSSIBLE
9247 A9 Ø7     DOSORT1  LDA #Ø7
9249 2Ø A3 93           JSR ARY2PLUS        ;ADD 7 TO DESC. BASE ADRS.
924C A9 Ø7              LDA #Ø7             ;+ 7 TO ARRY2 BASE DESC.
924E 2Ø AF 93           JSR ARY3PLUS
9251 A9 ØØ              LDA #ØØ
9253 8D B3 9Ø           STA SWAPFLAG
9256 8D B1 9Ø           STA COUNT           ;INIT PAIR COUNT
9259 8D B2 9Ø           STA COUNT+1
925C AØ ØØ     SORTLP2  LDY #ØØ             ;INIT Y
925E B1 52              LDA (ARRY2PTR),Y    ;GET LENGTH OF 1ST PAIR MEMBER
9260 8D AF 9Ø           STA PAIR1LEN
9263 C8                 INY
9264 B1 52              LDA (ARRY2PTR),Y    ;GET ADRS. OF 1ST PAIR MEMBER
9266 85 5Ø              STA PAIR1AD
9268 C8                 INY
9269 B1 52              LDA (ARRY2PTR),Y
926B 85 51              STA PAIR1AD+1
926D C8                 INY
926E B1 52              LDA (ARRY2PTR),Y    ;GET LENGTH OF 2ND PAIR MEMBER
9270 8D BØ 9Ø           STA PAIR2LEN
9273 C8                 INY
9274 B1 52              LDA (ARRY2PTR),Y    ;GET ADRS. OF 2ND. PAIR MEMBER
9276 85 9B              STA PAIR2AD
9278 C8                 INY
9279 B1 52              LDA (ARRY2PTR),Y
927B 85 9C              STA PAIR2AD+1
927D EE B1 9Ø           INC COUNT           ;INCREMENT COMPARE COUNT
928Ø DØ Ø3              BNE PAIRNE
9282 EE B2 9Ø           INC COUNT+1
9285 AD BØ 9Ø  PAIRNE   LDA PAIR2LEN        ;FIND SHORTER ELEMENT
9288 CD AF 9Ø           CMP PAIR1LEN
928B BØ Ø6              BCS PAIR2LNG
928D AE BØ 9Ø           LDX PAIR2LEN        ;PAIR MEMBER 2 IS SHORTER
929Ø 4C 96 92           JMP PAIR2SHT
9293 AE AF 9Ø  PAIR2LNG LDX PAIR1LEN        ;PAIR MEMBER 2 IS LONGER OR =
9296 DØ ØB     PAIR2SHT BNE COMPSTRT        ;IF SHORTEST=Ø;MAYBE=
9298 AD BØ 9Ø           LDA PAIR2LEN        ;COMPARE LENGTHS
929B CD AF 9Ø           CMP PAIR1LEN
929E BØ 11              BCS DONEYET         ;PAIR2LEN = OR > PAIR1LEN
92AØ 4C ØØ 93           JMP SWAP            ;P2LN=Ø&P1LN≠Ø
92A3 AØ ØØ     COMPSTRT LDY #ØØ             ;INIT Y
92A5 B1 9B     COMPLP   LDA (PAIR2AD),Y     ;COMPARE CHARACTERS
92A7 D1 5Ø              CMP (PAIR1AD),Y
92A9 9Ø 55              BCC SWAP            ;PAIR1 > PAIR2
92AB DØ Ø4              BNE DONEYET         ;PAIR1 < PAIR2
92AD C8                 INY
92AE CA                 DEX
92AF DØ F4              BNE COMPLP          ;MORE BYTES TO COMPARE
92B1 A9 Ø3     DONEYET  LDA #Ø3
92B3 2Ø A3 93           JSR ARY2PLUS        ;BUMP ARRY2PTR + 3
92B6 A9 Ø2              LDA #Ø2
92B8 2Ø AF 93           JSR ARY3PLUS        ;BUMP ARRY3PRT + 2
92BB AD B1 9Ø           LDA COUNT           ;SEE IF WE HAVE
92BE CD 62 93           CMP SIZE            ;COMPARED ALL ACTIVE PAIRS
92C1 FØ Ø3              BEQ DONEYET1
92C3 4C 5C 92           JMP SORTLP2         ;NO CONTINUE COMPARES
92C6 AD B2 9Ø  DONEYET1 LDA COUNT+1         ;MAYBE
92C9 CD 63 93           CMP SIZE+1
92CC FØ Ø3              BEQ DONEYET3        ;MORE COMPS THIS PASS?
92CE 4C 5C 92           JMP SORTLP2         ;NO - CONTINUE PASS
92D1 AD B3 9Ø  DONEYET3 LDA SWAPFLAG        ;SEE IF WE NEED MORE PASSES
92D4 FØ Ø9              BEQ DONEPASS        ;NO SWAPS-SO WE ARE DONE
92D6 2Ø 81 93           JSR SETPTR2         ;RESET ARRAY2 POINTERS
92D9 2Ø 8C 93           JSR SETPTR3         ;RESET ARRAY3 POINTERS
92DC 4C 3A 92           JMP DOSORT          ;CONTINUE SORT
92DF AD 64 93  DONEPASS LDA ZEROSV          ;RESTORE ZERO PAGE
92E2 85 5Ø              STA ARRY1PTR
92E4 AD 65 93           LDA ZEROSV+1
92E7 85 51              STA ARRY1PTR+1
92E9 AD 66 93           LDA ZEROSV+2
92EC 85 52              STA ARRY2PTR
92EE AD 67 93           LDA ZEROSV+3
92F1 85 53              STA ARRY2PTR+1
92F3 AD 68 93           LDA ZEROSV+4
92F6 85 54              STA ARRY3PTR
92F8 AD 69 93           LDA ZEROSV+5
92FB 85 55              STA ARRY3PTR+1
92FD 4C 95 D9           JMP DATA
93ØØ AØ ØØ     SWAP     LDY #ØØ             ;SWAP VALUES
93Ø2 AD BØ 9Ø           LDA PAIR2LEN        ;FROM MEMBER 2
93Ø5 91 52              STA (ARRY2PTR),Y    ;TO MEMBER 1
93Ø7 B1 54              LDA (ARRY3PTR),Y
93Ø9 8D B4 9Ø           STA INTEGER1        ;SAVE LOW ELEMENTS INDEX MSB
93ØC C8                 INY
93ØD A5 9B              LDA PAIR2AD
93ØF 91 52              STA (ARRY2PTR),Y
9311 B1 54              LDA (ARRY3PTR),Y
9313 8D B5 9Ø           STA INTEGER1+1      ;SAVE LOW ELEMENTS INDEX LSB
9316 C8                 INY
9317 A5 9C              LDA PAIR2AD+1
9319 91 52              STA (ARRY2PTR),Y
931B B1 54              LDA (ARRY3PTR),Y
931D 8D B6 9Ø           STA INTEGER2        ;SAVE HIGH ELEMENTS INDEX MSB
932Ø AD B4 9Ø           LDA INTEGER1
9323 91 54              STA (ARRY3PTR),Y    ;SWAP INDEX
9325 C8                 INY
9326 AD AF 9Ø           LDA PAIR1LEN
9329 91 52              STA (ARRY2PTR),Y
932B B1 54              LDA (ARRY3PTR),Y
932D 8D B7 9Ø           STA INTEGER2+1      ;SAVE HIGH ELEMENTS INDEX LSB
933Ø AD B5 9Ø           LDA INTEGER1+1
9333 91 54              STA (ARRY3PTR),Y    ;SWAP
9335 C8                 INY
9336 A5 5Ø              LDA PAIR1AD
9338 91 52              STA (ARRY2PTR),Y
933A C8                 INY
933B A5 51              LDA PAIR1AD+1
933D 91 52              STA (ARRY2PTR),Y
933F A9 Ø1              LDA #Ø1             ;SET SWAP FLAG
9341 8D B3 9Ø           STA SWAPFLAG
9344 AØ ØØ              LDY #ØØ
9346 AD B6 9Ø           LDA INTEGER2        ;COMPLETE INTEGER SWAP
9349 91 54              STA (ARRY3PTR),Y
934B AD B7 9Ø           LDA INTEGER2+1
934E C8                 INY
934F 91 54              STA (ARRY3PTR),Y
9351 4C B1 92           JMP DONEYET         ;CONTINUE SORT
9354           ;INTERNAL STORAGE AREAS
9354           SAVARRY1 DFS 2,Ø             ;HOLD ARRAY1 DESCPTR. ADRS.
9356           SAVARRY2 DFS 2,Ø             ;HOLD ARRAY2 DESCPTR. ADRS
9358           SAVARRY3 DFS 2,Ø             ;HOLD ARRY3 DESCPTR. ADRS.
935A           ELMNTLEN DFS 1,Ø             ;ELEMENT LENGTH
935B           STARTPOS DFS 1,Ø             ;START POSITION
935C           ENDPOS   DFS 1,Ø             ;END POSITION
935D           NEWAD    DFS 2,Ø             ;NEW ELEMENT ADDRESS
935F           NEWLEN   DFS 1,Ø             ;NEW ELEMENT LENGTH
936Ø           ELMNTPTR DFS 2,Ø             ;ELEMENT POINTER
9362           SIZE     DFS 2,Ø             ;SIZE OF ARRAY
9364           ZEROSV   DFS 6,Ø             ;ZERO PAGE SAVE AREA
9ØAF           PAIR1LEN EQU SETVEC          ;REUSE SETVEC(ONLY AT BRUN)
9ØBØ           PAIR2LEN EQU SETVEC+1
9ØB1           COUNT    EQU SETVEC+2
9ØB3           SWAPFLAG EQU SETVEC+4
9ØB4           INTEGER1 EQU SETVEC+5
9ØB6           INTEGER2 EQU SETVEC+7
936A           ;              <<<SUBROUTINES>>>
936A AØ Ø4     CHKONE   LDY #4              ;CHECK NO. DIM TO
936C B1 9B              LDA (LOWTR),Y       ;MAKE SURE IT IS A
936E C9 Ø1              CMP #1              ;ONE DIMENSION ARRAY
937Ø FØ Ø3              BEQ CHKONEXT        ;OK
```

(continued)

Alternate Index *(continued)*

```
9372 4C C9 DE           JMP SYNERR        ;DISP SYNTAX ERROR MESSAGE
9375 60       CHKONEXT  RTS
9376 AD 54 93 SETPTR1   LDA SAVARRY1      ;ESTABLISH WORK POINTER
9379 85 50              STA ARRY1PTR      ;FOR SOURCE ARRAY DESC.
937B AD 55 93           LDA SAVARRY1+1
937E 85 51              STA ARRY1PTR+1
9380 60                 RTS
9381 AD 56 93 SETPTR2   LDA SAVARRY2      ;ESTABLISH ARRY2 POINTER
9384 85 52              STA ARRY2PTR      ;FOR DESTINATION ARRAY DESC.
9386 AD 57 93           LDA SAVARRY2+1
9389 85 53              STA ARRY2PTR+1
938B 60                 RTS
938C AD 58 93 SETPTR3   LDA SAVARRY3      ;ESTABLISH ARRY3 POINTER
938F 85 54              STA ARRY3PTR      ;FOR INTEGER ARRAY DESC.
9391 AD 59 93           LDA SAVARRY3+1
9394 85 55              STA ARRY3PTR+1
9396 60                 RTS
9397 18       ARY1PLUS  CLC               ;ADD ACCUM TO ARRY1PTR
9398 65 50              ADC ARRY1PTR
939A 85 50              STA ARRY1PTR
939C A9 00              LDA #00
939E 65 51              ADC ARRY1PTR+1
93A0 85 51              STA ARRY1PTR+1
93A2 60                 RTS
93A3 18       ARY2PLUS  CLC               ;ADD ACCUM TO ARRY2PTR
93A4 65 52              ADC ARRY2PTR
93A6 85 52              STA ARRY2PTR      ;= ADDRESS OF FIRST ELMNT
93A8 A5 53              LDA ARRY2PTR+1
93AA 69 00              ADC #00
93AC 85 53              STA ARRY2PTR+1
93AE 60                 RTS
93AF 18       ARY3PLUS  CLC               ;ADD ACCUM TO ARRY3PTR
93B0 65 54              ADC ARRY3PTR
93B2 85 54              STA ARRY3PTR      ;= ADDRESS OF FIRST ELMNT
93B4 A5 55              LDA ARRY3PTR+1
93B6 69 00              ADC #00
93B8 85 55              STA ARRY3PTR+1
93BA 60                 RTS
93BB 18       DECSIZE   CLC               ;DECREMENT ELEMENT COUNT
93BC AD 62 93           LDA SIZE
93BF E9 00              SBC #00
93C1 8D 62 93           STA SIZE
93C4 AD 63 93           LDA SIZE+1
93C7 E9 00              SBC #00
93C9 8D 63 93           STA SIZE+1
93CC 60                 RTS
93CD A9 00    INITINT   LDA #00           ;INIT  INT ARRAY
93CF 8D B1 90           STA COUNT
93D2 8D B2 90           STA COUNT+1
93D5 A0 00    INITLOOP  LDY #00           ;INIT Y REG
93D7 AD B2 90           LDA COUNT+1       ;STORE COUNT IN ARRAY
93DA 91 54              STA (ARRY3PTR),Y
93DC C8                 INY
93DD AD B1 90           LDA COUNT
93E0 91 54              STA (ARRY3PTR),Y
93E2 A9 02              LDA #02           ;POINT TO NEXT ELEMENT
93E4 20 AF 93           JSR ARY3PLUS
93E7 EE B1 90           INC COUNT
93EA D0 03              BNE COUNTNE       ;NO NEED TO INC COUNT+1
93EC EE B2 90           INC COUNT+1
93EF AD 62 93 COUNTNE   LDA SIZE          ;SEE IF WE ARE DONE INITING
93F2 CD B1 90           CMP COUNT
93F5 D0 DE              BNE INITLOOP      ;NO
93F7 AD 63 93           LDA SIZE+1        ;MAYBE
93FA CD B2 90           CMP COUNT+1
93FD D0 D6              BNE INITLOOP
93FF 60                 RTS               ;ALL DONE
```

MICRO

AARDVARK

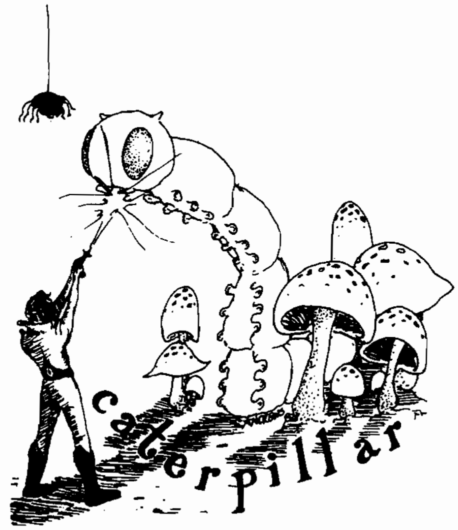
TRS-80 COLOR   OSI   VIC-64   VIC-20   SINCLAIR   TIMEX

QUEST – A NEW IDEA IN ADVENTURE GAMES! Different from all the others. Quest is played on a computer generated map of Alesia. Your job is to gather men and supplies by combat, bargaining, exploration of ruins and temples and outright banditry. When your force is strong enough, you attack the Citadel of Moorlock in a life or death battle to the finish. Playable in 2 to 5 hours, this one is different every time. 16k TRS-80, TRS-80 Color, and Sinclair. 13K VIC-20. $14.95 each.

ADVENTURES!!!

These Adventures are written in BASIC, are full featured, fast action, full plotted adventures that take 30-50 hours to play. (Adventures are interactive fantasies. It's like reading a book except that you are the main character as you give the computer commands like "Look in the Coffin" and "Light the torch.")

Adventures require 16k on TRS80, TRS80 color, and Sinclair. They require 8k on OSI and 13k on Vic-20. Derelict takes 12k on OSI. $14.95 each.



CATERPILLAR

O.K., the Caterpillar does look a lot like a Centipede. We have spiders, falling fleas, monsters traipsing across the screen, poison mushrooms, and a lot of other familiar stuff. COLOR 80 requires 16k and Joysticks. This is Edson's best game to date. $19.95 for TRS 80 COLOR.

PROGRAMMERS!
SEE YOUR PROGRAM IN THIS SPACE!!
Aardvark traditionally pays the highest commissions in the industry and gives programs the widest possible coverage. Quality is the keyword. If your program is good and you want it presented by the best, send it to Aardvark.

ESCAPE FROM MARS
(by Rodger Olsen)
This ADVENTURE takes place on the RED PLANET. You'll have to explore a Martian city and deal with possibly hostile aliens to survive this one. A good first adventure.

PYRAMID (by Rodger Olsen)
This is our most challenging ADVENTURE. It is a treasure hunt in a pyramid full of problems. Exciting and tough!

HAUNTED HOUSE (by Bob Anderson)
It's a real adventure—with ghosts and ghouls and goblins and treasures and problems — but it is for kids. Designed for the 8 to 12 year old population and those who haven't tried Adventure before and want to start out real easy.

DERELICT
(by Rodger Olsen & Bob Anderson)
New winner in the toughest adventure from Aardvark sweepstakes. This one takes place on an alien ship that has been deserted for a thousand years — and is still dangerous!

TUBE FRENZY
(by Dave Edson)
This is an almost indescribably fast action arcade game. It has fast action, an all new concept in play, simple rules, and 63 levels of difficulty. All machine code, requires Joysticks. Another great game by Dave Edson. TRS 80 COLOR ONLY. 16k and Joysticks required. $19.95.

CATCH'EM
(by Dave Edson)
One of our simplest, fastest, funnest, all machine code arcade games. Raindrops and an incredibe variety of other things come falling down on your head. Use the Joysticks to Catch'em. It's a BALL! — and a flying saucer! — and a Flying Y!— and so on. TRS 80 COLOR. $19.95.

BASIC THAT ZOOOMMS!!
AT LAST AN AFFORDABLE COMPILER!
The compiler allows you to write your programs in easy BASIC and then automatically generates a machine code equivalent that runs 50 to 150 times faster.

It does have some limitations. It takes at least 8k of RAM to run the compiler and it does only support a subset of BASIC—about 20 commands including FOR, NEXT, END, GOSUB, GOTO, IF, THEN, RETURN, END, PRINT, STOP, USR (X), PEEK, POKE, *, /, +, -, > , < ,=, VARIABLE NAMES A-Z, SUBSCRIPTED VARIABLES, and INTEGER NUMBERS FORM 0-64K.

TINY COMPILER is written in BASIC. It generates native, relocatable 6502 or 6809 code. It comes with a 20-page manual and can be modified or augmented by the user. $24.95 on tape or disk for OSI, TRS-80 Color, or VIC.

Please specify system on all orders

ALSO FROM AARDVARK — This is only a partial list of what we carry. We have a lot of other games (particularly for the TRS-80 Color and OSI), business programs, blank tapes and disks and hardware. Send $1.00 for our complete catalog.

AARDVARK - 80
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110

Phone Orders Accepted 8:00 a.m. to 4:00 p.m. EST. Mon.-Fri.

PET Vet

By Loren Wright

What's So Good About the Commodore 64?

It looks almost exactly like the VIC-20, but the $595 list price is twice that of the VIC's. There are a few external differences. The color of the case is light tan instead of off-white; the keyboard has a more comfortable feel; there are two controller ports instead of one; the power supply is considerably bigger and connects to a more elaborate jack on the computer; the cartridge port is narrower; the modulator is built into the computer with a deeply recessed channel 3/4 switch.

When you turn on the C64, more differences are apparent. There are 40 columns across, instead of 22, and it comes up with 38911 bytes free — more than ten times that of the VIC! The graphics (see last month's column) and sound capabilities are considerably more advanced than the VIC's. After that, comparison to the VIC is not very useful.

The C64 actually has 64K of RAM, and you do get nearly 8K more for BASIC than you do with a 32K PET. For machine language there's another 8K at $C000-$CFFF. However, if you want to strip down the C64's operating system to the essential routines or KERNAL (get a character, put a character, etc.), you can gain access to a lot more RAM for machine-language programs. You can copy the BASIC ROMs into RAM, make changes as you like, and run from the new RAM copy.

The difference is the 6510 processor with its built-in I/O port and tri-state address lines. This allows RAM and ROM to share the same address space, with the processor switching only one in at a time. For instance, the I/O devices (VIC-II, SID, CIA) and color RAM are addressed exactly the same place ($D000-$DFFF) as the character generator ROM (not to mention the RAM available there!). The 6510 is able to do all the necessary switching at the right times to pull this off. Unlike the 6509 (to be included in the PET/CBM B, P, and BX), the addressing range of the 6510 is still only 64K.

It is very encouraging that so much technical information is available on the C64. Before the C64 was released, Commodore had an information kit, including memory maps and development software, available to serious software developers. The *Programmer's Reference Guide* should be available by the time you read this. Unlike Commodore publications before the *VIC Programmer's Reference Guide*, the "Guide" fot the C64 is thorough, well done, and very useful.

Availability of software for the C64 is not as much a problem as first anticipated. Most PET programs can be converted easily to run on the C64. Many already have. C64 versions of such popular PET programs as WordPro 3, MAE, and VisiCalc should be available shortly. Over 300 educational programs are now offered by the Toronto PET Users' Group. I have already received review copies of C64 versions of "Tiny BASIC Compiler" from Abacus Software and "KMMM Pascal" from Wilserv Industries (available from AB Computers). There already is a fair amount of public domain software, including some nifty demonstrations, an assembler, a sprite editor, a character editor, and a SID monitor. These started out in Commodore's software developer's kit, but most users' groups should have these by now.

There is some cause for concern in the software area, though. The CP/M and IEEE cartridges have been delayed considerably. If you are counting on CP/M software for the C64 right away, don't hold your breath! Even when the cartridge does become available you will have to have each CP/M disk converted to CBM format.

Commodore has no immediate plans to release an IEEE adaptor, but two (and maybe three when you read this) such units are available from independent vendors. The fanciest unit, called the C64-LINK, sells for $185 (Canadian) from Richvale Communications (10610 Bayview Avenue, Richmond Hill, Ontario L4C 3N8, Canada). Not only does it interface to the IEEE, but it also adds BASIC 4 commands and a machine-language monitor. A less elaborate interface is available from Micro-Systems (11105 Shady Trail Suite 104, Dallas, TX 75229) for $109.95. It provides the IEEE interface only, under control of BASIC 2. A third company in Arizona has announced an IEEE adaptor, still under development. Reviews of the Micro-Systems and Richvale Communications units will appear soon in our "Reviews in Brief" department.

There are a few things I don't like about my new C64. Perhaps the biggest gripe is that as soon as I bought the computer I had to buy more equipment to get a workable system. I was able to get through an orientation period with a feeble, old, black and white TV and a borrowed CBM cassette. The first move was to purchase a C64 Link so I could use the CBM disk and printer from work. Then I bought a color TV. When I get tired of carrying the disk drive back and forth, I'll want my own. All the other computers in the C64's price range are designed in a similar modular fashion, so I must have been spoiled all this time by the PET's completeness! Two other gripes — BASIC 2 and the lack of a machine-language monitor — were solved with the addition of the C64 Link. It would also be nice to have a numeric keypad.

All in all, I'm happy with the purchase. The C64 will satisfy my needs for a computer that is both practical and recreational. I predict it will have a big impact on the market. Apple and Atari will have to make some fast moves to compete.

PET, VIC, and C64 BASIC Compatibility

If you own more than one Commodore computer, you will eventually want to be able to load programs written on one machine into another. If you are writing programs, your development software and firmware (assembler/editor, disassembler, Toolkit, POWER, etc.) is likely to be concentrated on one machine. Converting a program is usually a simple matter, With the exception of the MAX

PET Vet *(continued)*

machine, all Commodore computers use essentially the same BASIC. There are slight differences in the control characters implemented (color and programmable function keys on the VIC and C64; screen editing and window controls on the 8032 — unimplemented characters are ignored) and in the screen format (22 VIC columns, 40 for PET and C64, and 80 for 8032). BASIC 4 commands need to be replaced by BASIC 2 commands in the VIC, C64, and earlier PETs. Of course, more serious problems arise with machine-language programs and with BASIC programs that do PEEKs and POKEs to machine-dependent locations.

The cassette format and handling is exactly the same from machine to machine. Even though the VIC disk drive is serial rather than IEEE, the actual diskette can be handled by the PET disk drives (except the 8050).

But even though BASIC programs on these machines are basically compatible, there is a problem. BASIC text starts at different locations in the different machines. In the PET and CBM models, programs always start at $401. In an unexpanded VIC it's $1001, but with the 3K expansion it's $401. In the C64, BASIC usually starts at $801. The situation is far from hopeless, though. The VIC and C64 both have relocating loaders: a BASIC program, no matter where it was originally located or from which Commodore machine it was SAVEd, will automatically load at the current start of BASIC. The PET/CBM does not have this capability; it loads a program at the original location, but looks for it at $401. How do we get a program from $1001 or $801 to $401, where the PET expects to find it?

One way is to configure your C64 or VIC so that BASIC programs always start at $400. For the VIC, you need the 3K RAM expansion that fills in $400-$FFF. The VIC automatically adjusts to start BASIC at $401. For the C64, you need to move the screen to $8000 (where it is in the PET) and move the start of BASIC to $401. There is a short program called "C64 to PET" included on the developers' disk mentioned above, that does this.

Another, more general-purpose procedure is outlined below. It works with all but the very longest BASIC programs.

1. Type a one-line program into your PET (e.g., 1 REM)
2. Load the VIC or C64 program.
3. POKE 1025,1: POKE 1026,8 to move a program from $801. Or, POKE 1025,1:POKE 1026,16 to move a program from $1001.
4. Delete the original single line by typing its number, hit return, and the whole program will move to $401!

New Users' Group and Newsletter for the SuperPET

The SuperPET Users' Group (SPUG) is putting out a newsletter called the *SuperPET Gazette.* Membership is $10/year and includes a subscription.

Paul V. Skipski, Secretary
SuperPET Users' Group
4782 Boston Post Road
Pelham, NY 10803

The second issue was ten pages and included resource information, utility programs, and statements of purpose and direction.

MICRO™

Extending Newton-Raphson's Method to Evaluate Complex Roots

by P.P. Ong

This article discusses a standard procedure to compute the complex roots of a polynomial equation using the microcomputer. The accompanying program can be incorporated as a subroutine for applications programs. An extension to cover non-polynomial equations is also discussed.

N-R's Method
requires:

Any Microsoft BASIC (although written in Applesoft)

Many scientific and engineering applications require handling complex numbers and computing the complex roots of equations. Common practical examples involving complex quantities occur in wave attenuation calculations, solutions of differential equations, alternating current network, Fourier transformation, diffraction pattern analyses, and plane vector algebra. Most microcomputers are, as yet, not designed to handle complex numbers. Indeed, complex numbers are generally not covered in the standard or extended BASIC languages. Users wishing to modify a language to include such quantities invariably encounter an almost insuperable obstacle posed by the limited RAM capacity of the microcomputer. This seems a drawback, especially since the computations involve an iterative procedure for which the computer would be very efficient otherwise.

An illuminating consequential trend is found in the usage of the well-known Newton-Raphson numerical procedure to solve an algebraic equation. With the widespread use of the micro, this method has become so popular that it has by and large superseded the more conventional method of resorting to complicated mathematics to produce exact solutions. The greater accuracy of the latter method is not always required for real-life problems; in any case, there is often no possible solution by the exact method.

A standard Newton-Raphson procedure for complex roots is also available (e.g., see W.E. Grove, *Brief Numerical Methods*, Prentice-Hall (1966), pp. 9-14), yet it is seldom used in practice. This is because the numerical evaluations usually become too protracted and rarely conclude successfully. In fact, very few textbooks on numerical analysis treat complex root evaluations seriously.

I have developed a system to extend Newton-Raphson's method using de Moivre's theorem. It is now my standard routine and is applicable for both real and complex roots of any polynomial equation with real coefficients. The computer itself does not need to handle complex quantities — only standard trigonometric functions such as sine, cosine, and arc tangent, which are all built-in BASIC functions.

A detailed mathematical formulation is presented for those who want to know why the method works, but this section may be by-passed in a first reading.

Mathematical Formulation

Suppose a polynomial equation of degree n and having only real coefficients is given. It can be written in the form

(1) $F(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$

Its derivative is

(2) $F'(x) = a_1 + 2a_2x + 3a_3x^3 + \dots + na_nx^{n-1}$

Newton-Raphson's iteration formula is

(3) $x_{k+1} = x_k - \{F(x_k)/F'(x_k)\}$

Put

(4) $x_k = p + qi = A(\cos\Theta + i\sin\Theta) = Ae^{i\Theta}$

so that

(5) $A = SQR(p^2 + q^2)$

and

(6) $\Theta = \begin{cases} \tan^{-1}(q/p) & \text{for } p \neq 0 \text{ or } q = 0 \\ \pi/2 & \text{for } p = 0 \text{ and } q > 0 \\ -\pi/2 & \text{for } p = 0 \text{ and } q < 0 \end{cases}$

The case $x_k = p = q = 0$ is obviously trivial. To calculate the angular value of the arc tangent, care must be taken to ensure that it lies in the first quadrant for $p > 0$ and $q > 0$, in the second quadrant for $p < 0$ and $q > 0$, in the third quadrant for $p < 0$ and $q < 0$, and in the fourth quadrant for $p > 0$ and $q < 0$.

Using de Moivre's theorem

(7) $x_k^m = A^m(\cos m\Theta + \sin m\Theta)$

we can re-write equation (3) as

(8) $x_k + 1 = x_k - \{(r + si)/(t + ui)\}$

where

(9) $r = a_0 + a_1A\cos\Theta + a_2A^2\cos2\Theta + \dots + a_nA^n\cos n\Theta$

(10) $s = a_1A\sin\Theta + a_2A^2\sin2\Theta + a_3A^3\sin3\Theta + \dots + a_nA^n\sin n\Theta$

(11) $t = a_1 + 2a_2A\cos\Theta + 3a_3A^2\cos2\Theta + \dots + na_nA^{n-1}\cos(n-l)\Theta$

(12) $u = 2a_2A\sin\Theta + 3a_3A^2\sin2\Theta + \dots + na_nA^{n-1}\sin(n-1)\Theta$

From equation (8) the correction term is

$$(13)\quad x_{k+1} - x_k = -b\{\cos(\Phi - \psi) + i \sin(\Phi - \psi)\}$$

where

$$(14)\quad \Phi = \begin{cases} \tan^{-1}(s/r) & \text{for } r \neq 0 \text{ or } s = 0 \\ \pi/2 & \text{for } r = 0 \text{ and } s > 0 \\ -\pi/2 & \text{for } r = 0 \text{ and } s < 0 \end{cases}$$

and

$$(15)\quad \psi = \begin{cases} \tan^{-1}(u/t) & \text{for } t \neq 0 \text{ or } u = 0 \\ \pi/2 & \text{for } t = 0 \text{ and } u < 0 \\ -\pi/2 & \text{for } t = 0 \text{ and } u < 0 \end{cases}$$

and

$$(16)\quad b = SQR\{ (r^2 + s^2)/(t^2 + u^2) \}$$

Again, to compute the angles from the respective inverse tangent functions the proper angular quadrants have to be found. In equation (14) (the special case when $r = s = 0$ results in $x_{k+1} = x_k$) the solution is obviously obtained.

Equation (16) breaks down if both t and u vanish. This occurs when $F'(x) = 0$ and Newton-Raphson's method fails in this case. The computation has to be restarted with a different initial value for x_k.

Barring the above abortive case, the iteration procedure continues with x_{k+1} replacing x_k. The new values of p and q become

$$(17)\quad p \rightarrow p - b\cos(\Phi - \psi)$$

and

$$(18)\quad q \rightarrow q - b\sin(\phi - \psi)$$

Since complex roots occur in pairs for a polynomial with real coefficients, when one complex root is found its complex conjugate would also be a root. Furthermore, if $x = p \pm qi$ is a pair of complex roots, then F(x) has a quadratic factor

$$(19)\quad (x - p)^2 + q^2 = x^2 - 2px + (p^2 + q^2)$$

If a real root is found then $q = 0$ and F(x) has a linear factor $(x - p)$. By successive factorization of F(x) we can reduce the degree of the polynomial equation by one or two each time, and eventually all its roots can be obtained completely.

The Program

The above formulation is translated into a sub-program written explicitly in Applesoft BASIC. It can be readily modified to adapt to other micro systems. To assist the reader, the program is liberally filled with explanatory REMarks at each stage. It can be segmented at statement numbers 50000, 51000, 52000, etc. The leading statement of each segment clearly describes the purpose of the segment.

Because of the nature of the problem, there are an inconveniently large number of initial parameters that need to be supplied by the user. To minimize this, default values are automatically chosen whenever possible. The exact parameters describing the given equation must obviously be supplied by the user. All the other parameters are defaulted as follow:

a. Maximum iteration number allowed, IM = 30
b. Maximum error tolerance allowed, ER = 1E−8
c. Initial approximation of root: real part, P = 1; imaginary part, Q = i

Provision is available to re-select these defaulted values, especially after an unsuccessful iteration.

Since there is an inherent rounding error associated with any floating-point number, a perceptible, though normally small, error will be propagated after a large number of computation steps. This magnification of errors is roughly proportional to the degree of the polynomial, the coefficients, and the number of high-power terms involved. After many successive factorizations it is possible for the roots subsequently obtained to be off by approximately 0.0001% (see example 1). Although this discrepancy is usually negligible, a recourse is automatically provided in the program by going through a second stage re-computation of the *original* equation using each of the previously obtained results as starting approximation. This should eventually lead to new results with the originally stipulated accuracy.

The program is easily incorporated as a BASIC subroutine for any applications program. If necessary, it can first be renumbered (using the Applesoft Toolkit's LOADAPA, for instance), and then appended at the end of the user's application package. It is for this reason that the statement numbers are started high up at 50000, providing ample room for insertion of the user's master program. To access the subroutine replace its last END statement with a RETURN and call it with a GOSUB. Alternatively, it is also possible for the program to be SAVEd on a disk file and EXECed when required.

Applications

Two examples that demonstrate the application of the program are given below:

Example 1

Suppose you wish to find the intersections between the curves

$$(20)\quad y = 16 + 7x^5 - 13x^{11} + x^{13} + x^{14}$$

and

$$(21)\quad y = 12 - x + 5x^2 + 26x^3 + 2x^{12}$$

in a Cartesian coordinate system. This is equivalent to solving the complicated algebraic equation

$$(22)\quad 4 + x - 5x^2 - 26x^3 + 7x^5 - 13x^{11} - 2x^{12} + x^{13} + x^{14} = 0$$

Ordinarily this problem would be too formidable to attempt manually. However, with the present program the following set of answers (all accurate to seven decimal places) was printed on the screen in just over nine minutes (including a 15-second pause for screen reading after each successful iteration):

$x = 0.5100436, -0.3318626 \pm 0.4413179i, 1.0542850 \pm 0.3956750i, 0.5187938 \pm 0.9907267i, -0.9420515 \pm 0.3670825i, 2.1458455, -0.3315892 \pm 1.0217258i, -1.7955200 \pm 1.7664781i$

As an aside, it is worth noting that after the first-stage computation the last set of roots was $x = -1.7955314 \pm 1.7664794i$, which differed from the exact answers by about 1 part in 10^6. The accumulated propagated error in this illustration was 0.0001% — minimal considering the elaborate computations involved.

Example 2

Now try to solve the linear differential equation

$$(23)\quad \frac{d^{10}y}{dx^{10}} - 0.77\frac{d^8y}{dx^8} - 7.9\frac{d^6y}{dx^6} + 1.44\frac{d^4y}{dx^4} + 5.18\frac{d^2y}{dx^2} - 4.275 = 0$$

