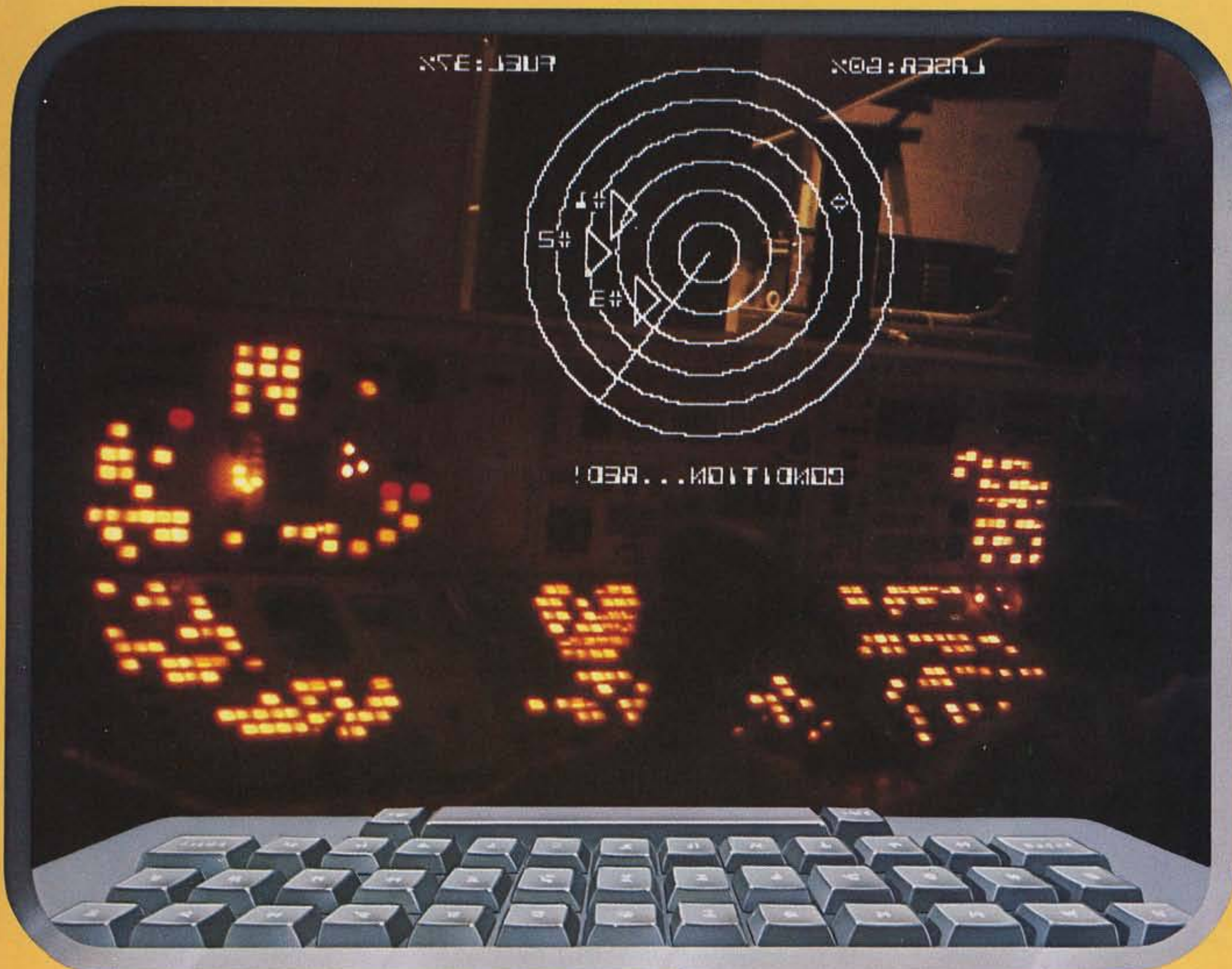
NO. 42 $2.50 NOVEMBER 1981

MICRO™

THE 6502/6809 JOURNAL



Special Games Feature

Auto Line Numbers for OSI Disk BASIC

Pascal Tutorial, Part 1

Apple II Digital Storage Oscilloscope

Turn your Apple into the world's most versatile personal computer.

The SoftCard™ Solution. SoftCard turns your Apple into two computers. A Z-80 and a 6502. By adding a Z-80 microprocessor and CP/M to your Apple, SoftCard turns your Apple into a CP/M based machine. That means you can access the single largest body of microcomputer software in existence. Two computers in one. And, the advantages of both.

Plug and go. The SoftCard system starts with a Z-80 based circuit card. Just plug it into any slot (except 0) of your Apple. No modifications required. SoftCard supports most of your Apple peripherals, and, in 6502-mode, your Apple is still your Apple.

CP/M for your Apple. You get CP/M on disk with the SoftCard package. It's a powerful and simple-to-use operating system. It supports more software than any other microcomputer operating system. And that's the key to the versatility of the SoftCard/Apple.

BASIC included. A powerful tool, BASIC-80 is included in the SoftCard package. Running under CP/M, ANSI Standard BASIC-80 is the most powerful microcomputer BASIC available. It includes extensive disk I/O statements, error trapping, integer variables, 16-digit precision, extensive EDIT commands and string functions, high and low-res Apple graphics, PRINT USING, CHAIN and COMMON, plus many additional commands. And, it's a BASIC you can compile with Microsoft's BASIC Compiler.

More languages. With SoftCard and CP/M, you can add Microsoft's ANSI Standard COBOL, and FORTRAN, or Basic Compiler and Assembly Language Development System. All, more powerful tools for your Apple.

Seeing is believing. See the SoftCard in operation at your Microsoft or Apple dealer. We think you'll agree that the SoftCard turns your Apple into the world's most versatile personal computer.

Complete information? It's at your dealer's now. Or, we'll send it to you and include a dealer list. Write us. Call us.

SoftCard is a trademark of Microsoft. Apple II and Apple II Plus are registered trademarks of Apple Computer. Z-80 is a registered trademark of Zilog, Inc. CP/M is a registered trademark of Digital Research, Inc.

MICROSOFT
CONSUMER PRODUCTS

Microsoft Consumer Products, 400 108th Ave. N.E., Bellevue, WA 98004. (206) 454-1315

JUDGE THE REST, THEN BUY THE BEST

Only GIMIX offers you **SOFTWARE SWITCHING** between **MICROWARE's OS-9** and **TSC's FLEX**. Plus you get the power of the GMXBUG system monitor with its advanced debugging utility, and memory manipulation routines. A wide variety of languages and other software is available for these two predominant 6809 Disk Operating Systems.

You can order a system to meet your needs, or select from the 6809 Systems featured below.

JUDGE THE FEATURES AND QUALITY OF GIMIX 6809 SYSTEMS

GIMIX' CLASSY CHASSIS™ is a heavyweight aluminum mainframe cabinet with back panel cutouts to conveniently connect your terminals, printers, drives, monitors, etc. A 3 position keyswitch lets you lock out the reset switch. The power supply features a ferro-resonant constant voltage transformer that supplies 8V at 30 amps, + 15V at 5 amps, and − 15V at 5 amps to insure against problems caused by adverse power input conditions. It supplies power for all the boards in a fully loaded system plus two 5 ¼'' drives (yes! even a Winchester) that can be installed in the cabinet. The Mother board has fifteen 50 pin and eight 30 pin slots to give you the most room for expansion of any SS50 system available. 11 standard baud rates from 75 to 38.4K are provided and the I/O section has its own extended addressing to permit the maximum memory address space to be used. The 2 Mhz 6809 CPU card has both a time of day clock with battery back-up and a 6840 programmable timer. It also contains 1K RAM, 4 PROM/ROM/RAM sockets, and provides for an optional 9511A or 9512 Arithmetic Processor. The RAM boards use high speed, low power STATIC memory that is fully compatible with any DMA technique. STATIC RAM requires no refresh timing, no wait states or clock stretching, and allows fast, reliable operation. The system includes a 2 port RS232 serial interface and cables. All GIMIX boards use gold plated bus connectors and are fully socketed. GIMIX designs, manufactures, and tests in-house its complete line of products. All boards are twice tested, and burned in electrically to insure reliability and freedom from infant mortality of component parts. All systems are assembled and then retested as a system after being configured to your specific order.

56KB 2MHZ 6809 SYSTEMS WITH GMXBUX/FLEX/OS-9 SOFTWARE SELECTABLE

With #58 single density disk controller **$2988.59**
With #68 DMA double density disk controller **$3248.49**
to substitute Non-volatile CMOS RAM with battery back-up, add 300.00
for 50 Hz export power supply models, add 30.00

Either controller can be used with any combination of 5'' and/or 8'' drives, up to 4 drives total, have data recovery circuits (data separators), and are designed to fully meet the timing requirements of the controller I.C.s.

5 ¼'' DRIVES INSTALLED IN THE ABOVE with all necessary cables

	SINGLE DENSITY		DOUBLE DENSITY		
	Formatted	**Unformatted**	**Formatted**	**Unformatted**	
40 track (48TPI) single sided	199,680	250,000	341,424	500,000	2 for **$700.00**
40 track (48TPI) double sided	399,360	500,000	718,848	1,000,000	2 for **900.00**
80 track (96TPI) single	404,480	500,000	728,064	1,000,000	2 for **900.00**
80 track (96TPI) double	808,960	1,000,000	1,456,128	2,000,000	2 for **1300.00**

Chart shows total capacity in Bytes for 2 drives.

Contact GIMIX for price and availability of 8'' floppy disk drives and cabinets; and 5'' and 8'' Winchester hard disk system.

128KB 2Mhz 6809 DMA Systems for use with TSC's UNIFLEX or MICROWARES's OS-9 Level 2

(Software and drives not included) **$3798.39**
to substitute 128KB CMOS RAM with battery back-up, add 600.00
for each additional 64KB NMOS STATIC RAM board, add 639.67
for each additional 64KB CMOS STATIC RAM board, add 988.64
for 50 Hz export power supply, add 30.00

NOTE: *UNIFLEX can not be used with 5'' minifloppy drives.*

GIMIX has a wide variety of RAM, ROM, Serial and Parallel I/O, Video, Graphics, and other SS50 bus cards that can be added now or in the future. Phone or write for more complete information and brochure.

THE SUN NEVER SETS ON GIMIX USERS

GIMIX Systems are found on every continent, except Antarctica. (Any users there? If so, please contact GIMIX so we can change this.) A representative group of GIMIX users includes: **Government Research and Scientific Organizations** in Australia, Canada, U.K., and in the U.S.; NASA, Oak Ridge, White Plains, Fermilab, Argonne, Scripps, Sloan Kettering, Los Alamos National Labs, AURA. **Universities:** Carleton, Waterloo, Royal Military College, in Canada; Trier in Germany; and in the U.S.; Stanford, SUNY, Harvard, UCSD, Mississippi, Georgia Tech. **Industrial users** in Hong Kong, Malaysia, South Africa, Germany, Sweden, and in the U.S.; GTE, Becton Dickinson, American Hoechst, Monsanto, Allied, Honeywell, Perkin Elmer, Johnson Controls, Associated Press, Aydin, Newkirk Electric, Revere Sugar, HI-G/AMS Controls, Chevron. **Computer mainframe and peripheral manufacturers,** IBM, OKI, Computer Peripherals Inc., Qume, Floating Point Systems. **Software houses;** Microware, T.S.C., Lucidata, Norpak, Talbot, Stylo Systems, AAA, HHH, Frank Hogg Labs, Epstein Associates, Softwest, Dynasoft, Research Resources U.K., Microworks, Analog Systems, Computerized Business Systems.

TO ORDER BY MAIL

SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear.

U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00.

Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S.. Our bank is the Continental Illinois National Bank of Chicago, account #73-32033 Visa or Master Charge also accepted.

GIMIX INC. reserves the right to change pricing and product specifications at any time without further notice.

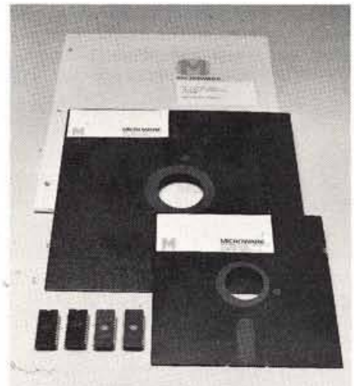
GIMIX Systems are chosen by the Pros because of quality, reliability and features.

GIMIX INC.

The Company that delivers
Quality Electronic products since 1975.

1337 WEST 37th PLACE, CHICAGO, IL 60609
(312) 927-5510 • TWX 910-221-4055

A TEAM OF 6809 SUPERSTARS:
Smoke Signal's Chieftain™ Computer, and Software by Microware



HERE'S THE *TOTAL* 6809-BASED SYSTEM FOR THOSE WHO DEMAND UNSURPASSED POWER, FLEXIBILITY AND RELIABILITY

After years of worldwide use in diverse and challenging applications, the outstanding performers in 6809 computer operations are SMOKE SIGNAL and MICROWARE. These leading companies are recognized as the undisputed choices **when there is no room for compromises**.

WHY SMOKE SIGNAL AND MICROWARE LEAD THE 6809 FIELD

Smoke Signal began pioneering research and development on 6800/6809-based computer systems back in 1977. Microware worked three years to perfect OS-9 and BASIC09.

Both companies have evolved outstanding 6809-based products from early engineering research, **and both pay almost fanatical attention to detail**. For example . . .

SMOKE SIGNAL'S 6809-based Chieftain™ computer series has **proven** its superiority in hundreds of demanding tasks. From gold-plated connectors to highest-quality materials throughout, each Chieftain™ is built to deliver absolute dependability from day one, and **stay** that way through years of service.

•ENDURANCE-CERTIFIED . . . an exclusive Smoke Signal quality-control measure that positively verifies a component is free of defects, and meets or exceeds all specifications.

Every Chieftain™ is meticulously ENDURANCE-CERTIFIED at 2.2 MHz. That's SMOKE SIGNAL's endorsement of product perfection.

MICROWARE's state-of-the-art OS-9 UNIX*-like operating system and the BASIC09 language have been developed in close coordination with computer manufacturers to maximize optimum system performance. The finest possible support and documentation further ensure satisfaction. Microware software performance is best summed up in this remark by a 25-year computer veteran:

"BASIC09 IS THE FINEST HIGH-LEVEL LANGUAGE I'VE EVER SEEN IN THE INDUSTRY!"

Thousands of engineers and programmers use MICROWARE software products as their standard time-saving tool . . . to execute process-control applications . . . and for other vital functions. COBOL and PASCAL are also available under the OS-9 operating system.

HOW THIS REMARKABLE TEAM OF COMPUTER SUPERSTARS CAN SERVE *YOU*

SMOKE SIGNAL's Chieftain™ computer provides an array of configurations ranging from 5¼-inch drives for single-user applications to multi-user, multi-tasking capabilities. Winchester hard-disk drive systems are also available.

In other words, **breathtaking power** with as little as 48k memory; Microware's OS-9 Level Two can access up to one full megabyte that your Chieftain™ can address!

One more sampling of the awesome processing potential at your fingertips with the Smoke Signal Chieftain™ computer:

MICROWARE'S Stylograph screen-oriented word processing package instantly makes Chieftain™ an easy-to-use document preparation system with comprehensive editing commands.

THERE'S MUCH, MUCH MORE! Call or write SMOKE SIGNAL for details on Chieftain™ computers and MICROWARE software.

SMOKE SIGNAL Dealer opportunities are still available . . . please request information.

☐ Send information about Chieftain™ computers and Microware software.

☐ Provide information about Smoke Signal's Dealer program.

Name ______________________

Address ______________________

City ______________ State ________ Zip ________

Telephone (   ) ______________________

*UNIX is a trademark of Bell Telephone Laboratories.

SMOKE SIGNAL BROADCASTING | MICROWARE

31336 VIA COLINAS
WESTLAKE VILLAGE, CA 91362
TEL (213) 889-9340

MICRO™

THE 6502/6809 JOURNAL

DEPARTMENTS

TUTORIAL

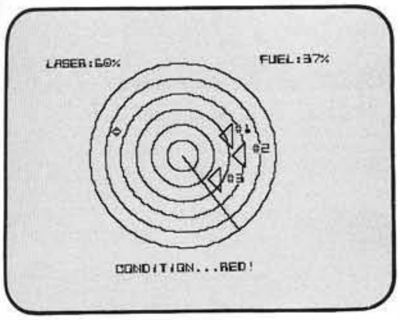
PROGRAMMING AIDS

GAMES

HARDWARE/SYSTEMS

APPLE

About the Cover



This month's cover depicts an exciting space-game scenerio. The games feature in this issue describes several space games, including Lunar Lander, Saucer Launch, and Galacticube. While MICRO rarely publishes games of any sort, we felt that it was time to make an exception. So we assembled a variety of game articles, and turned them into a feature section which you should find not only challenging but informative. Charge up those lasers... they're coming in.

The cover picture was taken inside a NASA simulator at Kennedy Space Center.

The cover graphic was generated on an Apple II, and output was provided by Computerland of Nashua.

(Cover photo by Ford Cavallari)

MICRO is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and Avon, MA 02322
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
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515

Subscription rates	*Per Year*
U.S.	$18.00
Foreign surface mail	$21.00
Air mail:	
Europe	$36.00
Mexico, Central America	$39.00
Middle East, North Africa	$42.00
South America, Central Africa	$51.00
South Africa, Far East, Australasia	$60.00

MICRO Editorial

Games, Games, Games ...

Long-time readers are probably surprised to find *games* in MICRO. Our long-standing editorial policy has been to limit games, unless they had "social redeeming value," since they may be found in many other magazines or may be purchased directly. We are relaxing our policy for this issue because we have not had any games for so long, and because the holiday season is approaching. So, have fun.

On the serious side of games, I still believe that too much time, effort and interest is being spent on them, to the detriment of other software developments. While it is the perogative of computerists to play and write games, MICRO does not want to emphasize this single aspect of the microcomputer. The games problem has grown into a significant social issue in recent months. The computer game, as used in the game arcades, has attracted large numbers of adolescents and is increasingly coming under public scrutiny and concern. A number of computer game arcades in our area have been required to limit their hours, to prevent children from playing during school hours. Other regulations are being contemplated.

If you have any ideas or comments on the game aspects of microcomputers, we would welcome letters to the Editor about this topic.

Up, Up, and Away ...

There is no way to escape the realities of inflation. Since MICRO was first published in 1977, at a subscription rate of $1.00 per issue, there has been a tremendous cost increase in almost every area of operation. Over several years, MICRO has doubled in size and increased its subscription price by 50% to the current $1.50 per issue. During the past year or two, the size of the staff has tripled, the cost of postage has gone up almost monthly, the price of paper is out-of-sight, the size of the magazine has increased 60%, and there are the general inflation effects. We have, therefore, decided to raise the subscription rate effective in January to $24.00 per year in the US, with adjustments in the foreign rates as well. To help make the increase less abrupt, we are accepting new subscriptions and renewals through the end of December 1981 at the current rates. There will also be a new two-year rate for US subscriptions ($42.00) which will help keep the cost down. The one-year subscriber will save 20% over the single issue price and the two-year subscriber will save 30%.

An Informal Computer Page

Since so much of MICRO is devoted to the serious side of microcomputing, we would like to balance this with a page of informal material in each issue. This would consist of cartoons, jokes, computer trivia, puzzles, jokes, limericks, bloopers, computer mishaps, strange computer photos, interesting computer graphics, and so on. This section will depend on you for input. There will be no payment for material submitted for this page, but you will be given full credit for your material. As a small incentive to start thinking about the informal side of computing, MICRO will offer a free one-year subscription to the best of suggestion of a title for this page. Entries must be received by the end of December 1981, and all decisions of the MICRO staff will be final.

MICRO Books

In addition to our monthly magazine, MICRO is interested in publishing relevant books. The *Best of MICRO* series which presented reprints from the early issues of MICRO (vol. 1: issues 1 to 6; vol. 2: issues 7 to 12; and vol 3: issues 13 to 24) indicated that there was a continuing interest in the fundamental material that was being printed in MICRO. Our first two specialized books, *MICRO on the Apple, Volume 1* and *What's Where in the Apple*, have met with great success. *MICRO on the Apple, Volume 2* is presently scheduled for publication in December 1981, with Volume 3 scheduled for mid 1981.

We are considering a number of other book projects, and welcome your suggestions. If you have a manuscript in mind, or in process, that you think would appeal to the MICRO readership, please contact us. We have a very good distribution network for 6502-and 6809-related materials, and a knowledgeable staff to assist in text preparation.

Robert M. Tripp

Precision Programming

Writing a structured program requires discipline on the part of the programmer. While a procedure-originated language will make the task easier, it is possible to write a structured program using BASIC.

Al Hamilton
12090 Brookston Drive
Springdale, Ohio 45240

Precision Programming

The real objective in programming should be to write correct programs from the start — not merely to emerge from debugging with no errors. Writing such correct programs from the start is a very possible human activity.

With the advent of compilers and other debugging aids, it has been easy to adopt an attitude of "let the compiler do it" in finding errors of syntax. But in the long run, this is a devastating attitude because it fosters ignorance and carelessness that slides to program logic that the compiler cannot uncover.

If your programming is a vocation rather than an avocation, there is no reason for you to take errors of syntax lightly. Syntax errors are either of ignorance or carelessness.

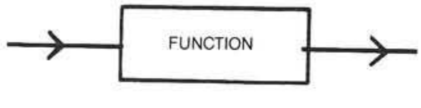
A professional writer of English, or even a well-educated non-professional, has little trouble in writing complete sentences or remembering to end sentences with a period. Writing with syntatic precision is a simple necessity and practically an unconscious skill for any competent programmer. True enough, the compiler will find syntax errors. However, there are many times when a syntax error will be reinterpreted as a correct syntax for another statement so that a logic error results of which neither the programmer or the compiler is aware.

Writing correct syntax is like playing a perfect game of Tic-Tac-Toe, not like sawing a board exactly in half. It is a combinatorial process which requires only a fixed and humanly possible degree of precision for correctness. For example, a complicated expression may end with five (or six) parentheses; but it will never end with 5.37521... parentheses. The difference between five and six is distinguishable in writing and reading, and whether it should be five or six depends only on previous characters of discrete kinds and locations in the expression.

The problem of writing correct program logic is more difficult than that of writing correct syntax. Most of this article is about writing correct logic. The reason for beginning with syntax errors is to identify an attitude of precision which will carry over with good effect into the problem of program logic.

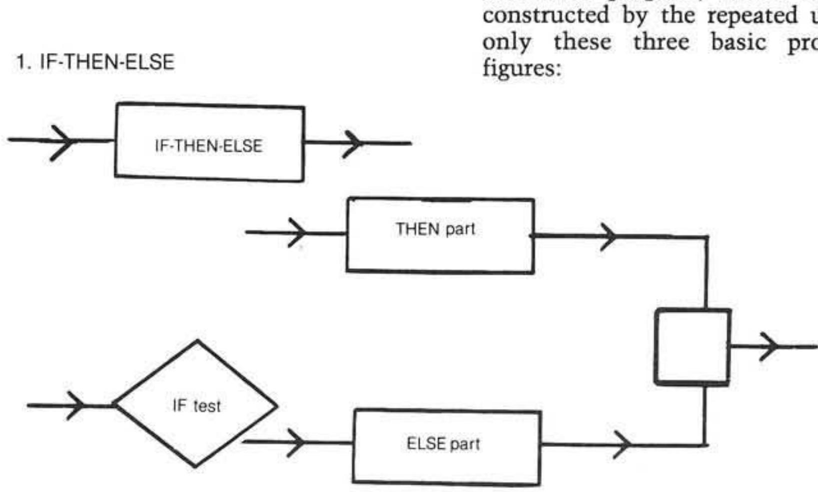
You can write programs with correct function logic by using principles of structured programming and program correctness which are applied in your line-by-line program construction. A programmer begins with a functional specification which describes what the program is to do. In his mind he converts that specification into program statements and then verifies that the statements created in fact do what the program was intended to do. In structured programming there is a precise description of this mental activity. It begins with the functional specification and repeatedly dissects it, a step at a time, into new functional subspecifications connected by program statements until the program is complete. It does not consist of a large leap in faith from a functional specification to loose collection of program statements which are fitted piece-by-piece into a program. The structured programming process analyzes functional specifications rather than synthesizing program statements. One brief way of understanding structured programming and how to prove the correctness of programs written in this way is this:

A. Any functional specification can be defined in terms of a mathematical function which maps inputs into outputs without regard to its internal construction. We show such a function (functional specification) as

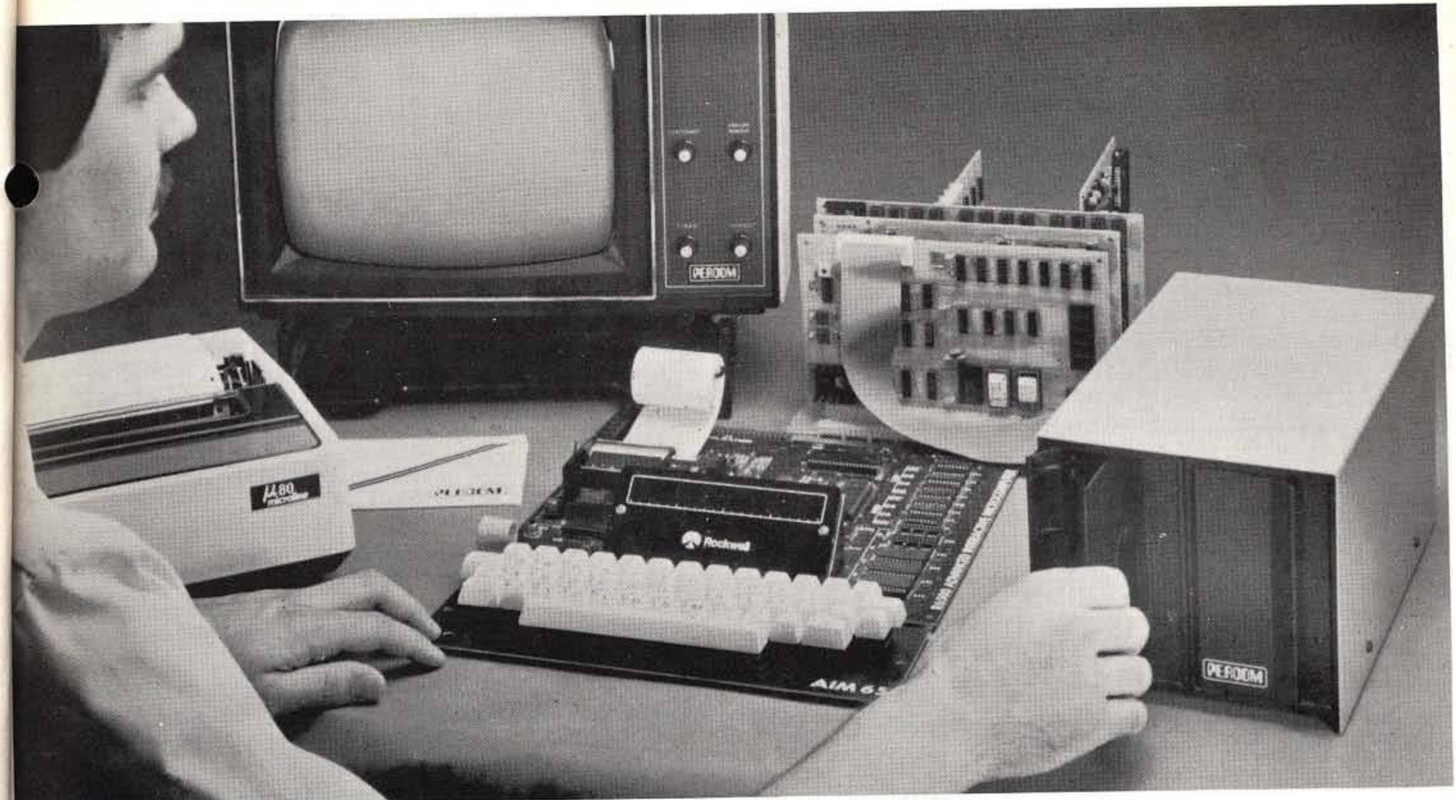


B. Any flowchartable program used to realize a function is equivalent to a structured program, which can be constructed by the repeated use of only these three basic program figures:

1. IF-THEN-ELSE



(Continued)



Introducing the M line . . .