```
10  REM ************************
11  REM *                      *
12  REM *   EXTENDING NEWTON-  *
13  REM *   RAPHSON'S METHOD   *
14  REM *  TO EVALUATE COMPLEX *
15  REM *        ROOTS         *
16  REM *                      *
17  REM *     P. P. ONG        *
18  REM *                      *
19  REM *                      *
20  REM ************************
50000  TEXT : HOME
50020  PRINT : PRINT ''THIS SUBR COMPUTES THE REAL AND COMPLEX''
50040  PRINT ''ROOTS OF ANY POLYNOMIAL EQUATION:'': PRINT
50060  PRINT ''F(X)=A(Ø)+A(1)×X+A(2)×X↑2+....+A(N)×X↑N''
50080  PRINT : INVERSE : PRINT ''SPECIFY THE FOLLOWING INPUTS:'': NORMAL
50100  PRINT : INPUT ''N = '';N: DIM A(N),SG$(N),AA(N),P(N),Q(N):NN = N
50120  FOR I = Ø TO N
50140  PRINT ''A(''I'') = '';: INPUT '''';A(I):AA(I) = A(I): NEXT
50160  PRINT
50180 ER = 1E - 8: REM  Set Error Tolerance
50200 IM = 3Ø: REM  Set Max.Iter.No.
50220 RT = Ø: REM   Init Root Counter
50240 PI = 3.141592654:FL = Ø: REM  Init Recomputation Flag
50260 PL$ = ''+'':MN$ = ''-''
50280  PRINT : INPUT ''DEFAULT FOR OTHER PARAMETERS? '';AN$: IF  LEFT$
     (AN$,1) = ''N'' THEN  PRINT : PRINT : GOTO 59040
50300  IF N = 1 THEN  GOSUB 58000
50320  IF N = Ø THEN 60000
50340 P = 1:Q = 1: REM SET FIRST ITER.VALUE OF X
50360 IR = Ø: GOSUB 56000
51000 IR = IR + 1: REM  Begin Iter.Loop
51020  IF IR > IM THEN 59020
51040  REM  Compute A and Theta
51060 A =  SQR (P × P + Q × Q)
51080  IF Q = Ø THEN Q = ER × ER: REM  Make Abs(Q)<>Ø
51100  IF P = Ø THEN TH = PI / 2 ×  SGN (Q): GOTO 51200
51120 TH =  ATN (Q / P)
51140  REM  Compute The Proper Quadrant For Theta
51160  IF TH < Ø THEN TH = TH + PI
51180  IF TH < PI AND Q < Ø THEN TH = TH + PI
51200 R = A(Ø): REM  Begin Compute R
51220  FOR I = 1 TO N:R = R + A(I) × A ↑ I ×  COS (I × TH): NEXT
51240 S = Ø: REM  Begin Compute S
51260  FOR I = 1 TO N:S = S + A(I) × A ↑ I ×  SIN (I × TH): NEXT
51280 T = Ø: REM  Begin Compute T
51300  FOR I = 1 TO N:T = T + I × A(I) × A ↑ (I - 1) ×  COS ((I - 1)
     × TH ): NEXT
51320 U = Ø: REM  Begin Compute U
51340  FOR I = 2 TO N:U = U + I × A(I) × A ↑ (I - 1) ×  SIN ((I - 1)
      × TH ): NEXT
51360  IF T = Ø AND U = Ø THEN 59340
51380 B =  SQR ((R × R + S × S) / (T × T + U × U))
51400  IF B < ER THEN 52000
51420  REM  Compute Phi and Psi
51440  IF R = Ø THEN FI = PI / 2 ×  SGN (S): GOTO 51540
51460 FI =  ATN (S / R)
51480  REM  Compute the Proper Quadrant for Phi
51500  IF FI < Ø THEN FI = FI + PI
51520  IF FI < PI AND S < Ø THEN FI = FI + PI
51540  IF T = Ø THEN SI = PI / 2 ×  SGN (U): GOTO 51640
51560 SI =  ATN (U / T)
51580  REM  Compute the Proper Quadrant For Psi
51600  IF SI < Ø THEN SI = SI + PI
51620  IF SI < PI AND U < Ø THEN SI = SI + PI
51640  REM Set New P and Q
51660 P = P - B ×  COS (FI - SI):Q = Q - B ×  SIN (FI - SI)
51680  IF FL = 1 THEN 51000: REM Don't Print on Recomputation
51700  PRINT ''I='' SPC( IR < 1Ø);IR; SPC( 2)''P=''P; TAB( 24)''Q=''Q:
       REM Print Result After Each Iter.
51720  GOTO 51000
52000  REM Sucessful Iteration
52020 RT = RT + 1: REM Count the No. of Sets of Roots
52040 P(RT) = P:Q(RT) = Q: REM Store Answers
52060  IF FL = 1 THEN  RETURN
52080  PRINT : PRINT ''NR'S METHOD IS SUCCESSFUL.''
52100  ON (1 + ( ABS (Q) < ER)) GOSUB 53000,55000: GOSUB 50300
53000  REM Routine for Complex Roots
53020  PRINT : PRINT ''A PAIR OF COMPLEX ROOTS ARE''
53040  PRINT : PRINT ''X = ''P'' (+/-) '' ABS (Q)'' × I'': PRINT
53060  FOR I = 1 TO 3000: NEXT
54000 N = N - 2: REM Reduce Polyn Degree by 2
54020  IF N = Ø THEN  RETURN
54040 H =  - 2 × P:K = P × P + Q × Q
54060  REM Reset Coeffs of F(X) After Extracting the Factor (X×X+H×X+K)
54080 A(Ø) = A(Ø) / K
54100 A(1) = (A(1) - H × A(Ø)) / K
54120  IF N = 1 THEN  RETURN
54140  FOR I = 2 TO N
54160 A(I) = (A(I) - H × A(I - 1) - A(I - 2)) / K
54180  NEXT : RETURN
55000  REM Routine for Real Root
55020  PRINT : PRINT ''A SINGLE REAL ROOT FOUND IS''
55040  PRINT : PRINT ''X = ''P
55060  FOR I = 1 TO 3000: NEXT
55080 N = N - 1: REM Reduce Polyn Degree by 1
55100  IF N = Ø THEN  RETURN
55120  REM Reset Coeffs of F(X) After Extracting the Factor (X-P)
55140 A(Ø) =  - A(Ø) / P
55160  FOR I = 1 TO N
55180 A(I) =  - (A(I) - A(I - 1)) / P: NEXT
55200  RETURN
56000  HOME : REM Display Screen Heading
56020  PRINT : PRINT ''======================================'': PRINT
     : PRINT
56040  PRINT ''NOW COMPUTING.....''
56060  FOR I = Ø TO N:SG$(I) = PL$
56080  IF A(I) < Ø THEN SG$(I) = MN$
56100  NEXT
56120  PRINT : PRINT : PRINT ''F(X) = '';:
        IF A(Ø) <  > Ø THEN  PRINT A(Ø);
56140  IF A(1) <  > Ø THEN  PRINT SG$(1); ABS (A(1))''×X'';
56160  IF N < 2 THEN  PRINT : PRINT : GOTO 56240
56180  FOR I = 2 TO N: IF A(I) <  > Ø THEN  PRINT SG$(I);
        ABS (A(I))''×X↑''I;
56200  NEXT : PRINT
56220  PRINT : PRINT ''WITH ER = ''ER'' AND IM = ''IM: PRINT
56240  PRINT ''______________________________________'': PRINT : RETURN
57000  REM Restore Original F(X)
57020 N = NN: FOR I = Ø TO NN:A(I) = AA(I): NEXT : RETURN
58000  REM  Compute Root of Residual Linear Fraction
58020 RT = RT + 1:P(RT) =  - A(Ø) / A(1)
58040  GOSUB 56000: PRINT : PRINT ''LAST ROOT (REAL) = ''P(RT)
58060  FOR I = 1 TO 3000: NEXT
58080 N = N - 1: RETURN
59000  REM Unsuccessful Cases
59020  HOME : PRINT ''MAX ITER. NO. EXCEEDED'': PRINT : PRINT
59030  POKE 34,3
59040 P = 1:Q = 1: REM Offset P and Q
59060  HOME : PRINT ''TYPE 1 TO RESELECT MAX ITER NO.'': PRINT
59080  PRINT ''TYPE 2 TO RESELECT ERR TOLERANCE'': PRINT
59100  PRINT ''TYPE 3 TO RESELECT INIT APROX. ROOT'': PRINT
59120  PRINT ''TYPE 4 TO RECOMPUTE'': PRINT
59140  PRINT ''TYPE 5 TO ABORT AND DISPLAY ROOTS''
59160  PRINT ''          OBTAINED SO FAR''
59170  POKE 34,Ø
59180  PRINT : GET CH$: PRINT : ON  VAL (CH$) GOTO 59220,59240,59260,
        50360,59320
59200  GOTO 59180
59220  INPUT ''NEW MAX ITER NO. = '';IM: GOTO 59060
59240  INPUT ''NEW ERR TOLERANCE = '';ER: GOTO 59060
59260  PRINT ''SUPPLY THE INIT APPROX OF ROOT BY''
59280  PRINT ''TYPING IN ITS REAL AND IMAG PARTS''
59300  PRINT ''(SEPARATED BY A COMMA): '': INPUT '''';P,Q: PRINT : PRINT :
        GOTO  59060
59320  PRINT : PRINT ''COMPUTATION ABORTED.'' : PRINT : PRINT ''LISTING OF
        ROOTS OBTAINED SO FAR:-'' : GOSUB 60200: END
59340  REM Case Where T=U=Ø
59360  PRINT : PRINT ''F'(X)=Ø AND NR'S METHOD FAILS''
59380  PRINT : PRINT ''RESELECT FIRST APPROX OF ROOT'': PRINT : GOTO 59260
60000  REM Compute to Minimize Propagation Errors
60020  GOSUB 57000
60040 FL = 1:RM = RT:RT = Ø
60060  HOME : PRINT ''PRELIMINARY LISTING OF ROOTS OF EQU.:-''
60080  GOSUB 60200: INVERSE : PRINT ''  PLEASE WAIT FOR RECOMPUTED RESULTS
        '': NORMAL
60100 P = P(RT + 1):Q = Q(RT + 1)
60120 IR = Ø: GOSUB 51000: IF RT < RM THEN 60100: REM Compute Next Root
60140  REM  Conclude and Display Summary Results
60160  PRINT  CHR$ (12)
60180  HOME : PRINT ''FINAL LISTING OF ROOTS OF EQUATION:-'': GOSUB 60200:
        GOTO 60300
60200  GOSUB 56060: PRINT
60220  FOR I = 1 TO RM: PRINT ''X = '';: IF  ABS (P(I)) > ER THEN  PRINT
       P( I);
60240  IF  ABS (Q(I)) > ER THEN  PRINT '' (+/-) '' ABS (Q(I))'' × I'';
60260  PRINT : PRINT
60280  NEXT : PRINT ''======================================'': RETURN
60300  END
```

First try the solution

(24) $y = Ae^{kx}$

where A and k are constants. By direct substitution and dividing the resulting equation throughout by $-Ae^{kx}$ you will obtain

(25) $k^{10} + 0.77k^8 - 7.9k^6 - 1.44k^4 + 5.18k^2 + 4.275 = 0$

The computer took four minutes to complete the first-stage computation and a further 45 seconds for the second stage to produce the answers:

$k = \pm 0.2692324 \pm 0.8097208i$, ± 1.5266484, ± 1.0626761, $\pm 1.7503178i$

The general solution to equation (23) is therefore

(26) $y = A_1\exp(k_1x) + A_2\exp(k_2x) + \ldots + A_{10}\exp(k_{10}x)$

where the A's are the ten integration constants and the k's are the respective real or complex roots obtained.

The Results

There is no problem of unattainable accuracy up to the limits of the accuracy of the computer. The computer merely must perform extra iterations to achieve the desired results. Convergence is usually very rapid except in regions of x where F'(x) is very small — a general defect of the Newton-Raphson method. To safeguard against this rare eventuality the computer prints the answers after each stage of iteration so that a quick visual inspection can be made. When this occurs a simple remedy is to re-run the program with a different initial trial root.

In the hundreds of equations I have solved using this method I have seldom found it unworkable. I'll leave it to the experts to do a rigorous analysis of the convergence and stability, or otherwise, of the iteration. It is enough to mention that the method will be inaccurate only if both the equations F(x) = 0 and F'(x) = 0 happen to share the same root. At the same time, a very high level of accuracy approaching the computer's own accuracy limits (e.g., ER < 10^{-6}) is expected. For example, when

(27) $F(x) = (1 + x^2)^2 = 1 + 2x^2 + x^4 = 0$

which has the same roots x = ± i as

(28) $F'(x) = 4x(1 + x^2) = 0$

is input with the error tolerance set at ER = 0.000001 the machine settles down to the slightly imperfect result of

x = 1.89330634E − 06 ± i

Conclusion

A general routine has been established that may be used to compute for both the complex and real roots of any polynomial equation. The routine itself does not involve complex numbers and is therefore appropriate for application on a microcomputer (or even a programmable calculator).

By going through two rounds of computations, residual errors propagated over the numerous computation stages can be eliminated, thereby ensuring the stipulated accuracy of the final results. The routine can also be incorporated in applications programs and called as a regular subroutine.

The method described can be extended to equations involving simple trigonometric, hyperbolic, or transcendental functions provided such a function can be expanded as a convergent power series and approximated to a polynomial by truncating at some arbitrary power. Such series are convergent only for ABS(x) < 1. For cases where ABS(x) < 1 a reciprocal transformation y = 1/x can often be tried successfully.

Dr. P.P. Ong has a Ph.D. in ionic physics from University College London. He is employed as a senior lecturer in the Department of Physics, National University of Singapore, and is a member of Institution of Electrical Engineers London. You may contact Dr. Ong at the Physics Department, National University of Singapore, Kent Ridge, Singapore, 0511.

MICRO

Signed Binary Multiplication is Unsigned

by Timothy Stryker

Two's complement notation has surprises in store for those writing integer multiplication routines. A little mathematical analysis shows why.

Multiplication Routine
requires:
6502 computer

Most programmers writing a signed-integer multiplication routine in assembly language would write it in what they consider the most straightforward manner. That is, they would find the absolute values of the multiplicand and the multiplier, multiply them together, and then adjust the sign of the product based on whether the signs of the original multiplicand and multipler were or were not the same.

It is a little-known fact of binary life that this method is not necessary in certain circumstances. In particular, if you plan to make the number of bits of precision in the product equal to the number of bits of precision in the input factors, then the nature of two's complement arithmetic causes the sign computations to come out right without any need for explicit sign handling on your part. Under these conditions, there is *no difference* between a signed and an unsigned integer multiplication routine. This applies whether you use a shift-and-add algorithm, Booth's algorithm, or any other basic multiplication algorithm.

Most programmers will snort in derision at such a proposition — it seems to run counter to all logic. The idea that, in the case of 16-bit numbers for example, multiplying a number by 2 and then inverting it should give the same result as multiplying it by 65534, ignoring all but the low-order 16 bits of the product, seems ludicrous. Nevertheless, that is the case. This article discusses why.

Remember that when you encode a negative integer in two's complement notation, you are actually using the sum of that number with 2 to the power of the number of bits in your word. In mathematical terms, you are encoding $-n$ as

$$2^m + (-n)$$

where m is the number of bits in the word. When you add a pair of two's complement numbers together, the reason that you don't have to special-case their signs is that you ignore all but the low-order m bits of the sum. Since any 2^m terms in the sum contribute only to bit positions above the m-th, these low-order m bits give you the right result. For example, adding 5 to -3 gives you

$$5 + 2^m - 3 = 2^m + 5 - 3 = 2^m + 2$$

the low-order m bits of which represent a 2. Adding -6 to -4 gives you

$$2^m - 6 + 2^m - 4 = 2 * 2^m - 6 - 4 = 2 * 2^m - 10$$

Listing 1