Now! Drive Systems for AIM, KIM and SYM Computers – from PERCOM.

Assembled and tested systems start at only $599.95, including the drive controller circuit card, disk-operating system, interconnecting cable, drive and comprehensive users manual.

- **The right storage capacity** – Available in 1-, 2- and 3-drive systems, with either 40- or 80-track drives.
- **Flippy storage** – Flippy drives (optional) let you flip a diskette and store data and programs on the second recording surface.
- **High Storage Capacity** – Formatted, one-side storage capacity is 102 Kbytes (40-track drive), 205 Kbytes (80-track drive).
- **Proven Controller** – The drive controller design is the same as the design used in the Percom 680X LFD mini-disk system. This system – introduced in 1977 – has given reliable service in thousands of applications. Two versions are available: the MFD-C65 for the AIM-65 expansion bus, and the MFD-C50 for the System-50 (SS-50) bus.
  - Includes an explicit data separator circuit that's reliable even at the highest bit densities.
  - Provides for on-card firmware.
  - Includes a motor inactivity time-out circuit.
  - Capable of handling up to four drives.
  - Capable of reading both hard- and soft-sectored diskettes.
- **DOS included** – The MFD disk-operating system works with the AIM monitor, editor, assembler, Basic and PL/65 programs; interface is direct, through user I/O and F1, F2 keys.
- **Reliability assurance** – Drives are burned-in 48 hours, under operating conditions, to flag and remove any units with latent defects.
- **Full documentation** – Comprehensive hardware and software manuals are included with each system. These manuals cover details from design to operation and applications.

Now! Expand your AIM-65 with Low-cost System-50 Modules.

The Percom M65/50 Interface Adapter connects your M-65 bus to Percom's System-50 (SS-50) motherboard, allowing you to expand your AIM, KIM or SYM with proven System-50 modules. You can add disk storage, memory modules,even a video display system. The M65/50 provides buffer-amplification of address, data and control lines. On-card decode circuitry lets you allocate address space either to the computer or to the expansion motherboard. Price: only $89.95.

System Requirements: AIM-65, KIM or SYM computer with expansion bus and four Kbytes RAM (min).

Yes . . . I'd like to know more about Percom MFD drive systems. Rush me free literature.

Send to
PERCOM DATA COMPANY, Inc., Dept. 65-M
11220 Pagemill Rd. Dallas, TX 75243

name ____________________
address ____________________
city ____________ state ____________
zip ____________ phone number ____________

MAIL TODAY!

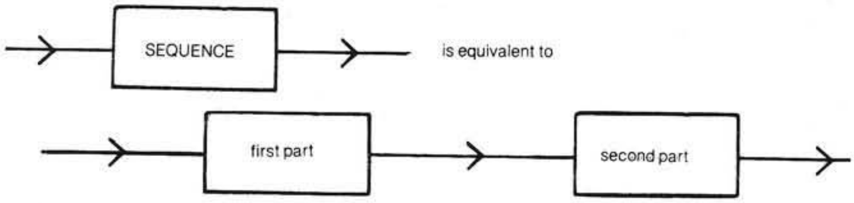
PERCOM

PERCOM DATA COMPANY, INC.
11220 PAGEMILL RD. DALLAS, TX 75243
(214) 340-7081

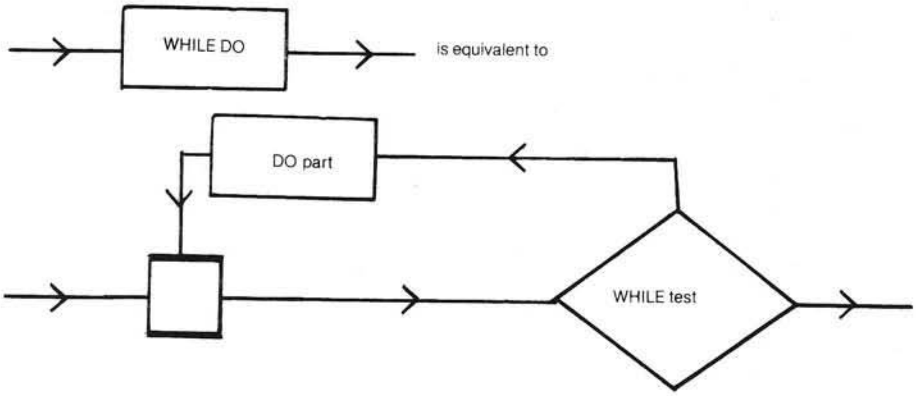
Toll-Free Order Number: **1-800-527-1222**

PRICES AND SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE.

2. SEQUENCE



3. WHILE DO



Each THEN-part, ELSE-part and DO-part is just a new function and can be replaced by another IF-THEN-ELSE, SEQUENCE or DO-WHILE figure in a subsequent expansion step.

The structured program construction process proceeds from an original functional specification as a series of decisions, which specify which figure and what resulting new tests and functions are required to expand the original and any intermediate functions required. When the functions required can be written directly as program statements, the expansion process is complete.

In a high level procedure-originated language such as Pascal and PL/I, these expressions can be written directly in matching statements. The relationship between program text and execution thus becomes especially clear. In a non-procedure-originated programming language like BASIC, the programmer requires more discipline to maintain proper programming structure. An example of how a proper program can be written in BASIC will be presented later.

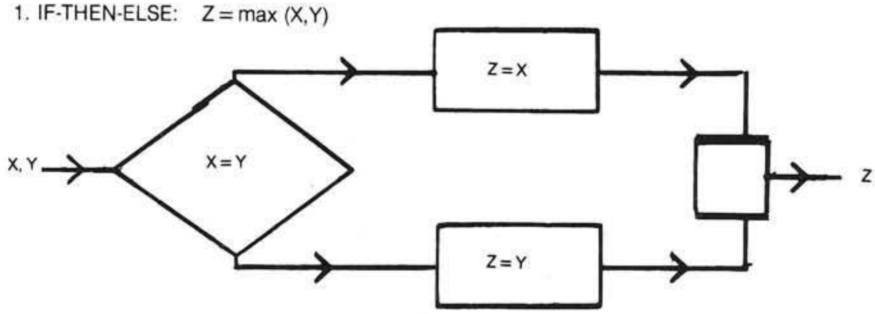
C. At each expansion step, the correctness of that step can be decided by answering a standard question that goes with that type of expansion. If the answer is yes, the step is correct and the program expansion can proceed. If the answer is no, the step is not correct and a new one should be defined. The questions are:

1. IF-THEN-ELSE — whenever the IF-test is true, does the THEN-part do the IF-THEN-ELSE; and whenever the IF-test is false, does the ELSE-part perform the IF-THEN-ELSE?
2. SEQUENCE — does the first-part followed by the second-part perform the sequence?
3. WHILE-DO — whenever the WHILE-test is true, does the DO-part followed by the WHILE-test perform the WHILE-DO; and whenever the WHILE-test is false, does the identity function (no-op program) perform the WHILE-DO?

The question for the IF-THEN-ELSE and sequence expansions are self-evident. The question for the WHILE-DO becomes self-evident by observing this sequence of equivalent expansions: expand the execution of the WHILE-DO into an IF-THEN-ELSE, and observe that the WHILE-DO reappears as the second-part of the sequence making up the THEN-part; the ELSE-part is the identity.

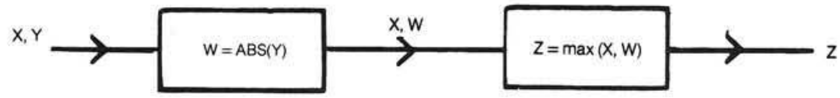
D. When steps 2 and 3 are carried out to the point where no subspecifications remain, the result is a complete program and the proof of its correctness has been completed as well. Some illustrations of individual steps with their correctness questions are:

1. IF-THEN-ELSE: Z = max (X,Y)



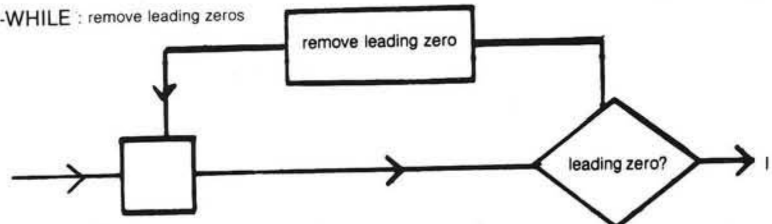
Whenever X >= Y, does Z = X perform Z = max (X, Y); and whenever X < Y, does Z = Y perform Z = max (X, Y)?

2. SEQUENCE: Z = max (X, ABS (Y))



Does W = ABS (Y), followed by Z = max (X, W), perform Z = max (X, ABS (Y))?

3. DO-WHILE : remove leading zeros



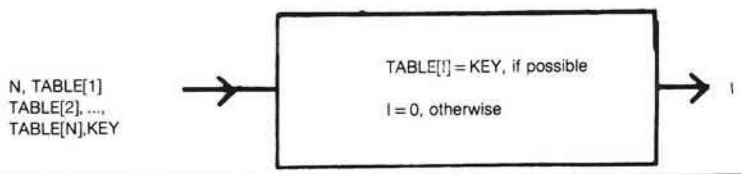
Whenever there is a leading zero, does "leading zero" followed by "remove leading zero" perform "remove leading zeros"; and whenever there is no leading zero, does doing nothing perform "remove leading zeros"?

In order to see these principles in action, consider the problem of searching for an item called "KEY" in a list called "TABLE", with a total of "N" elements, denoted TABLE[1], TABLE[2], ..., TABLE[N], respectively; we are to display the results of the search as an item called "I", which is to satisfy the relation:

"Sequence Question"

TABLE[I]: = KEY, if possible
I: = 0, otherwise.

Note we have defined a function in words. The argument is N+2 items namely "N", TABLE[1], TABLE[2], ..., TABLE[N], "KEY" and the value is "I", as diagrammed.



It is easy to invent a program for this function.

```
function search1(n:integer;key:real;
table:array [1..n] of real):integer;

 var i,j: integer;

 begin
  i: = 0;
  for j: = 1 to n do
   if table[j] = key
    then i: = j;

  search1: = i
 end;
```

It is not an efficient program, to be sure, but it seems to be correct. Why? First, it is a sequence of two subprograms whose functions are:

A. First-part: Set I to zero

B. Second-part: Find, if possible, a value for I for which TABLE[I] = KEY; otherwise, leave I unchanged.

The sequence question (see figure above) asks if the first-part followed by the second-part does the SEQUENCE. We believe so. The second-part above is itself a loop, but not a WHILE-DO loop. Instead it is the familiar indexed loop, which we will call a FOR-loop for short. It is worth our attention as an extra control beyond the three basic ones given above. This extra control is that the index of the FOR-loop is not altered in any way by the body, or DO-part of the FOR-loop. Then the FOR-loop becomes an extended sequence, with a first-part, second-part, third-part, ..., *nth*-part. The corresponding correctness question is a simple extension of the sequence question as well. The DO-part in this case is:

DO-part: IF TABLE[J] = KEY then set I to J,
otherwise leave I unchanged

And it is easy to see that the sequence of such DO-parts, for J = 1, J = 2, ..., J = N indeed does the FOR-loop (second-part above). Finally, the DO-part, is itself an IF-THEN (IF-THEN-ELSE with null ELSE) figure, and it is easy to see that it satisfies its functional requirements.

The SEARCH1 algorithm could be improved by the following logic. Note that this is not a valid Pascal program:

```
function search2;

 begin
  i: = 0;
  for j: = 1 to n while(i = 0) do
  (not valid Pascal logic)
   if table[j] = key
   then i: = j
 end;
```

Whereas SEARCH1 looked at every item in the table, this algorithm would stop looking after the first success in 'table.' Unfortunately, the FOR-WHILE construction is not valid in Pascal. Yet the effect of this conditional termination loop can be realized as written in SEARCH3:

```
function search3(n: integer;key:real;
table:array [1..n] of real):integer;

 var i,j: integer;

 begin
  i: = 0;
  j: = 1;
  while((j = n)and(i = 0)) do begin;
   if table [j] = key then
    i: = j;
   j: = j + 1
  end;

  search3: = i
 end;
```

Here, the FOR-WHILE loop becomes a sequence of a first-part for initialization and a second-part of WHILE-DO whose do-part includes incrementing the index. In this form the WHILE-DO question applies — it asks:

whenever J< = N and I = 0, does the DO-part followed by the WHILE-DO perform the WHILE-DO;

and:

whenever J > N or I< > 0, does doing nothing perform the WHILE-DO?

We can see that it does. If the KEY has not yet been found in the TABLE, and we have not looked at every item, then we can look at the next item and set I, J accordingly and still complete the task required of the WHILE-DO.

For performance the WHILE-DO should be made as small as possible:

```
function search4(n:integer;key:real;
table:array [1..n] of real):integer;

 var i: integer;

 begin
  i: = 1;
  while((table[i]< > key)and
   (i< = n))do i: = i + 1;
  if i  n then i: = 0;

  search4: = i
 end;
```

The IF-THEN-ELSE has been moved from within the WHILE-DO to a sequence following. In this form the WHILE-DO question asks:

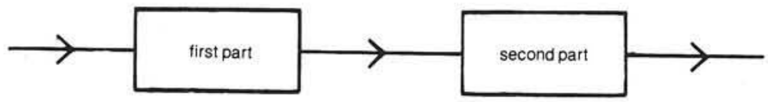
whenever TABLE[I]< >KEY and I< =N, does the DO-part (in this case i=i+1) followed by the WHILE perform the WHILE-DO;

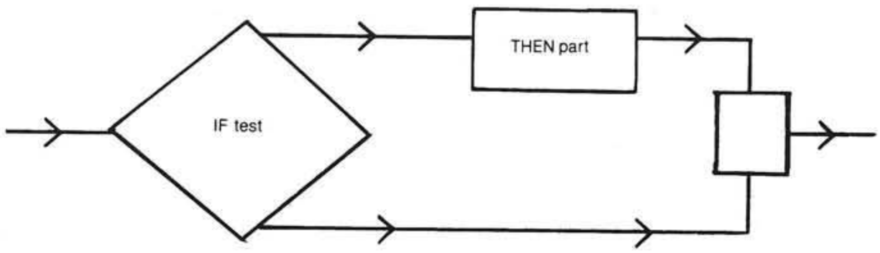
and:

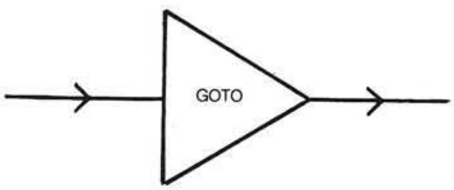
whenever TABLE[I] = KEY or I N, does doing nothing perform the DO-WHILE?

Now we have a single execution of the IF-THEN-ELSE to set I to 0 if no value of TABLE(I) was equal to KEY.

The programmer using BASIC does not have all three elements implemented. The elements available are the sequence and a degenerate IF-THEN-ELSE so that the discipline of coding the basic elements is added to the task of writing a proper program. We are therefore forced to use another construct with the two elements provided to build the three basic elements. The three constructs we start with are:



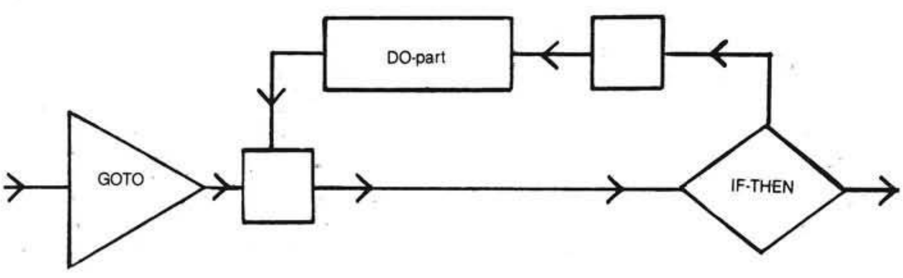




The program structure for the IF-THEN-ELSE and the WHILE-DO are:

A. IF-THEN-ELSE
  IF-test THEN GO-TO first-part
  second-part
  GO-TO collector
  first-part
  collector

B. WHILE-DO
  GO-TO WHILE-test
  collector
  DO-part
  IF-test THEN GO-TO collector



When the IF-THEN-ELSE has a null ELSE the following can be used:

C. IF-THEN
  IF-test THEN GO-TO collector
  THEN part
  collector

Another element that can be used instead of the WHILE-DO is the REPEAT-UNTIL:

D. REPEAT-UNTIL
  IF-test THEN GO-TO collector
  DO-part
  GO-TO REPEAT-UNTIL-test
  collector

The following sample programs will use Newton's method of successive approximations to find the square root of numbers obtained from a file. There are two WHILE-DOs, one WHILE not-end-of-file and one WHILE last approximation is not equal to the new approximation. Also count the input data items and display the count or "no input" at end of job.

Notice that the READLN (or INPUT) statement is at the bottom of the WHILE-DO and there is a priming READLN (or INPUT) statement in the intitializing phase of the program. The program flow is process, output, input, in that order (not input, process, output as in the non-structured programming approach.)

The syntax and function of a well-designed procedure-originated language can allow the programmer to code a program that reads like the functional specifications of the program with no need for remarks or comments.

The challenge to the BASIC programmer is greater.

Al Hamilton graduated from the University of Cincinnati as an Electrical Engineer. He has been programming in PL/I and Assembler for twelve years. Since obtaining an Apple II in June of 1979 he has been applying structured programming techniques to programming in BASIC.

(Continued on next page)

```
(**************************)
(*                        *)
(* Precision Programming  *)
(*       Hamilton         *)
(*                        *)
(*   listing 1: Pascal    *)
(*                        *)
(**************************)

program sample(infile);

   var infile: text;             (* input file *)
       number: real;             (* input number *)
       approximation: real;      (* square root approximation *)
       squareroot,square: real;  (* square root, square of number *)
       count: integer;           (* a counter *)


   begin

     count := 0;

     reset(infile,'*reals.data');
     readln(infile,number);

     while not eof(infile) do begin

        count := count+1;
        approximation := number/2;
        squareroot := (number/approximation+approximation)/2;

        while (squareroot<>approximation) do begin
           approximation := squareroot;
           squareroot := (number/approximation+approximation)/2
        end;

        square := number*number;

        writeln(number,square,squareroot);
        readln(infile,number)
     end;

     if count=0
     then writeln('No Input')
     else begin
        writeln('Count:',count);
        writeln('Successful end of job.')
     end;

     close(infile)

   end.
```

```
1  REM  **************************
2  REM  *                        *
3  REM  * PRECISION PROGRAMMING  *
4  REM  *       HAMILTON         *
5  REM  *                        *
6  REM  *   LISTING 2: BASIC     *
7  REM  *                        *
8  REM  **************************
9  REM


10 COUNT = 0
20 EOF = 0
30  ONERR  GOTO 50
40  GOTO 70
50 EOF = 1
60  GOTO 210
70  PRINT  CHR$ (4);
              "OPEN INFILE"
80  PRINT  CHR$ (4);
              "READ INFILE"
90  INPUT NUMBER
100  GOTO 210
110 COUNT = COUNT + 1
120 GUESS = NUMBER / 2
130 ROOT = (NUMBER /
          GUESS + GUESS) / 2
140  GOTO 170
150 GUESS = ROOT
160 ROOT = (NUMBER /
          GUESS + GUESS) / 2

170  IF ROOT <  > GUESS THEN 150
180 SQUARE = NUMBER * NUMBER
190  PRINT NUMBER,SQUARE,ROOT
200  INPUT NUMBER
210  IF EOF <  > 1 THEN  GOTO 110
220  IF COUNT = 0 THEN  GOTO 260
230  PRINT "COUNT: ";COUNT
240  PRINT "SUCCESSFUL EOJ"
250  GOTO 270
260  PRINT "NO INPUT"
270  PRINT  CHR$ (4);
          "CLOSE INFILE"

280  END
```

MICRO

Submitting an article to MICRO?

The following tips will help both you and our staff:

Text

- Send typewritten, double-spaced copy.
- Put your name and address on the first page of the article, and name on each page.
- Provide a summary of the article, and a brief biographical note.
- Use tables and figures — they're not only effective, but add visual interest to your article.

Listings

Make sure *printed* listings are:

- machine generated
- black ink on white paper
- commented
- thoroughly tested

We'd appreciate listings on *magnetic media.* Please provide:

- BASIC and binary files
- assembler source files
- specifications and loading instructions

If you have topic ideas you'd like to discuss, please call and talk with one of our editors. We look forward to receiving your manuscripts!

WHY THE MICROSOFT RAMCARD™ MAKES OUR SOFTCARD™ AN EVEN BETTER IDEA.

Memory — you never seem to have quite enough of it.

But if you're one of the thousands of Apple owners using the SoftCard, there's an economical new way to expand your memory dramatically.

16K ON A PLUG-IN CARD.

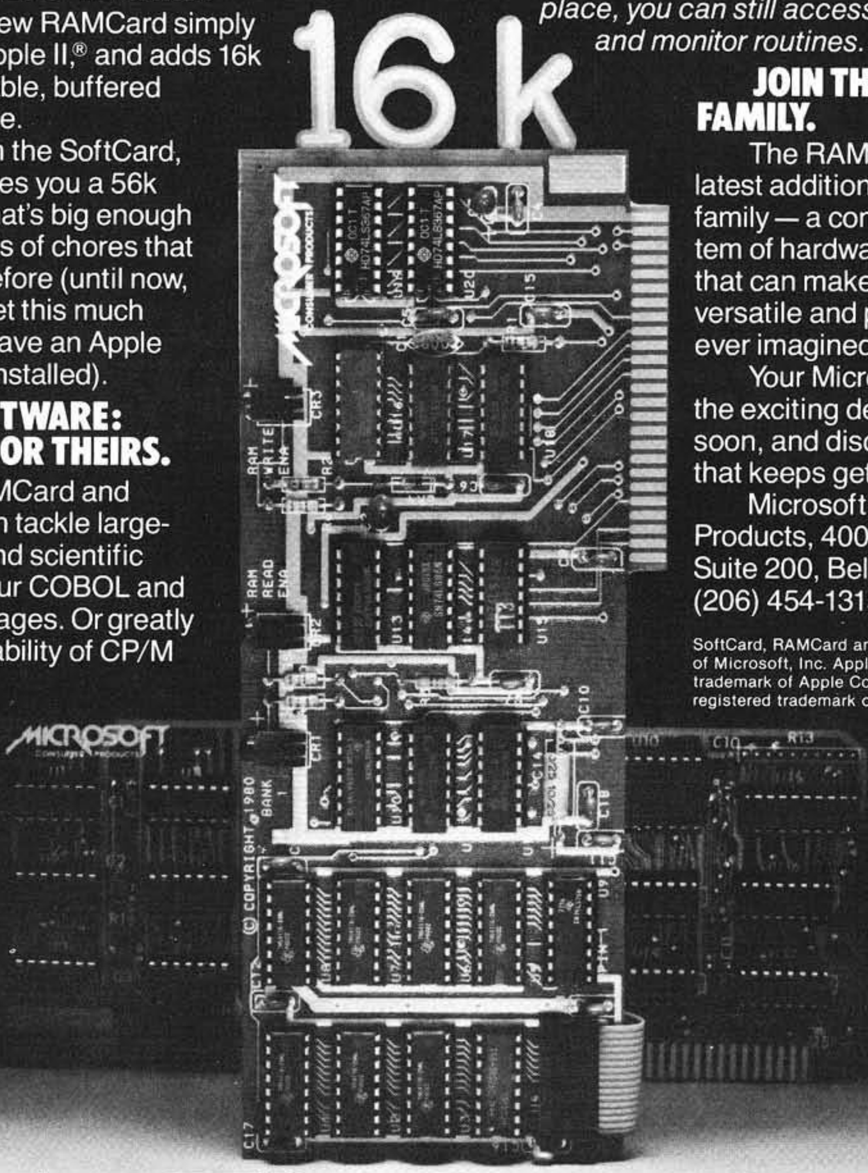
Microsoft's new RAMCard simply plugs into your Apple II,® and adds 16k bytes of dependable, buffered read/write storage.

Together with the SoftCard, the RAMCard gives you a 56k CP/M® system that's big enough to take on all kinds of chores that would never fit before (until now, the only way to get this much memory was to have an Apple Language Card installed).

GREAT SOFTWARE: YOURS, OURS, OR THEIRS.

With the RAMCard and SoftCard, you can tackle large-scale business and scientific computing with our COBOL and FORTRAN languages. Or greatly increase the capability of CP/M applications like the Peachtree Software accounting systems. VisiCalc™ and other Apple software packages can take advantage of RAMCard too.

And RAMCard gives you the extra capacity to develop advanced programs of your own, using the SoftCard and CP/M. *Even with the RAMCard in place, you can still access your ROM BASIC and monitor routines.*



JOIN THE SOFTCARD FAMILY.

The RAMCard is just the latest addition to the SoftCard family — a comprehensive system of hardware and software that can make your Apple more versatile and powerful than you ever imagined.

Your Microsoft dealer has all the exciting details. Visit him soon, and discover a great idea that keeps getting better.

Microsoft Consumer Products, 400 108th Ave. N.E., Suite 200, Bellevue, WA 98004. (206) 454-1315.

MICROSOFT™

Pascal Tutorial

Part 1

Victor R. Fricke
325 Ramapo Valley Road
Mahwah, New Jersey 07430

Lesson 1: Getting Started

A short time ago I installed my Apple Language Card, and with great expectations embarked on a course of painful self-education. The descriptions that follow are for a single drive system, which is what I have. I have found it very difficult to use Pascal with only one drive.

The first-time user is faced with two problems. First, you must learn the operating system for UCSD Pascal as implemented on the Apple. Second, you have to learn the Pascal language. This lesson will help you get started on both problems.

The Apple Pascal reference manuals provided with the system contain a wealth of information, but in a form which assumes a lot of knowledge on the part of the reader. They are included with Version 1.1 of the Pascal package, and are much better than the old manual that came with the Version 1.0 package. One of the manuals concentrates on the language, and the other on the operating system.

The Operating System

The operating system allows the computer to do many things, including understanding Pascal programs. When you bootload Pascal according to the directions in the appendix to the language manual, you are, in effect, running a "HELLO" program which is the Pascal operating system. The screen looks like this:

```
COMMAND:E(DIT, R(UN, F(ILE, C(OMP, L(IN

WELCOME APPLE0, TO APPLE II PASCAL 1.1
BASED ON UCSD PASCAL II.1
CURRENT DATE IS 19-MAY-81

(C) APPLE COMPUTER INC. 1979,1980
(C) U.C. REGENTS 1979
```

At this point the Pascal system is operating at the "COMMAND" level. The system functions are grouped into levels, and "COMMAND" is the highest level. The line across the top of the screen is similar in function to the "Menu" you see in many BASIC programs.

The line at the top of the screen, called the "prompt line," is different for each level of the system. However, you do not see all of it. The prompt line is longer than the 40 characters of the Apple display. This fact does not mean that you will not be able to see the rest of the line. The Pascal system has a feature known as horizontal rollover. This lets you shift your field of view from left to right, to see the other half of the line. This is done by typing "Control-A."

When you type in Control-A, the effect is as if the whole screen of text moved off the left side, and the right half of the screen moved into view. As a result, you have 80-character lines available, even though only the left half or the right half is visible at any time.