```
OBJECT      ASSEMBLY SOURCE

            * MULT:
            *
            * EXPECTS TO BE CALLED WITH TWO 16-BIT FACTORS
            * ON THE STACK: MULT REPLACES THEM WITH THEIR
            * 16-BIT PRODUCT AND RETURNS.  THE FACTORS AND
            * THEIR PRODUCT MAY BE THOUGHT OF AS EITHER
            * SIGNED OR UNSIGNED, IT MAKES NO DIFFERENCE.
            *
            * MULT IS RELOCATABLE AND USES NO SCRATCHPAD
            * MEMORY, ZERO-PAGE OR OTHERWISE.
            *
            * BY T. STRYKER 4/82 (WITH THANX AND A TIP OF
            * THE HAT TO C. GUILMARTIN AND K. WASSERMAN)
            *
A9 00       MULT   LDA #0         INITIALIZE PRODUCT TO 0
48                 PHA
48                 PHA
BA                 TSX            SET UP FOR STACK INDEXING
A0 10              LDY #16        DO SHIFT-AND-ADD 16 TIMES
5E 08 01    MLOOP  LSR $108,X     SHIFT FIRST FACTOR RIGHT
7E 07 01           ROR $107,X
90 13              BCC SHIFT      BRANCH IF ZERO SHIFTED OUT
18                 CLC            ADD LEFT-SHIFTED SECOND
BD 01 01           LDA $101,X          FACTOR TO PRODUCT
7D 05 01           ADC $105,X
9D 01 01           STA $101,X
BD 02 01           LDA $102,X
7D 06 01           ADC $106,X
9D 02 01           STA $102,X
1E 05 01    SHIFT  ASL $105,X     SHIFT SECOND FACTOR LEFT
3E 06 01           ROL $106,X
88                 DEY            DONE YET?
D0 DC              BNE MLOOP      BRANCH BACK IF MORE TO DO
68                 PLA            REPLACE FIRST FACTOR
9D 07 01           STA $107,X          WITH PRODUCT
68                 PLA
9D 08 01           STA $108,X
68                 PLA            REPLACE SECOND FACTOR
9D 05 01           STA $105,X          WITH RETURN ADDRESS
68                 PLA
9D 06 01           STA $106,X
60                 RTS            AND RETURN
```

the low-order m bits of which are equal to simply $2^m - 10$, namely, -10.

Now consider what happens when you multiply. The case in which both factors are positive need not be considered. The case in which one factor, f, is positive and the other, $-g$, negative, gives you

$$f * (2^m - g) = f*2^m - f*g$$

which, ignoring bit positions above the m-th, is none other than $-(f*g)$. Similarly, the case in which both factors, $-f$ and $-g$, are negative, gives you

$$(2^m - f) * (2^m - g) = 2^m*(2^m - f - g) + f*g$$

which, ignoring a rather large amount of gibberish above the m-th bit position, is simply f*g, as expected.

Listing 1 shows a relocatable 16-bit signed/unsigned integer multiplication routine for the 6502 that takes its arguments from the stack, pops them, and returns their product on the stack. It could be written more efficiently, of course — I have written it this way to make it completely machine-independent. Remember, though, that this approach does have definite limitations: in general, the multiplication of one 16-bit integer by another will yield a 32-bit product. Thus, a routine like the one shown is only applicable in cases where the product is known to fit in 16 bits (languages like RPL and FORTH, for example, typically make this assumption).

The approach given here could be economically applied in a fully general signed multiplication procedure on a processor possessing a hardware sign-extend operation. It would then be necessary only to sign-extend each 16-bit input factor to 32 bits before doing the multiply, yielding a fully general 32-bit result. Unfortunately, this finding does not apply to signed division. At any rate, whether useful to you or not, the above is certainly a surprising and illuminating result. There is more elegance and consistency lurking within the concept of two's complement notation than most of us realize.

Timothy Stryker may be contacted at Samurai Software, P.O. Box 2902, Pompano Beach, FL 33062.

MICRO™

MICRObits

Deadline for MICRObits: 20th of second month before publication; i.e., January 20th for March issue. Send typewritten copy (40-word limit) with $25.00 per insertion. (Subscribers: first ad at $10.00.)

MICRO™

APPLE Math Editor

by Robert D. Walker

This Apple Pascal program allows for easy construction, editing, and printing of mathematical formulas.

Math Editor
requires:

Apple II with Pascal
(optional: Dot-matrix printer such as Epson MX-80)

Anyone who has used a text editor for writing technical papers has encountered the problem of entering mathematical formulas into their text. If the formula is simple it may be typed into the text using the ASCII character set. More commonly, however, I find myself having to leave a blank area within the text and later writing the formula in with pencil. If you want a professional appearance this method is unacceptable. The following program, written in Apple Pascal, will solve this problem.

Although there is only one program here that does the formula editing, there are seven files used throughout this article. For this reason I recommend initializing a new disk, using "APPLE3: FORMATTER". Once this is done, change the volume name of this new disk from "BLANK:" to "MATH:" to make it easier to follow the article. It will also make the file names compatible with those included in the program listings.

Creating the Math Character Set

The math character set includes a special cursor used by the Math Editor, the Greek alphabet, math symbols not included in the ASCII character set, and small digits used for subscripting and superscripting. In addition, there is room for two user-definable characters. These images (81 total) are stored in the textfile "MATH: MATHSET. TEXT" (see listing 1). This textfile will be used to create the datafile "MATH: MATHSET. DATA", which contains these same images in a form readable by the Math Editor. *(Ed. Note: Listing 1 has three full-size samples. Figure 1 has a dot matrix reduction of the characters. They should all be entered in "X" format.)*

A few special rules must be followed when entering these images into the textfile. First, each image is an eight by eight dot matrix. Accordingly, each image occupies exactly eight lines of text, with each line having at least eight characters. Extra characters on each line are ignored and may be used for documentation. Second, the uppercase character "X" will show up as a white dot on the screen. All other characters will show up as black. Third, the first image of this textfile must be the special cursor. Fourth, there cannot be a linespace between images. Last, there must be 81 images (648 lines) in this textfile. Additional lines will be ignored.

Creating the math character datafile requires a small utility program (see listing 2, MATH: MATHCREATE. TEXT). This program should be entered and compiled. When executed, this program (MATH: MATHCREATE. CODE) will read "MATH: MATHSET. TEXT" and create "MATH: MATHSET. DATA".

The major advantage of this storage method is that the textfile "MATH: MATHSET. TEXT" can easily be edited to suit the user's needs. Once this is done the datafile can be created by simply executing "MATH: MATHCREATE. CODE".

If you have followed all the steps up to this point then the following files should exist:

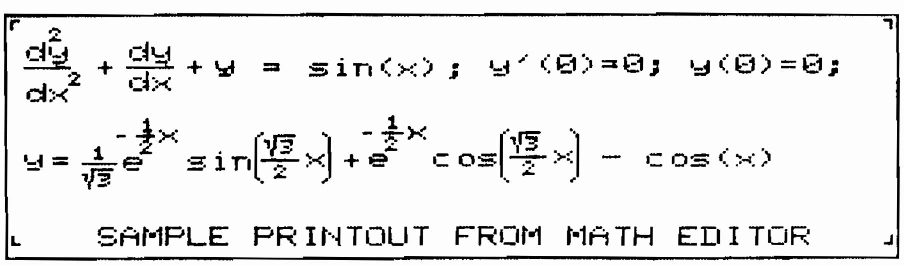
1. MATH:MATHSET.TEXT
2. MATH:MATHSET.DATA
3. MATH:CREATEMATH.TEXT
4. MATH:CREATEMATH.CODE

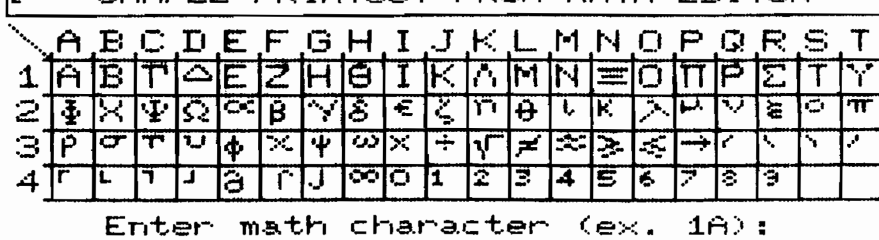
Math Editor—Program Operation

The Math Editor program uses the TURTLEGRAPHICS library unit to display the formulas and messages on the high-resolution screen. The math formula is displayed on the upper half of the screen, while all messages are displayed on the lower half.

The program is entirely menu-driven and calls on nine main procedures. Once a procedure is called, simply hitting the return key will return the program to the main menu.

The "A(SCII" command is used for putting text on the display. From the main program this procedure is called by typing "A". The user will then be prompted to enter the string. This





Listing 1: MATHSET.TEXT (Partial)

```
...X....  (Ø) CURSOR
...X....
........
XX...XX.
........
...X....
...X....
........
..XX....  (1) ALPHA
.X..X...
X....X..
XXXXXX..
X....X..
X....X..
X....X..
........
XXXXX...  (2) BETA
.X...X..
.X...X..
.XXXX...
.X...X..
.X...X..
XXXXX...
........
```

should be ended with a return character. The string will then be drawn in the lower left corner of the formula display area. Next, the moving menu will be displayed. The user will then use the keyboard for moving the string on the display.

The moving menu consists of six commands. "U)p", "D(own", "L(eft", and "R(ight" move the string on the display. When these commands are first encountered they move the string ten dots on each keypress. The "S(mall movement" command is for small movement of the string. This command causes the string to be moved only one dot per keypress. Once the string is in the desired position, it is frozen by using the "F(reeze" command. This causes program control to return to the main menu.

The "M(athset" command is used to draw math characters on the display. When this command is invoked it displays the entire math character set in a table (see figure 1). The user selects the character by entering the row number followed by the column letter. Once this is done, the character will be displayed and moved as explained above.

Some characters, such as parentheses, brackets, and the integral sign must be drawn at different sizes. These characters are drawn as two halves. The "D(ots" command (described below) is then used to draw the midsection of these split characters.

The "D(ots" command allows the user to put dots on the screen. The user first moves the cursor to the position where the first dot will be drawn and then freezes its position. The moving command is then used to determine where the next dot will be drawn. The "E(rase" command erases the last dot drawn. To exit this procedure simply hit the return key.

If a mistake is made while drawing a formula, then the "E(dit" command can be used to erase the most recently drawn character. For example, if the last operation was drawing a string on the display, then only the last character of the string will be erased. Likewise, if the last operation was drawing dots, then only the last dot drawn will be erased.

There are two commands used for loading and saving formulas on a "Math:" disk. First the "L(oad" command will clear the current display and load a previously stored formula. The "S(ave" command is used for saving the displayed formula. Both of these commands are written to avoid program interruption due to a disk I/O error.

Once a formula is constructed, the "P(rint" command is used to print a hardcopy of the display. The procedure

"PRINT" was specifically written for the Epson MX-80 equipped with Graftrax. This procedure takes about 100 seconds to print the formula display area of the screen.

The "C(lear" command is used to erase the current formula from the screen. This allows the user to start from scratch.

The "Q(uit" command simply verifies that the user wants to quit the program.

Conclusions

This program has the capacity for future expansion. For instance, an ambitious programmer might include a procedure for drawing variable size symbols such as parentheses and brackets. In addition, you might want to rewrite the editing procedure so that characters could be erased in any order. Both of these modifications increase the size of the program dramatically.

You may contact the author at 2850 Delk Rd., Apt. 2B, Marietta, GA 30067

***Listing 2:* MATHCREATE.TEXT**

```
(*$L PRINTER: *)

(*****************************************************************************************)
(**                                                                                    **)
(** This program creates the MATHSET.DATA file from the                                **)
(** MATHSET.TEXT file. MATHSET.DATA file is used by MATH EDITOR.                       **)
(**                                                                                    **)
(*****************************************************************************************)

PROGRAM CREATEMATHSETDATAFILE;

TYPE  CHARARRAY=PACKED ARRAY[0..80,0..7,0..7] OF BOOLEAN;

VAR CHTEXT: TEXT;
   CHARRAY: CHARARRAY;
   CHDATA: FILE OF CHARARRAY;
   S: STRING;
   I,ROW,COLUMN: INTEGER;

BEGIN
 RESET(CHTEXT,'MATH:MATHSET.TEXT');
 REWRITE(CHDATA,'MATH:MATHSET.DATA');
 FOR I:=0 TO 80 DO  (* read 81 image *)
  BEGIN
   WRITE(CHR(12));  (* clear screen *)
   FOR ROW:=7 DOWNTO 0 DO  (* invert image *)
    BEGIN
     READLN(CHTEXT,S);
     WRITELN(S);  (* echo image *)

     (* put image into character array *)
     (* 'X' is true, all other characters are false *)
     FOR COLUMN:=0 TO 7 DO CHARRAY[I,ROW,COLUMN]:=(S[COLUMN+1]='X')

    END
  END;
 CHDATA  =CHARRAY;
 PUT(CHDATA);
 CLOSE(CHDATA,LOCK)
END.
```

(Listing 3 begins on page 81)

Listing 3: MATHED.TEXT

```
(*$L PRINTER:*)

(* FIRST TEXTFILE - MATHED1 *)

(*****************************************************************************************)
(**                                                                                     **)
(** PROGRAM:  MATH EDITOR                                                               **)
(**                                                                                     **)
(** AUTHOR:   Robert D. Walker, Marietta, GA                                            **)
(**                                                                                     **)
(** SUMMARY: This program uses the Apple high-resolution graphics for displaying and editing **)
(**           math formulas. Once the formula is generated it may be edited, saved, loaded, or **)
(**           printed. The printout procedure was specifically written for the Epson MX-80 **)
(**           printer with Graftrax.                                                    **)
(**                                                                                     **)
(** HARDWARE:Apple II, Epson MX-80 printer with Graftrax                                **)
(**                                                                                     **)
(** LANGUAGE:Apple Pascal version 1.1                                                   **)
(**                                                                                     **)
(*****************************************************************************************)
(*$S+*)

PROGRAM MATHEDITOR;

USES TURTLEGRAPHICS;

CONST MAXDSP=1000; (* maximum no. of shapes drawn to screen *)

TYPE (* array of math characters *)
    CHARARRAY=PACKED ARRAY[0..80,0..7,0..7] OF
    BOOLEAN;

     (* record of shapes displayed on screen *)
     DSPRECORD=RECORD
           CHTYPE:  (ASCII,MATHSET,DOT);
           CHCODE,XPOS,YPOS:  INTEGER
          END;

VAR CHARRAY: CHARARRAY;
   DSPARRAY: ARRAY[1..MAXDSP] OF DSPRECORD;
   LINENUM,NUMDSP: INTEGER;
   CH: CHAR;
   PIXEL: BOOLEAN;

PROCEDURE INIT; (* read math character file and initialize variables *)
VAR CHDATA: FILE OF CHARARRAY;
BEGIN
 INITTURTLE;
 MOVETO(96,150); WCHAR(chr(1));
 MOVETO(106,149); WSTRING('MATH EDITOR');
 RESET(CHDATA,'MATH:MATHSET.DATA');
 CHARRAY:=CHDATA
 INITTURTLE;
 VIEWPORT(0,279,102,191); FILLSCREEN(WHITE);
 VIEWPORT(1,278,103,190); FILLSCREEN(BLACK);
 CHARTYPE(6);
 PIXEL:=TRUE;
 NUMDSP:=0
END;

PROCEDURE DRAWSHAPE(I: INTEGER); (* (re)draw a single shape *)
BEGIN
 WITH DSPARRAY[I] DO
       CASE CHTYPE OF
```

Listing 3 *(continued)*

```
                    (* draw ASCII character *)
          ASCII:    BEGIN
                     MOVETO(XPOS,YPOS);
                     WCHAR(CHR(CHCODE))
                    END;

                    (* draw math character *)
          MATHSET:  DRAWBLOCK(CHARRAY[CHCODE],
                    2,0,0,8,8,XPOS,YPOS,6);

                    (* draw single dot *)
          DOT:      DRAWBLOCK(PIXEL,
                    1,0,0,1,1,XPOS,YPOS,10)
      END
END;

PROCEDURE DRAWDISPLAY; (* redraw display with stored shapes *)
VAR DSPINDEX: INTEGER;
BEGIN
 VIEWPORT(1,278,103,190); (* erase display *)
 FILLSCREEN(BLACK);
 IF NUMDSP <> 0 THEN FOR DSPINDEX:=1 TO NUMDSP DO
  DRAWSHAPE(DSPINDEX);
 VIEWPORT(0,279,0,191)
END;

PROCEDURE CLEARMESSAGE; (* clear message area of screen *)
BEGIN
 VIEWPORT(0,279,0,101); FILLSCREEN(BLACK);
 VIEWPORT(0,279,0,191);
 LINENUM:=90 (* This is top line where messages are written. *)
END;

PROCEDURE WRITEMESSAGE(S: STRING); (* write message & update linenum *)
BEGIN
 MOVETO(0,LINENUM); WSTRING(S);
 LINENUM:=LINENUM-9;
 IF LINENUM < 0 THEN LINENUM:=0
END;

PROCEDURE IOERROR(I: INTEGER); (* display disk I/O error *)
VAR CH: CHAR;
BEGIN
 CLEARMESSAGE;
 WRITEMESSAGE('I/O ERROR ');

 (* write I/O ERROR number *)
 WCHAR(CHR((I DIV 10)+48));
 WCHAR(CHR((I MOD 10)+48));

 WRITEMESSAGE('');
 CASE I OF
   2:  WRITEMESSAGE('BAD DEVICE NUMBER');
   3:  WRITEMESSAGE('ILLEGAL OPERATION');
   5:  WRITEMESSAGE('LOST DEVICE- NO LONGER ON LINE');
   6:  WRITEMESSAGE('LOST FILE- NO LONGER IN DIRECTORY');
   7:  WRITEMESSAGE('BAD TITLE- ILLEGAL FILENAME');
   8:  WRITEMESSAGE('NO ROOM');
   9:  WRITEMESSAGE('NO DEVICE- VOLUME NOT ON LINE');
  10:  WRITEMESSAGE('NO SUCH FILE ON SPECIFIED VOLUME')
 END;
 WRITEMESSAGE('');
 WRITEMESSAGE('Hit <return> to continue...');
 READ(CH)
END;
```

Listing 3 *(continued)*

```
PROCEDURE GETSTRING(VAR S: STRING); (* assemble & display string *)
VAR S1: STRING;
    S2: STRING[1];
    CH: CHAR;
BEGIN
  S1:='';
  S2:=' ';
  REPEAT
    READ(CH);
    IF CH IN [' '..'z']
      THEN
        BEGIN
          WCHAR(CH);
          S2[1]:=CH;
          S1:=CONCAT(S1,S2)
        END;
    IF CH=CHR(8) THEN IF LENGTH(S1)>0
                    THEN
                      BEGIN
                        MOVETO(TURTLEX-7,TURTLEY);
                        WCHAR(S1[LENGTH(S1)]);
                        MOVETO(TURTLEX-7,TURTLEY);
                        S1:=COPY(S1,1,LENGTH(S1)-1)
                      END
  UNTIL EOLN;
  S:=COPY(S1,1,LENGTH(S1)-1)
END;

PROCEDURE MOVINGMENU; (* display moving menu *)
BEGIN
  CLEARMESSAGE;
  WRITEMESSAGE('U(p');
  WRITEMESSAGE('D(own');
  WRITEMESSAGE('L(eft');
  WRITEMESSAGE('R(ight');
  WRITEMESSAGE('S(mall movement');
  WRITEMESSAGE('F(reeze');
  WRITEMESSAGE('');
  WRITEMESSAGE('<return> - EXIT Without Freeze')
END;

(*$IMATHED2*)
(* SECOND TEXTFILE - MATHED2 *)

PROCEDURE DRAWASCII;
(*****************************************************************************************)
(**                                                                                     **)
(** This is a main procedure which draws ASCII characters.                              **)
(**                                                                                     **)
(*****************************************************************************************)
VAR I,INC,X,Y: INTEGER;
     S: STRING;
     CH: CHAR;
BEGIN
  CLEARMESSAGE;
  WRITEMESSAGE('Enter string:');
  VIEWPORT(1,278,103,190);
  X:=2;  Y:=104;  MOVETO(X,Y);
  GETSTRING(S);
  IF LENGTH(S)=0 THEN EXIT(DRAWASCII);
  MOVINGMENU;
  INC:=10;
```

Listing 3 *(continued)*

```
  REPEAT
    READ(CH);
    MOVETO(X,Y);
    WSTRING(S);
    IF EOLN THEN EXIT(DRAWASCII);
    CASE CH OF
      'U','u': Y:=Y+INC;
      'D','d': Y:=Y-INC;
      'L','l': X:=X-INC;
      'R','r': X:=X+INC;
      'S','s': INC:=1
    END;
    IF Y<103 THEN Y:=103; (* an odd thing happens outside viewport *)
    MOVETO(X,Y);
    WSTRING(S)
  UNTIL CH IN ['F','f'];

  (* Store ASCII string in display array *)
  FOR I:=1 TO LENGTH(S) DO
    BEGIN
      NUMDSP:=NUMDSP+1;
      WITH DSPARRAY[NUMDSP] DO
        BEGIN
          CHTYPE:=ASCII;
          CHCODE:=ORD(S[I]);
          XPOS:=(I-1)*7+X;
          YPOS:=Y
        END
    END
END;

PROCEDURE DRAWMATHSET;
(*****************************************************************************************)
(**                                                                                     **)
(** This is a main procedure which draws math characters.                               **)
(**                                                                                     **)
(*****************************************************************************************)
VAR I,INC,MATHCODE,X,Y: INTEGER;
     CH: CHAR;
  (* This procedure displays the selection of math characters. *)
  PROCEDURE DISPLAYMATHSET;
  BEGIN

    (* draw vertical lines *)
    FOR X:=0 TO 20 DO
      BEGIN
        PENCOLOR(NONE); MOVETO(X*11+28,41);
        PENCOLOR(WHITE); MOVETO(X*11+28,87)
      END;

    (* draw horizontal lines *)
    FOR Y:=0 TO 4 DO
      BEGIN
        PENCOLOR(NONE); MOVETO(26,Y*11+41);
        PENCOLOR(WHITE); MOVETO(248,Y*11+41)
      END;

    (* draw diagonal line *)
    PENCOLOR(NONE); MOVETO(28,85);
    PENCOLOR(WHITE); MOVETO(17,96);

    (* lable table *)
    PENCOLOR(NONE);
```

Listing 3 *(continued)*

```
  FOR X:=0 TO 19 DO
   BEGIN
     MOVETO(X*11+30,87);
     WCHAR(CHR(X+65))
   END;
  MOVETO(19,76); WCHAR('1');
  MOVETO(19,65); WCHAR('2');
  MOVETO(19,54); WCHAR('3');
  MOVETO(19,43); WCHAR('4');

 (* put mathset on screen *)
  FOR I:=1 TO 80 DO
   BEGIN
     X:=((I-1) MOD 20)*11+30;
     Y:=(3-((I-1) DIV 20))*11+43;
     DRAWBLOCK(CHARRAY[I],2,0,0,8,8,X,Y,6)
   END

 END; (* DISPLAYMATHSET *)

BEGIN (* DRAWMATHSET *)
 CLEARMESSAGE;
 DISPLAYMATHSET;
 LINENUM:=21;
 WRITEMESSAGE('Enter math character (ex. 1A):  ');

 (* Get two valid characters from user *)
 (* A <return> character will cause an exit from DRAWMATHSET *)
 REPEAT READ(CH) UNTIL (CH IN ['1'..'4']) OR EOLN;
 IF CH=' ' THEN EXIT(DRAWMATHSET);
 WCHAR(CH);
 MATHCODE:=(ORD(CH)-49)*20;
 REPEAT READ(CH) UNTIL (CH IN ['A'..'T','a'..'t']) OR EOLN;
 IF CH=' ' THEN EXIT(DRAWMATHSET);
 WCHAR(CH);
 IF CH IN ['A'..'T'] THEN MATHCODE:=MATHCODE+ORD(CH)-64;
 IF CH IN ['a'..'t'] THEN MATHCODE:=MATHCODE+ORD(CH)-96;

 (* Draw math character and move it. *)
 MOVINGMENU;
 VIEWPORT(1,278,103,190);
 X:=2; Y:=104; INC:=10;
 DRAWBLOCK(CHARRAY[MATHCODE],2,0,0,8,8,X,Y,6);
 REPEAT
  READ(CH);
  DRAWBLOCK(CHARRAY[MATHCODE],2,0,0,8,8,X,Y,6);
  IF EOLN THEN EXIT(DRAWMATHSET);
  CASE CH OF
   'U','u': Y:=Y+INC;
   'D','d': Y:=Y-INC;
   'L','l': X:=X-INC;
   'R','r': X:=X+INC;
   'S','s': INC:=1
  END;
  IF Y<103 THEN Y:=103;
  DRAWBLOCK(CHARRAY[MATHCODE],2,0,0,8,8,X,Y,6)
 UNTIL CH IN ['F','f'];

 (* Store math character in display array *)
 NUMDSP:=NUMDSP+1;
 WITH DSPARRAY[NUMDSP] DO
  BEGIN
   CHTYPE:=MATHSET;
   CHCODE:=MATHCODE;
```

Listing 3 *(continued)*

```
    XPOS:=X;
    YPOS:=Y
   END

 END; (* DRAWMATHSET *)
 PROCEDURE DRAWDOTS;
 (*******************************************************************************************)
 (**                                                                                       **)
 (** This is a main procedure which draws dots.                                            **)
 (**                                                                                       **)
 (*******************************************************************************************)
 VAR INC,STARTNUM,X,Y: INTEGER;
    CH: CHAR;
 BEGIN

  (* move cursor *)
  MOVINGMENU;
  X:=2; Y:=104; INC:=10;
   DRAWBLOCK(CHARRAY[0],2,0,0,8,8,X,Y,6);
  REPEAT
   READ(CH);
   DRAWBLOCK(CHARRAY[0],2,0,0,8,8,X,Y,6);
   IF EOLN THEN EXIT(DRAWDOTS);
   CASE CH OF
    'U','u': Y:=Y+INC;
    'D','d': Y:=Y-INC;
    'L','l': X:=X-INC;
    'R','r': X:=X+INC;
    'S','s': INC:=1
   END;
   DRAWBLOCK(CHARRAY[0],2,0,0,8,8,X,Y,6)
  UNTIL CH IN ['F','f'];
  DRAWBLOCK(CHARRAY[0],2,0,0,8,8,X,Y,6);

  (* draw dots *)
  X:=X+3; Y:=Y+4;
  CLEARMESSAGE;
  WRITEMESSAGE('U(p');
  WRITEMESSAGE('D(own');
  WRITEMESSAGE('L(eft');
  WRITEMESSAGE('R(ight');
  WRITEMESSAGE('E(rase');
  WRITEMESSAGE('');
  WRITEMESSAGE('<return> - EXIT');
  STARTNUM:=NUMDSP;
  DRAWBLOCK(PIXEL,1,0,0,1,1,X,Y,10);
  NUMDSP:=NUMDSP+1;
  WITH DSPARRAY[NUMDSP] DO
   BEGIN
     CHTYPE:=DOT;
     XPOS:=X;
     YPOS:=Y
   END;
  REPEAT
   READ(CH);
   CASE CH OF
    'U','u': Y:=Y+1;
    'D','d': Y:=Y-1;
    'L','l': X:=X-1;
    'R','r': X:=X+1;
    'E','e': IF (NUMDSP>STARTNUM)  (* erase dot *)
             THEN
              BEGIN
                PIXEL:=FALSE;
                DRAWBLOCK(PIXEL,1,0,0,1,1,X,Y,10);
```

Listing 3 *(continued)*

```
                NUMDSP:=NUMDSP-1;
                X:=DSPARRAY[NUMDSP].XPOS;
                Y:=DSPARRAY[NUMDSP].YPOS;
                PIXEL:=TRUE
               END;
   END;
   IF CH IN ['U','u','D','d','L','l','R','r']
    THEN
     BEGIN
      DRAWBLOCK(PIXEL,1,0,0,1,1,X,Y,10);
      NUMDSP:=NUMDSP+1;
      WITH DSPARRAY[NUMDSP] DO
       BEGIN
        CHTYPE:=DOT;
        XPOS:=X;
        YPOS:=Y
       END
     END
  UNTIL EOLN
 END;

 PROCEDURE EDIT;
 (****************************************************************************)
 (**                                                                        **)
 (** This is a main procedure which edits the display array by              **)
 (** by deleting the last stored shape.                                     **)
 (**                                                                        **)
 (****************************************************************************)
 VAR CH: CHAR;
 BEGIN
  CLEARMESSAGE;
  WRITEMESSAGE('E(rase');
  WRITEMESSAGE('');
  WRITEMESSAGE('<return> - EXIT');
  REPEAT
   IF NUMDSP=0 THEN EXIT(EDIT);
   READ(CH);
   IF CH IN ['E','e']
    THEN
     BEGIN
      NUMDSP:=NUMDSP-1;

      (* erase shape *)
      PIXEL:=FALSE;
      DRAWSHAPE(NUMDSP+1);
      PIXEL:=TRUE