If you are performing these actions on your computer as you read this article, try typing Ctrl-A. You should now see the right half of the prompt line:

```
K, X(ECUTE, A(SSEM, D(EBUG,? [1.1]
```

Notice that the last option in the prompt line is a ? which is an indication that there are more options available than those shown. To see them, press the "?" key. If you have been following directions, you will see:

```
LT, S(WAP, M(AKE EXEC
```

You are looking at the right half of the screen. By flipping back and forth, you will be able to see that the entire prompt line reads:

```
COMMAND: U(SER RESTART, I(NITIALIZE,
H(ALT, S(WAP, (MAKE EXEC
```

I will not attempt to describe all the options available at the command level, just the more common ones. The following is a summary of the options covered in this part of the series. More options will be covered in subsequent parts.

Edit

The Edit command gets you into the system editor. Just press "E" in the "COMMAND" level. The editor is used for establishing the workfile, editing the workfile, or editing a text file. It is at the second level of the system structure, and has its own prompt line.

At this point, I'll offer a word of description about the workfile. The workfile is a block of information in the working memory space, used to hold the file currently being worked on. A scratch copy of the workfile is also kept on the disk. It is stored as a text file called

```
SYSTEM.WRK.TEXT
```

Also, when SYSTEM.WRK.TEXT is compiled, the result is a code file that is saved on the disk with the name:

```
SYSTEM.WRK.CODE
```

Run

The Run option lets you run a Pascal program which is on the disk in the drive. If the program has not been compiled, the system compiles it first. Then, if it compiles successfully, it runs the compiled program. If any errors in the program are detected during compilation, the erroneous line is displayed, with an error message.

The system makes several assumptions that you should know about if you want to run a program. First, if you have established a workfile, it assumes that the workfile is the program you want to run. Second, it assumes that the disk in the drive contains the following files:

```
SYSTEM.COMPILER
SYSTEM.LIBRARY
SYSTEM.LINKER
```

If you attempt to run a program without these files in the drive, you will get a message:

```
MUST LINK FIRST
```

File

The File option gets you into the filer subsystem. In the filer mode, you can list the disk directory, delete files from the disk, check for bad spots on the disk, rename a file or a disk volume, clear the workfile, and so forth.

Compile

On your old (pre-Language Card) Apple, there was a system program like there is for the Pascal system — the BASIC interpreter. It took your BASIC program, looked at it one line at a time, checked for syntax errors, and, if it found none, executed the resulting instructions.

Although you may not have noticed, the result was a much slower operating computer. If a particular line of BASIC happened to be in a loop which executed many times, the interpreter would check it for syntax errors each time before executing it.

Obviously, the computer wasted a lot of time re-checking instructions. A much more efficient system program would be one which looked at the program as a whole only once, checked each statement for syntax only once, and then translated the entire program into a block of machine code. This process is called compiling, and that is what the Pascal system program does.

Execute

The execute option, as you already may have guessed, executes a machine code file which has been previously compiled.

More on the Filer

You access the filer by pressing the 'F' key while in the command level. After you do, the display looks like this:

```
FILER:G, S, N, L, R, C, T, D, Q [1.1]
```

Now let's look at the directory for the disk in the drive (it should be APPLE0: for a one-drive system). Press 'L'. The system then asks:

```
DIR LISTING OF?
```

to which you respond:

```
APPLE0:
```

Don't forget to have the colon (:) on the end of the name. It tells the system that you want the directory for the disk volume named APPLE0. Leaving the colon out initiates a search for a program named APPLE0 on the boot disk drive. As no such program exists, an error will occur. After the usual hums and whirrs, the display is as shown in table 1.

Table 1

```
FILER:G, S, N, L, R, C, T, D, Q [1.1]
APPLE0:
SYSTEM.PASCAL       41 22-SEP-80
SYSTEM.MISCINFO      1 14-MAY-79
SYSTEM.COMPILER     75 19-SEP-80
SYSTEM.EDITOR       47  4-SEP-80
SYSTEM.FILER        28 18-SEP-80
SYSTEM.LIBRARY      34 19-SEP-80
SYSTEM.CHARSET       2 14-JUN-79
SYSTEM.SYNTAX       14  1-AUG-80
8/8 FILES, 32 UNUSED, 32 IN LARGEST
```

Besides the listing of files on the disk, there is other information in this listing. The number to the right of the file name is the number of blocks occupied on the disk by the file. The message at the bottom can be interpreted to mean "There are eight files listed out of eight on the disk, 32 blocks are unused, and the largest contiguous chunk of free space has 32 blocks." That is, all the free blocks are together.

You will notice there is no file called SYSTEM.WRK.TEXT or SYSTEM. WRK.CODE. This means that the workfile is empty. You will also notice that there is no Pascal program file on the disk. If there were, it would have a name that ended in 'TEXT', such as TREE.TEXT. The meaning of this will become clear shortly.

While we are still in the filer, we will set up a sample program. Since SYSTEM.EDITOR is on APPLE0, and the sample Pascal programs are on APPLE3, we have to transfer the program file we want to examine onto APPLE0 from APPLE3.

Normally, when you enter the editor, it attempts to read the workfile from the disk in the drive. If there is no workfile, you get the message:

```
NO WORKFILE PRESENT
```

To establish an existing program text file as the workfile, you will have to use the GET command from the filer. If you do this to get a sample program from APPLE3, it seems to work all right until you replace APPLE0 and return to command level. When this is tried, you get the message:

```
WORKFILE LOST
```

Thus, since all the system programs which work on Pascal programs are on APPLE0, and the sample Pascal programs are on APPLE3, you will have to transfer the one you want onto APPLE0. Note that for two drive systems it will be necessary to transfer Pascal text files from APPLE3 to APPLE1. This is accomplished *via* the TRANSFER command. While still in the filer, hit 'T'. The question:

```
TRANSFER?
```

will appear on the screen. Place APPLE3 in the drive and answer with:

```
APPLE3:HILBERT.TEXT
```

The system next responds with the question:

```
WHERE?
```

to which you should answer:

```
APPLE0:HILBERT.TEXT
```

The system will then tell you:

```
PUT IN APPLE0
PRESS <SPACE> TO CONTINUE
```

In a short time the message:

```
APPLE3:HILBERT.TEXT --->
APPLE0:HILBERT.TEXT
```

will appear, signifying that the HILBERT text file containing the Pascal program, has been transferred to APPLE0.

After transferring the file, use the GET command to load it as the workfile. The system responds with:

```
GET?
```

to which you respond:

```
HILBERT
```

The system then loads the workfile with a copy of HILBERT, and gives you the message:

```
TEXT FILE LOADED
```

Now you can Q(UIT the filer and return to command level, then select E(DITOR. The Pascal program appears on the screen. You can explore the program by using Ctrl-A to flip between left and right half-pages, by pressing 'P' to scroll downwards by one full page, or change the scrolling direction by pressing ' − ' to scroll upwards or ' + ' to scroll downwards.

When you have examined the program and are ready to try running it, just press 'Q' to quit the editor. When you do, you will be presented with the following choice:

```
QUIT
     U(PDATE THE WORKFILE AND LEAVE
     E(XIT WITHOUT UPDATING
     R(ETURN TO THE EDITOR WITHOUT
        UPDATING
     W(RITE TO A FILE NAME AND RETURN
     S(AVE WITH SAME NAME AND
        RETURN
```

Select U and the system writes HILBERT. TEXT into the scratch copy of the workfile, called SYSTEM.WRK.TEXT.

Now you are ready to R(UN the HILBERT program. Just select 'R', and the system automatically assumes you intend to run the workfile.

Listing 2, a program named TURTLE, interprets turtlegraphics commands. The program is not an all-purpose turtlegraphics program. It will not interpret all valid turtle commands; only a few of them. However, it will run as listed.

To quote an old professor, it is left as an exercise to the reader to modify the program so that PROCEDURE NOTVALID intercepts all invalid commands. The reader can also modify PROCEDURE MIMIC to display the input string one character at a time as it is typed, so that a mistake can be caught and corrected.

An Editor Sample Session

Most of the following discussion will center around the use of the editor to create a file on the disk which contains the TURTLE program in listing 1. This version contains deliberate errors for you to fix. This will give you practice using the editor. After you enter this version you can edit it so that it is the same as the working version in listing 2.

The next step is to clear the workfile. At the F(iler level enter the L(ist command. Look at the directory and see if you have a file called SYSTEM.WRK.TEXT. If you do, the editor will assume that is the one you want to work on. To enter the TURTLE program, you will have to clear the workfile. If the current workfile is something you want to keep, it should be saved with a different name, or it will be lost for good.

When you are satisfied that clearing the workfile will not cause you to lose anything important, enter the N(ew command from the F(iler level. The prompt will say

```
THROW AWAY CURRENT WORKFILE?
```

Responding by 'Y' will clear the workfile so that you can enter something new using the E(ditor.

Next, Q(uit the filer and enter the E(dit command. The prompt should now be

```
EDIT:
NO WORKFILE IS PRESENT. FILE?
(< RET > FOR NO FILE < ESC-RET >
TO EXIT)
```

Respond by pressing < RET > (the 'return' key). The prompt will now change to

```
< EDIT:A(DJUST C(PY D(LETE F(IND
I(NSRT J(MP R(PLACE Q(UIT X(CHNG
Z(AP [1.1]
```

The cursor will be on the left end of the line below the prompt line, and the rest of the screen will be blank, since the workfile is now empty.

To start entering program text, press 'I' to go to the I(nsert mode. The prompt then becomes

```
INSERT:TEXT ( < BS > A CHAR, < DEL >
A LINE) [ < ETX > ACCEPTS < ESC >
ESCAPES]
```

The items in the prompt line which are enclosed in ' ' represent keystrokes. The prompt line is the same for all versions of UCSD-based Pascal systems, but not all computers have the same keys. I have prepared a table of equivalent keys for the Apple, which I gleaned by trial and error and digging into the references (see table 2).

Table 2

System	Apple Equivalent
< DEL >	Control-X
< ETX >	Control-C
< BS >	← (left arrow)
[	Control-K
]	Shift-M
↑	Control-O
↓	Control-L
< EOF >	Control-C
< TAB >	Control-I

Now, knowing the keystroke translations, you are ready to enter the TURTLE program. The first six lines of the program are comment lines, and are not interpreted as program lines by the compiler. They are recognized by the compiler by the (* and *) delimiters, which indicate start of comment and end of comment, respectively. This serves the same function as a REM statement in BASIC.

As you get to the end of each line, type a 'return', and the cursor will move to the beginning of the next line. You indent by starting the next line with spaces. When you type 'return' at the end of an indented line, the cursor returns to a position on the next line just below the first character on the line above (indented by the same amount). To return to a previous indentation position, use the left-arrow key to backspace to the starting position you want. Now type in the TURTLE program, one line at a time, as in listing 1.

When you get to the end of the text, you have to tell the system. The system expects to get an < ETX > to signify you are done entering text. The < ETX > means "end-of-text," and is typed 'Control-C' on the Apple. When you type 'Control-C', the system returns to E(dit level.

At this point you should update the disk copy of the workfile in case something happens. The TURTLE program file exists only in memory at this point, and you don't want to have to start over.

```
(********************)
(*                  *)
(*  Turtel program  *)
(*      Fricke      *)
(*                  *)
(*    listing 1     *)
(*                  *)
(********************)

(* * NOTE: This is NOT a working
   program. It contains intent-
   ional errors to be edited out
   as practice while reading this
   article.  Enter it in as shown,
   but DO NOT attempt to compile!  *)

program turtel;

uses turtelgraphics;

var input,output: string;
    argmnt,flag,lpos,rpos,i,j :integer;
    colr :screencolor;

procedure notvalid;

   begin
      input:='not a turtel command'
   end; (*notvalid*)

procedure getcolor;

   begin
      if pos('BLACK',input)>0
      then colr:=black
      else if pos('WHITE',input)>0
         then colr:=white
         else if
                 pos('NONE',input)>0
            then colr:=none
            elsenotvalid
   end; (*getcolor*)

procedure getarg;

   begin
      lpos:=pos('(',input);
      rpos:=pos(')',input);
      argmnt:=0;
      j:=1;
      for i:=rpos-1 downto lpos+1

      do begin
         argmnt:=argmnt+(ord(input[i])
                 -ord('0'))*j;
         j:=j*10
      end (*for*)

   end; (*getarg*)

procedure scanstring;   (* Look for turtle command *)

   begin
      if     pos('MOVE',input)>0
      then
         begin
         flag:=1;
         getarg
      end
```

(Continued)

When you Q(uit, select U(pdate. The workfile will then be saved on the disk with the name SYSTEM.WRK.TEXT, and the system will return to the command level. Now you can go back to the editor and continue working on the program without fear of losing what you have done so far.

The W(rite command can be used when you are happy with the current workfile and want to save it on the disk. In our case, if TURTLE were correct, we could use the W(rite command to save a copy in a file called TURTLE. The W(rite command could also be used if you wanted to temporarily stop working on TURTLE and start working on something else.

The UCSD system was set up for machines that have cursor-moving keys (up, down, left, and right arrows). The Apple has left and right arrows, but the up arrow is typed as 'Control-O', and the down arrow is 'Control-L'.

Notice that the Editor prompt line starts with

>EDIT:

The "greater than" sign can be thought of as an arrowhead pointing to the right; that is, forward through the text. This set direction affects three of the additional cursor-moving keys; the spacebar, the < return > key, and the < TAB > key ('Control-I' on the Apple). The spacebar moves the cursor to the right when the set direction is forward and to the left when it is backward.

The < return > key moves the cursor to the beginning of the next line in the set direction; it goes to the beginning of the previous line when the set direction is backwards. The < TAB > key ('Control-I) moves the cursor to the next tab positon in the set direction. There is a tab position every eight columns across the screen.

Editor's Note: The '+' or '−' keys work as well, as do the '.' and ',' keys.

To change the set direction, just press the '<' key. The prompt line becomes

< EDIT:

to show that the set direction is now backwards. Try experimenting with the '<' and '>' keys and the other cursor-moving keys we have discussed so far.

```
      else if pos('PENCOLOR',input)>0
         thenbegin
            flag:=2;
            getcolor
         end
         else if pos('TURN',input)>0
            then begin
               flag:=3;
               getarg
            end
            else if pos('CLEARSCREEN',input)>0
               then flag:=4
               else begin
                  notvalid;
                  mimic
               end
   end; (*scanstring*)

procedure mimic;

   var x,y: integer;

   begin
      x:=turtlex;
      y:=turtley;
      pencolor(none);
      moveto(1);
      wstring('                     '); (*21 spaces*)
      moveto(1);
      wstring(input);
      moveto(x,y);
      pencolor(colr)
   end; (*mimic*)

begin (* Main Program *)

   initturtel;
   readln(input);
   mimic;
   repeat
      scanstring;
      case flag of
         1:  move(argmnt);
         2: pencolor(colr);
         3: turn(argmnt);
         4: fillscreen(black)
      end; (*case*)
      readln(input);
      mimic
   until (length(input)=0); (* Exit by pressing 'RET' only *)
   textmode

end.
```

After a short while you will become bored with repeatedly typing right arrows, 'Control-L', etc., to move considerable distances through the text. Fortunately, you don't have to do that. The system has provided several additional features for "hot-rodding" the cursor. Say you want to move the cursor down eight lines. You have been typing 'Control-L' eight times to do this, but there is an easier way. Type '8Control-L', and *voila*, the cursor moves down eight lines. The '8' in front of the 'Control-L' is called a "repeat factor," and it can be applied to all cursor moving commands. The repeat factor can be any integer, or the character '/'. If '/' is used, it means "all the way to the end."

Another handy feature, the 'P' command, moves the cursor one whole screen page (24 lines) in the set direction.

Now that we know how to move the cursor around at the E(dit level, it is time to fix the errors in TURTLE. Compare your text to listing 2 and find a passage that needs to be deleted. For example, in the line that says

```
VAR
  INPUT, OUTPUT:STRING;
```

we want to make it say

```
VAR
  INPUT:STRING;
```

Move the cursor until it rests on the comma (the first character to be deleted) while at the E(dit level. Now, type 'D' to go to the D(elete level. The prompt line changes to

> DELETE: < >< MOVING COMMANDS >
< ETX > TO DELETE, < ESC > TO ABORT

but the text remains unchanged. Now, using the right arrow, trace over the letters in ', OUTPUT'. They will disappear from the screen, replaced by blanks. However, they are not gone from the text. If you move the cursor back, the letters reappear! The reason is that the deletion is not made from the workfile until you press ETX (Control-C). When you do, the deletion is made, and all the extra blanks are removed, closing up the resulting text.

The other cursor-moving commands work in a similar way. If you move the cursor down one line, the remainder of the line it started on and all the next line to the left of the cursor position disappears. Those are the characters that the cursor would have moved over in going from its starting position to its ending position one character at a time.

Now, let's find the line that says

```
MOVETO(1);
```

which we want to change to say

```
MOVETO(1,1);
```

While at the E(dit level, move the cursor to the right parenthesis; i.e., to the right of the '1', where we want the insertion to begin. Now press 'I' to get into the I(nsert mode. It looks like the rest of the line disappears, but it doesn't. If you flip over to the right half of the screen (by pressing 'Control-A'), you will see that the remaining characters have been moved all the way over to the right end of the line to make room for the insertion.

Next, type in the ',1' and then 'Control-C' to complete the insertion. The last two characters are moved back to the left to close up the line, and the insertion has been made.

Now, correct the indentation position of a line by placing the cursor on any character in the line and pressing 'A' to go to the A(djust level. The prompt line becomes

> ADJUST:L(JUST R(JUST C(ENTER <LEFT, RIGHT,UP,DOWN-ARROWS> <ETX> TO LEAVE

To move the whole line left, press the left arrow; to move it right, use the right arrow, and to move it all the way to the left edge of the screen, press 'L'. If you want to indent a whole block of lines the same amount, use 'Control-O' or 'Control-L', and the next line up or down will be indented to the same position. Pressing 'Control-C' implements the changes, as with the other commands.

For multiple changes of the same kind, use the R(eplace command. For example, you should really spell TURTLE, T-U-R-T-L-E, not T-U-R-T-E-L. You can use the R(eplace command to change TURTEL to TURTLE wherever it occurs. Put the cursor at the beginning of the file and press 'R'. The prompt line becomes

REPLACE[1]: L(IT V(FY <TARG><SUB> =>

The number in brackets is the number of times the command has been invoked. If you wanted to find all occurrences of TURTEL but did not know how many there were, you should have used the "infinite" repeat factor;

/R

The '/' repeat factor insures that all occurrences of TURTEL will be found.

Next press 'L' to select a literal search. Two types of searches can be made: literal and token. The default mode is a token search which looks for a string that is isolated by spaces on either side, while a literal search looks for all occurrences of the string, even those within a larger string.

Now press 'V' to select a V(erify search. If you do not, for all instances where the target string occurs, the substitute string will replace it. In the V(erify mode the system stops at each occurrence of the target string and asks if you really want to make the substitution.

The final two items in the prompt line are <TARG> and <SUB> which stand for the "target string" (what to search for in the existing text), and the "substitute string" (what to replace the target string with), respectively. These strings have to have the '/' delimiter at their beginnings and ends. There are other delimiters that can be used, but I always use '/'.

```
(*******************)
(*                 *)
(*  Turtle program *)
(*      Fricke     *)
(*                 *)
(*    listing 2    *)
(*                 *)
(*******************)

(* NOTE: This program requires
   UPPERCASE only as input.... *)

program turtle;

uses turtlegraphics;

var input: string;
    argmnt,flag,lpos,rpos,i,j: integer;
    colr: screencolor;

procedure notvalid;

   begin
      input:='not a turtle command'
   end; (*notvalid*)

procedure getcolor;

   begin
      if pos('BLACK',input)>0
      then colr:=black
      else if pos('WHITE',input)>0
         then colr:=white
         else if pos('NONE',input)>0
            then colr:=none
            else notvalid
   end; (*getcolor*)

procedure getarg;

   begin
      lpos:=pos('(',input);
      rpos:=pos(')',input);
      argmnt:=0;
      j:=1;
      for i:=rpos-1 downto lpos+1
      do begin
         argmnt:=argmnt+(ord(input[i])
                 -ord('0'))*j;
         j:=j*10
      end (*for*)
   end; (*getarg*)

procedure mimic;

   var x,y: integer;

   begin
      x:=turtlex;
      y:=turtley;
      pencolor(none);
      moveto(1,1);
      wstring('                     '); (*21 spaces*)
      moveto(1,1);
      wstring(input);
      moveto(x,y);
      pencolor(colr)
   end; (*mimic*)

procedure scanstring;    (* Look for turtle command *)
```

Listing 2 *(Continued)*

```
begin
   if pos('MOVE',input)>0
   then begin
      flag:=1;
      getarg
   end
   else if pos('PENCOLOR',input)>0
      then begin
         flag:=2;
         getcolor
      end
      else if pos('TURN',input)>0
         then begin
            flag:=3;
            getarg
         end
         else if pos('CLEARSCREEN',input)>0
                  then flag:=4
                  else begin
                     notvalid;
                     mimic
                  end
     end; (*scanstring*)

  begin (* Main Program *)

     initturtle;
     readln(input);
     mimic;
     repeat
        scanstring;
        case flag of
           1: move(argmnt);
           2: pencolor(colr);
           3: turn(argmnt);
           4: fillscreen(black)
        end; (*case*)
        readln(input);
        mimic
     until (length(input)=0); (* Exit by pressing 'RET' only *)

     textmode

  end.
```

So, in summary, starting from the E(dit level with the cursor at the beginning of the text, type

/RLV/TURTEL//TURTLE/

After you type the last '/' the replacement search will begin. The cursor will stop at the first line where 'TURTEL' appears:

PROGRAM TURTEL;

and the prompt line becomes

>REPLACE:< ESC >ABORTS, 'R' REPLACES, ' ' DOESN'T

If you want the replacement made, press 'R' and the line becomes

PROGRAM TURTLE;

and the search continues until all occurrences of 'TURTEL' have been found. If you want to leave one occurrence unchanged, just press the space bar and the system continues the search.

The last editor feature that will be covered is the C(opy command. This can be quite useful for moving blocks of text around.

When you make a change in the text of the workfile using I(nsert or D(elete, a copy of the change is made in a separate area of memory called the "copy buffer." As you add characters in the I(nsert mode, each additional character goes into the copy buffer, until you terminate the insertion with an < ETX > or < ESC >. The < ESC > cancels the insertion from having an effect on the workfile, but the copy in the copy buffer remains. Similarly, for a deletion, the deleted characters go into the copy buffer as they are removed from the screen.

If you had typed in PROCEDURE MIMIC after PROCEDURE SCANSTRING in the TURTLE program, you would get an error in compilation because PROCEDURE SCANSTRING invokes PROCEDURE MIMIC which has not yet been defined when the compiler tries to interpret PROCEDURE SCANSTRING. What you want to do is to move PROCEDURE MIMIC to a position before PROCEDURE SCANSTRING. You have to delete PROCEDURE MIMIC from its position after PROCEDURE SCANSTRING and insert it before PROCEDURE SCANSTRING. To do this, use the C(opy command.

The first step is to delete PROCEDURE MIMIC. Place the cursor on the 'P' in PROCEDURE. Press 'D' to go into the D(elete mode. Press < RETURN > repeatedly to erase successive lines of PROCEDURE MIMIC. When all the lines are gone, press < ETX > to effectuate the deletion. Remember, PROCEDURE MIMIC is gone from the screen, but it is still in the copy buffer.

Now move the cursor to the beginning of the blank line before PROCEDURE SCANSTRING. This positions the cursor at the point where we want the copy of PROCEDURE MIMIC to be inserted. Press 'C' to go to the C(opy level. The prompt line becomes

C(OPY: B(UFFER F(ROM FILE<ESC>

Next press 'B' for B(uffer, and the insertion is made.

You also have the option of copying from a disk file as well. This feature allows you to insert a debugged procedure from another program or to work on a large program in several pieces. To copy from a disk file, just press 'F' after the 'C' for C(opy. The prompt line will change to say:

> COPY: FROM WHAT FILE [MARKER,MARKER]?

Markers point out strategic places in a long text file. Unfortunately, the system will tell you how many markers you have used and their names, but not where they are. Markers also do not show on the screen. If you are interested in using them, consult the manual.

Auto Line Numbers for OSI Disk BASIC

Auto line numbers for OSI disk BASIC imitate large computers and make programming easier.

Lester Cain
1319 N. 16th
Grand Junction, Colorado 81501

Software support for the OSI is improving, but is still minimal, and users have to develop many of their own programs. Actual programming with flow charts and algorithms is part of the pleasure of developing your own program. But when it's time to input to the machine, some of the fun flies out the window. With all the necessary keying, line numbers are an added detriment and detract from the pleasures we do get from writing programs.

Some of us are familiar with large mainframe computers, which have an AUTO function and put out line numbers for you. This function is definitely a plus and should be available to all of us.

This manuscript explains a simple and easy-to-use program, which will give us an AUTO function to use in relieving some of the tedious burden of typing. Included are two listings, one in assembly language and the other in BASIC. This should work on the C1P disk BASIC also. The logic is easy to follow and could be put to use on ROM machines also, with different hooks. But we will leave that as an exercise for persons with ROM.

Listing 1 is the assembly language routine necessary to develop the program. In OSI disk BASIC, the routine to get a character from the keyboard and incorporate it into the BASIC Source begins at $558 which is LDX #$0. At the next address, or $55A, we will put in a hook to make BASIC jump to our AUTO program. This is accomplished in line 310 of listing 2. This will force information to go through our code before BASIC can do anything with our keyboard information.

Listing 1

```
                   ;* AUTO LINE NUMBERS *
                   ;*    FOR OSI C4P    *
                   ;*                   *
                   ;*    BY LES CAIN    *
                   ;*
                   SCL    EPZ $6C
                   SCH    EPZ $6D
                   BUF    EPZ $1A
                   ;
                   BASIC  EQU $055D
                   INPUT  EQU $0587
                   LINE   EQU $1CDC
                   ;
                   TH     EPZ $D8
                   FH     EPZ TH+1
                   LO     EPZ TH+2             ;DECIMAL 218
                   HI     EPZ TH+3
                   ;
                          ORG $8000
                   ;
8000 208705        START  JSR INPUT            ;BASIC INPUT ROUTINE SENT HERE
8003 48                   PHA                  ;SAVE CHARACTER
8004               ;
8004 C906          AUTON  CMP #$06             ;CTRL F
8006 D006                 BNE AUTOFF
8008 A900                 LDA #$00             ;YES, TURN ON AUTO
800A 85D8                 STA TH               ;AUTO FLAG
800C 85D9                 STA FH               ;FLAG TO BYPASS AUTO
800E               ;
800E C91B          AUTOFF CMP #$1B             ;ESC
8010 D004                 BNE BACK             ;TEST FLAGS
8012 E6D8                 INC TH               ;TURN OFF AUTO FLAGS
8014 E6D9                 INC FH
8016 A5D8          BACK   LDA TH               ;GET AUTO FLAG
8018 D004                 BNE BK               ;NOT A 0--BACK TO BASIC
801A A5D9                 LDA FH               ;CR FLAG MADE 0 WITH A CR
801C F004                 BEQ AUTO             ;IF 0 THEN CONTINUE WITH LINE #'S
801E 68            BK     PLA                  ;GET BACK SAVED CHARACTER
801F 4C5D05        BK1    JMP BASIC            ;BK TO BASIC WITH CHAR. IN ACC.
8022               ;
8022 68            AUTO   PLA                  ;PULL OFF SAVED CHARACTER
8023 A940                 LDA #$40             ;LO BYTE OF SCREEN ADDRESS
8025 856C                 STA SCL              ;INITIALIZE INDIRECT POINTERS
8027 A9D7                 LDA #$D7             ;HI BYTE OF SCREEN ADDRESS
8029 856D                 STA SCH              ;INITIALIZE HI BYTE
802B A5DA                 LDA LO               ;LO BYTE OF LINE NO.
802D 18                   CLC
802E 690A                 ADC #$0A             ;LINE NO. INCREMENT
8030 85DA                 STA LO               ;SAVE IT FOR NEXT TIME
8032 9002                 BCC ASOUT            ;NOT #$FF
8034 E6DB                 INC HI               ;ADD 256 TO LINE NO.
8036               ;
8036 A6DA          ASOUT  LDX LO               ;LO BYTE OF LINE NO.
8038 A5DB                 LDA HI               ;HI BYTE OF LINE NO.
803A               ;
803A               ;CONVERTS BINARY NUMBER TO ASCII STRING AND PUTS TO SCREEN
803A               ;USED TO OUTPUT LINE NO.'S WHEN LISTING
803A               ;
803A 20DC1C               JSR LINE             ;BASIC ROUTINE FOR THE LINE NO.
803D 98                   TYA                  ;GET Y FROM OUTPUT ROUTINE
803E AA                   TAX                  ;SAVE IT IN X
803F 48                   PHA
8040 88                   DEY                  ;BYPASS SPACE AFTER CURSOR
8041               ;
8041 B16C          SCR    LDA (SCL),Y          ;CHARACTER FROM SCREEN
8043 991A00               STA BUF,Y            ;PUT IN BASIC BUFFER-1
8046 88                   DEY
8047 D0F8                 BNE SCR              ;NOT TO END OF LINE NO. ON SCREEN
8049 68                   PLA                  ;GET BACK Y
```

(Continued on next page)

Now we are at routine START in the assembly routine. Since we put a hook here to make BASIC jump, we will have to perform the routine that was originally there, getting a key from the keyboard. At AUTON we test for a control 'F.' If this key is encountered here, the two Auto flags are set to zero and the program will fall through to the AUTO routine. If there is no control 'F,' then test for an ESC at AUTOFF. If there is an ESC, turn off Auto flags TH and FH and go back to BASIC with the character in the accumulator. If no ESC is found, test Auto flag TH. If TH is not zero then we test the secondary flag FH. This flag is turned off in the SCR routine, so constant line numbers are not output. If FH is zero then we are ready for a new line number and fall through to the AUTO routine.

AUTO is a simple addition and increments the line number by 10 at every pass. AUTO also initializes the indirect screen pointers. This need only be done once, but why take chances? BASIC might decide to stick something at these addresses.

One of the keys to our whole program is the ASOUT routine. The line number is loaded into the accumulator and the X index. A JSR to the BASIC routine LINE ($1CDC) will output an ASCII string from the binary values in LO and HI to the screen at cursor level. BASIC uses this routine to output line numbers when listing.

This brings us to the most important segment of the program — getting BASIC to accept the line number we have created. It must be in an acceptable format and in the input buffer. We use the Y index for LINE, and decrement it by one to get us to the cursor. Here storage is started into the buffer. After the line number is in, the X index is decremented and we write on top of the cursor with a space. BASIC uses X to point into the buffer. From here it's back to the keyboard with a space after the last digit of the line number. Here we also turn off the CR flag FH, by simply incrementing it.

Now for the last segment of the assembly program, the CR routine. We have put a hook into BASIC with the statement in line 270 of listing 2. BASIC will jump here when it finds a carriage return. We turn to the back of flag FH; if the main Auto flag TH is on, the AUTO process will continue until an ESC turns off both flags. To end the program we have a jump to $A6D. This puts the buffer pointer into the CHARGET routine and checks the syntax to determine if what we just did was an immediate command or a line number. It is a line number so all pointers will be reset and the line is entered into the BASIC source.

Listing 1 *(continued)*

```
804A A8                  TAY                 ;RESTORE Y FOR DISPLAY PURPOSES.
804B CA                  DEX                 ;BYPASS CURSOR, X IS BUFFER INDEX
804C E6D9                INC FH              ;TURN OFF CR FLAG
804E A920                LDA #$20            ;SPACE
8050 D0CD                BNE BK1             ;JUMP TO BASIC WITH SPACE IN ACC.
8052             ;
8052             ;PATCH FROM BASIC POKES TO RESTORE AUTO FLAG
8052             ;AFTER A CR IS RECEIVED INTO INPUT ROUTINE
8052             ;
8052 A900        CR      LDA #$00            ;TURN BACK ON AUTO FLAG
8054 85D9                STA FH              ;CR FLAG
8056 4C6D0A              JMP $0A6D           ;BACK TO BASIC ADDRESS PATCHED
                         END
```

Listing 2

```
100  REM  -- AUTO LINE NUMBERS --
110  REM  -- FOR OSI 4P AND 8P DISK SYSTEMS --
120  REM
130  REM  --WORKS FOR ANY SIZE MEMORY --
140  REM
150  REM  --POKE NEW HIGH MEMORY TO SAVE CODE --
160  REM
170 S =  PEEK (8960): POKE 132,143: POKE 133,S: RUN 180
180 P =  PEEK (8960)
190  REM
200  REM  -- X IS BEGIN ADDRESS TO POKE CODE --
210  REM
220 X = P * 256 + 144: FOR I = X TO X + 88: READ A: POKE I,A: NEXT
230  REM
240  REM   -- POKE A JUMP TO MACHINE CODE AT $584
250  REM  -- P IS THE HIGH BYTE --
260  REM
270  POKE 1412,76: POKE 1414,P: POKE 1413,226
280  REM
290  REM  -- POKE JUMP TO MACHINE CODE AT $55A --
300  REM
310  POKE 1370,76: POKE 1371,144: POKE 1372,P
320  REM
330  PRINT : PRINT "READY": PRINT
340  POKE 218,90: POKE 219,0: END : REM   90 IS BEGINNING LINE NO.
350  REM
360  REM  -- DATA FOR MACHINE CODE ROUTINE --
370  DATA 32,135,5,72,201,6,208,6,169,0,133,216,133,217,201,27
380  DATA 208,4,230,216,230,217,165,216,208,4,165,217,240,4,104,76
390  DATA 93,5,104,169,64,133,108,169,215,133,109,165,218,24,105,10
395  REM     --- THIS IS THE INCREMENT -----------------------------^
400  DATA 133,218,144,2,230,219,166,218,165,219,32,220,28,152,170,72
410  DATA 136,177,108,153,26,0,136,208,248,104,168,202,230,217,169,32
420  DATA 208,205,169,0,133,217,76,109,10
```

The BASIC program as shown is all that is necessary to have the AUTO function on our system. Line 170 determines the highest page of RAM on our system and sets the high end of BASIC work space to protect the object code. Statement 220 POKEs the code into the appropriate area of memory by reading the data and POKEing it to I. Statement 270 puts in the intercept jump to reset the secondary Auto flag. Statement 310 puts the hook for getting characters into the original BASIC routine, for our test routine. Since the machine code is complety relocatable, the only variable is P which BASIC puts in 8960 on boot in, indicating the highest page in RAM.

The REM statement in the data indicates the location of the beginning line number. This could be changed if we don't want to start a line number 100.

Both the listings are included here to give a choice of how we want to implement the AUTO routine. The assembly method could be used in the free area before BASIC workspace, on the minidisks. One word of caution here: some of the new software out has a revised keyboard routine in this area. This way the program would be available all the time and not used as free RAM. Or, the BASIC program could be run from BEXEC*. The BASIC listing was made using the AUTO function.

A few words here on the use of our finished program. The two flags are turned off at first and must be turned on with a Control F. After the program is on, it will continue to put out line numbers until an ESC is encountered. The ESC can be either in the line or before another line is put out. Simply press the space bar to continue after each carriage return. This is still a lot more convenient than typing line numbers in.

MICRO

Some Help for KIM

Part 1

The usefulness of the KIM memory dump routine is extended by utilizing software that dumps memory in a program-like format.

Wayne D. Smith
Box 8352
Austin Peay State University
Clarksville, Tennessee 37040

The KIM-1 microcomputer comes complete with an excellent set of software routines built into the ROM section of the computer. In fact, one of the strong points of the KIM is its use of software to perform many of the functions accomplished with hardware in other systems. The largest percentage of the KIM software is written in the form of subroutines, which makes it easy for the user to take advantage of already existing software when writing his own programs.

Two KIM routines that are used extensively with hand assembled programs are the memory dump program and the single-step software routine. These programs usually prove to be valuable to the user during the initial phases of program development. As useful as these routines are, however, both can be modified to provide additional features to the user. This month we will look at improving the memory dump format. In future months we will look at improving the single-step feature.

Listing Programs

The major deficiency in the memory dump routine lies in the format in which the data is displayed. The program is intended to provide both a hard copy listing and a reloadable paper tape. The format consists of a semi-colon, then the number of bytes to be printed (usually 18 hex), followed by the starting address, and finally the memory

```
                    ;
                    ; A PROGRAM TO LIST KIM PROGRAMS IN THE FORMAT:
                    ;
                    ;    ADDR OP OPERAND
                    ;
                    ; THE PROGRAM CHECKS THE OPERATION CODE
                    ; AND PRINTS THE CORRECT NUMBER OF OPERAND
                    ; BYTES FOR EACH TYPE INSTRUCTION.
                    ;
                    NOWL   .DE $E0          ;THE ADDR. OF THE OP.
                    NOWH   .DE $E1          ; CODE BEING PRINTED NOW.
                    SAL    .DE $17F5        ;WHERE LISTED PROG. STARTS
                    SAH    .DE $17F6        ; (TWO BYTES)
                    EAL    .DE $17F7        ;WHERE LISTED PROGRAM ENDS
                    EAH    .DE $17F8        ; +1 (TWO BYTES)
                    KIM    .DE $1C4F        ;KIM RETURN ADDRESS
                    CRLF   .DE $1E2F        ;KIM CARR. RETURN ROUTINE
                    PRTBYT .DE $1E3B        ;KIM PRINT A BYTE ROUTINE
                    OUTSP  .DE $1E9E        ;KIM PRINT A SPACE ROUTINE
                    ;
                           .BA $1000        ;(RELOCATABLE) .BA = ORG
                    ;
1000- 08            PGMDMP PHP              ;SAVE MACHINE STATUS
1001- D8                   CLD              ;CLEAR DECIMAL MODE
1002- 20 2F 1E             JSR CRLF         ;PRINT TWO BLANK LINES
1005- 20 2F 1E             JSR CRLF         ; FOR NEATNESS SAKE.
1008- AD F5 17             LDA SAL          ;SET "NOW" TO START. ADDR.
100B- 85 E0                STA *NOWL
100D- AD F6 17             LA SAH
1010- 85 E1                STA *NOWH
                    ;
1012- A5 E0         MAINLP LDA *NOW         ;CHECK FOR END OF DUMP
1014- CD F7 17             CMP EAL
1017- A5 E1                LDA *NOWH
1019- ED F8 17             SBC EAH
101C- 90 07                BCC ANOTHR       ;IF NOT, DO ANOTHER.
101E- 20 2F 1E             JSR CRLF         ;AT END, DO BLANK LINE,
1021- 28                   PLP              ; RESTORE STATUS,
1022- 4C 4F 1C             JMP KIM          ; AND BACK TO KIM.
                    ;
1025- 20 2F 1E      ANOTHR JSR CRLF         ;START A NEW LINE.
1028- A5 E1                LDA *NOWH
102A- 20 3B 1E             JSR PRTBYT       ;PRINT HIGH ORDER ADDRESS
102D- A5 E0                LDA *NOWL        ; AND
102F- 20 3B 1E             JSR PRTBYT       ; LOW ORDER.
1032- A2 03                LDX #$03         ;SET TO PRINT 3 BYTES.
1034- A0 00                LDY #$00         ;SET ADDR OFFSET TO 0
1036- B1 E0                LDA (NOWL),Y     ;GET NEXT OP CODE
1038- 48                   PHA             ; AND SAVE ON STACK.
1039- 29 08                AND #$08         ;IF BIT 3 IS ON, THEN
103B- D0 28                BNE PART2        ; GO TO PART 2.
103D- 68                   PLA              ;ELSE RECOVER OP CODE
103E- 48                   PHA
103F- 29 03                AND #$03         ;IF LSD IS 3 OR 7, THEN
1041- C9 03                CMP #$03         ; TREAT AS A ONE-BYTE
1043- F0 46                BEQ ONE          ; INSTRUCTION.
1045- 68                   PLA              ;ELSE RECOVER OP CODE
1046- 48                   PHA
1047- 29 0F                AND #$0F         ;IF LSD IS NOT 3, 7, OR 0,
1049- D0 41                BNE TWO          ; TREAT AS 2-BYTE INSTR.
104B- 68                   PLA              ;SINCE LSD IS ZERO,
104C- 48                   PHA              ; TEST MSD HERE.
104D- 29 70                AND #$70         ;IF MSD IS 0 OR 8, THEN
104F- F0 3A                BEQ ONE          ; TREAT AS 1-BYTE INSTR.
1051- 68                   PLA              ;ELSE,
1052- 48                   PHA
1053- 29 F0                AND #$F0         ;IF MSD IS 2, THEN
1055- C9 20                CMP #$20         ; TREAT AS A 3-BYTE INSTR.
1057- F0 34                BEQ THREE
1059- 68                   PLA              ;ELSE,
105A- 48                   PHA
105B- 29 D0                AND #$D0         ;IF MSD IS 4 OR 6, THEN
105D- C9 40                CMP #$40         ; TREAT AS A 1-BYTE
105F- F0 2A                BEQ ONE          ; INSTRUCTION.
1061- D0 29                BNE TWO          ;ALL OTHERS ARE 2 BYTES.
                    ;
1063- F0 AD         BTML   BEQ MAINLP       ;PATCH BRANCH TO MAINLP
1065- 68            PART2  PLA              ;(BIT 3 IS ON HERE)
1066- 48                   PHA              ;RECOVER OP CODE
```

(Continued on next page)

```
1067- 29 0B            AND #$0B      ;IF LSD IS B OR F,
1069- C9 0B            CMP #$0B      ; THEN TREAT AS A ONE-
106B- F0 1E            BEQ ONE       ; BYTE INSTRUCTION.
106D- 68               PLA
106E- 48               PHA
106F- 29 0D            AND #$0D      ;IF LSD IS 8 OR A,
1071- C9 08            CMP #$08      ; THEN TREAT AS A ONE-
1073- F0 16            BEQ ONE       ; BYTE INSTRUCTION.
1075- 68               PLA
1076- 48               PHA
1077- 29 0F            AND #$0F      ;ALL OTHERS EXCEPT LSD = 9
1079- C9 09            CMP #$09      ; ARE TREATED AS THREE-
107B- D0 10            BNE THREE     ; BYTE INSTRUCTIONS.
107D- 68               PLA           ;IF LSD IS 9,
107E- 48               PHA           ; THEN TEST MSD.
107F- 29 10            AND #$10      ;IF MSD IS 1,3,5,7,9,D,
1081- D0 0A            BNE THREE     ; OR F THEN 3-BYTE INSTR.
1083- 68               PLA
1084- 48               PHA
1085- 29 F0            AND #$F0      ;IF MSD IS NOT 8 THEN
1087- C9 80            CMP #$80      ; TREAT AS 2-BYTE INSTR.
1089- D0 01            BNE TWO       ;ALL OTHERS--1-BYTE.
108B- CA        ONE    DEX           ;ADJUST BYTES TO PRINT
108C- CA        TWO    DEX           ; AS REQUIRED
108D- 98        THREE  TYA           ;SAVE Y REGISTER,
108E- 48               PHA           ; (TWICE).
108F- 48               PHA
1090- 20 9E 1E         JSR OUTSP     ;PRINT A BLANK.
1093- 68               PLA           ;RECOVER Y REGISTER
1094- A8               TAY
1095- B1 E0            LDA (NOWL),Y  ;LOAD NEXT BYTE, AND
1097- 20 3B 1E         JSR PRTBYT    ; PRINT IT.
109A- 68               PLA           ;RECOVER Y REGISTER,
109B- A8               TAY           ; (AGAIN).
109C- C8               INY           ;INCREMENT ADDR OFFSET
109D- CA               DEX           ;DEC BYTES TO PRINT.
109E- D0 ED            BNE THREE     ;IF NOT DONE, REPEAT.
10A0- 18               CLC           ;INCREASE NOW BY THE
10A1- 98               TYA           ; NUMBER OF BYTES
10A2- 65 E0            ADC *NOWL     ; JUST PRINTED.
10A4- 85 E0            STA *NOWL
10A6- 90 02            BCC SKIP
10A8- E6 E1            INC *NOWH
10AA- 68        SKIP   PLA           ;CLEAR THE STACK.
10AB- A0 00            LDY #$00      ;SET Y FOR TWO-STEP
10AD- F0 B4            BEQ BTML      ; JUMP TO MAIN LOOP.
                ;
                       .EN
```


contents, followed by a 4-digit checksum. If more than one line is to be printed, subsequent lines are printed immediately below the preceding line. While this routine is certainly serviceable, it leaves a great deal to be desired in terms of readability, especially when it comes to listing a loaded program.

A better approach would be to have a routine that prints a program in the same form in which it was written. That is, first the address, then the operation code, and then the operand, if any, associated with that instruction. For clarity, the individual fields should be separated by blanks.

A program which performs this type of listing is shown in listing 1. The program is quite straightforward, and KIM subroutines are used whenever possible to reduce program length. The only tricky part of the program is the analysis of the operation codes to determine the instruction length for printing the operand field.

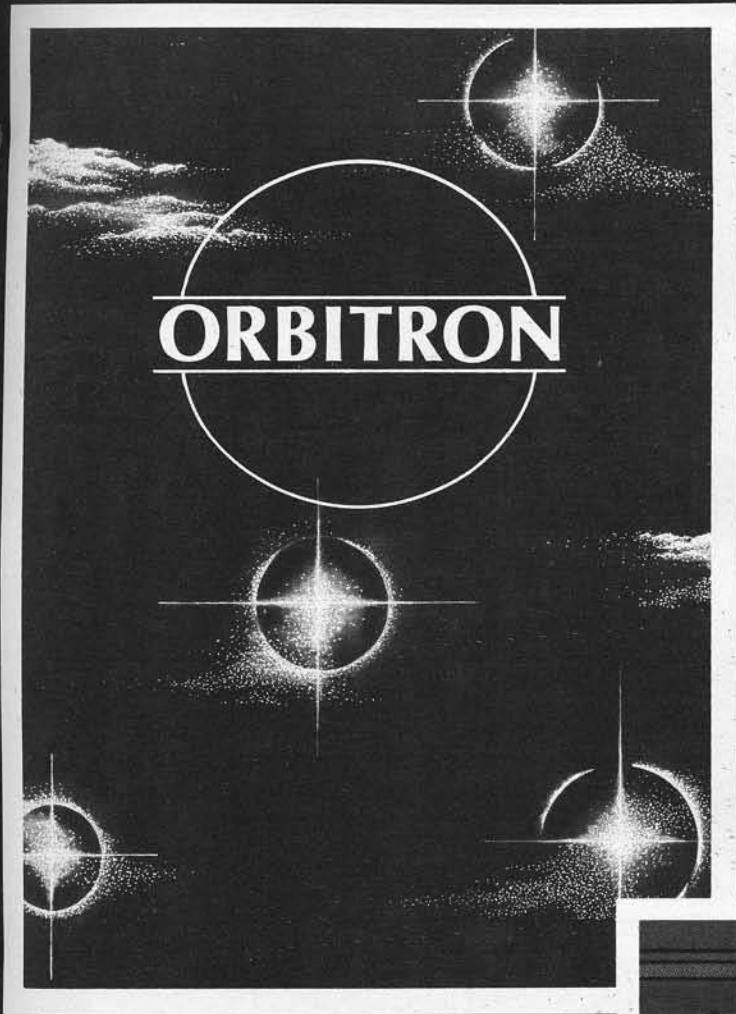
The program is used very much like the KIM dump routine except that the starting address of the program to be listed is loaded into 17F5 and 17F6 (low order byte first, as usual). The last address plus one is loaded into 17F7 and 17F8. The listing program is then executed normally. The program terminates with a JMP to the KIM monitor, but a subroutine return can be used if desired.

One word of caution is in order. The program is designed primarily for listing programs. If data is listed, the number of bytes printed per line will be determined by the first byte on that line, which is treated as an operation code. There may be one, two, or three bytes per line. Similarly, if the program is started in the middle of an instruction, the results will depend on the contents of the first byte. Experience has shown that the program will usually get back into synchronization after a few lines though.

The program is completely relocatable, and uses only two page zero addresses. These are locations E0 and E1, which are used to keep track of the location of the byte currently being printed.

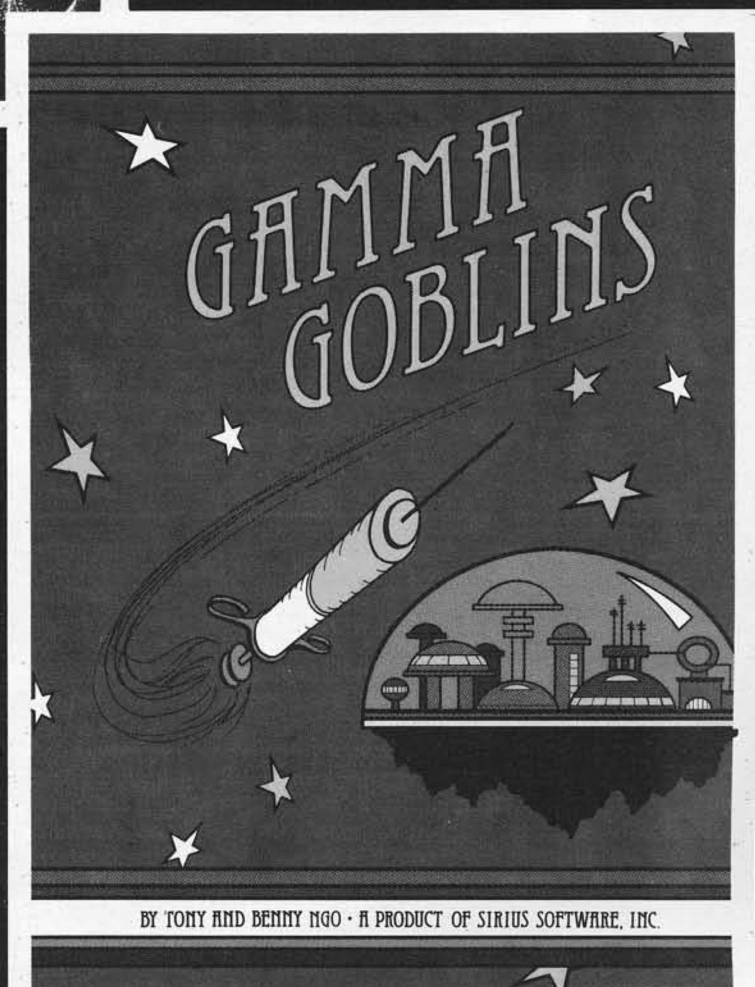
MICRO

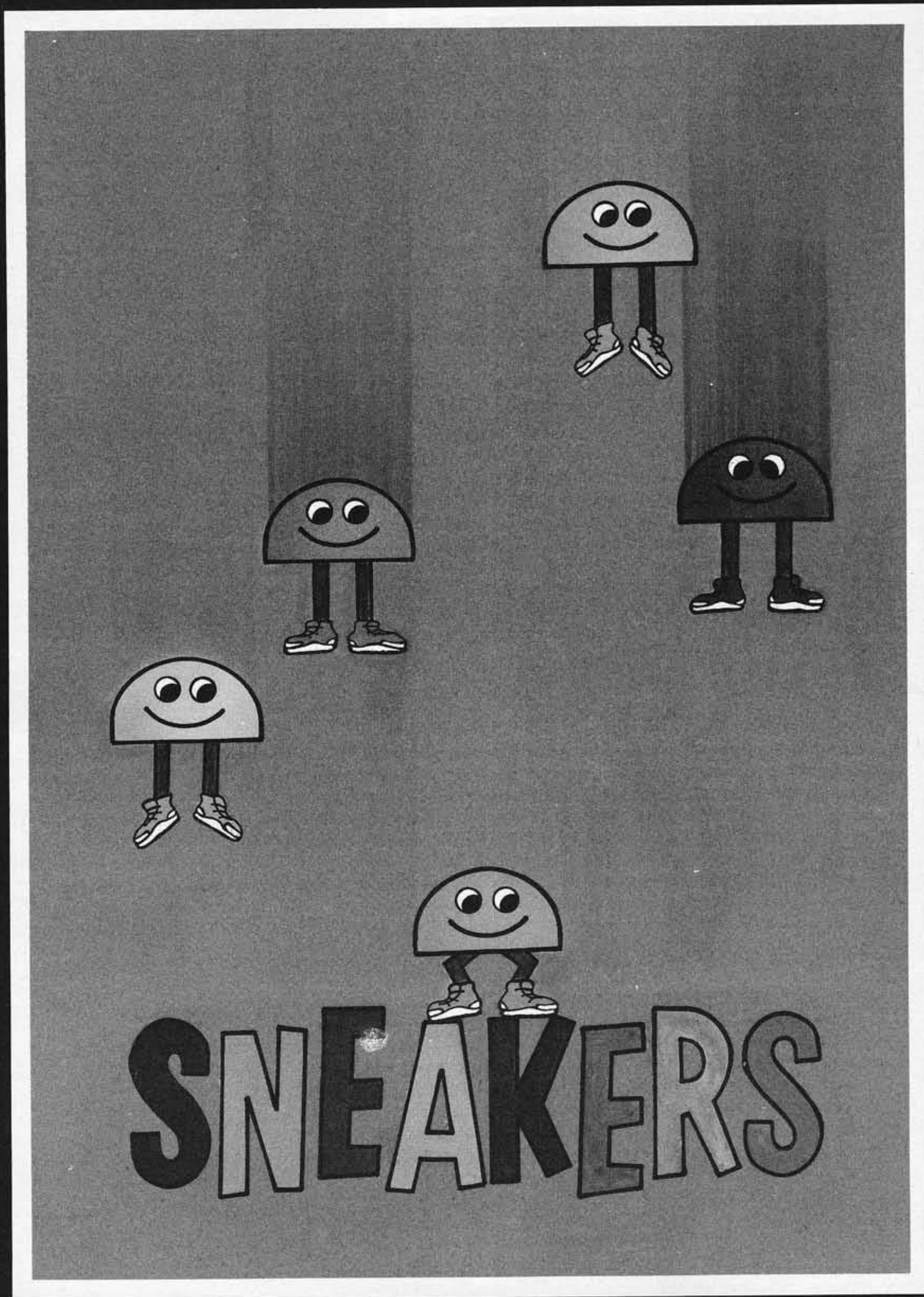
Where your secret weapon is the fourth dimension...



In the center of an orbiting space station you are protected only by a revolving force shield. Enemy forces are advancing from all directions and begin to place killer satellites in orbit around your station. And then, look out for the meteors!