     END
  UNTIL EOLN;
 END;

 (*$I-*)
 PROCEDURE LOAD;
 (****************************************************************************)
 (**                                                                        **)
 (** This is a main procedure which loads a previously saved display.       **)
 (**                                                                        **)
 (****************************************************************************)
 VAR DSPFILE: FILE OF DSPRECORD;
     S: STRING;
     I: INTEGER;
 BEGIN
  CLEARMESSAGE;
```

Listing 3 *(continued)*

```
  GETSTRING(S);
  IF LENGTH(S)=0 THEN EXIT(LOAD);
  S:=CONCAT('MATH:',S,'.DSP');
  RESET(DSPFILE,S);
  I:=IORESULTS;
  IF I<>0
   THEN
    BEGIN
     IOERROR(I);
     EXIT(LOAD)
    END
   ELSE  (* load DSPARRAY *)
    BEGIN
     NUMDSP:=0;
     WHILE NOT EOF(DSPFILE) DO
      BEGIN
       NUMDSP:=NUMDSP+1;
       DSPARRAY[NUMDSP]:=DSPFILE^;
       GET(DSPFILE);
       I:=IORESULTS;
       IF I<>0
        THEN
         BEGIN
          IOERROR(I);
          EXIT(LOAD)
         END
      END
    END
 END;

 PROCEDURE SAVE;
 (****************************************************************************)
 (**                                                                        **)
 (** This is a main procedure which saves the current display.              **)
 (**                                                                        **)
 (****************************************************************************)
 VAR DSPFILE: FILE OF DSPRECORD;
   S: STRING;
   DSPINDEX,I: INTEGER;
 BEGIN
  IF NUMDSP=0 THEN EXIT(SAVE); (* don't save empty display *)
  CLEARMESSAGE;
  WRITEMESSAGE('Enter File Name:  ');
  GETSTRING(S);
  IF LENGTH(S)=0 THEN EXIT(SAVE);
  S:=CONCAT('MATH:',S,'.DSP');
  REWRITE(DSPFILE,S);
  I:=IORESULTS;
  IF I<>0
   THEN
    BEGIN
     IOERROR(I);
     EXIT(SAVE)
    END
   ELSE (* save DSPARRAY *)
    BEGIN
     FOR DSPINDEX:=1 TO NUMDSP DO
      BEGIN
       DSPFILE^:=DSPARRAY[DSPINDEX];
       PUT(DSPFILE);
       I:=IORESULTS;
       IF I<>0
        THEN
```

Listing 3

```
            BEGIN
              IOERROR(I);
              EXIT(SAVE)
            END
          END;
        CLOSE(DSPFILE,LOCK)
      END;
  I:=IORESULTS;
  IF I<>0 THEN IOERROR(I)
END;
(*$I+*)

(*$R-*) (* speed up this procedure *)
PROCEDURE PRINT;
(*******************************************************************************)
(**                                                                           **)
(** This is a main procedure which prints the display. This                   **)
(** procedure is written for the EPSON MX-80 with GRAFTRAX.                   **)
(**                                                                           **)
(*******************************************************************************)
TYPE BYTE=0..255;

    (* use variant record to associate printing wires w/byte *)
    WIRES=PACKED RECORD CASE BOOLEAN OF
          TRUE:  (BO:  PACKED ARRAY[0..7] OF BOOLEAN);
          FALSE: (BY:  BYTE)
        END;

VAR I,X,Y,YINC:  INTEGER;
    A: PACKED ARRAY[1..278] OF BYTE;
    S: PACKED ARRAY[1..4] OF BYTE;
    W: WIRES;
    CH: CHAR;
    P: INTERACTIVE;
BEGIN
  CLEARMESSAGE;
  WRITEMESSAGE('Print screen (Y/N)? ');
  READ(CH);
  IF NOT (CH IN ['Y','y']) THEN EXIT(PRINT);
  CLEARMESSAGE;
  WRITEMESSAGE('Printing screen...');
  REWRITE(P,'PRINTER:');

  (* set line spacing 24/216 *)
  S[1]:=27; S[2]:=51; S[3]:=24;
  UNITWRITE(6,S[1],3,0,12);

  (* print screen *)
  FOR I:=10 DOWNTO 0 DO
    BEGIN
      WRITE(P,'':17); (* center printout *)
      S[1]:=27;  S[2]:=75;  S[3]:=22;  S[4]:=1;
      UNITWRITE(6,S[1],4,0,12);
      FOR X:=1 TO 278 DO
        BEGIN
          FOR YINC:=0 TO 7 DO
            W.BO[YINC]:=SCREENBIT(X,(I*8)+103+YINC);
          A[X]:=W.BY
        END;
      UNITWRITE(6,A[1],278,0,12);
      WRITELN(P)
    END;
```

Listing 3 *(continued)*

```
  S[1]:=27;  S[2]:=64;
  UNITWRITE(6,S[1],2,0,12)
END;
(*$R+*)

PROCEDURE CLEAR;
(*******************************************************************************)
(**                                                                           **)
(** This is a main procedure which clears the display.                        **)
(**                                                                           **)
(*******************************************************************************)
VAR CH: CHAR;
BEGIN
  CLEARMESSAGE;
  WRITEMESSAGE('Clear screen (Y/N)? ');
  READ(CH);
  IF CH IN ['Y','y'] THEN NUMDSP:=0
END;

PROCEDURE QUIT;
(*******************************************************************************)
(**                                                                           **)
(** This is a main procedure which checks if the user wants to quit.          **)
(**                                                                           **)
(*******************************************************************************)
VAR CH1: CHAR;
BEGIN
  CLEARMESSAGE;
  WRITEMESSAGE('Quit (Y/N)? ');
  READ(CH1);
  IF NOT(CH1 IN ['Y','y']) THEN CH:=' '
END;

BEGIN (* MATHEDITOR - main program *)
  INIT;
  REPEAT
    DRAWDISPLAY;
    CLEARMESSAGE;
    WRITEMESSAGE('A(SCII');
    WRITEMESSAGE('M(athset');
    WRITEMESSAGE('D(raw');
    WRITEMESSAGE('E(dit');
    WRITEMESSAGE('L(oad');
    WRITEMESSAGE('S(ave');
    WRITEMESSAGE('P(rint');
    WRITEMESSAGE('C(lear');
    WRITEMESSAGE('Q(uit');
    WRITEMESSAGE('');
    WRITEMESSAGE('Enter command:');
    REPEAT READ(CH) UNTIL CH IN ['A','a','M','m','D','d','E','e','L','l',
                       'S','s','P','p','C','c','Q','q'];
    CASE CH OF
      'A','a': DRAWASCII;
      'M','m': DRAWMATHSET;
      'D','d': DRAWDOTS;
      'E','e': EDIT;
      'L','l': LOAD;
      'S','s': SAVE;
      'P','p': PRINT;
      'C','c': CLEAR;
      'Q','q': QUIT
    END
  UNTIL CH IN ['Q','q'];
  WRITE(CHR(12))
END.
```

Using Long Integers for BCD Numbers in Pascal

by David C. Oshel

This article presents a bullet-proof string conversion for Pascal 1.1 long integers with implied decimal points.

BCDNUMS Demo
requires:
Pascal

English is an unimplemented programming language because it has no compiler; in some respects, Pascal belongs in the same category. While Pascal is a strong "top-down" programming language, some of its versions are particularly weak as "top-to-bottom" practical tools. Unlike extremely practical languages like FORTRAN and COBOL, Pascal has left a number of design decisions undecided, perhaps because few people have found uses for Pascal when other languages are available.

In the microcomputer field, however, Pascal's strengths outweigh some of its weaknesses, and it is virtually the only choice when program reliability and size are significant design criteria (with the possible exceptions of FORTH and Microsoft FORTRAN.) There is no question that Pascal is influential; the recent appearance of structured FORTRAN, or "FORTRAN77," is sufficient proof that concepts embodied in Pascal are worth learning well.

Pascal's worst failing is its inability to read numerical data efficiently. This article presents one method for interpreting Long Integers as accurate bcd numbers with decimal points. The difference between a Long Integer and a bcd number is usually that the bcd number has an implied decimal point; the Long Integer (its internal structure is not as interesting as its uses), can be interpreted several ways.

Long Integers can be used to represent dollar-and-cents amounts. All dollar amounts are represented as multiples of 100 cents, and the decimal point is understood to be two places in from the right. If you are working with millage rates or titrations, then you may understand the decimal point to be three, four, or five places in from the right, provided the bcd number is properly normalized. By using bcd numbers, complemented with appropriate special algorithms for multiplication and division, you may avoid the rounding errors that are sometimes the bane of ordinary floating-point variables. The price you pay is twofold: first, bcd numbers have a large overhead in terms of memory and disk space. Second, the Pascal interface to bcd numbers is an exponential function of aggravation.

Two procedures are outlined in this article. The first, a function called BCDVAL, scans an input string, converts it to a bcd number, and returns the boolean value TRUE if conversion was successful. The second procedure, called STRBCD, provides the inverse utility by converting a bcd number to an ASCII string. You may also select whether to affix the minus sign ahead of or behind the number, consistent with business practice.

The normalization constant "Rightsize" is actually a variable, as used here. It is global to both BCDVAL and STRBCD. Note one subtle point: all bcd numbers entered with a particular Rightsize are actually *typed* variables, but the Pascal operating system will have no inkling of the fact because all it sees are Long Integers. If you wish to inform Pascal that you are working with typed variables, you should declare a Record type, which maintains the bcd variable and its associated implied decimal point together. If program logic allows, you may still avoid the Record type (and attendant overhead) by normalizing short numbers "on the fly" to the longest required Rightsize before you attempt arithmetic on the variables. In most applications, neither maneuver will be necessary because most applications do not mix types. If you select Rightsize = 2, then you are working with dollars and cents.

The BCDVAL function protects itself from abnormal input and is safe to use with READLN. In other words, your program will not crash if BCDVAL encounters non-numeric characters, duplicate decimal points, etc. If the input is likely to cause a Long Integer range error, which can have dangerously unpredictable side-effects on the operating system, then BCDVAL takes its normal error exit and returns FALSE, along with the formal parameter BCD = 0. In the interest of program brevity there is no indication of what actually caused the error.

There is also a possibility that some valid inputs will take the overflow exit; this occurs when the length of the input, plus the variable Rightsize, exceeds the Long Integer parameter Maxlint (i.e., TYPE Lint = Integer [Maxlint].) You may avoid the problem either by declaring a larger Maxlint (and then re-compiling), or by writing a more intelligent overflow trap. The trap provided is conservative, and will correctly flag *all* overflow errors, plus a few of the *almost* cases. In Apple Pascal, Maxlint may range as high as 36; refer to the *Pascal Language Reference Manual* for details.

David Oshel works as a consultant designing small information management systems for Democratic political candidates. You may contact him at 1219 Harding Ave., Ames, IA 50010.

Listing 1: BCDNUMS Demo

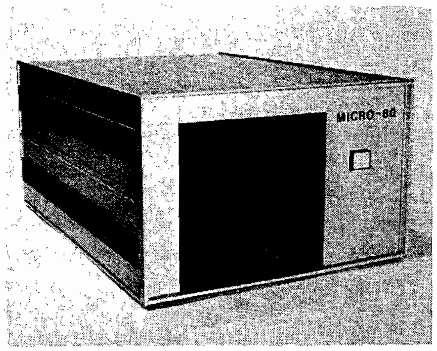
```
Program Bcdnums-Demo;
{ ★ Demonstrating the use of Apple Pascal 1.1 Long Integers as
  ★ Binary-coded-decimal numbers with implied decimal points.
  ★
  ★ David C. Oshel
  ★ 1219 Harding Ave.
  ★ Ames, Iowa 50010  —— March 17, 1982    }
CONST Maxlint = 16;   { Occupies 10 bytes }
      NULL  = '';     { Concatenation constants }
      SPACE = ' ';
      ZERO  = '0';
      RADIX = '.';        { Decimal point char in Bcdnums }
      MINUSIGN = '-';
TYPE  Charset = Set of Char;
      Bcdnum  = Integer[Maxlint];  { BCDVAL normalizes these to Rightsize }
VAR   Rightsize: Integer;   { Number of dec places in Bcdnums; default = 2 }
      { Demo program variables follow }
      P,Q : Bcdnum;
      S : String;
      Loop : Integer;
      Num,Minusloc  : Boolean;
Function  BCDVAL(VAR Numstr:String; VAR Bcd:Bcdnum):Boolean;
  { All BCDVAL's are normalized; e.g., Dollar values ($0.00 are
    represented internally as multiples of 100cents.  Normalization
    is then a simple function of Rightsize, as is radix insertion...
    The default is Rightsize=2, for Dollars-and-cents; note that input
    range errors when working with Long Integers cause a fatal SYSTEM
    crash...!   }
  VAR I,J,K,Len : Integer;
      Got-radix,Minus : Boolean;
      Numeric : Charset;
      Temp : String;
      T1   : String[1];
  Procedure Valerr;  {Overflow, or near enough to...}
    Begin
      Write(chr(7));
      Exit(Bcdval) {Conditions on Exit: Bcdval=False, Bcd=0}
    End; { Valerr }
  Procedure Truncate;
    Begin
      { Truncate extra digits right of radix... }
      J := Pos(Radix,Temp);
      If J <> 0 then
        Begin
          If (Length(Temp) - J + 1) > Rightsize then
             Temp:=Copy(Temp,1,J+Rightsize)
        End
    End;  { Truncate }
  Procedure Goodstring;
   { Shift all chars that belong in legal Bcd numbers
     into a temporary string accumulator & chop the remainder..  }
    VAR Okset : Charset;
    Begin  { Goodstring }
      Okset := Numeric + ['+','-',Radix];  { Ignore $, commas, etc. }
      For I := 1 to Len do
        Begin
          If Numstr[I] in Okset then
            Begin
              If Numstr[I] in ['+','-'] then
                Begin
                  Okset := Okset - ['+','-'];
                  If Numstr[I] = '-' then Minus := True
                End
              Else If Numstr[I] in Numeric + [Radix] then
                Begin
                  If Numstr[I] = Radix then Okset := Okset - [Radix];
                  T1[1] := Numstr[I];
                  Temp := Concat(Temp,T1)
                End
            End
        End
    End;  { Goodstring }
  Procedure Normalize;
    Begin
      While J < Rightsize do
        Begin
          Bcd := Bcd ×10;
          J := J + 1
        End
    End;  { Normalize }
  Begin  { Bcdval }
    BCDVAL := False;
    Numeric := ['0'..'9'];
    T1   := SPACE;
    Temp := NULL;
```

BCDNUMS Demo *(continued)*

```
    Bcd:=0; Len := Length(Numstr); Minus := False; Got-radix := False;
    If Len = 0 then Exit(Bcdval);
    If (Rightsize < 0) or (Rightsize > Maxlint) then Rightsize:=0;
    Goodstring;   { Zap spaces, $$, commas, any extra garbage }
    If Length(Temp)+Rightsize >= Maxlint then Valerr;
    Truncate;      { Drop extra digits on the right }
    Len := Length(Temp);  { New length... }
    If (Len = 0) or (Temp = Radix) then Exit(BCDVAL);  { Non-numeric input }
    J := 0;
    For I := 1 to Len do
      Begin
        If Temp[I] = Radix then Got-radix := True
        Else If Temp[I] in Numeric then
          Begin
            Bcd := (Bcd×10) + (Ord(Temp[I]) - Ord('0'));
            If Got-radix then J:=J+1 { Count decimal places }
          End
      End;
    Normalize;
    If Minus then Bcd := -Bcd;
    BCDVAL := True
  End;  { Bcdval }
Procedure STRBCD(VAR S:String; Bcd:Bcdnum; Suffixsign:Boolean);
  { Do the opposite of BCDVAL, i.e., convert a bcdnum to an ASCII string;
    If Suffixsign is True, then affix the Minus Sign, if required, to
    the end of the string, as in 100.00-}
  VAR I : Integer;
      Stemp : String;
      Sfix : String[1];
  Procedure Padleft;
      Begin
        Stemp:=NULL;
        For I := Length(S) to Rightsize do Stemp:=Concat(Stemp,ZERO);
        S:=Concat(Stemp,S)
      End;
  Begin
    If (Rightsize > Maxlint) or (Rightsize < 0) then
         Rightsize:=0
    Sfix := SPACE;
    Str(Bcd,S);
    If Bcd < 0 then
      Begin
        Delete(S,1,1); { Drop minus sign }
        Sfix := MINUSIGN
      End;
    If Length(S) <= Rightsize then Padleft;  { Do 0.ZZZZN Format }
    Insert(Radix,S,Length(S)-Rightsize+1);
    If Suffixsign then S:=Concat(S,Sfix) else S:=Concat(Sfix,S)
  End;  { Strbcd }
BEGIN { Main }
 Rightsize := 2; { Must be declared in Initialization part of a Unit }
 { DEMO CODE }
 Page(Output);
 Writeln('Demonstration of BCD numbers in Pascal');
 For Loop := 0 to 5 do
   Begin
     Rightsize := Loop;       { Range errors are checked by Bcdval }
     Minusloc := ODD(Loop);  { Decide Minus sign loc: True = Suffix }
     Q := 0;  { Summation accumulator }
     Repeat
       Writeln;
       Write('Input a number[',Rightsize,'] →] ');
       Readln(S);
       Num := Bcdval(S,P);
       If Num then
         Begin
           Q:=Q+P;
           Strbcd(S,P,Minusloc);
           Writeln(S:26);
           Writeln
         End
     until not Num;
     Strbcd(S,Q,Minusloc);
     Writeln;Writeln('Sum = ',S:20);Writeln
   End; { Loop }
 Writeln;Write('That''s all Folks...')
END. { Main }
```

MICRO

NEW FROM D & N MICRO PRODUCTS, INC.



MICRO-80 COMPUTER

Z80A CPU with 4MHz clock and CP/M 2.2 operating system. 64K of low power static RAM. Calendar real time clock. Centronics type parallel printer interface. Serial interface for terminal communications, dip switch baud rates of 150 to 9600. 4″ cooling fan with air intake on back of computer and discharge through ventilation in the bottom. No holes on computer top or side for entry of foreign object. Two 8″ single or double sided floppy disk drives. IBM single density 3740 format for 243K of storage on each drive. Using double density with 1K sectors 608K of storage is available on a single sided drive or 1.2 meg on a double sided drive. Satin finish extruded aluminum with vinyl woodgrain decorative finish. 8 slot backplane for expansion. 48 pin buss is compatible with most OSI boards. Uses all standard IBM format CP/M software.

Model 80-1200	$2995
2 8″ single sided drives, 1.2 meg of storage	
Model 80-2400	$3495
2 8″ double sided drives, 2.4 meg of storage	
Option 001	$ 95
Serial printer port, dip switch baud rate settings	

Software available in IBM single density 8″ format.

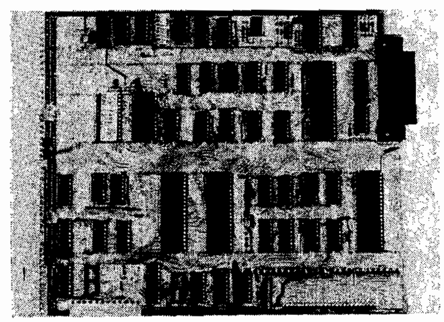
Microsoft	
Basic-80	$289
Basic Compiler	$329
Fortran-80	$410
Cobol-80	$574
Macro-80	$175
Edit-80	$105
Mu Simp/Mu Math	$224
Mu Lisp-80	$174
Digital Research	
PL/1-80	$459
Mac	$ 85
Sid	$ 78
Z-Sid	$ 95
C Basic-2	$110
Tex	$ 90
DeSpool	$ 50
Ashton-Tate	
dBase II	$595
Micropro	
Wordstar	$299
Mail-Merge	$109
Spellstar	$175
Super Sort I	$195
Pascal	
Pascal/MT +	$429
Pascal Z	$349
Pascal M	$355

Convert almost any static memory OSI machine to CP/M® with the D & N-80 CPU Board.

Z80A CPU with 4MHz clock. 2716 EPROM with monitor and bootstrap loader. RS-232 serial interface for terminal communications or use as a serial printer interface in a VIDEO system. Disk controller is an Intel 8272 chip to provide single or double density disk format. 243K single density or 608K double density of disk storage on a single sided 8″ drive. A double sided drive provides 1.2 meg of storage. DMA used with disk controller to unload CPU during block transfers from the disk drives. Optional Centronics type parallel printer port complete with 10 ft. cable. Optional Real Time Calendar Clock may be set or read using 'CALL' function in high level languages. Power requirements are only 5 volts at 1.4 amps. Available with WORDSTAR for serial terminal systems.

INCLUDES CPM 2.2

D & N-80 serial	$695
D & N-80 serial w/Wordstar	$870
D & N-80 video	$695
Option001	$ 80
parallel printer and real time calendar clock	



D & N-80 CPU BOARD

OTHER OSI COMPATIBLE HARDWARE

IO-CA10X Serial Printer Port — **$125**
Compatible with OS-65U and OS-65D software

IO-CA9 Parallel Printer Port — **$175**
Centronics standard parallel printer interface with 10 ft. flat cable

BP-580 8 Slot Backplane — **$ 47**
Assembled 8 slot backplane for OSI 48 pin buss

24MEM-CM9	**$380**	**24MEM-CM9F**	**$530**
16MEM-CM9	**$300**	**16MEM-CM9F**	**$450**
8MEM-CM9	**$210**	**8MEM-CM9F**	**$360**
BMEM-CM9F	**$ 50**	**FL470**	**$180**

24K memory/floppy controller card supports up to 24K of 2114 memory chips and an OSI type floppy disk controller. Available fully assembled and tested with 8, 16, or 24K of memory, with floppy controller (F). Controller supports 2 drives. Needs separated clock and data inputs. Available Bare (BMEM-CM9F) or controller only (FL-470). Ideal way to upgrade cassette based system

C1P-EXP Expansion Interface — **$ 65**
Expansion for C1P 600 or 610 board to the OSI 48 pin buss. Requires one slot in backplane. Use with BP-580 backplane

BIO-1600 Bare IO card — **$ 50**
Supports 8K of memory, 2 16 bit parallel ports may be used as printer interfaces. 5 RS-232 serial ports, with manual and Molex connectors

DSK-SW Disk Switch — **$ 29**
Extends life of drive and media. Shuts off minifloppy spindle motor when system is not accessing the drive. Complete KIT and manual

Disk Drives and Cables

8″ Shugart SA801 single sided — **$395**
8″ Shugart SA851 double sided — **$585**
FLC-6 6ft. cable from D & N or OSI controller to 8″ disk drive — **$ 69**
5 1/4″ MPI B51 with cable, power supply and cabinet — **$450**
FLC-5 1/4 8 ft. cable for connection to 5 1/4 drive and D & N or OSI controller, with data separator and disk switch — **$ 75**

Okidata Microline Printers

ML 82A Dot Matrix Printer — **$534**
120 CPS, 80/120 columns, 9.5″ paper width friction or pin feed

ML 83A Same as 82A except — **$89**
16″ paper width, 132/232 columns witt tractor feed

ML 84 Same as 82A except 200 CPS, — **$115**
16″ paper width, 132/232 columns, 2K buf fer, dot addressable graphics, with tracto feed

D & N Micro Products, Inc.
3684 N. Wells St.
Fort Wayne, Ind. 46808
(219) 485-6414

VISA MasterCard

TERMS $2.50 shipping, Foreign orders add 15%. Indiana residents add 4% sales tax.

MICRO™ Short Subjects

PET BASIC to Waterloo BASIC — *by Jerry D. Bailey*

Numerical Rounding — *by Chuck Muhleman*

PET BASIC to Waterloo Basic

Jerry D. Bailey, 9642 Remer, So El Monte, CA, 91733

As discussed in the October, 1982 PET Vet, Waterloo BASIC offers much more sophisticated program control structures than PET BASIC. Also, PET BASIC and Waterloo BASIC have different ways of implementing certain structures and functions. You'll need to do light to moderate editing including inserting blanks after key words when necessary.

There are several points you must consider when converting PET BASIC program files to Waterloo BASIC-readable program files:

1. The program file must be converted to a sequential file.
2. Line numbers must be forced to five characters with leading blanks.
3. Alphabetic characters must be converted to lower-case ASCII.

Use the following line in direct mode to convert a program in memory to PET ASCII. (Be sure the program does not start with line 0, since Waterloo BASIC will not accept a line 0.)