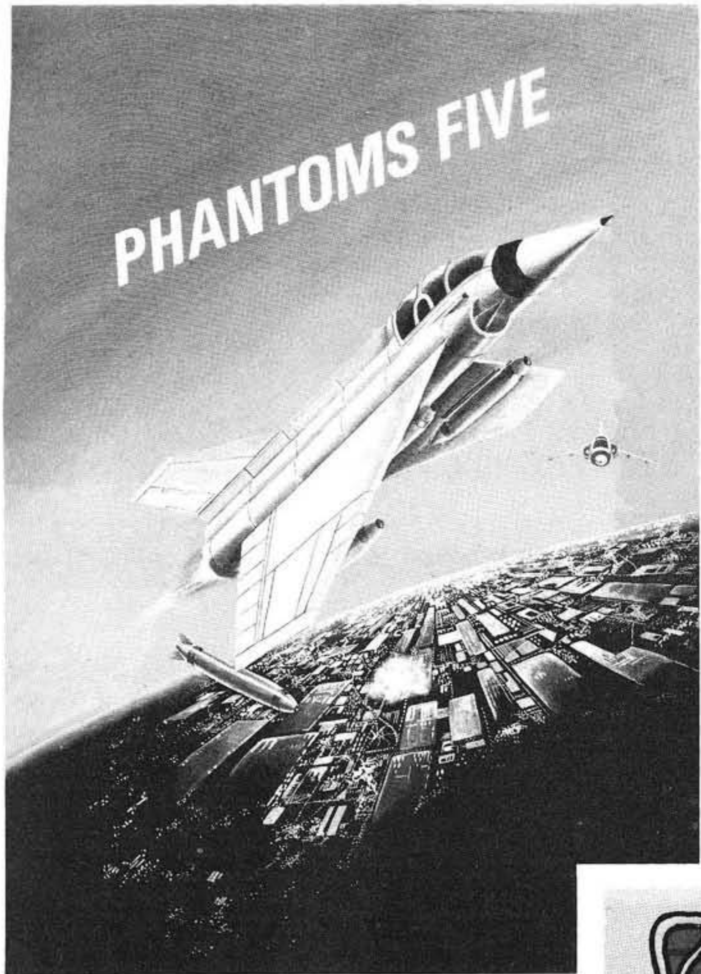
A "bloody" good game for the true-blue game freak. Your mission in this exploratory operation is to deliver whole blood to Hemophilia, a city in the sky, and return to Anemia Base before the Gamma Goblins overcome you. A real heart stopper!





What say we go out and stomp a few???

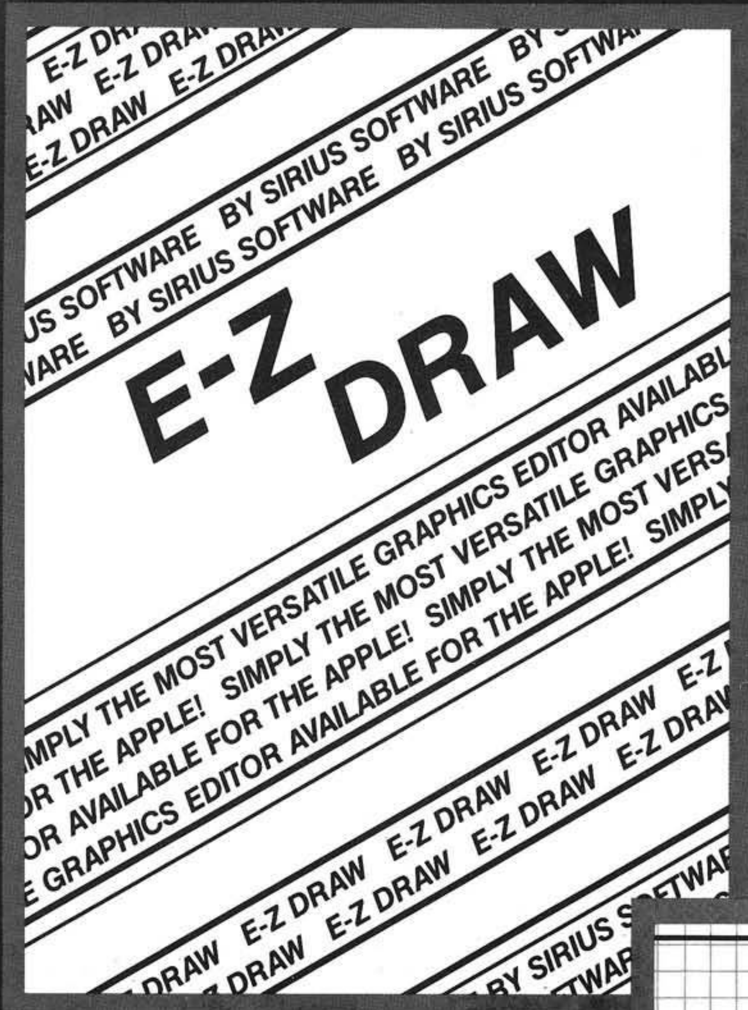
Endless Excitement Stomping Sneakers And A Swarm Of Other Creatures



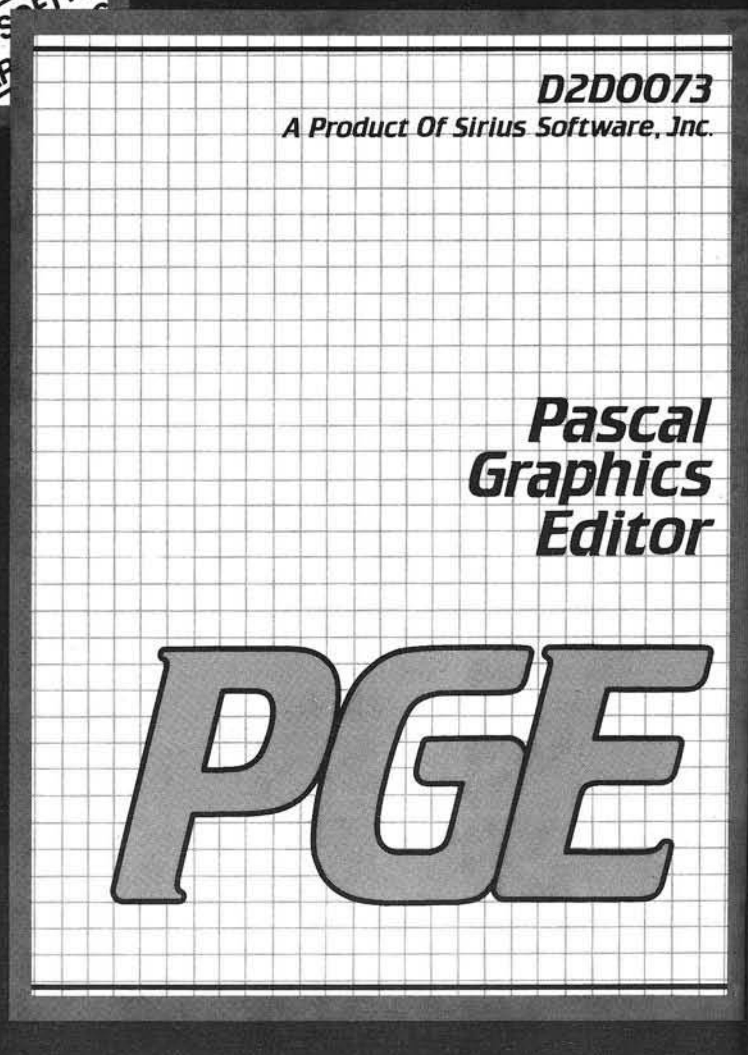
Phantoms Five simulates a fighter-bomber mission in real time, three dimensional color graphics. While you try to make your bombing run, you have to avoid being hit by anti-aircraft fire and fight off enemy aircraft as well.

?

Hatch some fun with the Spiders, Wolves, Lips, and Fuzzballs. Space Eggs will crack you up! Each package includes a multi-color T-shirt iron-on that says "I FRIED THE SPACE EGGS."



This is the graphics editing package we based our business on. Includes the Higher Text Character Generator by Ron & Darrell Aldrich and over 20 original and imaginative type styles.



The professional graphics editing package for use within the Pascal environment.

GORGON is here...

THE EARTH HAS ENTERED A TIME WARP . . . AND THE BATTLE HAS JUST BEGUN

From Here to Atari

James Capparell
297 Missouri Street
San Francisco, California 94107

What better place to discuss Atari features than in an issue devoted to games? Actually, I'm hesitant to draw the Atari-games parallel. The point that it is a premiere graphics machine has clouded the fact that it is a flexible, easy-to-use computer system with features and capability appealing to a diverse user group. And, the Atari has an installed user base estimated over 50,000.

Why do I believe that this equipment is ahead of its competitors? I feel its strongest features, relative to game programming, are sixteen graphics modes (high resolution 320 × 192), two direct memory access (DMA) video channels (sort of a simplified multiprocessing system), display list controlled memory mapped graphics, redefinable character sets, hooks for vertical blank interrupts and scan line interrupts, and of course four channels of sound (silent games are dull).

Mastering these features takes some time. Currently there are two essential manuals available from Atari: *The Operating Systems User Guide* and *Hardware Manual* #C016555. The cost for both is thirty dollars and worth every penny.

This month I would like to discuss one feature of the display list, smooth scrolling of the screen image. The Atari maps its memory to video *via* a LSI chip called ANTIC. This chip is a dedicated processor with its own instruction set. These instructions make up what is called a display list. The display list controls the graphics mode which will be displayed on the screen. Recall that there are sixteen modes, each specifying memory use, resolution and color. The display list tells ANTIC what part of the 6502 memory space to display, what mode to display, whether an interrupt should be generated, and whether horizontal and/or vertical scrolling should be enabled. It is this last feature which will be demonstrated.

There are two methods which can be used to scroll the image. The first is direct and easy to comprehend. The display list has, as part of its instructions, a feature called Load Memory Scan (LMS). This operator is three bytes long. The last two bytes are the address (low-high bytes, 6502 style) of the start of display memory. As a result, the entire address space is available for display under program control. This gives the observer a 'window' into memory. Scrolling windows are created by simply changing the two address bytes of the LMS. In other words, it is not data being moved through memory, but a window moving across the data residing in memory which causes the image to scroll.

Program 1 should give a good idea of 'coarse' vertical scrolling. I call it coarse since the image moves a full character space at a time. Lines 170 and 180 are really doing all the dirty work. The new display address is being inserted into the display list at this point after appropriate incrementing or decrementing of the address bytes. I've chosen to vertically scroll the entire image but it is an easy matter to set up a scrolling window within a background display. In fact, program 2 does just that, only in the horizontal direction.

I've also mixed two modes on the screen. The only complication here is the need to have more than one LMS instruction. The second LMS restores the pointer to memory prior to the horizontal intrusion. There is nothing to stop you from placing an LMS instruction on every mode line; each could be scrolling in independent directions.

Program 3 is meant to demonstrate the second scrolling method, smooth or fine scrolling. This is accomplished with the help of hardware scrolling registers, one for horizontal and another for vertical direction. When the appropriate bits are set in a display list instruction, the values in each of these registers control the amount of scan lines vertically or color clocks horizontally that each line will be displaced.

Listing 1

```
10 REM ** PROG3 **  FINE SCROLLING HORIZONTALLY AND VERICALLY
20 DLST=PEEK(560)+256*PEEK(561)
25 DMEM=PEEK(DLST+4)+PEEK(DLST+5)*256
30 SKIPH=INT((DMEM+280)/256):SKIPL=DMEM+280-SKIPH*256
35 VALL=0:VALH=2
40 POKE DLST+12,119:POKE DLST+13,VALL:POKE DLST+14,VALH
45 POKE DLST+15,66:POKE DLST+16,SKIPL:POKE DLST+17,SKIPH
50 IF PEEK(764)=255 THEN GOTO 50:REM SCAN KEYBOARD
55 IF PEEK(764)=14 THEN POKE 764,255:GOTO 200:REM UP ARROW ?
60 IF PEEK(764)=15 THEN POKE 764,255:GOTO 250:REM DOWN ARROW ?
65 IF PEEK(764)=6 THEN GOTO 300:REM LEFT ARROW ?
70 IF PEEK(764)=7 THEN GOTO 350:REM RIGHT ARROW ?
75 GOTO 50:REM IGNORE OTHER RESPONSES
200 Y=Y+1:IF Y<16 THEN GOTO 500
210 Y=0
215 VALL=VALL+40
220 IF VALL>240 THEN VALL=0:VALH=VALH+1
230 GOTO 450
250 Y=Y-1
255 IF Y>-1 THEN GOTO 500
260 Y=15
265 VALL=VALL-40
280 GOTO 445
300 X=X-1:IF X>-1 THEN GOTO 505
305 X=15
310 VALL=PEEK(DLST+13)+2
325 GOTO 445
350 X=X+1:IF X<16 THEN GOTO 505
355 X=0
360 VALL=PEEK(DLST+13)-2
440 IF VALL<0 THEN VALL=0:VALH=VALH-1
445 IF VALH<0 THEN VALH=0
450 POKE DLST+12,119:POKE DLST+13,VALL:POKE DLST+14,VALH
500 POKE 54277,Y:REM VERTICAL SCROLL REGISTER
505 POKE 54276,X:REM HORIZONTAL SCROLL REGISTER
510 GOTO 50
```

The limitation here is the amount of fine scrolling allowed. A line can be moved eight full color clocks horizontally and 16 scan lines vertically. When this amount is scrolled, the LMS address bytes must be incremented or decremented and the whole process must be started again. In this way smooth scrolling can be maintained.

The previously mentioned manuals are a necessity for commercial programmers. This machine has been completely disclosed and it's up to us to begin using these features.

Currently three games make full use of Atari graphics. They are *Missile Command* by Atari Inc., *Jawbreaker* by On-Line Systems, and *Dodgeracer* by Synapse Software. These three games use the graphics capability of this equipment and approach arcade level polish and style. Synapse also produces *Filemanager 800*, a database management package that's so easy to use and makes such excellent use of mixed mode displays that it's going to become a standard for emulation.

One other program that I must mention is *Eastern Front*, available from Atari Program Exchange (APX). This is a strategy war game that makes excellent use of smooth scroll technique. This game is a virtuoso performance of programming skill and probably excercises the internal features of the Atari more than any other product on the market.

I hope your curiosity has been stimulated. These techniques are just the beginning; I intend to offer more ideas and help over the coming months.

MICRO

Listing 2

```
10 REM ** PROG1 ** COARSE VERTICAL SCROLLING DEMO
15 REM PRESS UP/DOWN ARROWS TO MOVE DISPLAT THRU MEMORY
20 DLIST=PEEK(560)+PEEK(561)*256:REM GET START OF DISPLAY LIST
30 LMSL=DLIST+4:REM POINTER TO DISPLAY MEMORY
40 LMSH=DLIST+5
50 DISPLAYL=0:DISPLAYH=0:REM INITIALIZE ADDRESS OF DISPLAY MEMORY
55 REM READ KEYBOARD
60 IF PEEK(764)=255 THEN GOTO 60:REM WAIT FOR KEY
70 IF PEEK(764)=14 THEN POKE 764,255:GOTO 110:REM UP ARROW ?
80 IF PEEK(764)=15 THEN POKE 764,255:GOTO 140:REM DOWN ARROW ?
90 GOTO 60
100 REM MOVE DISPLAY WINDOW INTO LOWER MEMORY
110 DISPLAYL=DISPLAYL-40
120 IF DISPLAYL>=0 THEN GOTO 170:REM CAN'T DISPLAY NEGATIVE MEMORY
122 DISPLAYH=DISPLAYH-1:DISPLAYL=0
124 IF DISPLAYH<0 THEN DISPLAYH=0
126 GOTO 170
130 REM MOVE DISPLAY WINDOW INTO HIGHER MEMORY
140 DISPLAYL=DISPLAYL+40
150 IF DISPLAYL>240 THEN DISPLAYH=DISPLAYH+1:DISPLAYL=0
160 REM CHANGE DISPLAY MEMORY POINTER
170 POKE LMSL,DISPLAYL:REM PUT NEW DIPLAY ADDR IN DISPLAY LIST
180 POKE LMSH,DISPLAYH
200 GOTO 60:REM GO WAIT FOR KEYBOARD ENTRY
```

Listing 3

```
10 REM ** PROG2 ** COARSE HORIZONTAL SCROLLING DEMO
20 REM USE LEFT AND RIGHT POINTING ARROWS TO CONTROL SCROLL DIRECTION
25 LIST
30 DLST=PEEK(561)*256+PEEK(560)
35 DMEM=PEEK(DLST+4)+PEEK(DLST+5)*256
40 SKIPH=INT((DMEM+280)/256):SKIPL=DMEM+280-SKIPH*256
45 POKE DLST+15,66:POKE DLST+16,SKIPL:POKE DLST+17,SKIPH
50 ADDRL=DLST+13:ADDRH=DLST+14:VALL=0:VALH=3
55 POKE DLST+12,71:POKE ADDRL,VALL:POKE ADDRH,VALH
60 IF PEEK(764)=255 THEN GOTO 60:REM SCAN KE
65 IF PEEK(764)=7 THEN POKE 764,255:GOTO 100:REM RIGHT ARROW ?
70 IF PEEK(764)=6 THEN POKE 764,255:GOTO 140:REM LEFT ARROW ?
80 GOTO 50:REM ONLY ARROWS ARE LEGAL RESPONSE
90 REM SCROLL RIGHT
100 VALL=PEEK(DLST+13)+1:REM MOVE DISPLAY TO LEFT
110 IF VALL>255 THEN VALL=0:VALH=VALH+1
120 GOTO 55
130 REM SCROLL LEFT
140 VALL=PEEK(DLST+13)-1:REM MOVE DISPLAY TO RIGHT
150 IF VALL<0 THEN VALL=0:VALH=VALH-1
160 IF VALH<0 THEN VALH=0
170 GOTO 55
```

Lunar Lander

Animated Graphics in BASIC for the Color Computer

The TRS-80 Color Computer was designed for color graphics from the ground up. This program uses high-speed animation techniques available from BASIC for the Color Computer. Presented are several ideas and techniques that make the animation process easier or more realistic.

John Steiner
508 Fourth Ave. NW
Riverside, North Dakota 58078

The Color Computer was conceived with graphics as a central feature of its design. With Microsoft's powerful Extended BASIC, high speed animation is a reality without machine language. Using an old standby, Lunar Lander, I will demonstrate some of these techniques and how they are used. While techniques shown here are not as high speed as is possible with the system, they show the potential that is possible in a BASIC language program. See chart 1 for a description of how to prepare the Color Computer for graphics display.

Chart 1: **Graphics**

Initialization

A. Reserve Memory Pages
B. Choose Desired Mode
C. Choose Screen Color Set
D. Select Foreground and Background Colors

Table 1

PMODE	Pages Required
0	1
1	2
2	2
3	4
4	4

The first step is to clear the correct number of pages required for the program graphics displays. Next select the desired mode. This is done using the PMODE command, and each mode determines colors available and starting page of the graphics display. Next select the color set desired. This choice is dependent upon the PMODE, SCREEN and COLOR commands. (See my previous article, "The Radio Shack Color Computer" in last month's issue for more on this process.)

If you refer to table 1, you'll notice that the higher the resolution, and greater the color combination, the more pages of video memory are required. Each "page" of memory requires a minimum of 1.5K, and in Hi-Res, four pages are required to display one picture on screen. By using more than one page of graphics, the programmer can have several displays to be called when required. The Lunar Lander program has three separate background displays. Each display is drawn at the beginning of the game, and is called into view as it is needed. By drawing them in the beginning, the program does not have to stop and wait for the graphics to be drawn. In PMODE 0, since only one page is required, it is possible to store eight different displays for callup.

It is possible to draw images in memory without the drawing being visible to the player, and this technique is used in Lander. The SCREEN command calls the screen to be viewed, and by calling the text screen, or one of the previously drawn or empty pages, the drawings and placement of objects will not be visible.

The program gives directions, and then asks for a difficulty level. After obtaining the input from the player, a "standby" message is printed on the text screen. While the viewer waits, the program selects the correct pages, and draws the individual images on each page. This will be explained in detail later. However, one comment here is in order. If there is no need for a difficulty level or other player input, the instruction screen can be displayed while the graphics are being drawn. Upon completion of the draw routines, the "PRESS FIRE BUTTON TO CONTINUE" message appears, and the game can begin. It takes about five seconds to draw the screens.

This program is patterned after an arcade game. The lander is travelling horizontally and vertically toward the lunar surface, and the object is, of course, to bring the craft to a safe landing. As the craft nears the surface, the screen display switches, and a closeup view of the lander and immediate terrain is shown. Now the player must land safely, without too much speed, and on level ground. The program is well remarked, and the remarks may be left in if desired. The gravity calculation is not scientifically accurate, and you may experiment with it to determine what appeals to you. The program sets VV (vertical veloctiy) in line 370.

Selecting and drawing the landscape is your first order of business. The landscape is stored in two strings, L$ and R$. To draw the long range view of the landscape, PMODE 2,1 is selected. The 2 indicates the resolution and memory requirements, and the 1 indicates the page the information will begin drawing on. PMODE 2 requires two pages to display the entire screen, and since there will be a long range view combining L$ and R$ as well as two closeup views, L$ and R$

respectively, we will need six pages. Line 10 clears six pages and sets the mode. Line 20 clears the first screen, and calls the text screen to be viewed, to display the instructions.

B$ is a lower border, and may be left out if desired. The PAINT command colors the landscape, while RS$ is the landing craft, which is drawn in the upper lefthand corner of the screen. Next the ship is stored as an array using the GET command. (More on this later.)

After drawing the long range landscape, the program "turns the page" in line 110. The mode is still 2, but we have now selected starting page 3. A PCLS command clears the memory to background color, and the left landscape is drawn. One of the most useful options of the DRAW command, 'S'cale, is used to increase the length of L$ to fill the lower portion of the display with only the left portion of the landscape. Keep in mind that all this is going on while the "standby" message is being displayed.

The scale of the landing craft is also increased, and now is so large that the craft must be "PAINTED." After drawing and painting the left landscape, the pagestart is again changed. Line 150 selects PMODE 2,5, and clears the screen to begin drawing the right half of the terrain. The rocket is drawn in the upper left corner of this screen also. Once the craft gets close enough to the surface, if it is in the left half of the screen, page 3 is called, otherwise page 5 is called. In order to simplify the page changing routine, the craft is drawn on both pages. Once entering either page 3 or page 5, the program will eventually end in this mode. It is not possible to return to the long range view as the program is written. If you send the rocket to the top of the screen, it will stay there, because of a limiter on the vertical command.

I could have gotten more realistic and accurate closeups of the ground by using a middle landscape string. This requires two more pages of memory to store the landscape, and that does not leave enough memory for the program. One solution would be to clear pages 5 and 6 and quickly redraw the center landscape, should it be required. There is a pause as the paint commands are executed. In the interest of keeping action as fast as possible, I chose to have only a left and right option. After calling the new start page, the new landscape appears by executing the SCREEN command.

***Listing 1:* Lunar Lander Program**

```
10 PCLEAR6:PMODE2,1
20 PCLS:SCREEN0,1
30 GOSUB760
35 'LANDING CRAFT
40 DIMLC(19,20)
45 'LANDSCAPE
50 L$="R8E4R2E4R3E2R12F3R2F4R2F6R2E8R12E4R12R15E4R2E2R16F8"
60 R$="BM+0,+0;R1E4R12F2R1E2R10R12E4R1E4R1E4R2E3R2E5R3E6R3E5R2E7
   R3U2E3R1E4U2R2E4
R3
70 B$="S4;BM1,190;R253
75 'LANDING CRAFT
80 RS$="F2D3H2G2U3E2"
85 'DRAW LANDSCAPE
90 DRAW"C5;BM1,180;XL$;XR$;XB$;"
100 PAINT(250,180),5,5
110 PMODE2,3:PCLS
115 'DRAW LEFT LANDSCAPE
120 DRAW"S10;BM1,179;XL$;XB$;"
130 PAINT(100,170),5,5
140 DRAW"S12;BM40,40;XRS$;":PAINT(45,45),5,5
150 PMODE2,5:PCLS
155 'DRAW RIGHT LANDSCAPE
160 DRAW"S10;BM1,180;XR$;XB$;"
170 PAINT(190,180),5,5
180 DRAW"S12;BM40,40;XRS$;":PAINT(45,45),5,5
190 PMODE2,1
195 'DRAW SMALL CRAFT
200 DRAW"S4;BM20,20;XRS$;":DRAW"BM20,20;D3"
205 'STORE SMALL CRAFT
210 GET(10,15)-(30,36),LC,G
215 'INITIALIZE
220 DI=0:H=10:V=15:HV=3:VV=1
230 PRINT@448,"PRESS FIRE BUTTON TO CONTINUE.
240 P=PEEK(65280):IFP<>126 AND P<>254 THEN240
245 'CALL LARGE LANDSCAPE
250 SCREEN1,1
255 'UPDATE VELOCITY
260 A=JOYSTK(0):B=JOYSTK(1)
265 'SET TO NEW POSITION
270 IFA<16THENHV=HV-1:IFHV<-1THENHV=-1
280 IFA>48THENHV=HV+1:IFHV>3THENHV=3
290 IFB<16THENVV=VV-DF:IFVV<-3THENVV=-3
310 H=H+HV:IFH<0THENH=0
320 IFH>230THENH=230
330 V=V+VV:IFV<0THENV=0
335 'CLOSE ENOUGH TO LAND?
340 IFPPOINT(H+5,V+25)=5 AND DI=0THENGOSUB400
345 'CHECK FOR TOUCHDOWN
350 IF PPOINT(H+19,V+22)=5 AND DI=1THEN570
360 IFPPOINT(H+6,V+22)=5 AND DI=1THEN570
365 'IF NOT, INCREASE SPEED, PUT NEW POSITION
370 VV=VV+.2:IFVV>3THENVV=3
380 PUT(H,V)-(H+20,V+21),LC,PSET
390 GOTO260
395 'SELECT LANDING FIELD
400 DI=1:IFH<126THEN420
410 IFH>125THEN490
415 'CALL LEFT LANDSCAPE
420 PMODE2,3
425 'STORE LARGE CRAFT
430 GET(30,37)-(50,58),LC,G
440 PUT(30,37)-(50,58),LC,NOT
450 PUT(30,37)-(50,58),LC,AND
460 H=H*2:IFH>230THENH=230
470 V=(V+10)/2
480 SCREEN1,1:RETURN
485 'CALL RIGHT LANDSCAPE
490 PMODE2,5
495 'STORE LARGE CRAFT
500 GET(30,37)-(50,58),LC,G
510 PUT(30,37)-(50,58),LC,NOT
520 PUT(30,37)-(50,58),LC,AND
530 IFH<180THENH=H/2
540 IFH<80THENH=H-40
550 V=(V/2)-10
560 SCREEN1,1:RETURN
565 'CHECK FOR LEVEL GROUND
570 IFPPOINT(H+18,V+22)<>5 AND PPOINT(H+2,V+22)=5 THENLG=1ELSELG=0
580 IFPPOINT(H+18,V+22)=5 AND PPOINT(H+2,V+22)<>5 THEN LG=1ELSELG=0
585 'SETS CRAFT DOWN
590 FORI=1TOVV+4
600 PUT(H,V)-(H+20,V+21),LC,OR
610 V=V+1
620 NEXT
630 FORI=1TO500:NEXT
640 IFVV<=0THENVV=1
645 'FINISH ROUTINE
650 IFLG=1THEN700
660 ON VV GOTO670,680,690
```

```
670 CLS0:PRINT@224,"GOOD LANDING":GOTO710
680 CLS0:PRINT@224,"NOT SO HOT, BUDDY, YOU REALLY":PRINT"SHOOK 'EM UP!!"
    :GOTO710
690 CLS0:PRINT@224,"THERE WERE NO SURVIVORS!":GOTO710
700 CLS0:PRINT@224,"YOU'VE GOT TO LAND ON LEVEL":PRINT"GROUND."
    :PRINT"THERE WERE  NO SURVIVORS!
710 PRINT@448,"TO PLAY AGAIN, PRESS FIRE BUTTON
720 FORI=1TO500
730 A=PEEK(65280):IFA=126 OR A=254 THEN RUN
740 NEXT
745 'RESTORE POWER UP CONDITION
750 CLEAR200:PMODE0,1:PCLEAR4:END
755 'TITLE SUBROUTINE
760 CLS:PRINT@10,"LUNAR LANDER
770 PRINT"YOUR COMPUTER HAS AN URGENT":PRINT"MESSAGE!
780 PRINT"THE CREW OF THE LANDING CRAFT":PRINT"HAS REPORTED AUTO-DESCENT
790 PRINT"EQUIPMENT FAILURE! YOU MUST":PRINT"CONTROL THE DESCENT WITH MANUAL
800 PRINT"FACILITIES. USE THE RIGHT
810 PRINT"JOYSTICK TO VARY THE CRAFT'S":PRINT"SPEED. BRING 'EM HOME SAFELY!
820 PRINT:PRINT"ENTER DIFFICULTY":PRINT"ENTER 1(NOVICE) TO 3(EXPERT)"
830 A$=INKEY$:IFA$=""THEN830
840 IFA$<"1"ORA$>"3"THEN830
850 DF=VAL(A$):CLS:PRINT@224,"STANDBY":RETURN
```

In other words, the PMODE does not call the display to be viewed, it only prepares it for display. Select the mode and startpage, then call the screen. Remember SCREEN has four options, which are listed in table 2. The MODE and page select options bring a lot of versatility to the graphics display. If the two-color mode doesn't appeal to you, try a lower res mode. PMODE 1 is a four-color mode, and you can choose different colors if desired, for landscape and craft. The PAINT commands, however, will have to be modified, as well. Due to the different colors involved, you cannot just change the mode.