```
dopen#8, "FILENAME",w:cmd8:list
```

When the cursor returns, enter:

```
dclose:xx
```

This will give a syntax error, but the file will be closed and the cursor will return properly. Now you can use the following program to format the file for access by Waterloo BASIC's OLD command.

Be sure to substitute the appropriate names in lines 10 and 30. The two GET#8's in line 20 discard the carriage returns that CMD puts at the front of the file. Line 40 checks for the "r" in "ready", which marks the last line in the file. All other lines will begin with a space. Line 50 converts the alphabetic characters to true ASCII lower case. Line 60 builds the output string and checks for the end-of-line carriage return. Line 70 searches for the space following the line number. Line 80 pads the line so that the line number always occupies the first five spaces, padded with leading blanks, and writes the line to the output file. Line 90 goes back for the next program line.

This is not a particularly friendly program, in that it simply stops on errors and requires the file names to be written into the program. But it will get the job done. After the file is up in Waterloo BASIC, it will probably not run right away. You'll need to do light to moderate editing.

Bailey — Conversion Routine

```
10 DOPEN#8,"TEST";IFDSTHENPRINTDS$:STOP
20 N$=CHR$(0):GET#8,A$:GET#8,A$
30 DOPEN#9,"WATTEST",W:IFDSTHENPRINTDS$:STOP
40 B$="":GET#8,A$:IFA$="R"THENDCLOSE:END
50 GET#8,A$:PRINTA$;:A=ASC(A$+N$):IFA>64ANDA<91THENA=A+32
60 B$=B$+CHR$(A):IFA-13GOTO50
70 FORI=2TOLEN(B$):IFMID$(B$,I,1)<>" "THENNEXTI:STOP
80 PRINT#9,LEFT$("      ",6-I)B$;:IFDSTHENPRINTDS$:STOP
90 GOTO40
```

Numerical Rounding

Chuck Muhleman, Computer-ease, Box 806, Marion, IN 46952

You may calculate numbers properly using all possible digits internal to a computer or calculator. You should *not* state the answer with all the digits shown. The accuracy of any answer is only as good as the accuracy of the least accurate input to the problem. Thus, when the calculation is completed, the answer usually must be rounded to show the proper accuracy. I say "usually" because some calculations give exact values: 2*3, or 2.33*3 = 6.99.

When you do a series of calculations involving several formulae, the answer should be rounded. For financial problems, such as computing an amortization table, the answer should be rounded to the nearest cent. For instance, in an amortization computation the interest due for a specific month is computed, rounded to the nearest cent, subtracted from the monthly payment, and the balance applied to the principal. After all, you do not normally make payments less than $0.01 each, and the loan repayment must be larger than the interest due or the loan will never be repaid.

Remember that answers do not need to involve fractions for rounding to apply. For example, when considering the populations of cities rounded to the nearest 1000, it would be improper to give an answer such as 53 162, say, for a population average of all the cities within a state. The proper answer would be 53 000. (Note that numbers over four digits on either side of the decimal are separated by spaces, not commas. This is now preferred in deference to the European practice of using commas where Americans use decimal points.)

Do not use algorithms which just truncate the answer; i.e., drop the unwanted fractional parts. Why use a sophisticated computer, then give an answer similar to that of an elementary school student?

Another important point to remember is when the fractional part is exactly equal to 0.5, then the answer should be rounded to the nearest *even* number. That is, 3.15 should be rounded to 3.2, as should be 3.25.

(continued)

Short Subjects *(continued)*

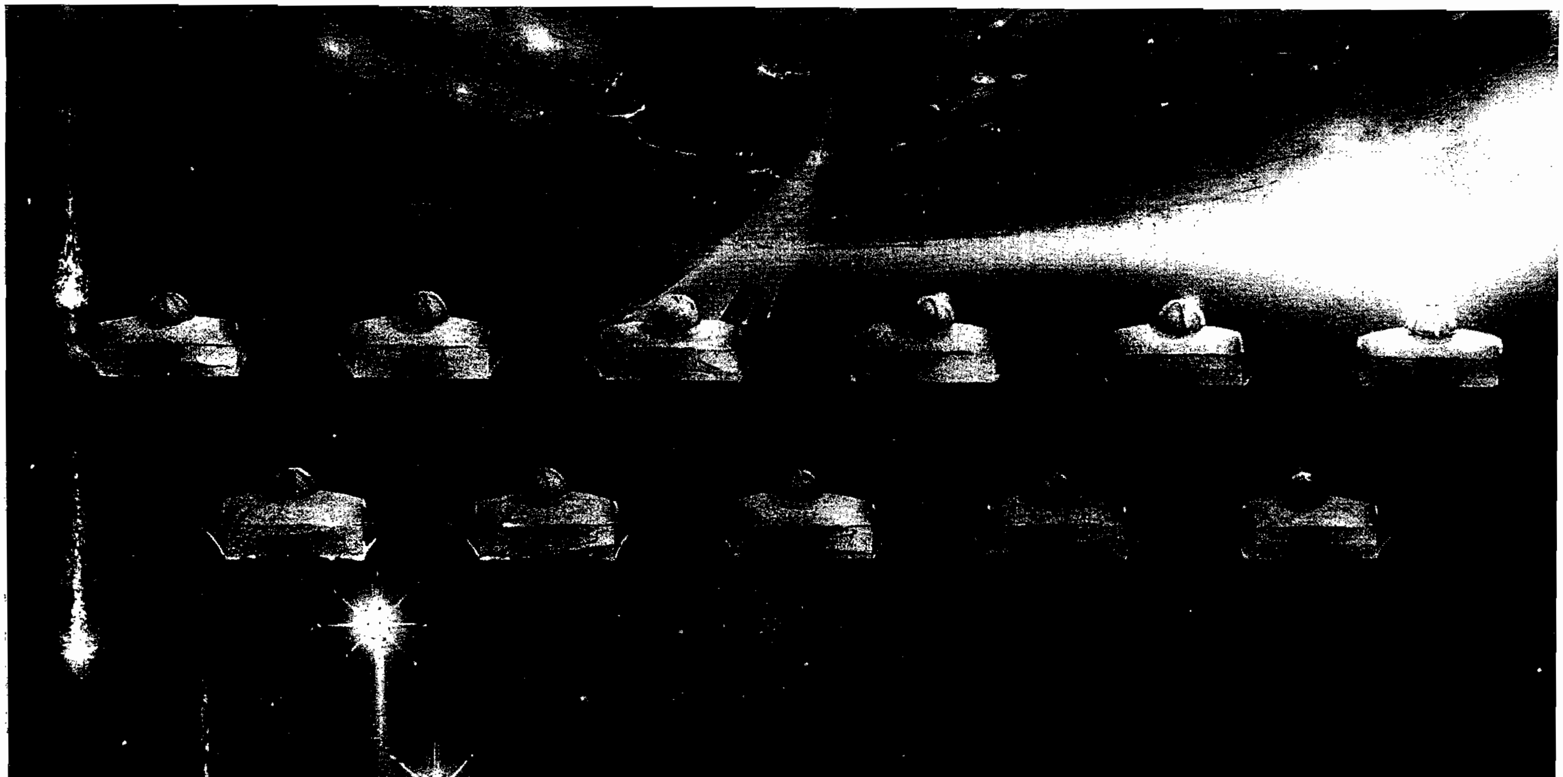
The common way to round is to convert the number to an integer using logarithms, round, then find the antilog to get an answer. This method is fine, except the manipulation must be different for numbers less than or greater than 1. Also, the math algorithms internal to a computer may give some strange answers because they manipulate using binary numbers, not decimals.

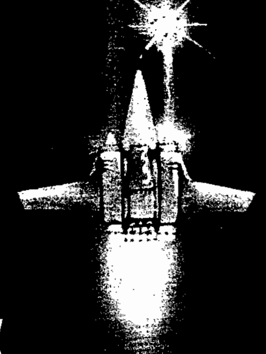
The accompanying algorithm, in BASIC, will round numbers input as variable N1 to the number of significant figures input as variable M1. This algorithm also rounds the fraction 0.5 properly.

To handle the strange answers caused by the binary numbers, string functions may be employed when the magnitude of the answer is known. This is generally true for a given problem, such as an amortization table or a table of wire resistance. Other useful math subroutines and algorithms are included in the program listing.