The secret behind high-speed animation in BASIC is the GET and PUT set of commands. GET specifies a double dimensioned array that literally stores the color numbers of each pixel in the specified area. GET(10,10)-(30,30),LC stores a rectangle 20 × 20 square. The array must be dimensioned properly in the beginning. Once the array is stored in memory, it can be PUT anywhere on the screen.

PUT(50,50)-(70,70),LC puts the array back on the screen at the new coordinates. Notice that there are now two objects on the screen. GET does not remove the screen display, so other techniques must be applied to remove it. In Lander the craft is in the exact center of the display rectangle. By limiting the display movement to only a few units, the display will be overwritten by the new PUT command. This results in relatively high speed. However, if you store a background array (an array that matches the background color) and PUT it over the old array position, the new position can be anywhere you want it. Speed is not actually increased, but the illusion of speed is enhanced greatly.

It is possible to use a single array to store more than one object. This technique is used here, as there is no room for two arrays, for the large and small landers respectively. A new GET command, specifying the same array name and size (in this case, LC), is executed to store the image of the large craft when the small craft nears the surface. Just after calling the correct PMODE, and before executing the SCREEN command, the large craft is stored. The lander must be removed from the left corner of the screen so that it may be placed in its approximate relative position to the surface. This is accomplished using the graphics option of the GET command.

To select the option, just add a ,G suffix. When the G is used, the correct suffix for PUT must be included. Options are listed in table 3. Lines 430 to 450 or 500 to 520 store the craft, PUT the craft back using the NOT option, and PUT it back again using the AND option. To see how this works, let's look at what is happening.

Table 2: Screen Choice

SCREEN 0,X text modes

SCREEN 1,0

- Two-color mode: black/green
- Four-color mode: blue/red/yellow/green

SCREEN 1,1

- Two-color mode: black/buff
- Four-color mode: buff/cyan/magenta/orange

A PUT using the NOT option reverses the logic of the destination rectangle. What was set in the location is reset on the screen, and *vice versa*. What is placed in the display is the exact opposite of the landing craft and background. Next, using the AND option, we PUT the craft in the same location again. The AND option sets an array point only if it was previously set in both the array AND the screen location chosen. Since we PUT an exact opposite of the array using the NOT, there are no locations that are set in the stored array AND display location. Therefore all points are reset, and the craft disappears. We cannot see this happening, because the SCREEN command has not yet been executed.

Now that the new craft has been safely stored, and removed from the screen, we can modify the variables to correspond to the new locations, and continue. The graphics options are required in all two-color modes, by the way. Due to the internal structure of the language, if this option is not used, you will not always get what you PUT on the screen.

The logic that determines velocity and placement of the craft is in lines 250 to 410. The right joystick is read, and lateral velocity determined from the horizontal reading. Vertical velocity is determined in line 370 by gravity, as well as by the vertical joystick reading. The difficulty level, DF, is the upward vertical velocity. It roughly corresponds to the thrust of the rocket. The higher difficulty level moves the rocket faster vertically up the screen. Three is the upper limit of motion, as a number higher than this will leave traces of the previous position on the screen. Lines 340 to 360 check for landscape. Specifically, line 340 selects the page switch subroutine when the craft ap-

Table 3: Graphics Options

PSET	Sets dot on screen to match array
PRESET	Resets dot on screen that was set in array
AND	Sets dot on screen only when screen and array are set
OR	Sets dot on screen when either array or screen is set
NOT	Reverses the state of screen whether array was set or reset

proaches the surface. DI is set to 1 in the subroutine, and this line is ignored during the last part of the program.

As the ship is about to touch down, lines 350 and 360 check the left and right sides of the craft looking for terrain. If terrain is found, the program goes to a set down routine. This routine first checks to see if the ground is level, then a FOR-NEXT loop sets the craft down. Line 600 uses the OR option to set the craft down, as it descends, without erasing the terrain. The OR option sets a screen point, if the point in the array is set OR the point on the screen is set.

Velocity, VV, is added to the loop to give a demonstration of the danger of landing at too high a velocity. If the landing is perfect the craft settles to the top of the lunar surface. If VV is increased, the craft settles deeper and deeper into the lunar dust. A velocity of 3 is fatal to the occupants, as is landing on uneven ground. There are ending messages corresponding to the nature of the landing, and a request to play again.

The first time this program is run, there will appear an OK message after the STANDBY message is displayed. This problem is due to the initialization of Extended Color BASIC's stack. Just type RUN again, and the program will work.

Using Extended BASIC for graphics is an easy process to learn, and I hope I have shown some useful techniques in animation. If you have any questions, write to me and include a stamped self-addressed envelope.

Happy Landings.

John Steiner is an electronics instructor in the Fargo, North Dakota, school system. Before this he worked as an Audio and Communications technician in consumer and business electronics. He owns a color computer and is waiting for a disk system and other peripherals.

MICRO™

Galacti-Cube

You are the Captain of a starship exploring the outer limits of our universe. You have discovered a gigantic cube floating in space. Through the only opening you have flown your ship inside, but now you can't find your way back out!

Bob Bishop
155 Michael Lane
Santa Cruz, California 95060

GALACTI-CUBE is a simple maze game in three dimensions. You are in a 3 × 3 × 3 array of cubical compartments and must find your way out in no more than 40 moves, or else you lose. Moves are made by hitting the keys N, S, E, W, U, or D to move north, south, east, west, up or down, respectively. Although it appears small, a 3 × 3 × 3 cubical maze actually has 27 rooms in it, which can make the task of finding your way through deceptively non-trivial.

The program is written entirely in Apple II Integer BASIC and requires at least 8K bytes of memory. In fact, since the program uses no machine language, graphics, or special sound effects, it could probably be converted over to other CRT-type computers (such as the PET, TRS-80, etc.) without too much difficulty.

A few words about some obvious cautions might be in order. The program assumes that the text screen is the standard 24 × 40 Apple II screen. The PRINT statements in the program (especially around lines 5000-5900) must be entered carefully with exactly the specified messages and number of spaces as shown in the listing, or else things might not line up properly on the screen. Also, you might want to include a liberal amount of CTRL-Gs (bells) in some of the print statements, such as in lines 560-580. (In my version, I have put a bell between each letter of the message.) In other places, like line 380, I have explicitly indicated a bell *via* a REM statement.

```
10 REM *** GALACTI-CUBE ***
20 REM  R. J. BISHOP
30 DIM BOX(27),QUE(27),NODE(6),BIT(6),A$(5)
40 GOSUB 9000
50 GOSUB 1000
60 VTAB 23: TAB 5: PRINT "(HIT ANY KEY TO START THE GAME) ";
70 GOSUB 4000: GOSUB 5000
90 LOC=14:OLD=LOC:FUEL=40
100 REM MAIN LOOP
110 GOSUB 2000
150 CALL -936: PRINT : PRINT : PRINT "   COMMAND:"
160 PRINT : TAB 7: GOSUB 4000: CALL -936
165 IF A$="" THEN 150
170 IF A$(1,1)#"F" THEN 250
180 CALL -936: PRINT : PRINT "  YOU HAVE ";FUEL
190 PRINT : PRINT "  FUEL UNITS"
210 FOR K=1 TO 1000: NEXT K: GOTO 150
250 Z=(OLD-1)/9+1
260 Y=(((OLD-1)/3) MOD 3)+1
270 X=((OLD-1) MOD 3)+1
300 IF A$="E" THEN X=X+1
310 IF A$="W" THEN X=X-1
320 IF A$="N" THEN Y=Y+1
330 IF A$="S" THEN Y=Y-1
340 IF A$="U" THEN Z=Z+1
350 IF A$="D" THEN Z=Z-1
360 LOC=X+3*(Y-1)+9*(Z-1)
370 IF LOC<>OLD THEN 390
380 PRINT "": GOTO 150:REM CONTROL-G
390 IF X<1 OR X>3 OR Y<1 OR Y>3 THEN 700
400 IF BOX(OLD)>=32 AND Z=0 THEN 800
410 VAL=BOX(OLD): IF VAL>=32 THEN VAL=VAL-32
420 IF VAL>=16 AND Z=4 THEN 800
430 IF Z<1 OR Z>3 THEN 700
450 BITS=BOX(OLD)
460 WAY=BITS-2*(BITS/2):BITS=BITS/2
470 IF WAY=0 AND A$="E" THEN 700
480 WAY=BITS-2*(BITS/2):BITS=BITS/2
490 IF WAY=0 AND A$="W" THEN 700
500 WAY=BITS-2*(BITS/2):BITS=BITS/2
505 IF WAY=0 AND A$="N" THEN 700
510 WAY=BITS-2*(BITS/2):BITS=BITS/2
515 IF WAY=0 AND A$="S" THEN 700
520 WAY=BITS-2*(BITS/2):BITS=BITS/2
525 IF WAY=0 AND A$="U" THEN 700
530 WAY=BITS-2*(BITS/2):BITS=BITS/2
535 IF WAY=0 AND A$="D" THEN 700
540 WAY=BITS-2*(BITS/2):BITS=BITS/2
550 FUEL=FUEL-1: IF FUEL>0 THEN 100
560 CALL -936: PRINT "   YOU ARE"
565 PRINT
570 PRINT "   OUT OF"
575 PRINT
580 PRINT "    FUEL!";
590 GOTO 830
700 CALL -936: PRINT " THAT DIREC-"
710 PRINT : PRINT " TION HAS AN"
720 PRINT : PRINT " OBSTRUCTION";
730 FOR K=1 TO 1000: NEXT K: GOTO 150
800 CALL -936: PRINT "YOU FOUND THE"
810 PRINT : PRINT " EXIT IN ONLY"
820 PRINT : PRINT "  ";41-FUEL;" MOVES!";
830 GOSUB 2700
840 FOR K=1 TO 2500: NEXT K
850 CALL -936: END
900 END
1000 REM GENERATE THE MAZE
1010 FOR K=1 TO 27
1020 BOX(K)=128
1030 NEXT K
1040 BOX(14)=0
1050 QUE(1)=14:QBIG=1
1060 XQBIG=1
1100 FOR K=1 TO QBIG
1110 IND=QUE(K)
1140 KNT=0:ROAD=1:DEL=1
1150 FOR J=0 TO 2
1160 SET=3*DEL
1170 FOR L=0 TO 1
1180 NDX=IND+DEL
1190 IF NDX<1 THEN 1400
1200 IF (NDX-1)/SET<>(IND-1)/SET THEN 1400
1250 IF BOX(NDX)<128 THEN 1400
1300 KNT=KNT+1:NODE(KNT)=NDX:BIT(KNT)=ROAD
1400 DEL=-DEL:ROAD=ROAD+ROAD
1450 NEXT L
1460 DEL=SET
1470 NEXT J
1500 IF KNT=0 THEN 1600
1510 NDX= RND (KNT)+1:XQBIG=XQBIG+1
1520 QUE(XQBIG)=NODE(NDX)
1530 BOX(IND)=BOX(IND)+BIT(NDX)
1540 TIB=2*BIT(NDX)
1550 IF TIB=4 OR TIB=16 OR TIB=64 THEN TIB=TIB/4
1590 BOX(NODE(NDX))=BOX(NODE(NDX))+TIB-128
1600 NEXT K
1610 QBIG=XQBIG: IF QBIG<27 THEN 1100
```


(Continued on next page)

```
1700 HOLE=2* RND (2)+6* RND (2)+18* RND (2)+1
1710 OPEN=16: IF HOLE<14 THEN OPEN=32
1720 BOX(HOLE)=BOX(HOLE)+OPEN
1800 RETURN
2000 REM UPDATE THE DISPLAY
2005 GOSUB 2700
2010 Z=(OLD-1)/9+1
2020 Y=(((OLD-1)/3) MOD 3)+1
2030 X=((OLD-1) MOD 3)+1
2040 VTAB 13-Y-Y
2050 TAB 8*Z+X+X-7
2060 PRINT "-"
2110 Z=(LOC-1)/9+1
2120 Y=(((LOC-1)/3) MOD 3)+1
2130 X=((LOC-1) MOD 3)+1
2140 VTAB 13-Y-Y
2150 TAB 8*Z+X+X-7
2170 POKE PEEK (36)+ PEEK (40)+256* PEEK (41),109
2200 BITS=BOX(LOC)
2210 VT=20:T=34:A$="EAST": GOSUB 2500
2220 VT=22:T=34:A$="WEST": GOSUB 2500
2230 VT=20:T=28:A$="NORTH": GOSUB 2500
2240 VT=22:T=28:A$="SOUTH": GOSUB 2500
2250 VT=20:T=24:A$="UP": GOSUB 2500
2260 VT=22:T=23:A$="DOWN": GOSUB 2500
2300 GOSUB 2600
2400 OLD=LOC
2450 RETURN
2500 WAY=BITS-2*(BITS/2):BITS=BITS/2
2510 MODE=127: IF WAY THEN MODE=255
2520 POKE 50,MODE: VTAB VT: TAB T: PRINT A$: POKE 50,255
2550 RETURN
2600 VTAB 19: TAB 5
2610 POKE 32,2
2630 POKE 33,14
2660 POKE 34,17
2680 POKE 35,22
```



```
2690 RETURN
2700 POKE 32,0
2710 POKE 33,40
2720 POKE 34,0
2730 POKE 35,24
2750 RETURN
4000 REM 'GET' FROM THE KEYBOARD
4010 POKE -16368,0
4020 CHAR= PEEK (-16384): IF CHAR<128 THEN 4020
4030 POKE -16368,0:A$="?"
4080 IF CHAR=141 THEN A$=""
4090 IF CHAR=196 THEN A$="D"
4100 IF CHAR=197 THEN A$="E"
4110 IF CHAR=198 THEN A$="F"
4120 IF CHAR=206 THEN A$="N"
4130 IF CHAR=211 THEN A$="S"
4140 IF CHAR=213 THEN A$="U"
4150 IF CHAR=215 THEN A$="W"
4200 RETURN
5000 REM DRAW DISPLAY
5010 CALL -936: PRINT "      YOUR LOCATION            COMPASS"

5020 PRINT : PRINT "  (BOT)   (MID)   (TOP)        REFERENCE"

5030 PRINT : TAB 34: PRINT "N"
5040 PRINT : TAB 34: PRINT "!"
5050 TAB 34: PRINT "!"
5060 TAB 29: PRINT "W <--*--> E"
5070 TAB 34: PRINT "!"
5080 TAB 34: PRINT "!"
5090 PRINT : TAB 34: PRINT "S"
5100 VTAB 6
5110 FOR K=1 TO 3
5120 PRINT : PRINT "  - - -   - - -   - - -"
5130 NEXT K
5140 VTAB 16: TAB 21: PRINT "OBSTRUCTION SENSORS"
5200 POKE 50,63
5210 VTAB 5: PRINT "                          "
5220 FOR K=1 TO 7
5230 PRINT " ";: TAB 9: PRINT " ";: TAB 17: PRINT " ";:
     TAB 25: PRINT " "
5240 NEXT K
5250 PRINT "                          "
5300 VTAB 18: TAB 21: PRINT "                   "
5310 FOR K=1 TO 5
5320 TAB 21: PRINT " ";: TAB 39: PRINT " "
5330 NEXT K
5340 TAB 21: PRINT "                   ";

5400 VTAB 15: PRINT
5410 PRINT "                  "
5420 FOR K=1 TO 7
5430 PRINT " ";: TAB 18: PRINT " "
5440 NEXT K
5450 PRINT "                  ";
5500 POKE 50,255
5900 RETURN
9000 CALL -936: VTAB 10
9010 TAB 10: PRINT "*** GALACTI-CUBE ***"
9020 PRINT : TAB 19: PRINT "BY"
9030 PRINT : TAB 14: PRINT "ROBERT BISHOP"
9040 FOR K=1 TO 1500: NEXT K
9050 CALL -936
9110 PRINT "     YOU ARE THE CAPTAIN OF A STAR-SHIP"
9120 PRINT "EXPLORING THE OUTER LIMITS OF OUR UNI-"
9130 PRINT "VERSE.  YOU HAVE DISCOVERED A GIGANTIC"
9140 PRINT "CUBE FLOATING IN SPACE.  THROUGH THE"
9150 PRINT "ONLY OPENING YOU HAVE FLOWN YOUR SHIP"
9160 PRINT "INSIDE, BUT NOW YOU CAN'T FIND YOUR WAY"
9170 PRINT "BACK OUT!"
9190 PRINT "     FROM YOUR EXPLORATIONS YOU HAVE"
9200 PRINT "LEARNED THAT THE CUBE IS DIVIDED INTO"
9210 PRINT "AN ARRAY OF 3X3X3 CUBICAL COMPARTMENTS"
9220 PRINT "AND YOU ARE CURRENTLY IN THE CENTER-"
9230 PRINT "MOST ONE."
9250 PRINT "     YOUR SHIP IS EQUIPPED WITH A DIS-"
9260 PRINT "PLAY INDICATING YOUR LOCATION.  THE"
9270 PRINT "OBSTRUCTION SENSORS INDICATE WHICH DI-"
9280 PRINT "RECTIONS (FLASHING) ARE BLOCKED.  YOU"
9310 PRINT "MOVE YOUR SHIP BY HITTING THE FIRST"
9320 PRINT "LETTER OF THE DIRECTION YOU WANT TO GO."
9330 PRINT "YOUR FUEL SUPPLY (WHICH IS DISPLAYED BY"
9340 PRINT "HITTING THE LETTER, F) WILL ONLY LET"
9350 PRINT "YOU MAKE UP TO 40 MOVES.  GOOD LUCK!"
9999 RETURN
```

MICRO

the BEST ⇦ AND ⇨ the BEST

keyboard buffer

Lazer MICRO SYSTEMS INC

& SHIFT KEY UPPER/LOWER CASE CONTROL

$119.95

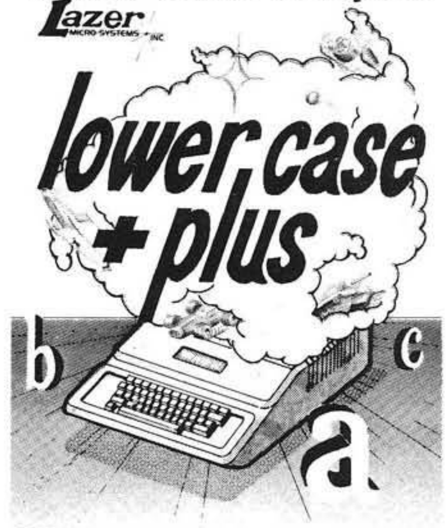
+ More buffer than others.
+ Clear buffer control.
+ SHIFT key entry of upper/lower case.
+ Easy CTL key access to special chars " | ~ ` { } _ [\ □ ".
+ Allows BASIC programs with standard INPUT to support Lower Case without software modification.
+ A lower case adapter is required to display lower case.

Separately, they have more features and out perform all the rest. But together as a team they perform even better. Look for the Graphics +Plus soon. It's a RAM based character generator to compliment the Lower Case +Plus. It will allow you to define the character set to your needs. You could load German, French, Scientific, Engineering or any other special characters into the Graphics +Plus and use it as if the Apple II was designed specially for that application. And that's not all. If you define the characters as graphics, you can do extremely fast HI-RES type graphics on the text screen without all those cumbersome and slow HI-RES routines and 8K screen. For all the details on this triad of products, send for our free booklet "Lower case adapters and keyboard buffers from the inside out". This booklet gives all the details about lower case adapters and keyboard buffers in general. It also has a section on the Graphics +Plus (RAM based character generator).

Lazer MICRO SYSTEMS INC.
1791-G Capital
Corona, CA 91720
(714)735-1041

lower case adapter

Lazer MICRO SYSTEMS INC



GRAPHICS & LOWER CASE CHARACTER GENERATOR FOR THE APPLE II COMPUTER

$69.95

+ Normal & Inverse Lower Case.
+ 2 complete character sets on board.
+ Graphics character font built in.
+ Expansion socket allows access to external character sets.
+ 2716 EPROM compatible char generator.
+ More supporting software. (on diskette)
+ Keyboard +Plus & Graphics +Plus designed around the Lower Case +Plus.

DOSOURCE 3.3

for the Apple II

A source listing of DOS 3.3

Disassembled & commented by Randy Hyde

We took our DISASM/65 disassembler program, disassembled Apple's DOS 3.3, and added meaningful lables and comments to create DOSOURCE 3.3, a perfect companion to "Beneath Apple DOS" by Don Worth and Pieter Lechner*. DOSOURCE clearly lists each routine used by Apple DOS.

DOSOURCE is a LISA 2.5 compatible source listing of DOS 3.3. LISA 2.5 owners can load and reassemble DOS at other locations for special applications (such as in a RAM card). DOSOURCE is also a text file that can be loaded into your favorite assembler and converted for use with it. DOSOURCE is also an assembled listing that you can dump to a printer for reference purposes.

With DOSOURCE you can:

-> Reassemble DOS 3.3 at different addresses.
-> Utilize several useful routines found within DOS, such as decimal input and output. Many routines within DOS are as useful as routines found within the Apple monitor...only you didn't know about them until now!!
-> Remove portions of DOS, that you may not need, freeing memory for program use. Most programs do not need the "RENAME", "INIT", "BSAVE", "BRUN", "BLOAD", "CATALOG", etc. commands while they are running. As much as 4K can be removed from DOS without affecting your programs operation. Think about it the next time you get a MEM FULL error or need to declare an array that's just a little bit too big.
-> Learn lot's of 6502 programming tricks - DOS 3.3 is full of 'em. And you can learn them by studying the source listing.
-> Make "Patches" to DOS 3.3 and understand exactly whats going on. No more "guessing game" resulting in unreliable software.

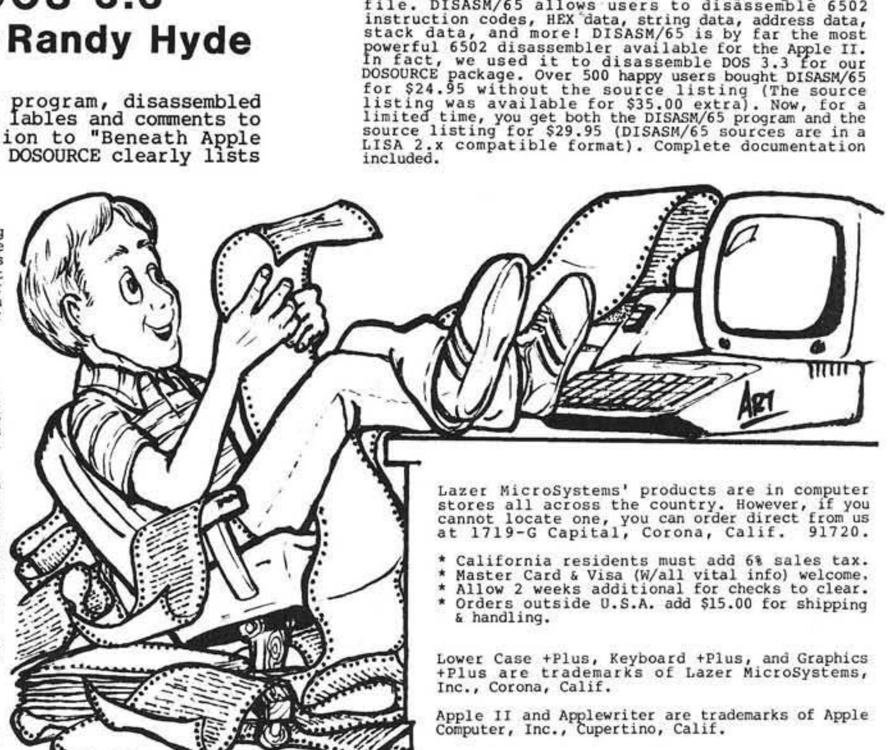
SPECIAL INTRODUCTORY PRICE $39.95
with "Beneath Apple DOS" $55.00

* Beneath Apple DOS is published by Quality Software. Suggested list $19.95

DISASM/65

by Randy Hyde

DISASM/65 is a LISA compatible 6502 disassembler for the Apple II. DISASM/65 takes unadorned machine code and converts it to an understandable assembly language text file. DISASM/65 allows users to disassemble 6502 instruction codes, HEX data, string data, address data, stack data, and more! DISASM/65 is by far the most powerful 6502 disassembler available for the Apple II. In fact, we used it to disassemble DOS 3.3 for our DOSOURCE package. Over 500 happy users bought DISASM/65 for $24.95 without the source listing (The source listing was available for $35.00 extra). Now, for a limited time, you get both the DISASM/65 program and the source listing for $29.95 (DISASM/65 sources are in a LISA 2.x compatible format). Complete documentation included.



Lazer MicroSystems' products are in computer stores all across the country. However, if you cannot locate one, you can order direct from us at 1719-G Capital, Corona, Calif. 91720.

* California residents must add 6% sales tax.
* Master Card & Visa (W/all vital info) welcome.
* Allow 2 weeks additional for checks to clear.
* Orders outside U.S.A. add $15.00 for shipping & handling.

Lower Case +Plus, Keyboard +Plus, and Graphics +Plus are trademarks of Lazer MicroSystems, Inc., Corona, Calif.

Apple II and Applewriter are trademarks of Apple Computer, Inc., Cupertino, Calif.

The Games People Buy

By Mary Ann Curtis
and Marjorie Morse

You can be sure that all computer users, even the most serious ones, have at one time or another played games on their machines. We decided to take a look at this lucrative market, from the point of view of the dealer and the manufacturer.

Dealers

ComputerLand of Nashua, New Hampshire, noted that *Raster Blaster*, an arcade-style pinball game written by Bill Budge of BudgeCo in California, has been a big seller. Other New England dealers also reported that *Raster Blaster* is in demand.

Another popular game — both in New England and on the west coast — is *Ultima* from California Pacific Computer Company. This dragons and dungeons game is in marked contrast to *Raster Blaster*. Rather than providing quick action and immediate success or failure, *Ultima* can take up to a month to play, and the user must remember many variables.

On the west coast, *Castle Wolfenstein* (MUSE), *Falcons* (Piccadilly Software), and *Apple Panic* (Broderbund), are all popular now, along with *Raster Blaster*. Southern and midwestern trends follow the same pattern.

Generally, dealers told us that adults are the primary purchasers of games. However, one store manager said that everyone "from eight to eighty" buys games. Users' backgrounds have little bearing on the type of game they'll buy. For instance, a Boston dealer said he has seen a systems analyst buy *Apple Panic*. And a spokesperson at Computer Town Inc., in Salem, New Hampshire, pointed out that "...a businessman is looking for something he doesn't have to think about — a fast-action game; that's why the space games and pinball games sell well." Educational-game software is carried by most dealers, but does not sell nearly as well as the purely entertainment variety.

Most of the popular games fall in the $25 to $30 price range. Game software is sold almost exclusively on disks. Dealers said that they've had cassette games on their shelves for years. One salesperson pointed out that the single-user games (you against the computer) are the most popular.

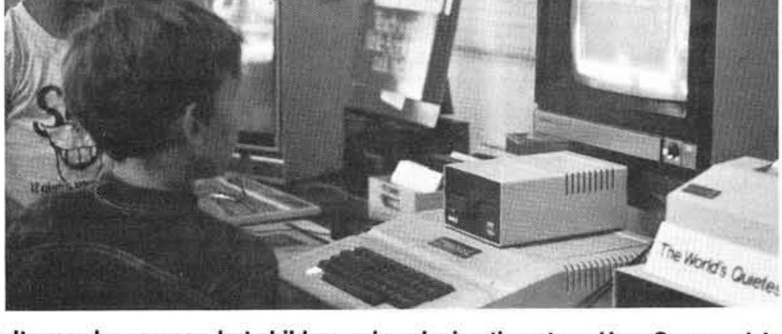
Popularity of a game seems to relate directly to its time on the market. A saleswoman in Houston, Texas, said that the most recent game is usually the most popular. A California dealer agreed — games are constantly changing; the hottest is the newest. And the stores have to be aware of these trends. At ComputerCity in Salem, New Hampshire, they believe you have to keep up with the fads; a store must carry the game when it's popular.

Manufacturers

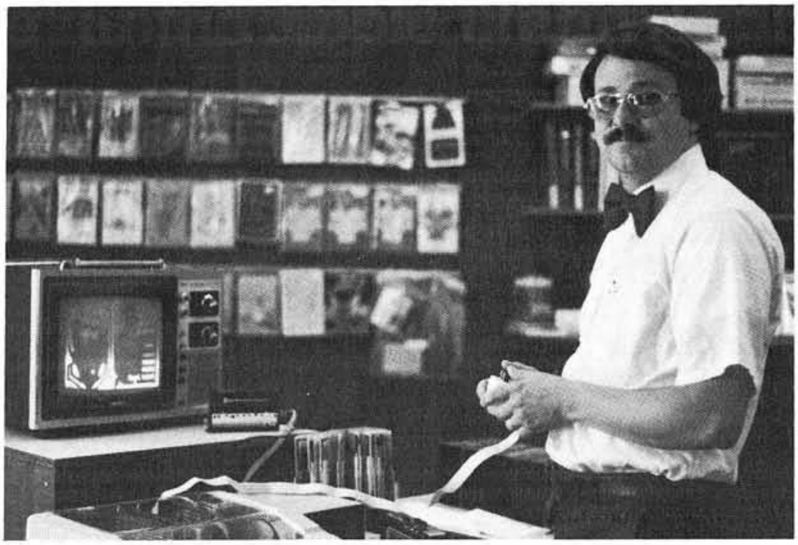
An entertaining, successful game must be easy to understand and play, and must use sound effects, according to Tom Jackson, Director of Marketing at MUSE Software. The designer of *Castle Wolfenstein*, MUSE's current best seller, obviously had these elements in mind. Jackson believes that *Castle* will become a trendsetter because it effectively combines the appeal of both adventure and arcade games.

Doug Carlson of Broderbund Software offered a similar opinion about the features that make a first-rate game. He told us that consumers want good sound and animation plus rules they can learn fast. Anything too intimidating isn't attractive. (*Apple Panic*, Broderbund's best selling game, is arcade-style, and features a little man chased by man-eating apples.)

One popular arcade-style game is *Sneakers*, made by Sirius Software. "It's a fast-action game, and that's what's selling now," said manager Jim Ackermann. Sirius' *Space Eggs*, another arcade game, is their all-time best seller, and is still moving well.



Adults may buy games, but children enjoy playing them too. Here Sean and John at Computer Town in Salem, New Hampshire, compete in a game of *Sneakers*.



Jonathan Wood of Computerland in Nashua, New Hampshire, demonstrates *Raster Blaster*, the store's current best seller.

BudgeCo, producers of *Raster Blaster*, was new to the industry in April, 1981. So far their remarkably successful game is BudgeCo's only product. Bill Budge, author of *Raster Blaster*, for now is the company's sole programmer.

Quality Software offers *Starbase Hyperion*, a space simulation arcade game for the Atari, while Instant Software makes *Space Shuttle*, an air flight simulation game.

Mary Reed, Director of Marketing at Instant Software, thinks that the public wants simulation games with good graphics and varying levels of play. A Creative Computing Software spokesperson believes arcade games should have these same features. He added that the game should have a place to record the player's score — to play against next time, or to show off.

Creative's *Advanced Air Traffic Controller* is a simulation game whose popularity has been increased by the national air traffic controllers' strike. As with arcade games, the spokesman had some ideas on features that strategy and adventure games should include. Games with "seemingly unattainable goals" will never sell because the player gets too frustrated, he said. He maintained that if the goal of the game may be reached only after playing for a long time (30 minutes or more), the game should reward the player from time to time to give him encouragement.

If these comments sound to you like rules a teacher would follow, you're on the right track. Simulation games are popular with educators. *Robot Wars* teaches programming skills, and *Three Mile Island* helps players understand how a thermonuclear reactor works — but at the same time is challenging and fun. (Both of these games are manufactured by MUSE.)

Strategic Simulations in California manufactures strategy games. Two of their recent best sellers are *Warp Factor* and *Computer Baseball*, both for the Apple. According to a company spokesperson, most of their games are distributed to the east and west coasts, with very few going to the south or midwest.

Strategic's software is generally war-game in nature. The games take several hours to complete and are designed to challenge you.

Two new releases from Strategic are *Battle of Shiloh* for the Apple or TRS-80, and *Tigers in the Snow*, a game based on a WWII battle.

Adventure, fantasy, and role-playing games are also popular now. Jim Connelley of Automated Simulations reported *Temple of Apshai* to be his number one best seller. This role-playing game won the Hobby Industry Association of America's "Best Computer Game of 1980" award in July of this year. Automated Simulations does not sell arcade games.

Rich Richmond, marketing manager at Adventure International, said that computer games, in general, have become sophisticated in their use of sound and graphics, but that "game theory has been amazingly poor." He defined game theory as the thought and imagination that go into design of a game (the concept behind the moves, sequences, etc.) and he cited Chess as having the all-time best game theory.

Richmond said that computer game manufacturers have had to copy arcade games for survival, but due to new copyright laws they will no longer be able to rely on this method. He thinks that eventually computer game producers will have to use more imagination to provide the player with more long-term satisfaction from playing a game.

Programmers

We found that some software manufacturers have their games written in-house. For instance, MUSE employs the well known game author Silas Warner as a programmer. MUSE, though, like all other manufacturers we contacted, accepts games from outside authors. Adventure International claimed that 80% of its games were written outside.

If you are writing a game and wish to sell it to a software company, we suggest that you keep in mind these suggestions.

1. Keep directions as simple and as brief as posible.
2. Offer more than one level of play.
3. Provide comments on play from the computer (especially humorous).
4. Devote special attention to graphics — make objects look realistic.
5. For adventure-type games, the more variables, the better (i.e., monsters, treasures, aids, places to hide, etc.).
6. Provide more than one means to combat the enemy.
7. Include sound whenever possible.

If you still have problems writing your game, two game manufacturers will soon offer more help: Avant-Garde's *Hi-Res Secrets* and Broderbund's *The Arcade Machine*.

MICRO

we beat the price...

ATARI

800™ $749

ATARI
Computers for people.™

ATARI 810 DISC DRIVE

$444

ATARI SOFTWARE

CX404 Word Processor	$119.00
CX405 PILOT	$68.00
CX413 MICROSOFT BASIC	$68.00
CX4101 Invitation to Programming 1	$17.00
CX4102 Kingdom	$13.00
CX4103 Statistics	$17.00
CX4104 Mailing List	$17.00
CX4105 Blackjack	$13.00
CX4106 Invitation to Programming 2	$20.00
CX4107 Biorhythm	$13.00
CX4108 Hangman	$13.00
CX4109 Graph It	$17.00
CX4110 Touch Typing	$20.00
CX4111 Space Invaders	$17.00
CX4112 States & Capitals	$13.00
CX4114 European Countries & Capitals	$13.00
CX4115 Mortgage & Loan Analysis	$13.00
CX4116 Personal Fitness Prog.	$59.00
CX4117 Invitation to Programming 3	$20.00
CX4118 Conversational French	$45.00
CX4119 Conversational German	$45.00
CX4120 Conversational Spanish	$45.00
CX4121 Energy Czar	$13.00
CX4125 Conversational Italian	$45.00
CX6001 U.S. History	$23.00
CX6002 U.S. Government	$23.00
CX6003 Supervisory Skills	$23.00
CX6004 World History	$23.00
CX6005 Basic Sociology	$23.00
CX6006 Counseling Proced.	$23.00
CX6007 Principal of Act.	$23.00
CX6008 Physics	$23.00
CX6009 Great Classics	$23.00
CX6010 Business Comm.	$23.00
CX6011 Basic Psychology	$23.00
CX6012 Effective Writing	$23.00
CX6014 Principals of Econ.	$23.00
CX6015 Spelling	$23.00
CX6016 Basic Electricity	$23.00
CX6017 Basic Algebra	$23.00
CX8106 Bond Analysis	$20.00
CX8107 Stock Analysis	$20.00
CX8108 Stock Charting	$20.00
CXL4001 Education System Master	$21.00
CXL4002 Basic Computing Language	$46.00
CXL4003 Assembler Editor	$46.00
CXL4004 Basketball	$24.00
CXL4005 Video Easel	$24.00
CXL4006 Super Breakout	$30.00
CXL4007 Music Composer	$45.00
CXL4009 Chess	$30.00
CXL4010 3-D Tic-Tac-Tow	$24.00
CXL4011 Star Raiders	**$39.00**
CXL4012 Missle Command	**$32.00**
CXL4013 Asteroids	**$32.00**
CXL4015 TeleLink	$20.00
Compuhome	$74.95
Visicalc	$149.00
Letter Perfect (Word Processor)	$119.00
Source	$89.00

Atari® Peripherals:

400 16K	$329.00
410 Recorder	$59.00
822 Printer	$359.00
825 Printer	$ CALL
830 Modem	$159.00
850 Interface	$ CALL

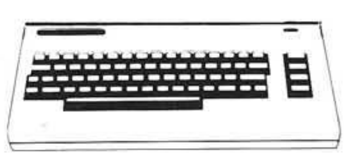
Atari® Accessories

New DOS 2 System	$21.00
CX70 Light Pen	$64.00
CX30 Paddle	$18.00
CX40 Joy Stick	$18.00
CX853 16K RAM	$89.00
Microtek 16K RAM	$75.00
Microtek 32K RAM	$169.00

commodore

CBM 8032 $1099

4016	$799.00
4032	$999.99
8096	$1795.00
CBM4022 Printer	$629.00
Tally 8024	$1699.00
CBM C2N Cassette Drive	$69.00
CBM4040 Dual Disk Drive	$999.00
CBM8050 Dual Disk Drive	$1349.00



VIC 20 $259

Vic-TV Modual	$19.00
Vic Cassette	$69.00
Vic Disk Drive	$ Call
Vic 6 Pack program	$44.00

CBM Software

WordPro3 Plus	$199.00
WordPro4 Plus	$299.00
Commodore Tax Package	$399.00
Visicalc	$149.00
BPI General Ledger	$329.00
OZZ Information System	$329.00
Dow Jones Portfolio	$129.00
Pascal	$239.00
Legal Time Accounting	$449.00
World Craft 80	$289.00
Word Check	$180.00
Create-A-Base	$239.00
Power	$89.00
Socket-2-Me	$20.00
Jinsam	Call

Disks

CX8100 Blank Disk (5)	$22.00
Sycom Blank Disk (10)	$29.00
Maxell Blank Disk (10)	$36.00
Maxell Blank Disk (10)	$46.00

Printers

Call for prices on the new NEC models.

Epson MX-70
Epson MX-80
Epson MX-80 FT
Diablo 630
} Call for Prices

TEC 1500 Starwriter 25cps	$1495.00
TEC 1500 Starwriter 45cps	$1795.00

No Risk, No Deposit On Phone Orders, COD or Credit Card, Shipped Same Day You Call *

* on all in stock units

IN PA, CALL (717) 327-9575

(800) 233-8950

To Order:

Phone orders invited (800 number is for order desk only). Or send check or money order and receive free shipping. Pennsylvania residents add 6% sales tax. Add 3% for Visa or M.C. Equipment is subject to price change and availability without notice. Please call between 11 AM & 6 PM.

COMPUTER MAIL ORDER
501 E. 3RD ST., WILLIAMSPORT, PA 17701

Saucer Launch

Atari arcade games contain specialized video processors to control and manipulate their game images. The Atari 800 computer system contains much the same hardware. Saucer Launch is a combination of BASIC and assembler level programming designed to use these Atari video processors. Although a 24K byte Atari system is required for Saucer Launch, the game itself should help provide insights into this versatile personal computer.

Mike Dougherty
7659 West Fremont Ave.
Littleton, Colorado 80123

Saucer Launch Scenario

You are the Gunner's Apprentice and one of the few remaining survivors of the Starfleet patrol. Sent as an envoy of peace, the patrol was attacked by a squadron of robot saucers. A combination of number and surprise has rendered the rest helpless. No fellow patrol ships are responding to your short range communications; long range communications with the Starfleet have been severed.

Apparently this innocent planetoid, P-XA123, has been responsible for the robot saucers sighted in this sector of the galaxy. Devastatingly accurate once launched, a sufficient fleet of robot saucers could overpower the main Starfleet. The only chance is for your patrol ship to destroy at least 60 percent of the robot saucers while they are being launched.

With considerable effort, the crew has managed to repair the laser cannon. This cannon, steered by the Master Gunner's control stick, fires automatically when the laser cross hairs overlap the target. Unfortunately, the patrol ship is hovering immobilized in the rarefied atmosphere of P-XA123 — the saucers must be destroyed as they are launched.

The last attack left the Master Gunner critically wounded. Since you are the best trained member of the crew, it is up to you to eliminate 60 percent of the saucers launched and save the Starfleet.

A word of warning: the automatic sensors of P-XA123 are capable of detecting the approaching Starfleet. As the Starfleet unwittingly moves into an ambush, the saucers will be launched at an ever-increasing pace.

Good luck! The fate of the Starfleet lies completely in *your* hands.

Introduction

This article has been written to serve several purposes. First, the Atari 800 computer has several specialized "video processors" allowing outstanding game displays. Second, the Atari version of BASIC supports an easy interface to assembler level subroutines — a needed feature for implementing many real time programs. Third, the combination of properly designed hardware and good software achieves results unobtainable with either alone. Fourth, an "arcade-like" game is possible in Atari BASIC while remaining a respectable programming task. Hopefully, at the end of this article, the reader will have a better understanding of the Atari 800 computer system. At the very least, "Saucer Launch" will serve as a model to disect for that better understanding.

The two manuals, *Operating System User's Manual* and *Hardware Manual*, which may be ordered directly from Atari, contain a detailed description of the hardware used in Saucer Launch. One word of warning: the special display modes of Saucer Launch interact with BASIC memory locations causing unusual Atari behavior. Before using other programs, switch the Atari off and on to reset the BASIC memory.

Definition of Saucer Launch

Saucer Launch involves manipulating a cross hair pattern *via* joystick #1 (STICK(0)) to touch a computer-generated "saucer" target. The saucers move upward along straight lines at a random angle. Any contact or collision between the cross hair pattern and the saucer pattern counts as a "hit." If the saucer reaches an edge limit prior to contact, the round is scored as a "miss." One hundred saucer launches from random positions on the ground comprise a game. The goal of Saucer Launch is to hit (destroy) 60 of the 100 saucers launched. The overall average speed of the game increases linearly with each saucer launched.

The following features were designed into Saucer Launch:

- The game player is looking out from the viewport of the Master Gunner over the saucer launch field.
- The cross hair pattern of the laser cannon is controlled by the joystick.
- A hit is rewarded with a combination of sound and light as the saucer target explodes.
- A miss is signaled with a "buzzing" sound.
- The saucers grow smaller the higher (further) they get, becoming harder to hit if the player's reaction time is slow.
- The average speed of the game is programmable and increases with each saucer.
- The saucers travel in essentially straight lines at random upward angles.
- A saucer emits an "engine" sound to allow audio feedback during game playing.
- To increase the "psychological stress" of the game, a background

drone increases in pitch as the game progresses.

- A running score of hits, misses, and percentage is output after each round.

From an examination of these features, a combination of BASIC and assembler level subroutines is required. The sound and color manipulations, the running score, and the random number generation are most conveniently programmed in BASIC. However, to respond rapidly to the joystick and for rapid overall movement, assembler level subroutines are required. Thus, only by combining the best features of both languages is Saucer Launch practical.

Saucer Movement

One problem that had to be solved prior to designing Saucer Launch was the straight line saucer movement at a random angle. The method chosen to encode the saucer movement involved interpreting a byte as a pattern of eight separate move commands. Each bit position indicates whether the saucer is to move (a one bit) or not to move (a zero bit) for that given step. A set of four bytes determines the saucer movement in the y positive (YP), y minus (YM), x positive (XP), and x minus (XM) directions (up, down, right, left respectively). By repeating the cycle, every eighth step leaves the saucer at a new position along a straight line. If the bit patterns are chosen at random, then this "straight line" path of the saucer will form a "random" angle.

Specifically, assuming that "MASK" is initialized to 1, the following logic will move the saucer one step and prepare for the next step:

if (YP AND MASK) not equal to zero then move saucer UP

if (YM AND MASK) not equal to zero then move saucer DOWN

if (XP AND MASK) not equal to zero then move saucer RIGHT

if (XM AND MASK) not equal to zero then move saucer LEFT,
shift MASK left one bit

if (MASK = 0) then set MASK to 1

where "AND" is the "logical and" operation. An example of this type of movement is represented in table 1. For Saucer Launch, motion in the y minus direction, YM, was set to zero so that the saucers would always launch upward.

***Table 1:* Saucer Motion via Bit Patterns**

Assume that YP = 247 = $F7 = B'1111 0111'
XP = 89 = $59 = B'0101 1001'
YM = XM = 0

Step	Bit Mask	Mask .and. YP	Mask .and. XP	Movement
1	0000 0001	0000 0001	0000 0001	UP, RIGHT
2	0000 0010	0000 0010	0000 0000	UP
3	0000 0100	0000 0100	0000 0000	UP
4	0000 1000	0000 0000	0000 1000	RIGHT
5	0001 0000	0001 0000	0001 0000	UP, RIGHT
6	0010 0000	0010 0000	0000 0000	UP
7	0100 0000	0100 0000	0100 0000	UP, RIGHT
8	1000 0000	1000 0000	0000 0000	UP
9	0000 0001	0000 0001	0000 0001	UP, RIGHT
10	0000 0010	0000 0010	0000 0000	UP
11	0000 0100	0000 0100	0000 0000	UP
12	0000 1000	0000 0000	0000 1000	RIGHT

. . . . . and so on

***Table 2:* Summary of BASIC Program**

Lines	Function
1000-1999	POKE and relocate the USR function and USR data base into memory.
2000-2999	Initialize the Player/Missile graphics; draw background
3000-4999	Main game loop. For 100 rounds: select target position and YP,YM,XP,XM initialize collision registers wait a random interval before launching saucer initiate sounds for this round call assembler level USR function to control movement if a collision, then call hit subroutine at 5000 if a miss, then call miss subroutine at 6000 output current score next round if less than 60 hits, then call failed subroutine at 8000 else call success subroutine at 9000 GOTO 2000 for next game
5000-5999	Hit target subroutine; blow up the saucer with sound and color
6000-6999	Missed target subroutine; erase saucer and buzz the player
7000-7999	Draw the saucer launch world
8000-8999	Mission failed subroutine; destroy the Starfleet
9000-9999	Mission succeeded subroutine; congratulate the player

BASIC USR Interface

The USR function is described in Chapter 11 of the Atari *BASIC Reference Manual*. Unfortunately, the USR code and data base required by Saucer Launch is greater than the free memory available in page 6 (addresses $0600 -$06FF). Saucer Launch solves the problem by storing the data base in page 6 and POKEing the code into a character string, CODE$. Since the location of CODE$ depends upon the size of Saucer Launch, the USR functions are relocated "on the fly." Each address requiring relocation is assembled relative to an origin of zero. Then the relocated address is simply this calculation offset to zero plus the starting address of CODE$. However, if the program is stopped and modified, re-run the entire program. Atari BASIC moves the variables as necessary to make room for the program lines. After movement of CODE$, any relocated addresses will point to the wrong place in memory.

Conclusion

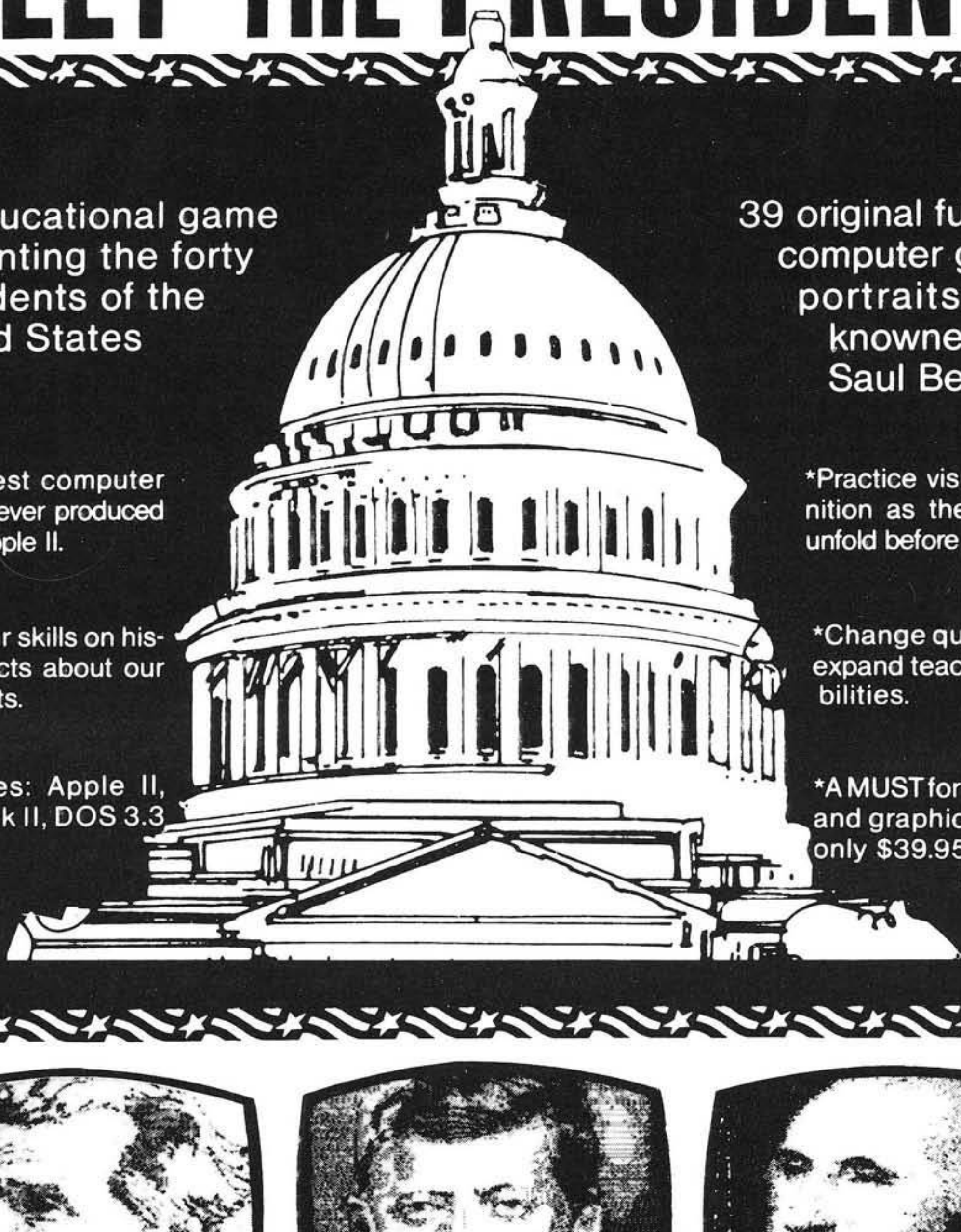
The final game of Saucer Launch is outlined in table 2 and listings 1 and 2. In most cases, I used very straightforward programming techniques with occasional unnecessary precaution; Saucer Launch will not win any awards in efficiency or compactness. The game with all comments requires a 24K byte Atari system. The overall speed of the game is sufficient to make the defense of the Starfleet a definite challenge.