```
10 REM ''MATH''  C&J Supply 7/5/81  Bytes=3298
11 REM  Remember this system computes to 9 sig. figures
82 DEF FNDC(X) = INT((X + .ØØØ1) × 1ØØ) :
   REM Stop $ RD Error; ''/1ØØ'' for answer
83 EØ = 2.71282813
84 DEF FNLG(X) = LOG(X)/LOG(1Ø) :REM LN to LOG
85 DEF FNSN(X) = ATN(X /SQR(1-X↑2)) : REM ARCSIN
86 DEF FNCS(X) = (PI/2)-FNSN(X) : REM ARCCOS
87 DEF FNHØ(X) = (EØ↑X-EØ↑(-X))/2 : REM SINH
88 DEF FNJØ(X) = (EØ↑X+EØ↑(-X))/2 : REM COSH
90 DEF FNTC(X) = (X-32)×5/9 : REM Farenheit to  Celsius
91 DEF FNTF(X) = X×9/5+32 : REM Clesius to Farenheit
92 PRINT : INPUT ''Enter: Number Decimal to RT  ''; N1
93 DEF FNR(X) = INT(X×1Ø↑N1+Ø.5)/1Ø↑N1
3800 REM Math Subroutines
3801 REM———Deg, Min, Sec to Radians
3802 A = D+M/6Ø + S/36ØØ
3803 A = (A×PI/18Ø-2×PI×INT(A/36Ø))
3809 RETURN
3820 REM———Radian to Deg, Min, Sec
3821 A = 36ØØ×18Ø×R/PI : REM Total Sec
3822 B = INT(A/36ØØ) : REM Total Deg
3823 C = INT(B/36Ø) : REM Number Circle
3824 D = B-36Ø×C : REM Deg
3825 Z = A-B×36ØØ
3826 M = INT(Z/6Ø) : REM Min
3827 S = Z-M×6Ø : REM Sec
3829 RETURN
3859 REM———Significant figures
3860 IF N1 = Ø THEN 3879
3866 INPUT ''Enter # Sig Fig ''; M1
3868 N2 = ABS(N1) :M2 = LOG(N2)/LOG(1Ø) : M3 = INT(M2+1ØØ)
3869 N3 = N2×1Ø↑(M1-(M3-99)) : N4 = INT(N3+.5)
3872 IF N3-INT(N3)< >.5 THEN 3878
3874 N5 = INT(N3)-INT(N3/1Ø)×1Ø : IF INT(N5/2)< >N5/2 THEN 3877
3876 N4 = INT(N3) : GOTO 3878
3877 N4 = INT(N3+1)
3878 N1 = N4×1Ø↑((M3-99)-M1)×SGN(N1)
3879 RETURN
3899 END
```

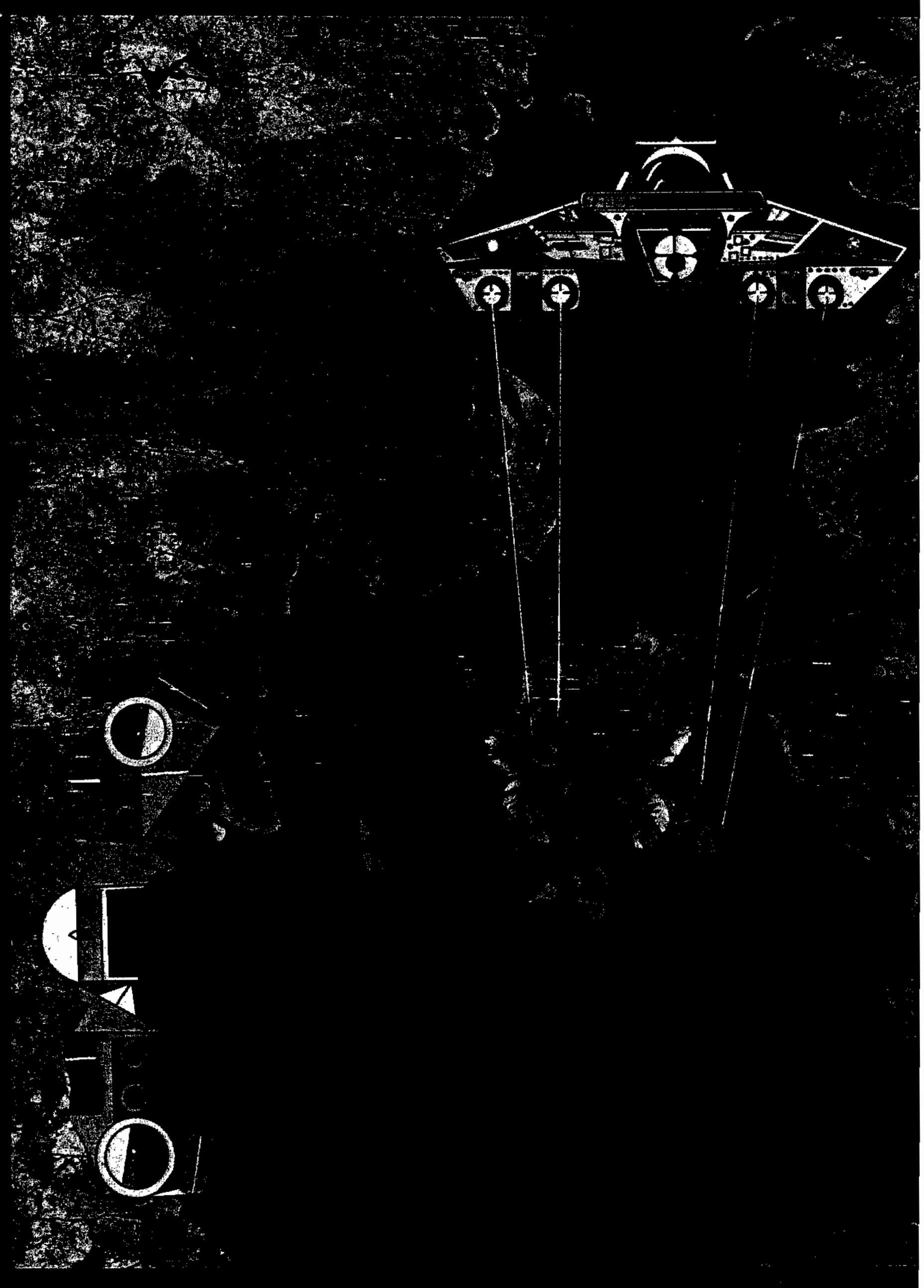
MICRO





HEROES WANTED!

It takes the stuff of heroes to challenge Tharolia. For this gigantic planet is guarded by robotic spacefighters and automatic defenses programmed for one purpose: kill.



Our interstellar expansion will come to a halt unless you can blast through the swarms of automated ships, enter the devilish Tharolian Tunnels —elude the traps, and fight your way through tunnel after tunnel.

It will take uncommon dexterity, speed and courage to master the tunnels and destroy the robotic defense system. Heroes only are wanted, the faint hearted need not apply.

$29.95 for Apple II* joystick/paddles

At your computer store, or:

DATAMOST INC

9748 Cozycroft Ave., Chatsworth, CA 91311. (213) 709-1202.

VISA/MASTERCARD accepted. $2.00 shipping/handling charge (California residents add 6½% sales tax.)

*Apple II is a trademark of Apple Computer, Inc

MICRO™

CoCo Bits

By John Steiner

This month, in addition to news, I discuss some of the books available for the Color Computer and the 6809. I also examine how to set up a high-resolution graphics display on the CoCo.

Last month I mentioned rumors of a new Color Computer built by Radio Shack, available through RCA. The machine, TDP System 100, should be available from RCA dealers by the time you read this. Considering the power of the Color Computer, I expect other CoCo "clones" will appear soon.

A unique accessory now available from several companies is an expansion unit. The unit plugs into the existing ROM pack and provides several extra expansion slots where drive controller, printer card, ROM packs, and other accessories can remain connected at all times. I look forward to testing one of these units for utility and ease of use. I will keep you posted.

By the time you read this, I am sure many of you will know of the death of Mr. Arnold C. Pouch. A retired IBM programmer, Mr. Pouch was one of the first to realize the power of Color Computer graphics. His Motion Picture Programming techniques for CoCo set a standard in graphics programs, and he created many other excellent utilities, including Disk Doctor. His company, Superior Graphics Software, will continue to operate, according to Mrs. Pouch.

When the snow flies and the early winter darkness descends, many people settle down with a good book by the roaring fire. What better way to spend an evening than to read about your favorite computer? In the past few months, several books have been released providing information and programs for both CoCo and the 6809. Subjects range from general purpose programs to high-resolution graphics tutorials.

A good programming book for beginners is one of a series written for many different home computers: Bob Albrecht's *TRS-80 Color BASIC*, published by John Wiley & Sons. Mr. Albrecht assumes no previous knowledge of BASIC, and in an entertaining manner teaches much about CoCo. The book lacks only a discussion of the features of Extended Color BASIC.

Two excellent books covering graphics are *TRS-80 Color Computer Graphics* by Don Inman with Dymax (Reston Publishing Co.), and *Color Computer Graphics* by William Barden, Jr. (Radio Shack). Mr. Inman's explanation of graphics operations and sound and joystick usage is well written. An entire chapter on machine-language USR routines is included.

Mr. Barden explains the graphics commands found in Extended Color BASIC, and covers the details of the 6847 video display generator and the CoCo memory map. The appendices are a rich source of video display information. For $5.95, *Color Computer Graphics* is the least expensive book mentioned here.

TRS-80 Programs and Applications, by Alfred Baker, contains many beginner-level programs for the CoCo. If you are new to programming, you will like the fully documented program listings. (This book is not for the intermediate or advanced programmer.) One of the first programs in the book is a joystick test routine that checks the keyboard to see if a joystick fire button has been depressed. In ROM 1.0, this is acceptable. In ROM 1.1, the keyboard is isolated from the joystick port and pressing the fire button will have no effect on this program.

For CoCo owners who want to learn machine-language programming on the Color Computer, there is little to choose from. Two new releases may be available by the time you read this. William Barden, Jr., is writing a book for use with EDTASM+. It will be available from Radio Shack. Don Inman, who wrote an excellent book on machine language for the Apple, will soon have one for the Color Computer.

Books covering the 6809 processor and machine-language programming are more plentiful. A good reference source, *The MC6809 Cookbook* by Carl D. Warren (TAB Books), is written for the experienced programmer upgrading to the 6809. *6809 Microcomputer Programming and Interfacing*, by Andrew Staugard, Jr., (Howard W. Sams & Co.), discusses the 6820 and 6821 PIAs in addition to the 6809. The 6821 is used in I/O applications for the Color Computer.

Probably the most detailed discussion of 6809 programming and applications is found in Lance Leventhal's *6809 Assembly Language Programming*. It is possible to learn machine-language programming from the beginning with this text, but the routines will not work unmodified on your CoCo assembler unless you can work on memory page zero.

Programming the 6809, by Rodney Zaks and William Labiak, covers elementary and intermediate programming techniques. Again, program modification may be required for the Color Computer.

This list is by no means complete, and discusses only books that I have had an opportunity to purchase. If you have information concerning other CoCo or 6809 reference sources, let me know and I will mention them in future columns.

Extended Color BASIC is probably one of the easiest graphics extensions available. In addition, other useful features are available in the ECB ROM. Although the ECB manual is easy to understand, it leaves some questions concerning the proper set-up of graphics screens. The simple, step-by-step list below should help ensure that all options are covered.

CoCo Graphics Screen Initialization

1. Reserve the correct number of graphics pages. CoCo needs 1.5K of memory for each page reserved. The number of pages needed depends on the resolution you want and the number of separate screens to be held in memory. PCLEAR reserves between one and eight pages for graphics use.
2. Choose the mode you want. The mode determines the resolution and color combinations displayed. There

are three two-color modes and two four-color modes. PMODE identifies the mode and page on which the image will be placed.
3. Choose the color set you want and call the graphics screen. Use the SCREEN command to set this up. You can choose from two text screens and two color sets.
4. Select the foreground and background colors you want. This step allows you to create a display without specifying color in individual commands. COLOR sets this parameter.

All Extended Color BASIC graphics programs use these commands in one form or another to set up the display. Efficient use of these commands makes high speed, high-resolution graphics programming available to the BASIC programmer.

Next month, in addition to news, I will take a look at some hooks to RAM from the BASIC ROMs. A list pager program will demonstrate the use of these hooks.

MICRO

MICRO™ New Publications

Survey consists of three sections: Personal Computers, Engineering, and General Interest. It is published bimonthly. For further information and to receive a sample pre-publication issue, send name and address with $2 to KVA Associates, 2821 Camino del Mar, Del Mar, CA 92014. Phone (714) 755-0041.

The Computer Tutor, Learning Activities for Homes and Schools, by Gary W. Orwig and William S. Hodges. Winthrop Publishers, Inc. (17 Dunster Street, Cambrdige, MA), 1982, 203 pages, 8½ × 11 inches, paperback.
ISBN: 0-87626-147-0 $10.95

Programming the 6809, by Rodnay Zaks and William Labiak. Sybex (2344 Sixth St., Berkeley, CA 94710), 1982, 520 pages, paperback.
ISBN: 0-89588-078-4 $14.95

Introduction to WordStar, by Arthur Naiman. Sybex (2344 Sixth St., Berkeley, CA 94710), 1982, 220 pages, paperback.
ISBN: 0-89588-077-6 $8.95

Data Communications for Microcomputers: With Practical Applications and Experiments, by Elizabeth A. Nichols, Joseph C. Nichols, and Keith R. Musson. McGraw Hill Book Company (1221 Avenue of the Americas, New York, NY 10020), 1982, 264 pages, paperback.
ISBN: 0-07-046480-4 $16.95

WRITE, EDIT, & PRINT: Word Processing with Personal Computers, by Donald H. McCunn. Design Enterprises of S.F. (P.O. Box 14695, San Francisco, CA 94114), 1982, paperback.
ISBN: 0-932538-06-1 $24.95

Word Processing Primer, by Mitchell Waite and Julie Arca. BYTE Books/McGraw Hill (1221 Avenue of the Americas, New York, NY 10020), 1982, 188 pages, paperback.
ISBN: 0-07-067761-1 $14.95

Fortran Programs for Scientists and Engineers, by Alan R. Miller. Sybex (2344 Sixth Street, Berkeley, CA 94710), 1982, 320 pages, paperback.
ISBN: 0-89588-082-2 $15.95

Introduction to the UCSD p-System, by Charles W. Grant and Jon Butah. Sybex (2344 Sixth St., Berkeley, CA 94710), 1982, 300 pages, paperback.
ISBN: 0-89588-061-X $14.95

VIC Innovative Computing, by Clifford Ramshaw. Melbourne House Publishers (347 Reedwood Drive, Nashville, TN 37217), 1982, 151 pages, paperback.
ISBN: 0-86161 108-X $14.95

User's Guide to PET/CBM Computers, by Jeffrey R. Weber. Weber Systems (8437 Mayfield Road, Cleveland, OH 44026), 1982, 324 pages, paperback.
ISBN: 0-9604892-8-2 $14.95

Basic BASIC-English Dictionary for the Apple, PET, and TRS-80, by Larry Noonan. dilithium Press (Beaverton, OR), 1982, 150 pages, paperback.
ISBN: 0-918398-54-1 $10.95

MICRO™

Reviews in Brief

Product Name: **The Disk Doctor**
Equip. req'd: TRS-80C disk, 32K
Price: $49.95
Manufacturer: Superior Graphic Software
406 Little Mountain Road
Waynesville, NC 28786

Description: *The Disk Doctor* is a disk-salvage program for the Color Computer. The doctor will assist in transferring files from a "crashed" disk onto a different disk. Programs can be recovered even if the disk directory and allocation tables have been destroyed. Included with the package are an eight-page operation guide, a sample "crashed" disk, and the system disk. *Disk Doctor* is written in BASIC, and can be easily modified.

Pluses: The program is capable of reconstructing lost disk files in their entirety, no matter what their file type. Running *Disk Doctor* on machine-language files before the disk crashes gives a printout of ML start, end, and execute addresses. A wealth of information on preventative maintenance of disks is provided.

Minuses: *The Disk Doctor* has only one minor limitation — when salvaging a file larger than 12,288 bytes and you try to continue the transfer operation past the last graphics page, the transfer program overwrites *The Disk Doctor.* If this happens, the transfer procedure must be restarted from the beginning. This file size limitation is noted in the instructions and is easily avoided if you are careful.

Documentation: *The Disk Doctor* is full of "medical" information on sick disks. There are no lists of instructions, but the operator is guided through a disk reconstruction using the included crashed disk.

Skill level required: A knowledge of the disk file structure is helpful, and the user is referred to the Color Computer disk-system manual for preliminary information.

Reviewer: John Steiner

Product Name: **OSI Greatest Hits**
Equip. req'd: OSI 1P or Superboard
Price: $29.95 plus $1.50 shipping
Manufacturer: Victory Software Corp.
2027-A S.J. Russell Circle
Elkins Park, PA 19117

Description: *Greatest Hits* is a collection of original programs for the OSI 1P computer. The programs fill both sides of two cassette tapes. Tape one contains 18 game programs; Night Rider, Cosmic Debris, Minos, Street Sweeper, Ridge Cruiser, and Worm are excellent games with good graphics effects. The second tape contains 16 utility programs. The two tapes represent nearly two hours of programs at 300 baud.

Pluses: The user can list, study, and modify these BASIC programs. A number of machine-code subroutines are used to increase the graphics speed and create some impressive effects. Smooth movement is obtained by clever use of the OSI character set. Some programs even let the player record his initials when he makes a new high score — just like the "real" arcades.

Minuses: The programs with graphics are written specifically for the 1P screen and ROM BASIC. These programs will need modification to run on any other OSI machine. The keys used to move UP, DOWN, LEFT, and RIGHT are not consistent among the various games. The user will have to modify the programs to fit his particular joystick hook-up. The utility programs are not of wide general interest; however, you may find one or two that fit your needs.

Documentation: A brief description and rules for each game are provided. Program listings are available at additional cost.

Skill level required: Good hand-eye coordination and fast reflexes.

Reviewer: E.D. Morris

Product Name: **Basic'**
Equip. req'd: Apple II or Apple II Plus
One disk drive
Price: $129.00
Manufacturer: Delta Micro Systems, Inc.
P.O. Box 15951
1022 Harmony Street
New Orleans, LA 70175
(504) 895-1481

Description: This program development system for BASIC consists of a text editor, a preprocessor, a menu program, and a special disk operating system that detects the presence of a special protection chip that plugs into the game I/O socket of the Apple. The heart of the software is a preprocessor program that accepts a program written in the *Basic'* language (a structured dialect of BASIC) and produces Applesoft programs as output. *Basic'* provides the advantages of structured control statements that ordinary BASIC does not have: REPEAT-UNTIL, CASE, IF-ELSE, and PROC (named procedures without parameters).

Pluses: *Basic'* provides an easy way for those already familiar with BASIC programming to learn the principle of structured programming.

Minuses: The system imposes limits on the size of source files. This could make large program development awkward, since the editor evidently does not make it easy to shift chunks of one source file to another. Thus, when one file fills up and you want to insert a bunch of new code in the middle of it, it would be necessary to key in the tail end of that file all over again in another text file. [*Editor's note:* there is a procedure in the manual to eliminate rekeying. Computer Assisted Analysis and Interactive Sports Systems have also developed large-scale BASIC programs using this system.] The system is menu-driven and does not allow the use of an eighty-column display card. *Basic'* generates Applesoft source code, which is equivalent to the source code written in *Basic'*.

Documentation: Well written, concise, and attractively packaged.

Skill level required: Knowledge of BASIC and a desire to learn structured programming.

Reviewer: Richard Vile

Product Name:	**ColorZAP**
Equip. req'd:	TRS-80C, RS disk system
Price:	$49.95
Manufacturer:	Software Options, Inc. 19 Rector Street New York, NY 10006

Description: *ColorZAP* is a BASIC program with machine-language routines that allow the user to examine, change, or copy data on Color Computer diskettes. *ColorZAP* will access four drives and remains in memory so that all drives are available for program use. The program will display all sectors, or display sectors in a given file. Sectors can be verified for accuracy.

Pluses: *ColorZAP* quickly moves from sector to sector, forward or backward by pressing the + or − key. Direct access to any sector is allowed. A cursor-controlled screen editor modifies individual bytes. All changes are made in memory and transferred to the disk only when you are ready. A convert routine allows you to convert granule numbers into the track and sector numbers required for data file access.

Minuses: None noted.

Documentation: A 24-page manual describes program operation in detail and provides valuable information on disk system parameters.

Skill level required: A solid basic knowledge of disk file structure is necessary, especially when trying to reconstruct a defective or killed file.

Reviewer: John Steiner

(continued)

Product Name:	**VIE (VIC IEEE Interface)**
Equip. req'd:	VIC-20 (5K or more) PET to IEEE cable IEEE Device(s)
Price:	$99.95
Manufacturer:	Micro-Systems 11105 Shady Trail #104 Dallas, Texas 75229

Description: *VIE* is a cartridge-like unit that plugs into the expansion port of the VIC-20 and enables the VIC to communicate with IEEE devices like the PET/CBM 2022 printer and 4040 disk drives. A PET to IEEE cable (not provided) attaches to the side of the VIE. The VIE also has an expansion slot so other cartridges may be attached to the VIC without removing the VIE. Approximately 1K of ROM software is built into the VIE which is accessed by a SYS40000 command to initialize the interface. Once initialized, communication with any IEEE device is by direct commands or from within a program.

Pluses: The VIE is reliable and extremely easy to use. It takes no memory away from the VIC, nor interferes with normal operation. It's attractively packaged and comes in a hard plastic case that matches the VIC's case and color.

Minuses: The expansion slot on the VIE is positioned so that added cartridges are vertical. Inserting cartridges into the VIE can cause stress on the VIC's expansion connection. Adding a piece of wood under the VIE will solve this problem.

Documentation: The VIE comes with a single page of documentation that covers everything quite well.

Skill level required: The user should understand the commands needed to communicate with his IEEE devices.

Reviewer: David Malmberg

Product Name:	**Type 'N Talk Text-to-Speech Synthesizer**
Equip. req'd:	Virtually all personal computers
Price:	$249.00 (plus cable) Interface cable for VIC and 64 - $34.95 For other computers - $24.95
Manufacturer:	Votrax, Inc/ Federal Screw Works Division Consumer Products Group 500 Stephenson Highway Troy, MI 48084

Description: *Type 'N Talk* is a completely self-contained text-to-speech synthesizer that attaches to your computer *via* an RS-232C serial port. When you open a file and write to this port using BASIC or other languages, the text you write is converted to speech. The speech sounds mechanical, but the overall quality is good and understandable. If you have a VIC-20 or Commodore-64, you will need a special cable available from Votrax that attaches to the user port. The *Type 'N Talk* has its own microprocessor and buffer (with enough capacity to hold a minute's worth of speech), so speech can occur while the host computer is doing something else. The text-to-speech synthesizer creates speech from electronic phonemes that give an unlimited vocabulary and the ability to speak languages other than English. The unit has a built-in amplifier with volume and frequency controls and a jack to plug in a speaker (not provided).

Pluses: *Type 'N Talk* works well once you overcome the lack of any practical examples in the documentation. The unit is fun (especially for children) and is an impressive demonstration of your computer's power.

Minuses: See Documentation.

Documentation: A 32-page manual primarily addressed to engineers and/or hardware experts. There is no additional documentation.

Skill level required: None.

Reviewer: David Malmberg

Product Name:	**Printographer**
Equip. req'd:	Apple II Plus Any of the popular printers
Price:	$49.95
Manufacturer:	Southwestern Data Systems P.O. Box 582 Santee, CA 92071

Description: A versatile screen-dump program designed for ease of use interfaces routines for most of the Apple-compatible printers currently available. Features to "crop" a picture permit you to print only desired parts of a picture. The manufacturer's standard backup facility provides a maximum of three copies to be made.

Pluses: Pictures can be positioned on a page both horizontally and vertically. A magnification feature allows you to blow up and print just a portion of a picture. A subroutine permits printing under Applesoft control.

Minuses: None noted.

Documentation: Well written; numerous illustrated examples speed the familiarization process.

Skill level required: A beginning BASIC programmer should have no trouble.

Reviewer: Chris Williams

Product Name:	**Chromasette Magazine**
Equip. req'd:	TRS-80C w/Extended BASIC
Price:	$45.00/year or $5.00 each
Manufacturer:	Chromasette Magazine P.O. Box 1087 Santa Barbara, CA 93102

Description: *Chromasette* is a monthly magazine with approximately six programs on cassette for the Color Computer. Programs range in nature from games to utilities or home-management software. All tapes include a graphics cover program. Some programs are written especially for the CoCo disk system.

(continued)

Pluses: An interesting newsletter that accompanies the cassette provides information on using the programs, bugs found in previous issues, reader modifications, and short program listings not found on the tape. It is an inexpensive source of CoCo software for a wide variety of applications. Where possible, tape-to-disk conversion information is included.

Minuses: Not all programs are usable on all machines. You must have Extended BASIC to run most of the programs.

Documentation: Provided in the accompanying newsletter; typically four to six pages.

Skill level required: Programs are provided for all levels, from novice to hardware hacker.

Reviewer: John Steiner

Product Name:	**VIC Expansion Module**
Equip. req'd:	VIC-20 (5K or more)
Price:	$49.95
Manufacturer:	Parsec Research P.O. Drawer 1766 Fremont, CA 94538

Description: The *VIC Expansion Module* plugs into the expansion port in the back of your VIC and enables you to have up to three cartridges operable simultaneously. Using this module, you can add up to 32K of additional RAM memory, or combinations of RAM and utility cartridges like the toolkit or machine-language monitor.

Pluses: It works reliably. The module is well made with double-gold plating throughout. It is designed to rest flush with the table, so inserting cartridges will not put any stress on the VIC expansion connectors.

Minuses: The unit's black color does not go well with the rest of the VIC's color scheme.

Documentation: Clear and concise. The module also comes with a detailed memory map and instructions on how to set the DIP switches in Commodore's 8K RAM cartridges to correspond to any 8K block.

Skill level required: None.

Reviewer: David Malmberg

Product Name:	**80C Disassembler**
Equip. req'd:	TRS-80 Color Computer plus printer
Price:	$49.95
Manufacturer:	The Micro Works P.O. Box 1110 Del Mar, CA 92014

Description: *80C Disassembler* is a tape-based 6809 assembly-language disassembler specifically tailored to the Color Computer. The output from the disassembler can be any one of three formats. Users with an 80-column printer can specify the "full output" mode that lists an address, the machine code (one to five bytes for a 6809), the effective address specified by the instruction, ASCII characters represented by the code, and the assembly-language statement deduced from the code. Users with a narrow-format printer can get the "full output" printed with a special indented format that eases interpretation of the data. During program set-up, the user can specify various types of code areas within the program.

Pluses: This software is highly user-oriented; the experienced user can glean a maximum of information in a minimum of time.

Minuses: Not available in ROM; but such use may be too specialized to warrant production.

Documentation: Thorough and clearly written.

Skill level required: Novice assembly-language programmer.

Reviewer: Ralph Tenny

Product Name:	**The Software Automatic Mouth (S.A.M.)**
Equip. req'd:	Apple II with Applesoft 48K RAM and DOS 3.3
Price:	$124.95
Manufacturer:	Don't Ask Computer Software 2265 Westwood Blvd. Suite B-150 Los Angeles, CA 90064

Description: This software voice synthesizer generates clear speech from strings of phonemes (speech sounds represented by about 50 unique letter combinations), with programmable pitch, speed, and inflection. A program to translate English text directly to speech is included. To use *S.A.M.* from Applesoft, BLOAD *S.A.M.*, assign the string to be spoken to SA$, CALL an address, and *S.A.M.* speaks. Connection to an external speaker is through the included plug-in card.

Pluses: Speech is clear and expressive, easy to generate and manipulate from Applesoft or assembly language. The disk is unprotected. This is an outstanding and fascinating product at a reasonable price.

Minuses: *S.A.M.* programs will not work on other Apple models without a converter card.

Documentation: The excellent, thorough manual includes a 1500-word phonetic spelling dictinary.

Skill level required: None to enjoy the demonstration programs; ordinary knowledge of BASIC to use *S.A.M.* in programs.

Reviewer: Jon R. Voskuil

MICRO

PRODUCTS FOR ATARI* 400/800 FROM ELCOMP

BOOKS:

ATARI BASIC – Learning by using
An excellent book for the beginner. Many short programs and learning exercises. All important features of the ATARI computers are described (screen drawings, special sounds, keys, paddles, joysticks, specialized screen routines, graphics, sound applications, peeks, pokes, and special stuff). Also suggestions are made that challenge you to change and write program routines.
Order # 164 — **$7.95**

GAMES for the ATARI Computer
This book describes advanced programming techniques like player-missile-graphics and use of the hardware-registers. Contains many ready to run programs in BASIC and one called GUNFIGHT in machine-Language.
Order # 162 — **$7.95**

SOFTWARE IN BASIC FOR ATARI

Invoice Writing for Small Business
This program makes writing invoices easy. Store your products in DATA statements with order-number, description, and price. The program later retrives the description and price matching to the entered order-number. The shipping cost and the discount may be calculated automatically depending on the quantity ordered or entered manually. The description to the program tells you how to change the program and adapt it to your own needs. Comes with a couple of invoice forms to write your first invoices on to it.

Order # 7201	cassette version	$29.95
Order # 7200	disk version	$39.95

Mailing List
This menu driven program allows the small business man to keep track of vendors and customers. You can search for a name or address of a certain town or for an address with a certain note. 50 addresses are put into one file.

Order # 7212	cassette version	$19.95
Order # 7213	disk version	$24.95

Inventory Control
This program is menu driven. It gives you the following options: read/store data, define items, entry editing, inventory maintenance (incoming-outgoing), reports. The products are stored with inventory number, manufacturer, reorder level, present level, code number, description.

Order # 7214	cassette version	$19.95
Order # 7215	disk version	$24.95

Programs from Book # 164
The programs from book no. 164 on cassette. (Book included)
Order # 7100 — **$29.00**

Game Package
Games on cassette. (Bomber, tennis, smart, cannon fodder, etc.)
Order # 7216 — **$9.95**

Microcomputer Hardware Handbook (845 pages)
Descriptions, pinouts and specifications of the most popular microprocessors and support chips.
A MUST for the hardware buff.
Order-No. 29
$14.95

Care and Feeding of the Commodore PET
Eight chapters exploring PET hardware. Includes repair and interfacing information. Programming tricks and schematics.
Order # 150 — **$9.95**

ELCOMP

Books + Software for ATARI VIC-20 OSI SINCLAIR TIMEX

Payment: check, money order, VISA, MASTER-CHARGE, Euroscheck.
Orders from outside USA: add 15% shipping. CA residents add 6.5% tax
*ATARI is a registered trademark of ATARI Inc.
*VIC-20 is a registered trademark of Commodore

ELCOMP Publishing Inc.,
53 Redrock Lane
Pomona CA 91766,
Phone: (714) 623–8314

SOFTWARE IN MACHINE LANGUAGE for ATARI

ATMONA-1
This is a machine language monitor that provides you with the most important commands for programming in machine-language. Disassemble, dump (hex and ASCII), change memory location, block transfer, fill memory block, save and load machine-language programs, start programs. Printer option via three different interfaces.

Order # 7022	cassette version	$19.95
Order # 7023	disk version	$24.95
Order # 7024	cartridge version	$59,00

ATMONA-2
This is a tracer (debugger) that lets you explore the ATARI RAM/ROM area. You can stop at previously selected address, opcode, or operand. Also very valuable in understanding the microprocessor. At each stop, all registers of the CPU may be changed. Includes ATMONA-1.

Order # 7049	cassette version	$49.95
Order # 7050	disk version	$54.00

ATMAS
Macro-Assembler for ATARI-800/48k. One of the most powerful editor assemblers on the market. Versatile editor with scrolling. Up to 17k of source-Code. Very fast, translates 5k source-code in about 5 seconds. Source code can be saved on disk or cassette. (Includes ATMONA-1)

Order # 7099	disk version	$89.00
Order # 7999	cartridge version	$129.00

ATAS
Same as ATMAS but without macro-capability. Cassette-based.

Order # 7098	32k RAM	$49.95
Order # 7998	48k RAM	$49.95

ATEXT-1
This wordprocessor is an excellent buy for your money. It features screen oriented editing, scrolling, string search (even nested), left and right margin justification. Over 30 commands. Text can be saved on disk or cassette.

Order # 7210	cassette version	$29.95
Order # 7216	disk version	$34.95
Order # 7217	cartridge version	$69.00

GUNFIGHT
This game (8k machine-language) needs two joystieks. Animation and sound. Two cowboys fight against each other. Comes on a bootable cassette.
Order # 7207 — **$19.95**

ELCOMP FORTH for the ATARI

ELCOMP FORTH is an extended Fig-Forth-version, Editor and I/O package included. Utility package includes decompiler, sector copy, Hex-dump (ASCII), ATARI Filehandling, total graphic and sound, joystickprogram and player missile.

Order # 7055	disk	$39.95

Floating point package for ELCOMP FORTH with trigonometric functions (0 – 90°).

Order # 7230	disk	$29.95

Learn-FORTH from ELCOMP
A subset of Fig-Forth for the beginner. On disk (32k RAM) or on cassette (16k RAM).
Order # 7053 — **$19.95**

Expansion boards for the APPLE II

The Custom Apple + Other Mysteries
A complete guide to custom zing the Apple Software und Hardware
Order-No. 680 — **$24.95**

We also stock the boards which are used in the book "The Custom Apple . . ." (barebords)

6522 I/O Board No. 605	**$39.00**
EPROM Burner No. 607	**$49.00**
8K EPROM/RAM Board No. 609	**$29.00**
Prototyping board for the Apple II No. 604	**$29.00**
Slot repeater board for the Apple II No. 606	**$49.00**

Order two boards and get the book free !

No. 6153 Learn FORTH for the Apple II	(C)	$19.95
No. 6154 Learn FORTH for the Apple II	(D)	$24.95
No. 6155 ELCOMP FORTH for the Apple II (Extended Fig-Forth, Editor, graphics package)	(D)	$39.95
No. 6156 Floating point for ELCOMP FORTH	(D)	$29.95

Hardware – ADD-ONS for ATARI

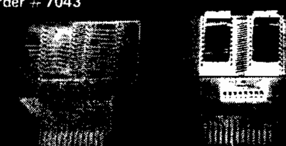
PRINTER INTERFACE
This construction article comes with printed circuit board and software. You can use the EPSON printer without the ATARI printer interface. (Works with gameports 3 and 4).
Order # 7211 — **$19.95**

RS-232 Interface for your ATARI 400/800
Software with connector and construction article.
Order # 7291 — **$19.95**

EPROM BURNER for ATARI 400/800
Works with gameports. No additional power supply needed. Comes compl. assembled with software (2716, 2732, 2532).
Order # 7042 — **$179.00**

EPROM BURNER for ATARI 400/800 KIT
Printed circuit board incl. Software and extensive construction article.
Order #7292 — **$49.00**

EPROM BOARD (CARTRIDGE)
Holds two 4k EPROMs (2532). EPROMs not included.
Order # 7043 — **$29.95**



EPROM BOARD KIT
Same as above but bare board only with description.
Order # 7224 — **$14.95**

ATARI, VIC-20, Sinclair, Timex and OSI

New – for your ATARI 400/800

Astrology and Biorythm for ATARI (cass. or disk).
Order # 7223 — $29.95

Birth control with the ATARI (Knaus Ogino)
Order # 7222 cass. or disk — $29.95

Books + Software for VIC-20 (requires 3k RAM Exp.)

# 4870	Wordprocessor for VIC-20, 8k RAM	$19.95
# 4883	Mailing List for VIC-20, 16k RAM	$14.95
# 141	Tricks for VICs - The VICstory Progr.	$9.95
# 4880	TIC TAC VIC	$9.95
# 4881	GAMEPACK I (3 Games)	$14.95
# 4885	Dual Joystick Instruction	$9.95

INPUT/OUTPUT Programming with your VIC
Order # 4886 — $9.95

# 4896	Miniassembler for VIC-20	$19.95
# 4881	Tennis, Squash, Break	$9.95
# 4894	Runfill for VIC	$9.95

Universal Experimenter Board for the VIC-20
(Save money with this great board). This board plugs right into the expansion slot of the VIC-20. The board contains a large prototyping area for your own circuit design and expansion. The construction article shows you how to build your own 3k RAM expander and ROM-board.
Order # 4844 — **$18.95**

Software for SINCLAIR ZX-81 and TIMEX 1000

# 2399	Machine Language Monitor	$9.95
# 2398	Mailing List	$19.95

Programming in BASIC and machine language with the ZX-81 (82) or TIMEX 1000.
Order # 140 (book) — $9.95

Books for OSI

# 157	The First Book of Ohio	$7.95
# 158	The Second Book of Ohio	$7.95
# 159	The Third Book of Ohio	$7.95
# 160	The Fourth Book of Ohio	$7.95
# 161	The Fifth Book of Ohio	$7.95

# 151 8K Microsoft BASIC Ref. Man.	$9.95
# 152 Expansion Handbook for 6502 and 6802	$9.95
# 153 Microcomputer Appl. Notes	$9.95

Complex Sound Generation
New revised applications manual for the Texas Instruments SN 76477 Complex Sound Generator.
Order # 154 — **$6.95**

Small Business Programs **Order # 156**
Complete listings for the business user. Inventory, Invoice Writing, Mailing List and much more. Introduction to Business Applications. — **$14.90**

MICRO™
Software Catalog

Name: **Concentrated Chemical Concepts**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: 3.3 DOS, disk drive
Description: This package of drill and practice programs covers the entire course in introductory general, organic, and biological chemistry for health science majors. The programs are intended for introductory college or advanced placement programs in high schools. No computer experience is necessary and complete documentation accompanies the programs.
Price: $300.00 - Part I
$225.00 - Part II
$150.00 - Part III
$550.00 - all nine disks
Part I (general) includes four disks, Part II (organic) includes three disks, and Part III (biology) includes two disks.
Author: Richard Cornelius
Available:
John Wiley & Sons, Inc.
Eastern Distribution Center
Order Processing Dept.
1 Wiley Drive
Somerset, NJ 08873

Name: **Pie Man**
System: Apple II Plus
Memory: 48K
Language: Machine language
Description: As the pies come out of the oven onto a conveyor belt, you (the baker's apprentice) must get a can of whipped cream, squirt it on the pie, grab a cherry, put it on the pie, then take the finished pie and put it in the pie bin. Watch out for flour sacks and grease spots on the floor and the slightly tipsy wedding-cake baker delivering his creations. If you let seven pies fall to the floor, you're fired.
Price: $29.95
Includes disk and instruction booklet.
Author: Eagle Berns, Michael Kosaka
Available:
Penguin Software
830 4th Avenue
Geneva, IL 60134

Name: **MicRo Quiz II**
System: TRS-80 Model III, VIC-20
Memory: 16K - TRS-80 Model III
8K - VIC-20
Language: BASIC
Description: *MicRo Quiz II* is a subject-independent CAI authoring package with class evaluation features that requires *no* computer programming knowledge.
Price: $39.95
Includes comprehensive, easy-to-use instruction manual.
Available:
M-R Information Systems, Incorporated
P.O. Box 73
Wayne, NJ 07470

Name: **Sensible Speller**
System: Apple II, Apple II Plus
Memory: 48K
Language: 5 versions available: Pascal, DOS 3.3, CP/M, Word Handler, and Super-Text
Hardware: One or two disk drives
Description: A spelling-verification program designed specifically for the Apple. The official *Random House Dictionary*, Concise Edition (80,000 plus words) is included in both diskette and hardcover form. The average time to proofread a 10-page document (about 3350 words) is one minute if there are no spelling mistakes or two minutes, 15 seconds for many spelling mistakes. Shorter documents will take less time.
Price: $125.00
Includes instruction manual, two copies of the *Sensible Speller* program diskette, a main dictionary diskette, a hardcover copy of the *Random House Dictionary*, Concise Edition, and binder to hold the complete package.
Author: Charles Hartley
Available:
Sensible Software
6619 Perham Drive
West Bloomfield, MI 48033
(313) 399-8877

Name: **INTROL-C/6809 Compiler**
System: 6809 running FLEX, UniFLEX, OS-9, Z80/8080 running CP/M
Memory: 48K plus 8K -FLEX-09
40K free memory -OS-9
60K - CP/M
Language: C
Description: This is a full C compiler system for developing programs in C for 6809-based target applications. The software package includes a C compiler, 6809 assembler, linking loader, run-time library, and library manager. *INTROL-C/6809* supports virtually all standard C as defined by Kernighan and Ritchie. It is efficient both in terms of size and speed of execution. Compiled programs are re-entrant, relocatable, and ROMable.
Price: From $475
Includes floppy disk, user's manual, and one-year maintenance program.
Author: Richard Pennington
Available:
Introl Corp.
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

Name: **Filmtape**
System: Apple II Plus
Memory: 48K
Language: Applesoft, compiled by Microsoft 'TASC'
Hardware: DOS 3.3 and printer
Description: *Filmtape* is designed to aid film editors and others who need rapid, frame-accurate translation of film times into television time codes. Working with a cut workprint and up to four O/S rolls, it can help trim up to 80% from on-line video editing times, often paying for itself in one session. Users may mix types of film and time code.
Price: $395.00
Includes diskette, manual, full support.
Available:
Editing Services
615 Fairground
Plymouth, MI 48170
(313) 459-4618

Name: **Economic Order Quantity (E.O.Q.) Package**
System: Apple II or IBM Personal Computer
Memory: 48K/Apple
64K/IBM
Hardware: Apple II — disk II disk controller and at least one disk II disk drive.
IBM — 80-column video monitor, optional printer
Description: The Execuware™ *Economic Order Quantity (E.O.Q.) Package* provides the businessman with a tool to minimize overall inventory costs. The package calculates the Economic Order Quantity and the Order Point. The sensitivity analysis affords the user the opportunity to perform "what if" analysis and to determine which variables cause the EOQ and the Order Point to vary significantly. Probability theory is used to simulate the variable demand, thereby insuring realistic inventory levels at all times.
Price: $174.95
Includes instruction manual and diskette.
Author: Execuware™ Microcomputer Software Division of Aeronca, Inc.
Available:
Apple and IBM Personal Computer dealers

Name: **The Count**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft BASIC
Hardware: DOS 3.3, one or more disk drives
Description: A winning Blackjack system; an interactive program which teaches strategies for playing-card counting, and betting for a winning Blackjack game.
Price: $24.95
Includes disk and manual.
Author: Pear Software
Available:
Insoft, Inc.
10175 S.W. Barbur Blvd.
Suite 202B
Portland, OR 97219

(continued)

Software Catalog *(continued)*

Name: **Hockey**
System: Atari 400/800
Memory: 16K RAM
Language: Assembler (Machine)
Hardware: Two, three, or four joysticks, cassette recorder or disk drive
Description: *Hockey* is a high-speed video action game for two, three, or four players. It is played on an enclosed rink, with scoreboard including clock overhead. Game players use joysticks to control the action. Offensive players skate with the puck, pass, and shoot. Defensive players steal the puck and intercept passes. Goalies block shots. *Hockey* includes "smart" players who perform automatically.
Price: $29.95
Available:
Gamma Software
P.O. Box 25625
Los Angeles, CA 90025
(213) 473-7441

Name: **AMPER-SORT/ MERGE II (A-S/M II)**
System: Apple II
Memory: 48K
Language: Applesoft BASIC and machine language
Hardware: DOS 3.3, disk drive
Description: *AMPER-SORT/ MERGE* is a general-purpose disk sort/merge utility for Apple DOS text files. Its machine-language file can sort 1000 records in seconds, alphanumerically (ascending or descending order) on up to five fields, random or sequential text files, and merge two to five pre-sorted files into a single file. It is compatible with most data-base programs that create standard DOS 3.3 text files. New features are S&H's super fast VisiFile index sort (callable from *within* VisiFile for effortless use) and a fast random access file index sort.
Price: $69.95
Includes disk and documentation.
Author: Alan G. Hill
Available:
S&H Software
58 Van Orden Road
Harrington Park, NJ 07640
(201) 768-3144

Name: **OMNIPACK**
System: Apple II, Apple III
Memory: 48K minimum - Apple II
128K minimum - Apple III
Language: BASIC and 6502 machine language
Hardware: At least one disk drive, printer optional
Description: *OMNIPACK* consists of three separate programs for which data files are fully interchangeable. *OMNIFILE* is a powerful RAM-based file management system and report generator, with global editing, built-in statistical functions, and flexible output formatting. *OMNIGRAPH* is a versatile data-plotting program for constructing X-Y plots, bar charts, and pie charts. *OMNITREND* is a powerful multiple-regression trend-analysis program.
Price: $129.95 - Apple II
$169.95 - Apple III
Includes two diskettes and user's manual.
Author: M.K. Booker
Available:
Educational Computing Systems
136 Fairbanks Road
Oak Ridge, TN 37839
(615) 483-4915

Name: **A.S.A.P.**
System: Apple II, Apple II Plus
Memory: 48K
Language: Apple Run-Time Environment for Pascal
Hardware: One disk drive (5¼), THE MILL 6809 co-processor
Description: The system allows a wide variety of popular programs to utilize the power of THE MILL. Similar to the Pascal Speed Up System, *A.S.A.P.* works with software intended for the Run-Time Environment — including PFS and VisiSchedule. It increases speed in processing, compilation, and printing.
Price: $295.00
Includes *A.S.A.P.* software and THE MILL.
Author: SB Programming
Available:
Stellation Two
The Lobero Bldg.
P.O. Box 2342
Santa Barbara, CA 93120
(805) 966-1140

Name: **Interface**
System: Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: DOS 3.3, disk drive
Description: *Interface* reads numerical tables in three formats, transforms and rearranges rows or columns, fits curves to data, and outputs files in several formats. The program's primary function is to translate from VisiCalc to Apple Plot, while adding flexibility and preventing erroneous graphs. It also supplements VisiCalc with rank ordering, alphabetizing and curve fitting, and outputs tables to Apple Writer or VisiCalc itself.
Price: $30.00
Includes instructions and copyable program disk.
Available:
Bill Starbuck
2100 E. Edgewood
Shorewood, WI 53211
(414) 963-9750

Name: **Fractions**
System: PET
Memory: 16K
Language: BASIC
Hardware: Cassette player or disk drive
Description: An overview program and a placement test program begin this carefully-structured sequence of 24 interactive programs. Eleven tutorial programs, each backed by a fun and challenging enrichment game program, help students (grade five and up) develop the confidence, concepts, and skills needed to master fractions.
Price: $175.00 for 12 tapes or 6 diskettes
Includes teacher's guide and software.
Author: Joanne Benton
Available:
Quality Educational Designs
P.O. Box 12486
Portland, OR 97212
(503) 287-8137

Name: **Guadalcanal Campaign**
System: Apple II, Apple II Plus, or Apple III
Memory: 48K
Language: BASIC
Hardware: One disk drive
Description: The 294-turn campaign game takes into account every Japanese and American warship that participated historically in the campaign. Each is rated for speed, cargo/plane-carrying capacity, damage points, number of main guns, secondary anti-aircraft guns and torpedo tubes. An abridged campaign (184 turns) is available as well as four mini-games which take only two to four hours to play. Game has both solitare and two-player versions.
Price: $59.95
Includes one disk, rulebook, and two maps.
Author: Gary Grigsby
Available:
Strategic Simulations Inc.
465 Fairchild Dr.
Suite 108
Mt. View, CA 94043

Name: **Tax Dodge**
System: Atari 400/800
Memory: 32K
Language: 6502 machine language
Description: *Tax Dodge* is a scrolling maze game in which the taxed citizen tries to collect as much money as possible without being hit by the tax collectors.
Price: $39.95
Author: Jon Freeman, Anne Westfall
Available:
Island Graphics
Box U
Bethel Island, CA 94511

Name: **Seafox**
System: Apple II, Apple II Plus, Atari 400/800
Memory: 48K
Language: Machine language
Hardware: Apple - keyboard, joystick, and paddle
Atari - joystick
Description: You are in control of a lone submarine looking for a convoy of enemy ships and its escort. Dodge exploding depth charges, avoid menacing mines, and evade speeding torpedos in an effort to eliminate the foe. You will need superior maneuvering ability, great courage, and a welcome aquatic ally to survive.
Price: $29.95
Author: Ed Hobbs
Available:
Broderbund Software, Inc.
1938 Fourth Street
San Rafael, CA 94901
(415) 456-6424

(Continued

Name: **DSS/F Decision Support System/ Finance**
System: Apple
Memory: 64K
Language: Pascal
Hardware: Disk drive, serial interface, Hewlett-Packard one, two, four, and eight pen plotters, or Houston Instrument plotters
Description: Micro *Decision Support System/Finance* (DSS/F) is a financial modeling and graphics system that assists managers, planners, and others with no previous computer knowledge to perform financial forecasting and reporting, investment analysis, cash flow forecasting, budgeting, consolidations, and strategic planning. Features include English modeling language, financial function, graphics, report generator, and sophisticated power.
Price: $1500.00
Includes manuals, software, and support.
Available:
Ferox Microsystems
1701 N. Fort Myer Drive
Arlington, VA 22209
Attn. Phil Evans
(703) 841-0800

Name: **Real Estate Analysis Package (REAP)**
System: Apple II, Apple II Plus
Memory: 48K
Hardware: One disk drive and printer
Description: The *Real Estate Analysis Package* performs property income analysis, calculates after-tax results if sold or exchanged, highlights tax sheltering effects, simulates inflation, and enables you to know when to buy, hold, or dispose of property. It allows up to 20-year projections and utilizes the Rule of 78's, ACRS Depreciation Methods, and multiple and/or assumable loans. A must for investors, tax advisors, and accountants.
Price: $274.95 suggested retail
Includes user's manual and a diskette.
Author: Execuware™
Available:
Computer retail stores

Name: **Super-Text™ 40/56/70**
System: Apple II
Memory: 48K
Language: Machine language
Hardware: Disk drive
Description: You can choose a 40-, 56-, or 70-column screen display without any additional hardware. *Super-Text* gives you the best features in word processing for easy text handling all the way through. It introduces the *Character Designer* for creative special display characters and includes *Autolink*, the file-linking system for one-step search and replace or print functions.
Price: $125.00
Includes tutorial documentation, quick reference card, and dual disk back-up.
Author: Ed Zaron
Available:
Muse Software
347 N. Charles St.
Baltimore, MD 21201
or computer stores

Name: **T/MAKER III**
System: Apple II, IBM PC, Osborn, NorthStar, or any system offering CP/M
Memory: 48K minimum
Description: *T/MAKER III* uses a unique visual syntax to facilitate easy yet powerful word processing/text editing, list management/tabulation, spreadsheet/scientifc calculations, load/unload data, and many other functions.
Price: $275.00 retail
Includes software, manual, and tutorial/quick reference booklet.
Author: Peter Roizen
Available:
TMAKER
1742 Willow Rd.
Suite 206
Palo Alto, CA 94304
(415) 326-6103
Call or write for distributor information

Name: **Mind Bogglers-1**
System: Atari 400/800
Memory: 16K - cassette
24K - disk
Hardware: Disk drive or cassette recorder
Description: Three thought-provoking mind bogglers are *Capture* — a strategy game in which you and the computer fight for control of the board (based on *Othello*); *Mystery Box* — a game in which you shoot rays into the mystery box to find the hidden atoms; and *Simon Says* — a memory teaser in which you must repeat the computer's pattern. The game adapts to the player's skill level.
Price: $15.95 - cassette
$19.95 - disk
Includes cassette or diskette and user's guide.
Available:
Versa Computing, Inc.
3541 Old Conejo Rd.
Suite 104
Newbury Park, CA 91320
(805) 498-1956

Name: **PTD-6502**
System: Apple II, Apple II Plus
Memory: 16K mimimum
Language: 6502 machine language
Hardware: Autostart ROM for fast breakpoint
Description: This BASIC-like compiled language debugs 6502 machine language and is relocatable. Nearly all commands can be executed in immediate mode or as part of a program with line numbers. Check on complex compound conditions at 1000 instructions/second, then see the 128 executed instructions prior to the condition.
Price: $49.95
Includes relocator source code.
Author: Edwin Rosenzweig
Harlan Harrison
Available:
Pterodactyl Software
1452 Portland Ave.
Albany, CA 94706

Name: **El Diablero**
System: Radio Shack Color Computer
Memory: 16K
Language: Assembly language
Hardware: Disk or cassette
Description: *El Diablero* is an adventure game extraordinaire! You wake, dazed and confused, in the middle of a southwestern desert. You had been learning the techniques of sorcery from an old man who told you that an evil sorcerer, a diablero, was his enemy. Now your teacher is missing and you are alone. You can't seem to remember the techniques you learned except the curious verse....
Price: $19.95 - cassette
$24.95 - disk
plus $2.00 S/H
Includes cassette or disk and instructions.
Author: Kenneth Kalish
Available:
Computerware
Box 668
Encinitas, CA 92024
(714) 436-3512

Name: **The Big Math Attack**
System: Apple II Plus, Atari
Memory: 48K - Apple II
16K - Atari cassette
24K - Atari disk
Language: Applesoft, Atari BASIC
Description: This program skillfully combines the excitement and challenge of an arcade game with basic math skills of addition, subtraction, multiplication, and division. An equation is launched from a spaceship. The player must enter the correct answer before the equation lands on the city. Grades one to six.
Price: $25.00 disk
$20.00 Atari cassette
Author: Schreiber & Schrebier
Available:
T.H.E.S.I.S.
P.O. Box 147
Garden City, MI 48135-0147
or from dealers

Name: **Alphabet Squares**
System: Apple II
Memory: 48K
Hardware: Disk drive
Description: An ideal program for the young computer user learning the ABC's. Excellent hi-res color graphics present three familiar objects: A is for airplane, B is for bird, etc. You use the joystick or paddles to move a pointer graphic from the letter to the correct picture. If correct, the graphic will expand to full screen as a reward. An exciting progam for that young user who would like to use the computer too.
Price: $29.95
Includes floppy diskette, user guide.
Available:
Versa Computing, Inc.
Suite 104
3541 Old Conejo Rd.,
Newbury Park, CA 91320
(805) 498-1956

(Continued)

Name: **Market Time**
System: CP/M, Apple Z80 card, IBM PC, Osborne 1, etc.
Memory: 34K
Language: A Compiled BASIC
Hardware: Requires Cursor Control and two disk drives
Description: *Market Time*, an easy-to-use menu-driven program, provides a data base of selected market statistics that can be analyzed with moving averages and plotted on screen or printer to spot market turning points. It also features an expandable data base to allow entering additional market statistics of the user's choice.
Price: $75.00
Includes program disk, data file disk, and user's manual in three-ring binder.
Available:
Hourglass Systems
P.O. Box 312,
Glen Ellyn, IL 60137
(312) 690-1855

Name: **AAARRRGGG!!!**
System: Atari 400/800
Memory: 16K - cassette
32K - disk
Language: Hybrid
Description: The fabric of space has been weakened by atomic bomb testing! Strange little creatures are popping through from another dimension, cluttering up the Earth. You have to catch as many as you can before your time runs out. Act quickly though, because the highest point-value creatures disappear the fastest. If you can catch the "SUPER AAARRRGGG," you'll get bonus time and super bonus points, but don't get poisoned by the glowing green radioactive creatures!
Price: $18.95 plus $2.00 S/H
Author: Bob Retelle
Available:
Pretzelland Software
2005 Whittaker Rd.
Ypsilanti, MI 48197
(313) 483-7358
or local dealers

Name: **Shuttle Intercept**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: One disk drive and a game paddle
Description: *Shuttle Intercept* takes you on a daring rescue mission into deep space. Your spacecraft is directed to retrieve friendly satellites bearing vital data and must fight or avoid enemy craft, satellites, missiles, and meteors.
Price: $34.95
Author: John Van Ryzin
Available:
The Hayden Software Co.
600 Suffolk Street
Lowell, MA 01853

Name: **Fast Figure**
System: CP/M, Apple, Z80, IBM PC, Wang MYP, DECmate, Prime Microdata, Osborne 1, NorthStar
Memory: 54K
Language: A compiled BASIC
Hardware: Cursor control, two disk drives, and Z80 card
Description: *Fast Figure*, a new electronic spreadsheet program with helping menus, offers sophisticated business calculations such as depreciation, present value and net present value, internal rate of return, compound growth, standard deviation, and what-if analysis in a package any business can easily afford. *Fast Figure's* three-dimensional file-sharing feature lets the user create additional multiple spreadsheets from one file without time consuming reentry of data.
Price: $150.00
Includes program disk and user's manual.
Available:
Hourglass Systems
P.O. Box 312
Glen Ellyn, IL 60137
(312) 690-1855

Name: **Pot 'O Gold**
System: Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: Disk drive, DOS 3.3, and paddles
Description: *Pot 'O Gold* is a medley of 46 games that includes such classics as *Eliza, Color Math, Keyboard Organ, Othello, Dragon Maze, Hexpawn*, and *Pinball*. With the Echo II speech synthesizer, text appearing on the screen will have a voice accompaniment.
Price: $39.95
Author: Jim Day
Available:
Rainbow Computing Inc.
19517 Business Center Dr.
Northridge, CA 91324
(213) 349-0300

Name: **Modula-2**
System: Apple II, Apple III, Z80/8080, TI9900
Memory: 64K
Language: Modula 2, Apple Pascal, UCSD Pascal Version 2.0
Description: The *Modula-2* language, designed by Pascal's creator Niklaus Wirth, provides a simple but powerful alternative for systems programming in assembly language, Pascal, C, and ADA. Features include modules, processes, separate compilation, dynamic array parameters, and low-level machine access.
Price: $550.00
Includes compiler, librarian, run-time library, and interpreter.
Available:
Volition Systems
P.O. Box 1236
Del Mar, CA 92014
(714) 481-2286

Name: **TransFORTH**
System: Apple II or Apple II Plus (Apple III version sold separately)
Memory: 48K
Language: Machine language
Hardware: DOS 3.3, one or more disk drives
Description: *TransFORTH* is a compiled programming language similar to FORTH that features floating-point capability and scientific functions, DOS 3.3 compatibility, versatile array structures, extensibility, and structured interactive programming.
Price: $125.00
Includes disk and manual.
Author: Paul Lutus
Available:
Insoft, Inc.
10175 S.W. Barbur Blvd.
Suite 202B
Portland, OR 97219

Name: **The Filer**
System: Apple II, Apple II Plus
Memory: 48K
Language: 6502 Assembly
Hardware: DOS 3.3, one or more disk drives
Description: This is a general utility system for the Apple. Features include FAST copy program, disk speed and check, copy, delete, lock, unlock files, change booting program, and a catalog with space on the disk.
Price: $19.95
Includes one disk with instructions.
Available:
Central Point Software, Inc.
P.O. Box 19730-203
Portland, OR 97219

Name: **Darkstar™**
System: Timex-Sinclair 1000 (ZX81), Apple II, Atari 800/400
Memory: 16K RAM/Sinclair
48K RAM/Apple II and Atari
Language: BASIC
Hardware: Standard cassette tape deck — Sinclair
Disk drive with DOS 3.3 — Apple
810 disk drive or 410 recorder — Atari
Description: This program solves problems associated with the photographic darkroom. It provides exposure times needed for changes to magnification, lens opening, and print density for both black-and-white and color materials, for both chromogenic or dye-bleach materials. It provides color-printing filter-pack checking and correcting for color balance, neutral density, and filter factors, and development times for black-and-white films over a wide temperature range, as a function of the user's ideal processing time at 68°F. Expert-type program.
Price: $99.95/Sinclair tape
$129.95/Apple/Atari disk
$129.95/Atari tape
Includes 34 pages of documentation.
Author: Bob Nadler
Available:
F/22 Press
P.O. Box 141
Leonia, NJ 07605

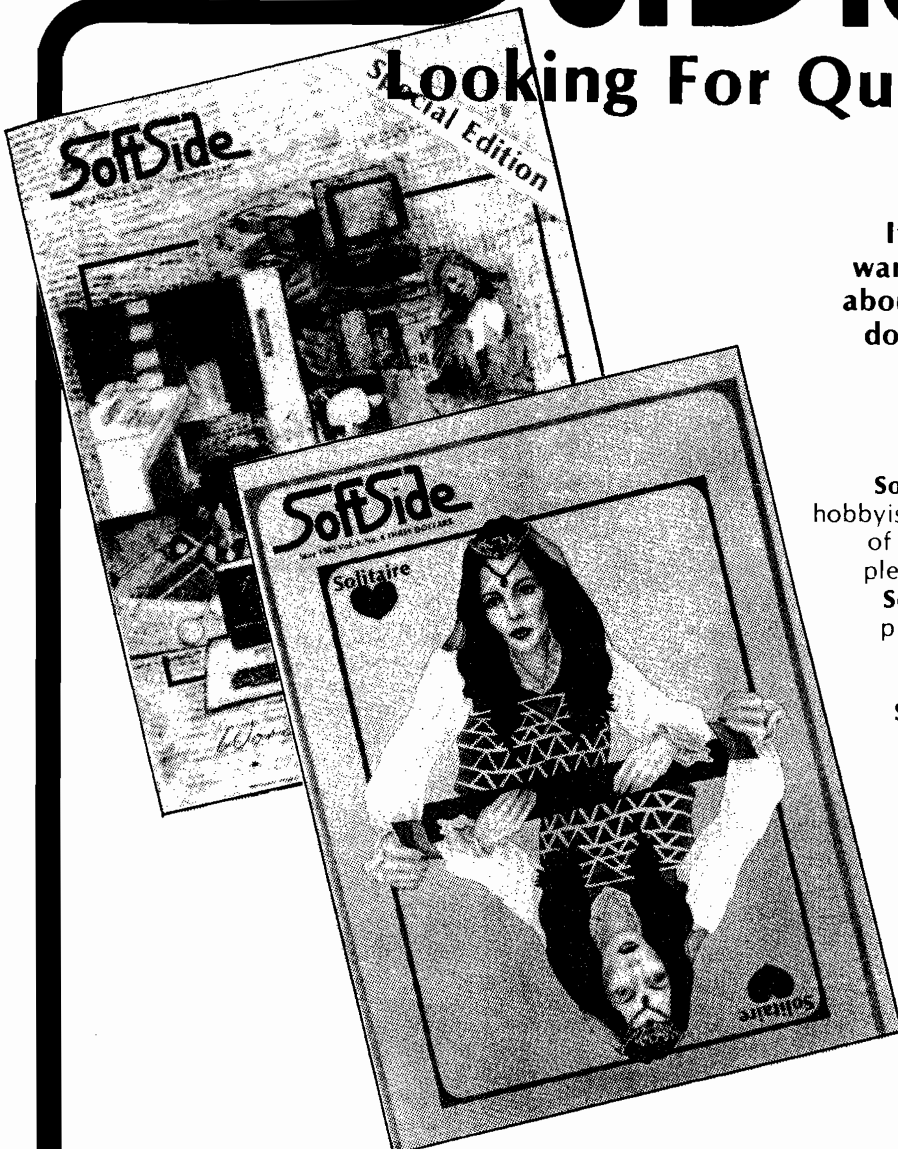
MICRO

SoftSide™

Looking For Quality Software?

READ THIS:

If the high price of commercial software and the lack of clear information about your microcomputer has got you down, here's the solution you've been waiting for!



SoftSide Magazine

SoftSide is a favorite of computer users and hobbyists alike. They rely on it as a prime source of programs, reviews and articles for the Apple™, ATARI®, and TRS-80® microcomputers. **SoftSide** is the magazine for the microcomputer owner who wants to learn BASIC programming, learn MORE about BASIC programming, or just wants to have FUN! **SoftSide** gives you the BASIC code listings of several programs — adventures, utilities, games, simulations, you name it — for your computer EVERY MONTH.

There's more:

- **Reviews** — of the software and hardware products you want to know about.
- **Articles** — about all aspects of BASIC programming.
- **Tutorials** — on graphics, use of important commands, and more.
- **Programs** — each month **SoftSide** publishes a variety of program for the Apple, ATARI® and TRS-80®.
- **Columns** — which discuss different topics including: computer graphics, picking the right modem for you and marketing your software — just to name a few.
- **Input from our readers** — each month we devote a space in the magazine to let our readers give us some feedback about **SoftSide**.
- **Hints & Enhancements** — programmers and readers provide us with enhancements, to our programs, and programming tips.

Use coupon to order. Mail to: **SoftSide** Publications, 6 South St., Milford, NH 03055

As you can see, you'll receive pages and pages of information and entertainment from SoftSide. Here's the best part:

A subscription to SoftSide is just $24 a year. That's 12 issues for only $2 each! What a value!

☐ **YES! Send me the first copy of my SoftSide subscription right away!**

$24/yr for USA and Canada only. For orders to APO/FPO or Mexico — $40/yr. Other foreign orders — $62/yr.

I own a ☐ Apple ☐ ATARI® ☐ TRS-80®

Name ______________________
Address ______________________
City/State ______________ Zip ________

☐ Check is enclosed
☐ MasterCard ☐ VISA

Name of Cardholder ______________________
MC# and Interbank#/VISA# ______________________
Exp. Date ______________________
Signature ______________________

MICRO™

Hardware Catalog

Name: **VC-PLUS 80-column**
System: Apple II
Memory: 48K
Hardware: Legend 128K or 64K RAM expansion card(s)
Description: Add memory to Personal Software's 16-sector VisiCalc program using Legend memory cards and the new *VC-PLUS* with 80-column capability. Your Apple II can have more power than an Apple III at a fraction of the cost.
Price: Free with purchase of RAM card.
$20.00 as an update.
Includes diskette and operation manual.
Available:
Legend Industries, Ltd.
2220 Scott Lake Rd.
Pontiac, MI 48054
(313) 674-0953

Name: **Kraft Precision Joystick**
System: Apple II, IBM PC, TRS-80 Color Computer
Description: *Kraft Precision Joystick* features instantly selectable spring-centering or free-floating stick modes at the flip of a switch. High-quality potentiometers ensure greater linearity and better stick performance. Full one-year warranty.
Price: $64.95 - $69.95
Available:
Contact Kraft Systems for name of nearest dealer
(714) 724-7146

Name: **MAC INKER**
Description: *MAC INKER* automatically re-inks ribbons for any printer at an average cost of five cents per ribbon. Operation is simple. The ink contains a special lubricant that helps improve the life of the print-head.
Price: $54.95
Includes one two-ounce ink bottle (approximately six months of intensive use).
Available:
Computer Friends
100 North West 86th Ave.
Portland, OR 97229
(503) 297-3231

Name: **6522 Parallel I/O Card**
System: Commodore VIC-20
Language: BASIC or assembly language
Description: This card is designed to plug directly into the VIC's expansion port. It provides two programmable 8-bit ports with expanded handshake capability that allow the user to interface any parallel peripheral device to the VIC-20. It also includes two 16-bit programmable timer/counters and a serial data port. The on-board switch-selectable address feature allows the alteration of the card's memory location within the system and provides for the use of multiple cards when an expansion chassis is utilized.
Price: $69.95 — assembled and tested
$59.95 — kit
Includes the user guide and application notes.
Available:
Fountain Intelligent Devices Company
P.O. Box 913
Palo Alto, CA 94302
OEM and dealer inquiries welcome

Name: **DISKBUB**
Description: *DISKBUB* is a compact bubble-memory board with 128K bytes of data storage. It will interface to the FLEX™ operating system using a 68XX-based microprocessor with a 30-pin ss 50 I/O bus. *DISKBUB* acts like a disk but has the advantages of bubble memory, high reliability data storage, and operation in harsh environments. *DISKBUB* can be used to boot up systems, replacing the need for disks altogether. Its applications include process control, automation, data logging, and robotics. It can be used virtually anywhere a computer must withstand a harsh environment.
Price: $995.00
Available:
Universal Data Research Inc.
2457 Wehrle Drive
Buffalo, NY 14221

Name: **EPROM Pack**
System: TRS-80 Color Computer
Memory: 4K and up
Language: BASIC or Extended BASIC
Description: The *EPROM Pack* is a plug-in cartridge for the Color Computer that allows up to 16K bytes of user ROM to be added simply and quickly to the machine. Four sockets are contained in the pack to allow 2732-type EPROMs to be inserted. Additional programs, like assemblers, word processors, graphics, and games can be permanently available to the computer.
Price: $39.95
Includes EPROM Pack cartridge and full instructions.
Available:
Maple Leaf Systems
Box 2190, Station "C"
Downsview, Ontario, Canada M2N 2S9

Name: **Atari Bank Select Memory**
System: Atari 400
Memory: 64K
Description: The board consists of 48K RAM with four banks of 4K RAM addressed above the 48K limit to insure that the 48K is continuous and 52K RAM is always available. It also means a ROM cartridge will never affect the availability of the bank select RAM. The 4K RAM banks allow for a larger hard-wired RAM size and *all* Atari software and peripherals are compatible.
Price: $249.95 suggested retail
Available:
Mosaic Electronics
P.O. Box 708
Oregon City, OR 97045
(800) 547-2708

Name: **Computer Case**
System: Commodore 64
Description: CM703 holds the Commodore 64 computer, one or two 1541 disk drives, power supply, and other equipment. CM704 holds the Commodore 64 computer and dataset program recorder (plus other equipment). These cases provide portability and a convenient method of storage, free from possible damage and dust accumulation. The computer and software are protected from tampering and unauthorized use by replacing and locking the lid.
Price: $119.00 - CM703
$109.00 - CM704
Available:
Computer Case Company
5650 Indian Mound Court
Columbus, OH 43213
(800) 848-7548
Or most computer stores

Name: **NOVADAPTER**
Description: *NOVADAPTER* consists of two 25-pin D-connectors, 25 short wires with pins crimped-on, some B-crimps, and a hood. Using the short wires you can wire between the pin positions and create the cable connection quickly. Ideal for extension cables, gender changers, and null modems. It replaces all existing cables with 25-pin D-connectors.
Price: $30.00
Available:
Innovative Supplies & Accessories Inc.
P.O. Box 61149
Dallas, TX 75261
(214) 641-8090

Name: **ROM Simulator**
Description: This is a new fast-responding ROM simulator that is capable of emulating virtually any ROM, programmable ROM, or erasable PROM. The simulator occupies one card slot of any IEEE standard S100 bus computer. The P&E board also simulates memory-response time for experimenting with various timing possibilities. When not in use as a simulator, the board can function as additional RAM for the microprocessor or as an I/O port-driven memory extension unit.
Price: $600.00
Includes 2K RAM and complete manual.
Available:
P&E Microcomputer Systems, Inc.
P.O. Box 2044
Woburn, MA 01880
(617) 944-7585

Name: **PET Joystick Interface**
System: PET/CBM
Description: This versatile interface card adds joystick/paddle capabilities to all PET/CBM computers. The device enables the PET to accept input directly from two Apple joysticks, four Apple game paddles, or two Atari joysticks. The interface is complete and ready to plug into the user port. All modes of operation are software-selectable. The device features short access time (less than 10 milliseconds/joystick) and high-resolution digitization (greater than 8 bits). Fast machine-language input routines, callable from a BASIC program, are included.
Price: $49.95
Includes interface card,power supply, documentation, and sample software.
Available:
J Systems Corp.
1 Edmund Place
Ann Arbor, MI 48103
(313) 662-4714

Name: **The Spectrum Stick**
System: Color Computer
Memory: 4K-64K
Language: Microsoft BASIC
Hardware: Joystick
Description: *The Spectrum Stick* has the following features: hair trigger firebutton, swivel ball-type component joystick to give you a smooth and true feel, red LED power indicator to remind you to shut off the Color Computer after the TV, brush-aluminum knob, and extra-long cable.
Price: $39.95 plus $2.00 S/H
Includes joystick, firebutton, case, and cable.
Available:
Spectrum Projects
93-1586 Drive
Woodhaven, NY 11421
(212) 441-2807

Name: **RS-232 Expansion Cable**
System: Color Computer
Memory: 4K and up
Hardware: "Y" cable
Description: The *RS-232 Expansion Cable* allows two devices to be connected to the serial I/O port at the same time. A printer and modem can be hooked in-line without constantly swapping cables.
Price: $19.95 plus $1.00 S/H
Available:
Spectrum Projects
93-1586 Drive
Woodhaven, NY 11421
(212) 441-2807 Voice
(212) 441-3755 Computer

Name: **BUSMAN**
System: Commodore PET/CBM
Description: *BUSMAN* provides dual IEEE-488 busses; one for "local" peripherals used excusively by the installed system; the other allows multiple *BUSMAN*s to be networked together to share "common" peripherals. It maintains the stand-alone ability of Commodore systems plus networking.
Price: $595.00 each
Available:
Lem Data Products
P.O. Box 1080
Columbia, MD 21044

Name: **MULTIPORT™**
System: TRS-80 Color Computer
Memory: 4K-32K
Language: BASIC
Description: The *MULTIPORT* is a hardware device containing four sockets that allow Color Computer peripherals (disks, program cartridges, I/O cards, etc.) to be on-line at all times and selectable under software control.
Price: $99.50
Includes fully assembled and tested *MULTIPORT* and documentation.
Available:
Maple Leaf Systems
P.O. Box 2190, Station "C"
Downsview, Ontario
Canada M2N 2S9

Name: **SYSTEM 200™**
System: Apple II, Apple III, IBM PC, etc.
Description: These modular, solid-oak units feature contemporary design with unique disk storage capabilities allowing random selection of any disk by label. The units house disk drives, manuals, monitor, and accessories as well. All the units function individually or collectively with available add-on modules in user-customized configurations.
Price: From $31.95 to $279.95 per module, FOB shipping point
Includes *Floppy Fingers*™ diskette holders, *Floppy Drawers*™ diskette filing system, *The Bridge* ™ hardware organizer.
Available:
Venice Woodworking Co.
12810 Venice Boulevard
Los Angeles, CA 90066
(213) 390-4885

Name: **RS-232 Expansion**
System: Color Computer
Memory: 4K and up
Hardware: "Y" cable
Description: The *RS-232 Expansion Cable* allows two devices to be connected to the serial I/O port at the same time. A printer and modem can be hooked in-line without constantly swapping cables.
Price: $19.95 plus $1.00 S/H
Available:
Spectrum Projects
93-1586 Drive
Woodhaven, NY 11421
(212) 441-2807 Voice
(212) 441-3755 Computer

Name: **ALIS Instrumentation Modules**
System: 48K Apple II or Apple II Plus
Memory: 2.5K RAM maximum
Language: 6502 Machine and Applesoft RAM or ROM
Hardware: Disk drive, printer optional
Description: The ALIS family of data acquisition and control modules for an Apple II computer provides an economical multi-function laboratory or industrial instrumentation system. Hardware and augmented BASIC software permit 8- or 12-bit analog input/output, and multi-function digital I/O at rates up to 10K Hz under ALIS software control. The digital module provides 32 bidirectional lines, 2 16-bit hardware clocks, and up to 14 serviceable interrupt conditions.
Price: $1149.00 8-bit analog input
$1517.00 12-bit analog input
$1787 Digital I/O
$613.00 - $991.00 Analog output
Includes PC card(s), cables, terminal box, AMPERALIS and real time graphics software on diskette, and manual.
Available:
Eco-Tech, Inc.
2990 Lake Lansing Rd.
P.O. Box 776
East Lansing, MI 48823
(517) 337-9226

Name: **Executive Compu-Cover**
System: All microcomputers
Description: The *Executive Compu-Cover* is an attractive, high-quality leatherette cover for computers, disk drives, monitors, printers, and other peripheral equipment. These covers will prevent dust and foreign matter from entering and damaging equipment when not in use.
Price: $14.95 ppd. - Apple II
Other prices on request.
Available:
Executive Compu-Cover
76-51 169 Street
Queens, NY 11366
(212) 969-1079

Name: **Apple-Verter Model APX 800**
System: Apple II
Description: This plug-in video to RF modulator for Apple II operates in high VHF band (Ch. 7-10), tunable. It attaches inside the Apple II w/VELCRO, then plugs into an existing video/power connector. The die-cast aluminum housing executes frequency stability with no assembly needed. Built-in 5V regulator allows use with other computer systems.
Price: $29.75
Available:
ATV Research or local dealers

Name: **BUBBLE™**
System: APPLE II
Memory: 128K Bubble memory
Hardware: Intel 7110
Language: DOS 3.3 with patches for Pascal and CP/M
128K Non-volitile memory with on-board boot prom eliminates disk and operates in any environment. It is three to four times faster and 1000 times more reliable than Floppy. *Bubble* boots directly from the module or operates in conjunction with disk system.
Price: $875.00
Includes software for DOS 3.3 emulation and boot prom.
Available:
MPC Peripherals Corporation
9424 Chesapeake Drive
San Diego, CA. 92123
or your local Computerland Store

MICRO

MICRO™

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

6809 Bibliography

93. Color Computer News (August, 1982)

Lester, Lane P., "Mileage Monitor," pg. 40-42.
An automotive program for the Color Computer.

Bogan, John R., "Digits," pg. 44-45.
Mix alphanumerics with your graphics programs with this program for the Color Computer.

Phelps, Andrew, "Comment Corner," pg. 46-48.
Documentation of the "get next character" routine of the TRS-80 Color Computer.

Sullivan, Steve, "Venus Lander," pg. 49-54.
A game for the 6809-based Color Computer.

Hawks, Christopher R., "Homebrew 64K Conversion," pg. 57-58.
A hardware article for TRS-80 Color Computer users.

Hogg, Frank, "64K Korner," pg. 59-60.
Miscellaneous notes on memory assignments in the 6809-based TRS-80 Color Computer.

Rothstein, Mark, "On Modifying Packaged ROM Programs," pg. 61-64.
Notes on ROM program modification including a listing for an "Automated Address Modification Program" for the Color Computer.

Giovanoni, Richard, "Learning Curves — A Real Life Use of Microcomputers in Aerospace," pg. 65-76.
A Color Computer curve graphics program. Listing and examples.

Hornsby, James A., "Justprin," pg. 77-83.
A word-processing program featuring proportional spacing for the Color Computer.

Hunt, Craig, "Air Raid," pg. 86-91.
A graphics game for the 6809-based Color Computer.

94. MICRO, No. 52 (September, 1982)

Clark, Hal, "6809 Macros for Structured Programming," pg. 57-63.
This article presents a technique for using 6809 assembler macros to allow structured assembly-language programming.

Suckle, Leonard I., "Market Projection Program for the Color Computer," pg. 67-77.
A sophisticated business program implemented on the 6809-based TRS-80 Color Computer.

Staff, "MICRO Reviews in Brief," pg. 103-106.
Reviews include several pieces of software for the 6809 user.

Dial, Wm. R., "6809 Bibliography," pg. 110.
Some 32 references to the 6809 literature are cited.

Staff, "MICRO Software Catalog," pg. 111-116.
Several software programs for 6809-related equipment are cited.

Staff, "MICRO Hardware Catalog," pg. 117.
Hardware for 6809 systems are listed.

95. 80-U.S. Journal, 5, No. 9 (September, 1982)

Tangeman, Richard, "Color Computer Assembler/Disassembler," pg. 38-45.
Get inside your 6809E-based Color Computer with this utility.

Davis, Lynn, "Three Color Computer Video Tips," pg. 96-97.
Several listings for the TRS-80 Color Computer involving switching PMODEs for effect.

96. '68' Micro Journal, 4, Issue 9 (September, 1982)

Anderson, Ronald W., "FLEX User Notes," pg. 8-11.
Notes on addition and subtraction routines with 6809 FLEX. Notes on FORTH, Pascal, "C", BASIC, and assembler languages.

Nay, Robert L., "Color User Notes," pg. 11-12.
Notes of interest to TRS-80 Color Computer users.

Distefano, Tony, "Color Clinic," pg. 13-14.
A hardware modification to alter the background color on the 6809-based Color Computer.

Commo, Norm, " 'C' User Notes," pg. 14-18.
Discussion of initialization code for the 'C' user.

Urie, Paul M., "Telecon C," pg. 18-20.
Discussion of new utilities for 6809 systems. Includes some benchmark tests.

Zeff, Robert, "Simple Winchester Interface," pg. 20-21.
Hardware and procedure for implementing a Winchester hard disk on 6809 systems.

Pass, E.M., "6800 to 6809," pg. 24-26.
Converting 6800 assembler language to 5809 assembler language.

Watson, Ernest Steve, "Home Accounting Program," pg. 26-28.
Part III reads in from a data file information concerning the proposed budget.

Williams, Don, "Structured Assembler/Translate Subroutines," pg. 36-38.
Macros and subroutines based on the 6809 assembler.

97. Dr. Dobb's Journal 7, Issue 9 (September, 1982)

Dorner, George C., "68XX Blurb," pg. 7.
A users' group is forming for those interested in the 6809 microprocessor, the OS-9 operating system, and UNIX-like systems on 68xx machines.

98. PET Benelux Exchange 3, No. 2 (July, 1982)

Anon., "SuperPET and 8096," pg. 3 (insert).
Short note on the 6809-based SuperPET.

99. BYTE 7, No. 9 (September, 1982)

Stuart, John, "Three Dee Tee," pg. 34-50.
A computer game for the 6809-based TRS-80 Color Computer based on Rubik's Cube and Tic Tac Toe.

Staff, "Software Received," pg. 494-496.
Reviews a couple of programs for the 6809-based systems.

100. Compute! 4, No. 9 (September, 1982)

Chastain, Linton S., "TRS-80 Color Computer Energy Monitor Graphics," pg. 130-131.
A companion program for the 'Energy Monitor' program cited earlier.

101. 80 Micro (October, 1982)

Calle, Carlos, "Personal Finance," pg. 38.
A finance program for the Color Computer.

Norman, Scott L., "Newtalk," pg. 38-39.
A utility for those doing machine-language or assembly-language programming on a 6800 or 6809 system.

Stone, Stephen G., III, "Color Scripsit."
Word processing for the TRS-80 Color Computer.

Barden, William, Jr., "The Color Computer on Parade — Part I," pg. 82-87.
All about graphics on the 6809-based Color Computer.

Stark, Peter A., "Income Tax Estimator," pg. 168-182.
Use your 6809-based Color Computer for the IRS.

MICRO

TRS-80® Color Computer, 6809E-based computer, manufactured by Radio Shack,® a division of Tandy Corporation.

TRS-80C comes standard with 4K RAM, color BASIC-in-ROM, cassette, and serial interface. RS options include expansion to 16K RAM, joysticks, cassette recorder, printer, disk system. Additional options available are expansion to 64K, FLEX, and OS-9 operating systems. Standard screen output to TV; may be modified for use with monitor.

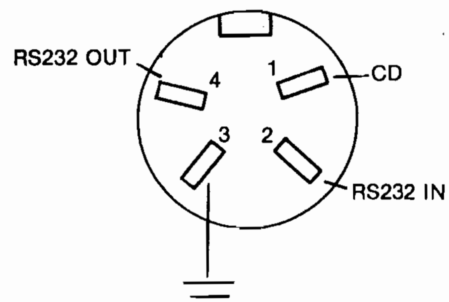
MEMORY MAP

Decimal	Hex	Memory Contents
0-255	0-FF	Direct page RAM
256-273	100-111	Interrupt Vector's
274-276	112-114	USRJMP — Jump to BASIC's USR routine
278-279	116-117	RND Function SEED
282	11A	Keyboard Alpha lock — 0 = not locked, FF = locked
283-284	11B-11C	Keyboard delay constant
285-337	11D-151	BASIC token directory
338-345	152-159	Keyboard rollover table
346-349	15A-15D	Joystick pot values
350-425	15E-1A9	System jump vectors
426-1023	1AA-3FF	I/O Buffers
1024-1535	400-5FF	Video display
1536-4095	600-0FFF	Program and variable storage (4K RAM)
1536-16383	600-3FFF	Program and variable storage (16K RAM)
1536-32767	600-7FFF	Program and variable storage (32K RAM)
32768-40959	8000-9FFF	Extended Color BASIC
40960-49151	A000-BFFF	COLOR BASIC (8K ROM)
49152-65279	C000-FEFF	Program Pak Memory
65280-65535	FF00-FFFF	Input/Output

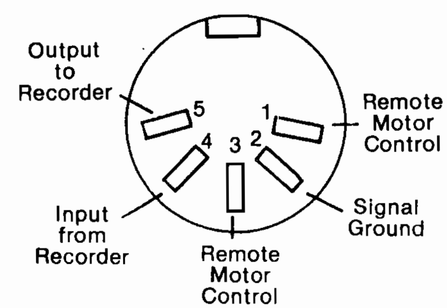
RS232 PRINTER VARIABLES

Variable	Hex Address	Dec. Address	Hex Value	Dec. Value	Quant.
Baud Rate	95	149	01CA	1-202	120 Baud
			00BE	0-180	300 Baud
			0057	0-87	600 Baud
			0029	0-41	1200 Baud
			0012	0-18	2400 Baud
Line Delay	97	151	0001	0-1	None
			4000	64-0	.288 Sec.
			8000	128-0	.576 Sec.
			FFFF	255-255	1.15 Sec.
Line Width	9B	155	10	16	16 Char/Line
			20	32	32 "
			40	64	64 "
			84	132	132 "
			FF	255	255 "
Comma Field Width	99	153	10	16	16
Last Comma Field	9A	154	70	112	112

RS232C Interface Pinout



Cassette Interface Pinout



High Logic (Logic 1) voltage greater than +3 volts
Low Logic (Logic 0) voltage less than −3 volts

It is assumed that: the printer generates a busy output when not ready. The printer automatically will carriage return at the end of a line. The data format is 1 Start Bit (Logic 0), 2 Stop Bits (Logic 1), 7 Data Bits (LSB First), and No Parity Bit.

TRS-80C

MICRO™ Data Sheet #12

Decimal	Hex	Keyboard	Inverse Screen	6809
0	00		@	NEG D
1	01		A	—
2	02		B	—
3	03	Break	C	COM D
4	04		D	LSR D
5	05		E	—
6	06		F	ROR D
7	07		G	ASR D
8	08	←	H	ASL/LSL D
9	09	→	I	ROL D
10	0A	↓	J	DEC D
11	0B		K	—
12	0C	CLR	L	INC D
13	0D	ENTER	M	TST D
14	0E		N	JMP D
15	0F		O	CLR D
16	10		P	—
17	11		Q	—
18	12		R	NOP I
19	13	# @	S	SYNC I
20	14		T	—
21	15	# ←	U	—
22	16		V	LBRA R
23	17		W	LBSR R
24	18		X	—
25	19		Y	DAA I
26	1A		Z	ORCC #
27	1B		[	—
28	1C		\	ANDCC #
29	1D		]	SEX I
30	1E		↑	EXG I
31	1F		←	TFR I
32	20	Space	Space	BRA R
33	21	!	!	BRN R
34	22	"	"	BHI R
35	23	#	#	BLS R
36	24	$	$	BHS/BCC R
37	25	%	%	BLO/BCS R
38	26	&	&	BNE R
39	27	'	'	BEQ R
40	28	(	(	BVC R
41	29	)	)	BVS R
42	2A	*	*	BPL R
43	2B	+	+	BMI R
44	2C	,	,	BGE R
45	2D	-	-	BLT R
46	2E	.	.	BGT R
47	2F	/	/	BLE R
48	30	0	0	LEAX ,
49	31	1	1	LEAY ,
50	32	2	2	LEAS ,
51	33	3	3	LEAU ,
52	34	4	4	PSHS #
53	35	5	5	PULS #
54	36	6	6	PSHU #
55	37	7	7	PULU #
56	38	8	8	—
57	39	9	9	RTS I
58	3A	:	:	ABX I
59	3B	;	;	RTI I
60	3C	<	<	CWAI I
61	3D	=	=	MUL I
62	3E	>	>	—
63	3F	?	?	SWI I

Decimal	Hex	Keyboard	Normal Screen	6809
64	40	@	@	NEGA I
65	41	#A	A	—
66	42	#B	B	—
67	43	#C	C	COMA I
68	44	#D	D	LSRA I
69	45	#E	E	—
70	46	#F	F	RORA I
71	47	#G	G	ASRA I
72	48	#H	H	ASLA/LSLA I
73	49	#I	I	ROLA I
74	4A	#J	J	DECA I
75	4B	#K	K	—
76	4C	#L	L	INCA I
77	4D	#M	M	TSTA I
78	4E	#N	N	—
79	4F	#O	O	CLRA I
80	50	#P	P	NEGB I
81	51	#Q	Q	—
82	52	#R	R	—
83	53	#S	S	COMB I
84	54	#T	T	LSRB I
85	55	#U	U	—
86	56	#V	V	RORB I
87	57	#W	W	ASRA I
88	58	#X	X	ALSB/LSLB R
89	59	#Y	Y	ROLB I
90	5A	#Z	Z	DECB I
91	5B	#↓	[	—
92	5C	#CLR	\	INCB I
93	5D	#→	]	TSTB I
94	5E	↑	↑	—
95	5F	#↑	←	CLRB I
96	60		Space	NEG ,
97	61	A	!	—
98	62	B	"	—
99	63	C	#	COM ,
100	64	D	$	LSR ,
101	65	E	%	—
102	66	F	&	ROR ,
103	67	G	'	ASR ,
104	68	H	(	ASL/LSL ,
105	69	I	)	ROL ,
106	6A	J	*	DEC ,
107	6B	K	+	—
108	6C	L	,	INC ,
109	6D	M	-	TST ,
110	6E	N	.	JMP ,
111	6F	O	/	CLR ,
112	70	P	0	NEG E
113	71	Q	1	—
114	72	R	2	—
115	73	S	3	COM E
116	74	T	4	LSR E
117	75	U	5	—
118	76	V	6	ROR E
119	77	W	7	ASR E
120	78	X	8	ASL/LSL E
121	79	Y	9	ROL E
122	7A	Z	:	DEC E
123	7B		;	—
124	7C		<	INC E
125	7D		=	TST E
126	7E		>	JMP E
127	7F		?	CLR E

Decimal	Hex	Graphic Screen	BASIC		6809
128	80	GREEN	FOR	INT	SUBA #
129	81	GREEN	GO	ABS	CMPA #
130	82	GREEN	REM	USR	SBCA #
131	83	GREEN	'	RND	SUBD #
132	84	GREEN	ELSE	SIN	ANDA #
133	85	GREEN	IF	PEEK	BITA #
134	86	GREEN	DATA	LEN	LDA #
135	87	GREEN	PRINT	STR$	—
136	88	GREEN	ON	VAL	EORA #
137	89	GREEN	INPUT	ASC	ADCA #
138	8A	GREEN	END	CHR$	ORA #
139	8B	GREEN	NEXT	EOF	ADDA #
140	8C	GREEN	DIM	JOYSTK	CMPX #
141	8D	GREEN	READ	LEFT$	BSR R
142	8E	GREEN	RUN	RIGHT$	LDX #
143	8F	GREEN	RESTORE	MID$	—
144	90	YELLOW	RETURN	POINT	SUBA D
145	91	YELLOW	STOP	INKEY$	CMPA D
146	92	YELLOW	POKE	MEM	SBCA D
147	93	YELLOW	CONT		SUBD D
148	94	YELLOW	LIST		ANDA D
149	95	YELLOW	CLEAR		BITA D
150	96	YELLOW	NEW		LDA D
151	97	YELLOW	CLOAD		STA D
152	98	YELLOW	CSAVE		EORA D
153	99	YELLOW	OPEN		ADCA D
154	9A	YELLOW	CLOSE		ORA D
155	9B	YELLOW	LLIST		ADDA D
156	9C	YELLOW	SET		CMPX D
157	9D	YELLOW	RESET		JSR D
158	9E	YELLOW	CLS		LDX D
159	9F	YELLOW	MOTOR		STX D
160	A0	BLUE	SOUND		SUBA ,
161	A1	BLUE	AUDIO		CMPA ,
162	A2	BLUE	EXEC		SBCA ,
163	A3	BLUE	SKIPF		SUBD ,
164	A4	BLUE	TAB(		ANDA ,
165	A5	BLUE	TO		BITA ,
166	A6	BLUE	SUB		LDA ,
167	A7	BLUE	THEN		STA ,
168	A8	BLUE	NOT		EORA ,
169	A9	BLUE	STEP		ADCA ,
170	AA	BLUE	OFF		ORA ,
171	AB	BLUE	+		ADDA ,
172	AC	BLUE	–		CMPX ,
173	AD	BLUE	*		JSR ,
174	AE	BLUE	/		LDX ,
175	AF	BLUE	↑		STX ,
176	B0	RED	AND		SUBD E
177	B1	RED	OR		CMPA E
178	B2	RED	>		SBCA E
179	B3	RED	=		SUBA E
180	B4	RED	<		ANDA E
181	B5	RED	!		BITA E
182	B6	RED			LDA E
183	B7	RED			STA E
184	B8	RED			EORA E
185	B9	RED			ADCA E
186	BA	RED			ORA E
187	BB	RED			ADDA E
188	BC	RED			CMPX E
189	BD	RED			JSR E
190	BE	RED			LDX E
191	BF	RED			STX E

Decimal	Hex	Graphic Screen	6809
192	C0	BUFF	SUBB #
193	C1	BUFF	CMPD #
194	C2	BUFF	SBCB #
195	C3	BUFF	ADDD #
196	C4	BUFF	ANDB #
197	C5	BUFF	BITB #
198	C6	BUFF	LDB #
199	C7	BUFF	—
200	C8	BUFF	EORB #
201	C9	BUFF	ADCB #
202	CA	BUFF	ORB #
203	CB	BUFF	ADDB #
204	CC	BUFF	LDD #
205	CD	BUFF	—
206	CE	BUFF	LDU #
207	CF	BUFF	—
208	D0	CYAN	SUBB D
209	D1	CYAN	CMPB D
210	D2	CYAN	SBCB D
211	D3	CYAN	ADDD D
212	D4	CYAN	ANDB D
213	D5	CYAN	BITB D
214	D6	CYAN	LDB D
215	D7	CYAN	STB D
216	D8	CYAN	EORB D
217	D9	CYAN	ADCB D
218	DA	CYAN	ORB D
219	DB	CYAN	ADDB D
220	DC	CYAN	LDD D
221	DD	CYAN	STD D
222	DE	CYAN	LDU D
223	DF	CYAN	STU D
224	E0	MAGENTA	SUBB ,
225	E1	MAGENTA	CMPB ,
226	E2	MAGENTA	SBCB ,
227	E3	MAGENTA	ADDD ,
228	E4	MAGENTA	ANDB ,
229	E5	MAGENTA	BITB ,
230	E6	MAGENTA	LDB ,
231	E7	MAGENTA	STB ,
232	E8	MAGENTA	EORB ,
233	E9	MAGENTA	ADCB ,
234	EA	MAGENTA	ORB ,
235	EB	MAGENTA	ADDB ,
236	EC	MAGENTA	LDD ,
237	ED	MAGENTA	STD ,
238	EE	MAGENTA	LDU ,
239	EF	MAGENTA	STU ,
240	F0	ORANGE	SUBB E
241	F1	ORANGE	CMPB E
242	F2	ORANGE	SBCB E
243	F3	ORANGE	ADDD E
244	F4	ORANGE	ANDB E
245	F5	ORANGE	BITB E
246	F6	ORANGE	LDB E
247	F7	ORANGE	STB E
248	F8	ORANGE	EORB E
249	F9	ORANGE	ADCB E
250	FA	ORANGE	ORB E
251	FB	ORANGE	ADDB E
252	FC	ORANGE	LDD E
253	FD	ORANGE	STD E
254	FE	ORANGE	LDU E
255	FF	ORANGE	STU E

Decimal	Hex	6809
4129	1021	LBRN R
4130	1022	LBHI R
4131	1023	LBLS R
4132	1024	LBHS/LBCC R
4133	1025	LBCS/LBLO R
4134	1028	LBNE R
4135	1027	LBEQ R
4136	1028	LBVC R
4137	1029	LBVS R
4138	102A	LBPL R
4139	102B	LBMI R
4140	102C	LBGE R
4141	102D	LBLT R
4142	102E	LBGT R
4143	102F	LBLE R
4159	103F	SWI/2 I
4227	1083	CMPD #
4236	108C	CMPY #
4238	108E	LDY #
4243	1093	CMPD D
4252	109C	CMPY D
4254	109E	LDY D
4255	109F	STY D
4259	10A3	CMPD ,
4268	10AC	CMPY ,
4270	10AE	LDY ,
4271	10AF	STY ,
4275	10B3	CMPD E
4284	10BC	CMPY E
4286	10BE	LDY E
4287	10BF	STY E
4302	10CE	LDS #
4318	10DE	LDS D
4319	10DF	STS D
4334	10EE	LDS ,
4335	10EF	STS ,
4350	10FE	LDS E
4351	10FF	STS E
4415	113F	SWI/3 I
4483	1183	CMPU #
4492	118C	CMPS #
4499	1193	CMPU D
4508	119C	CMPS D
4515	11A3	CMPU ,
4524	11AC	CMPS ,
4531	11B3	CMPU E
4540	11BC	CMPS E

6809 COLUMN

I = INHERENT
\# = IMMEDIATE
D = DIRECT
E = EXTENDED
R = RELATIVE
, = INDEXED

KEYBOARD

\# = SHIFT

Advertiser's Index

MICRO INK is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.

Next Month in MICRO

February: Language Feature

- **APPLE Pascal Hi-Res Screen Dump —** Use this high-resolution graphics dump to send APPLE Pascal Turtlegraphics to an Epson printer with Graftrax.
- **APL on the SuperPET—** APL offers powerful features, high execution speeds, and a cryptic character set. This article discusses APL's history and advantages, with specific reference to the Waterloo version on the SuperPET.
- **Parameter Passing in Assembly Language —** The author describes various methods for passing parameters to and from assembly-language programs. The Motorola 6502, 6809, 68000, and National Semiconductor 16032 are emphasized.
- **EDIT: A FORTH Screen-Oriented Editor —** EDIT uses the Atari 800 display as a text window into a FORTH disk screen and allows full use of the Atari special function keys to prepare FORTH applications.

Plus...

FORTH for the 6809
LISP for the APPLE
OSI PROM BASIC
IEEE Control with Logical Files on the PET
OSI Renumber BASIC
Applesoft BASIC Routine for CAI

Columns

APPLE Slices
PET Vet
From Here to ATARI
CoCo Bits

Departments

Reviews in Brief
Software and Hardware Catalogs
New Publications
FORTH Data Sheet

Model Rocket Simulation in BASIC

by David Eagle

This article describes a program to determine the altitude performance of single-stage model rockets, including burnout conditions, flight time, and maximum altitude of a model rocket.

ROCKET1
requires:
BASIC

ROCKET1 solves the problem of vertical model rocket motion by using several assumptions that allow the equation of motion to be solved exactly or analytically. These assumptions involve the boost phase of flight where an average thrust and average model rocket mass are assumed. The atmospheric density and drag coefficient are also assumed to be constant during the entire model rocket flight. ROCKET1 also compensates for non-standard launch sites that are not at sea level and launchings on hot or cold days.

User Inputs and Selections

ROCKET1 will prompt the user for the necessary inputs. A description of these requests and a discussion of how the user should respond follows. Information that pertains to the model rocket engine characteristics is available from manufacturers' catalogs.

LAUNCH SITE ALTITUDE (METERS)?
The user responds with the altitude of the launch site relative to sea level. This altitude is input in meters and is positive for sites above sea level and negative for sites below sea level.

LAUNCH SITE TEMPERATURE (DEG F)?
The user responds with the temperature at the launch site in decimal degrees Fahrenheit.

THRUST DURATION (SECONDS)?
The user inputs the total thrust duration of the model rocket engine in seconds.

TOTAL IMPULSE (NEWTON-SECONDS)?
The user responds with the total impulse of the model rocket engine in the units of newton-seconds.

INITIAL MASS (GRAMS)?
The user inputs the lift-off or gross mass of the entire model rocket in grams.

PROPELLANT MASS (GRAMS)?
The user responds with the propellant mass of the model rocket engine in grams.

FRONTAL DIAMETER (MM)?
The user inputs the maximum body tube diameter of the model rocket in millimeters.

DRAG COEFFICIENT?
The user responds with the drag coefficient of the complete model rocket. This number is non-dimensional.

After the program has run it will prompt the user for another selection. A description of each prompt follows. The user responds with "Y" if he/she desires the particular selection, or "N" if not.

ANOTHER SELECTION (Y=YES, N=NO)?
The user responds with "N" to exit the program.

ANOTHER LAUNCH SITE (Y=YES, N=NO)?
The user responds with "Y" to compute the model rocket flight performance at another launch site.

ANOTHER ROCKET ENGINE (Y=YES, N=NO)?
The user responds with "Y" to compute a model rocket's flight performance with a different model rocket engine.

DIFFERENT MASS OR DRAG (Y=YES, N=NO)?

Sample Run

```
    PROGRAM ROCKET1

LAUNCH SITE ALTITUDE (METERS)? 0

LAUNCH SITE TEMPERATURE (DEG F)? 59

THRUST DURATION (SECONDS)? 1.2

TOTAL IMPULSE (NEWTON-SECONDS)? 5

INITIAL MASS (GRAMS)? 40

PROPELLANT MASS (GRAMS)? 8.33

FRONTAL DIAMETER (MM)? 18

DRAG COEFFICIENT? .321

    BURNOUT ALTITUDE (METERS)                74.0671213

    BURNOUT VELOCITY (METERS/SECOND)         119.35939

    COAST TIME (SECONDS)                     7.93162468

    TOTAL FLIGHT TIME (SECONDS)              9.13162468

    MAXIMUM ALTITUDE (METERS)                451.393595
    ANOTHER SELECTION (Y=YES, N=NO)?
```