Mike Dougherty graduated from the University of Tennessee in 1977 with an M.S. degree in Computer Science, and is currently working at Martin Marietta Aerospace in Denver, Colorado. 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.

```
1 REM ...   S A U C E R   L A U N C H
2 REM
3 REM ...        by  Mike Dougherty
4 REM
5 REM
100 DIM CODE$(512),BYTE$(2)
200 GRAPHICS 0:POKE 752,1
1000 REM
1001 REM
1002 REM ...   LOAD USR FUNCTION
1003 REM
1004 REM
1010 A=ADR(CODE$):REM ADDR OF USR FUNCTION
1020 AHI=INT(A/256):REM HIGH BYTE
1030 ALO=A-AHI*256:REM LOW BYTE
1035 PRINT "Loading USR function."
1040 GOSUB 1200:REM LOAD FUNCTION
1045 PRINT "Loading USR data base."
1050 GOSUB 1800:REM LOAD DATA BASE
1060 SOUND 0,0,0,0
1070 PRINT "Press trigger to continue.";
1080 DUMMY=RND(1):IF STRIG(0)<>0 THEN GOTO 1080
1090 GOTO 2000
1200 REM
1201 REM ...   READ/POKE USR
1202 REM
1210 READ BYTE$:REM READ A BYTE OF OBJECT CODE
T CODE
1220 IF BYTE$=".." THEN RETURN
1230 IF BYTE$="**" THEN 1300
1240 GOSUB 1400:REM CONVERT BYTE$ TO BYTE
1250 POKE A,BYTE:A=A+1
1260 GOTO 1210
1300 REM
1301 REM ...   RELOCATE ADDRESS
1302 REM
1310 READ BYTE$:GOSUB 1400:LO=BYTE+ALO
1320 READ BYTE$:GOSUB 1400:HI=BYTE+AHI
1324 IF LO>255 THEN LO=LO-256:HI=HI+1:GOTO 1324
1330 POKE A,LO:A=A+1
1340 POKE A,HI:A=A+1
1350 GOTO 1210
1400 REM
1401 REM ...   BYTE$ --> BYTE
1402 REM
1410 BYTE=0
1420 V=ASC(BYTE$(1)):GOSUB 1450
1430 V=ASC(BYTE$(2)):GOSUB 1450
1440 RETURN
1450 IF V<58 THEN BYTE=BYTE*16+V-48
1460 IF V>57 THEN BYTE=BYTE*16+V-55
1465 SOUND 0,BYTE,10,8
1470 RETURN
1500 REM
1501 REM ... USR OBJECT CODE
1502 REM
1510 DATA 68,F0,0A,C9,07,F0,07
1512 DATA AA,68,68,CA,D0,FB,60
1514 DATA 68,8D,01,06,68,8D,00,06
1516 DATA 68,8D,03,06,68,8D,02,06
1518 DATA 68,68,8D,04,06
1520 DATA 68,68,8D,05,06
1522 DATA 68,68,8D,06,06
1524 DATA 68,68,8D,07,06
1526 DATA 68,68,8D,08,06
1527 DATA A9,01,85,CE
1528 REM
1530 REM ... MAIN USR LOOP
1532 REM
1534 DATA 20,**,9F,00,F0,0E
1536 DATA 20,**,60,00,AD,0C,D0
1538 DATA D0,0F,20,**,0A,01,4C,**,3B,00
1540 DATA A9,00,85,D4,A9,00,85,D5,60
1542 DATA A9,01,85,D4,A9,00,85,D5,60
1550 REM
1552 REM ... MOVE PLAYER SUBROUTINE
1554 REM
1556 DATA AD,00,06,85,D0
1558 DATA AD,01,06,85,D1
1560 DATA A2,00,AD,00,D3,29,0F
1562 DATA 85,CF,29,01,D0,03,20,**,2D,01
1564 DATA A5,CF,29,02,D0,03,20,**,16,01
1565 DATA A5,CF,29,04,D0,03,20,**,5D,01
1566 DATA A5,CF,29,08,D0,03,20,**,45,01
1568 DATA BD,0B,06,A8,A2,40
1570 DATA 20,**,F8,00,60
1572 REM
1574 REM ...   MOVE TARGET SUBROUTINE
1576 REM
1578 DATA 06,CE,D0,04,A9,01,85,CE
1580 DATA AD,02,06,85,D0
1582 DATA AD,03,06,85,D1,A2,01
1584 DATA AD,04,06,25,CE,F0,06,20,**,2D,01,D0,01,60
1586 DATA AD,05,06,25,CE,F0,06,20,**,16,01,D0,01,60
1588 DATA AD,06,06,25,CE,F0,06,20,**,45,01,D0,01,60
1590 DATA AD,07,06,25,CE,F0,06,20,**,5D,01,D0,01,60
1592 DATA A2,01,BD,0B,06,A8,4A,4A
1594 DATA 29,38,AA,20,**,F8,00
1596 DATA A9,01,60
1600 REM
1602 REM ...   MOVE OBJECT SUBROUTINE
1604 REM
1606 DATA A9,08,8D,0D,06
1608 DATA BD,0E,06,91,D0,E8,C8
1610 DATA CE,0D,06,D0,F4,60
1612 REM
1614 REM ...   DELAY SUBROUTINE
1616 REM
1618 DATA AC,08,06,A2,05,CA,D0,FD
1620 DATA 88,D0,F8,60
1622 REM
1624 REM ...   DOWN SUBROUTINE
1626 REM
1628 DATA BD,0B,06,C9,BA,F0,0D
1630 DATA FE,0B,06,EA,EA,A8
1632 DATA A9,00,91,D0,A9,01,60
1634 DATA A9,00,60
1636 REM
1638 REM ...   UP SUBROUTINE
1640 REM
1642 DATA BD,0B,06,C9,1C,F0,0E
1644 DATA DE,0B,06,18,69,07,A8
1646 DATA A9,00,91,D0,A9,01,60
1648 DATA A9,00,60
1650 REM
1652 REM ...   RIGHT SUBROUTINE
1654 REM
1656 DATA BD,09,06,C9,CC,F0,0E
1658 DATA FE,09,06,BD,09,06
1660 DATA 9D,00,D0,EA,EA,A9,01,60
1662 DATA A9,00,60
1664 REM
1666 REM ...   LEFT SUBROUTINE
1668 REM
1670 DATA BD,09,06,C9,2D,F0,0E
1672 DATA DE,09,06,BD,09,06
1674 DATA 9D,00,D0,EA,EA,A9,01,60
1676 DATA A9,00,60
```

```
1700 DATA ..
1800 REM
1801 REM ...   LOAD USR DATA BASE
1802 REM
1810 A=1550:REM BASE ADDRESS OF TARGETS
1820 READ BYTE$
1830 IF BYTE$=".." THEN RETURN
1840 GOSUB 1400
1850 POKE A,BYTE:A=A+1
1860 GOTO 1820
1900 REM
1901 REM ...   TARGET PATTERNS
1902 REM
1904 DATA 00,00,00,08,1C,00,00,00
1906 DATA 00,00,00,18,3C,18,00,00
1908 DATA 00,00,18,24,3C,18,00,00
1910 DATA 00,00,1C,2A,3E,1C,00,00
1912 DATA 00,08,1C,2A,3E,1C,08,00
1914 DATA 00,1C,3E,55,7F,3E,1C,00
1916 DATA 00,1C,3E,55,7F,7F,1C,08
1918 DATA 00,3C,7E,A5,FF,FF,7E,18
1920 REM
1921 REM ...   CROSS HAIR AIM
1922 REM
1924 DATA 00,38,10,92,FE,92,10,38
1926 REM
1928 REM ...   EXPLOSION PATTERNS
1930 REM
1932 DATA 00,00,24,18,18,24,00,00
1934 DATA 81,42,2C,A6,65,3C,42,81
1936 DATA 10,00,44,90;02,10,02,89
1938 DATA 00,00,00,00,00,00,00,00
1990 DATA ..
2000 REM
2001 REM ... INITAILIZE THE PLAY
2002 REM
2005 HITS=0:MISSES=0
2010 GRAPHICS 8:POKE 752,1:REM SET HIGH RES
2015 GOSUB 7000:REM DRAW REST OF WORLD
2020 POKE 559,62:REM SET PLAYFIELD SIZE
2030 POKE 704,88:REM COLOR REGISTER FOR PLAYER #0
2040 POKE 705,24:REM COLOR REGISTER FOR PLAYER #1
2050 SPACE=PEEK(106)-8:REM GRAPHICS PAGE
2060 POKE 54279,SPACE:REM PLAYER/MISSLE BASE
     ADDRESS REGISTER
2070 POKE 53277,3:REM ENABLE PLAYER MISSLE
     DIRECT MEMORY ACCESS
2080 POKE 53256,0:REM SIZE OF PLAYER # 0
2090 POKE 53257,0:REM SIZE OF PLAYER # 1
2100 BASE=6*256:REM USR DATA BASE
2110 P0=SPACE*256+1024:REM BIT MAP FOR PLAYER #0
2120 P1=SPACE*256+1280:REM BIT MAP FOR PLAYER #1
2200 CX=133:POKE BASE+9,CX:REM INIT AIM
2210 CY=110:POKE BASE+11,CY
3000 REM
3001 REM ...   GAME LOOP
3002 REM
3010 FOR LOOP=1 TO 100
3020 SETCOLOR 2,8,0
3030 TX=RND(1)*60+110:POKE BASE+10,TX
3040 TY=RND(1)*10+170:POKE BASE+12,TY
3050 YP=INT(RND(1)*255+1)
3060 YM=0
3070 XP=INT(RND(1)*255+1)
3080 XM=INT(RND(1)*255+1)
3100 POKE 53248,PEEK(BASE+9)
3200 POKE 53249,PEEK(BASE+10)
3400 POKE 53278,0:REM INIT COLLISION REGISTER
4200 POKE 77,0:REM NO ATTRACT MODE
4300 FOR WAIT=1 TO RND(1)*50:NEXT WAIT
4400 SOUND 0,10,0,5:SOUND 3,255-LOOP,10,2
4450 D=200-LOOP:REM VARIABLE DELAY LOOP
4500 X=USR(ADR(CODE$),P0,P1,YP,YM,XP,XM,D)
4610 IF X=1 THEN GOSUB 5000
4620 IF X=0 THEN GOSUB 6000
4700 PRINT :PRINT
4710 HITS=HITS+X:MISSES=MISSES+(1-X)
4720 PRINT "Targets HIT: ";HITS,"  MISSED: "
   ;MISSES
4730 PRINT "Percentage: ";INT(HITS/(HITS+MISSES)
   *1000)/10;"%"
4800 NEXT LOOP
4900 REM
4901 REM ... ROUND OVER
4902 REM
4910 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 3,0,0,0
4920 POKE 53277,0:REM NORMAL DISPLAY
4950 IF HITS<60 THEN GOSUB 8000:GOTO 2000
4960 GOSUB 9000:GOTO 2000
5000 REM
5001 REM ... HIT THE TARGET
5002 REM
5010 A=BASE+86:CLEAR=P1+PEEK(BASE+12)
5020 FOR IMAGE=4 TO 1 STEP -1
5025 SETCOLOR 2,3,IMAGE*2
5030 SOUND 0,50,8,IMAGE*2
5035 SOUND 1,60,0,IMAGE*2
5040 FOR LINE=0 TO 7
5050 POKE CLEAR+LINE,PEEK(A)
5060 A=A+1
5070 NEXT LINE
5080 NEXT IMAGE
5085 SOUND 0,0,0,0:SOUND 1,0,0,0
5087 SETCOLOR 2,8,0
5090 RETURN
6000 REM
6001 REM ... MISSED THE TARGET
6002 REM
6010 CLEAR=P1+PEEK(BASE+12)
6020 FOR LINE=0 TO 7
6030 POKE CLEAR+LINE,0
6035 SOUND 0,200,12,8
6040 NEXT LINE
6045 SOUND 0,0,0,0
6050 RETURN
7000 REM
7001 REM ...   DRAW LAUNCH BASE
7002 REM
7010 COLOR 1
7020 PLOT 90,159:DRAWTO 160,100
7030 PLOT 230,159:DRAWTO 160,100
7040 PLOT 0,146:DRAWTO 160,100
7050 PLOT 319,146:DRAWTO 160,100
7060 PLOT 0,120:DRAWTO 160,100
7070 PLOT 319,120:DRAWTO 160,100
7080 PLOT 0,136:DRAWTO 319,136
7090 PLOT 0,121:DRAWTO 319,121
7100 PLOT 0,113:DRAWTO 319,113
7110 PLOT 0,107:DRAWTO 319,107
7120 PLOT 0,103:DRAWTO 319,103
7180 PLOT 0,100:DRAWTO 319,100
7190 PLOT 0,159:DRAWTO 319,159
7200 DRAWTO 319,0
7210 DRAWTO 0,0
7220 DRAWTO 0,159
7300 RETURN
8000 REM
8001 REM ...   MISSION FAILED !!!
8002 REM
8010 PRINT :PRINT :PRINT
     "S T A R F L E E T   S U R R O U N D E D"
8015 SOUND 1,100,10,6
8020 FOR BLAST=1 TO 12
8030 C=INT(RND(1)*3)+2
8032 X=RND(1)*200+50:Y=RND(1)*50+10
8040 FOR EXPLODE=10 TO 0 STEP -(RND(1)*3+1)
8050 SOUND 0,10,0,EXPLODE
8055 SETCOLOR 2,C,EXPLODE
8060 PLOT X+RND(1)*4,Y+RND(1)*4
8062 PLOT X+RND(1)*4,Y+RND(1)*4
8070 NEXT EXPLODE
8080 NEXT BLAST
8090 SOUND 0,0,0,0:SOUND 1,200,10,8
8100 SETCOLOR 2,0,0
8110 PRINT :PRINT :PRINT HITS;"%
        -- M I S S I O N   A B O R T E D"
8112 FOR WAIT=1 TO 150:NEXT WAIT
8114 SOUND 1,0,0,0
8116 GRAPHICS 0:POKE 752,1
```

(Continued)



*actual screen photo

```
8120 PRINT :PRINT "Well HOTSHOT,
     ... want to try again ?"
8130 PRINT :PRINT "Press triger to continue."
8140 IF STRIG(0)<>0 THEN GOTO 8140
8200 RETURN
9000 REM
9001 REM ...    MISSION SUCCEDED
9002 REM
9010 PRINT :PRINT :PRINT HITS;"%
     -- MISSION WELL DONE."
```

```
9020 FOR BLAST=1 TO 10
9030 FOR WAIT=RND(1)*100+100 TO 1 STEP -10
9040 SOUND 0,WAIT,10,8
9050 SETCOLOR 2,13,WAIT/20
9060 NEXT WAIT
9070 NEXT BLAST
9080 SOUND 0,0,0,0
9090 GRAPHICS 0:POKE 752,1
9100 PRINT "Press trigger to continue."
9200 IF STRIG(0)<>0 THEN GOTO 9200
9300 RETURN
```

Listing 2

```
                     ;        SAUCER LAUNCH USR FUNCTION
                     ;
                     ;        X = USR(ADDR,P0,P1,YP,YM,XP,XM,DELA)
                     ;
                     ;               WHERE ADDR - STARTING ADDRESS OF USR FUNCTION
                     ;                     P0   - PLAYER 0 DISPLAY MEMORY
                     ;                     P1   - PLAYER 1 DISPLAY MEMORY
                     ;                     YP   - Y POSITIVE MOVEMENT
                     ;                     YM   - Y MINUS    MOVEMENT
                     ;                     XP   - X POSITIVE MOVEMENT
                     ;                     YM   - Y MINUS    MOVEMENT
                     ;                     DELA - TIME DELAY CONSTANT
                     ;
                     ;
                     ;        USR RETURNS     0  - IF SAUCER NOT HIT
                     ;                        1  - IF SAUCER HIT


00D0                          PLAY=$00D0           STORAGE FOR INDIRECT POINTER TO DISPLAY MEMORY
00CE                          MASK=$CE             BIT POSITION OF SAUCER MOVE
00CF                          JOY=$00CF            CURRENT JOY STICK VLAUE
00D4                          VALUE=$00D4          USER RETURN ARGUMENT

D300                          PORT=$D300           JOYSTICK 0 PORT
D000                          HORZ=$D000           BASE HORIZONTAL POSITION REGISTER
D00C                          COLSN=$D00C          PLAYER 0 COLLISION REGISTER FOR OTHER PLAYERS

                              *=$0600              USER VARIABLE / DATA BASE STORAGE

                     BASE

0600 00 00           P0       .WORD   0            PLAYER 0 DISPLAY MEMORY
0602 00 00           P1       .WORD   0            PLAYER 1 DISPLAY MEMORY
0604 00              YPLUS    .BYTE   0            SAUCER MOVEMENT BIT PATTERNS
0605 00              YMINUS   .BYTE   0
0606 00              XPLUS    .BYTE   0
0607 00              XMINUS   .BYTE   0
0608 00              DELA     .BYTE   0            DELAY TO SLOW DOWN GAME
0609 00 00           POSX     .BYTE   0,0          CURRENT POSITION OF CROSSHAIRS
060B 00 00           POSY     .BYTE   0,0          CURRENT POSITION OF SAUCER
060D 00              COUNT    .BYTE   0            TEMPORARY

060E                          TARG=*               SAUCER,CROSSHAIR,EXPLOSION PATTERNS
                     ;                             SEE BASIC PROGRAM FOR ACTUAL PATTERNS
                     ;        ASSEMBLE CODE AT $1000 FOR ASSEMBLER'S CONVIENCE.  LATER MODIFY
                     ;        ALL ADDRESSES REQUIRING RELOCATION TO OFFSETS TO $0000 -- SIMPLY
                     ;        SUBTRACT $1000 FROM THOSE ADDRESSES

                              *=$1000


1000 68              USR      PLA                  NUMBER OF ARGUMENTS
1001 F0 0A                    BEQ     UABORT       ZERO IS WRONG NUMBER
1003 C9 07                    CMP     #$07         EXPECT 7 ARGUMENTS IN USR CALL
1005 F0 07                    BEQ     PARMOK       OK

1007 AA                       TAX                  BAD USR CALL -- CLEAN UP STACK
1008 68              PLALP    PLA                  EACH ARGUMENT IS 2 BYTES LONG
1009 68                       PLA
100A CA                       DEX
100B D0 FB                    BNE     PLALP
100D 60              UABORT   RTS                  ABORT USR CALL, NO ACTION

100E 68              PARMOK   PLA                  MOVE ARGUMENTS INTO DATA BASE
100F 8D 01 06                 STA     P0+1         (HIGH ORDER IS FIRST)
1012 68                       PLA
1013 8D 00 06                 STA     P0           THE FIRST ARGUMENT IS THE FIRST TO
1016 68                       PLA                  BE PULLED FROM THE STACK..
```

```
108E 29 08              AND    #$08        RIGHT BIT ?
1090 D0 03              BNE    *+5         NO
1092 20 45 11           JSR    RIGHT       YES, MOVE RIGHT

1017 8D 03 06           STA    P1+1
101A 68                 PLA
101B 8D 02 06           STA    P1

101E 68                 PLA                USE ONLY LOW BYTE OF DIRECTION PATTERNS
101F 68                 PLA
1020 8D 04 06           STA    YPLUS
1023 68                 PLA
1024 68                 PLA
1025 8D 05 06           STA    YMINUS
1028 68                 PLA
1029 68                 PLA
102A 8D 06 06           STA    XPLUS
102D 68                 PLA
102E 68                 PLA
102F 8D 07 06           STA    XMINUS
1032 68                 PLA                DELAY IS BETWEEN 0...255
1033 68                 PLA
1034 8D 08 06           STA    DELA
1037 A9 01              LDA    #$01        INITIALIZE BIT POSITON FOR SAUCER
1039 85 CE              STA    MASK        MOVEMENT MASK


                ;       MAIN USR LOOP   ...   CONTROL MOVEMENT OF SAUCER AND CROSSHAIRS


103B 20 9F 10   LOOP    JSR    MOVTRG      MOVE SAUCER TARGET
103E F0 0E              BEQ    MISSED      HIT LIMIT OF SCREEN -- MISSED
1040 20 60 10           JSR    MOVPLA      MOVE PLAYER CROSSHAIRS
1043 AD 0C D0           LDA    COLSN       CHECK FOR COLLISION WITH ANOTHER PLAYER
1046 D0 0F              BNE    HIT         HIT SOMEONE -> SAUCER
1048 20 0A 11           JSR    DELAY       NOTHING, STALL A BIT
104B 4C 3B 10           JMP    LOOP        REPEAT UNTIL SOMETHING HAPPENS


104E A9 00      MISSED  LDA    #$00        SAUCER MADE IT OFF THE SCREEN BEFORE
1050 85 D4              STA    VALUE       A HIT -- RETURN A ZERO
1052 A9 00              LDA    #$00
1054 85 D5              STA    VALUE+1
1056 60                 RTS


1057 A9 01      HIT     LDA    #$01        GOT THE SAUCER -- RETURN A ONE

1059 85 D4              STA    VALUE
105B A9 00              LDA    #$00
105D 85 D5              STA    VALUE+1
105F 60                 RTS


                ;       MOVE THE PLAYER CROSSHAIRS ACCORDING TO THE BITS SET IN THE
                ;       JOYSTICK PORT.  SHARE UP/DOWN/RIGHT LEFT ROUTINES WITH
                ;       SAUCER MOVE ROUTINES.


1060 AD 00 06   MOVPLA  LDA    P0          INITIALIZE POINTER TO DISPLAY MEMORY
1063 85 D0              STA    PLAY
1065 AD 01 06           LDA    P0+1
1068 85 D1              STA    PLAY+1
106A A2 00              LDX    #$00        PLAYER 0 INDEX
106C AD 00 D3           LDA    PORT        GET JOYSTICK VALUE DIRECTLY
106F 29 0F              AND    #$0F        CLEAN TO ONLY STICK(0)

1071 85 CF              STA    JOY         SAVE FOR LATER
1073 29 01              AND    #$01        CHECK UP BIT ?
1075 D0 03              BNE    *+5         NO STICK CONTACT -- SKIP UP
1077 20 2D 11           JSR    UP          YES, MOVE UP

107A A5 CF              LDA    JOY         USE SAME VALUE OF JOYSTICK
107C 29 02              AND    #$02        DOWN BIT ?
107E D0 03              BNE    *+5         NO -- SKIP DOWN
1080 20 16 11           JSR    DOWN        YES, MOVE DOWN

1083 A5 CF              LDA    JOY
1085 29 04              AND    #$04        LEFT BIT ?
1087 D0 03              BNE    *+5         NO
1089 20 5D 11           JSR    LEFT        YES, MOVE LEFT

108C A5 CF              LDA    JOY
```

```
1095 BD 0B 06           LDA    POSY,X           GET CURRENT Y POSITION ON SCREEN
1098 A8                 TAY
1099 A2 40              LDX    #64              OFFSET INTO TARG FOR CROSSHAIRS PATTERN
109B 20 F8 10           JSR    MOVOBJ           MOVE NEW PATTERN ONTO (POSSIBLY) NEW POSITION
109E 60                 RTS

                ;       MOVE THE SAUCER TARGET ACCORDING TO THE FOUR BIT PATTERNS
                ;       YPLUS, YMINUS, XPLUS, XMINUS.  CHOOSE SAUCER PATTERN TO
                ;       DRAW IN NEW POSITION BASED UPON Y POSITION ON  SCREEN.

109F 06 CE      MOVTRG  ASL    MASK             MOVE TO NEXT BIT POSITION
10A1 D0 04              BNE    BITON            BIT STILL IN MASK
10A3 A9 01              LDA    #$01             WHOOPS, SHIFTED BIT OUT
10A5 85 CE              STA    MASK             RE-INITIALIZE BIT MASK

10A7 AD 02 06   BITON   LDA    P1               INITIALIZE POINTER TO DISPLAY MEMORY
10AA 85 D0              STA    PLAY
10AC AD 03 06           LDA    P1+1
10AF 85 D1              STA    PLAY+1
10B1 A2 01              LDX    #$01             PLAYER 1 INDEX

10B3 AD 04 06           LDA    YPLUS            YPLUS DIRECTION
10B6 25 CE              AND    MASK             TEST UP BIT ?
10B8 F0 06              BEQ    NOYP             NO
10BA 20 2D 11           JSR    UP               YES, MOVE UP
10BD D0 01              BNE    *+3              DID NOT HIT TOP
10BF 60                 RTS                     HIT TOP => THROUGH

10C0 AD 05 06   NOYP    LDA    YMINUS           DOWN ?
10C3 25 CE              AND    MASK
10C5 F0 06              BEQ    NOYM             NO
10C7 20 16 11           JSR    DOWN             YES
10CA D0 01              BNE    *+3
10CC 60                 RTS                     HIT BOTTOM

10CD AD 06 06   NOYM    LDA    XPLUS            RIGHT ?
10D0 25 CE              AND    MASK

10D2 F0 71              BEQ    RIGHT            NO
10D4 20 45 11           JSR    RIGHT            YES
10D7 D0 01              BNE    *+3
10D9 60                 RTS                     HIT RIGHT LIMIT


10DA AD 07 06   NOXP    LDA    XMINUS           LEFT ?
10DD 25 CE              AND    MASK
10DF F0 06              BEQ    NOXM             NO
10E1 20 5D 11           JSR    LEFT             YES
10E4 D0 01              BNE    *+3
10E6 60                 RTS                     HIT LEFT LIMIT

10E7 A2 01      NOXM    LDX    #$01             INDEX TO PLAYER 1
10E9 BD 0B 06           LDA    POSY,X           GET CURRENT Y POSITION
10EC A8                 TAY                     INDEX TO MOVE PATTERN
10ED 4A                 LSR    A                Y POSITION / 4 -- 0<=Y<=63
10EE 4A                 LSR    A
10EF 29 38              AND    #$38             CLEAN UP TO MULTIPLES OF 8
10F1 AA                 TAX                     USE AS INDEX INTO 8 SAUCER TARGET PATTERNS
10F2 20 F8 10           JSR    MOVOBJ           MOVE 8 BYTES OF NEW TARGET SHAPE
10F5 A9 01              LDA    #$01             NO HIT
10F7 60                 RTS

                ;       MOVE 8 BYTES FROM TARG,X TO (PLAY),Y  --  THIS WILL REDRAW PATTERN
                ;       AT A (POSSIBLY) NEW POSITION ON THE SCREEN

10F8 A9 08      MOVOBJ  LDA    #$08             MOVE 8 BYTES -- USE TEMPORARY STORAGE
10FA 8D 0D 06           STA    COUNT
10FD BD 0E 06   MVPLAL  LDA    TARG,X           GET TARGET PATTERN BYTE
1100 91 D0              STA    (PLAY),Y         PUT INTO PROPER DISPLAY MEMORY
1102 E8                 INX                     NEXT PAIR OF BYTES
1103 C8                 INY
1104 CE 0D 06           DEC    COUNT            ALL 8 ?
1107 D0 F4              BNE    MVPLAL           NO
1109 60                 RTS


                ;       BY THE 7TH ARGUMENT IN THE USER CALL (ONE BYTE VALUE).

110A AC 08 06   DELAY   LDY    DELA             GET BASIC DELAY 0...255 (1 IS FASTEST)
```

```
110D A2 05      DLOOP1  LDX    #$05          ARBITRARY CONSTANT TO MAKE A BASIC DELAY
110F CA         DLOOP2  DEX                  OF 200 SLOW, 100 FAST
1110 D0 FD              BNE    DLOOP2
1112 88                 DEY
1113 D0 F8              BNE    DLOOP1
1115 60                 RTS

                ;       SET PARAMETERS FOR DOWN MOVEMENT (UP IN DISPLAY MEMORY)
                ;
                ;       NOTE:  THE MOVOBJ ROUTINE ACTUALLY REDRAWS THE PATTERN AT THE
                ;              NEW POSITION.  THUS ONLY 1 BYTE NEEDS TO BE ZEROED BY
                ;              THE DOWN/UP ROUTINES.

1116 BD 0B 06   DOWN    LDA    POSY,X        GET CURRENT POSITION
1119 C9 BA              CMP    #186          AT SCREEN LIMIT ???
111B F0 0D              BEQ    NODOWN        YES, IGNORE MOVEMENT

111D FE 0B 06           INC    POSY,X      . NO, MARK NEW POSITION
1120 EA                 NOP                  SYNCHRONIZE ALL MOVEMENT SUBROUTINES TO
1121 EA                 NOP                  TAKE SAME NUMBER OF MACHINE CYCLES (TIME)
1122 A8                 TAY
1123 A9 00              LDA    #$00          ZERO OLD TOP -- WILL NOT BE OVERWRITTEN
1125 91 D0              STA    (PLAY),Y      WITH NEW PATTERN
1127 A9 01              LDA    #$01          SUCCESSFUL MOVE
1129 60                 RTS

112A A9 00      NODOWN  LDA    #$00          NO ACTUAL MOVE (IGNORE TIME SYNC)
112C 60                 RTS

                ;       SET PARAMETERS FOR UP MOVE (DOWN IN DISPLAY MEMORY)

112D BD 0B 06   UP      LDA    POSY,X        GET CURRENT Y POSITION
1130 C9 1C              CMP    #28           AT SCREEN LIMIT ???
1132 F0 0E              BEQ    NOUP          YES, IGNORE MOVE

1134 DE 0B 06           DEC    POSY,X      . NO, MARK NEW POSITION
1137 18                 CLC                  POINT TO BOTTOM OF OLD PATTERN
1138 69 07              ADC    #$07
113A A8                 TAY
113B A9 00              LDA    #$00          ZERO IT
113D 91 D0              STA    (PLAY),Y
113F A9 01              LDA    #$01          SUCCESSFUL MOVE
1141 60                 RTS

1142 A9 00      NOUP    LDA    #$00          NO MOVE
1144 60                 RTS
                ;       SET PARAMETERS FOR RIGHT MOVE.
                ;
                ;       NOTE:  RIGHT/LEFT MOVEMENT IS DONE BY THE HARDWARE SET IN THESE
                ;              TWO ROUTINES.
1145 BD 09 06   RIGHT   LDA    POSX,X        GET CURRENT X POSITION
1148 C9 CC              CMP    #204          AT SCREEN LIMIT ???
114A F0 0E              BEQ    NORGT         YES, NO MOVE

114C FE 09 06           INC    POSX,X        NO, UPDATE TO NEW POSITION
114F BD 09 06           LDA    POSX,X        USE NEW POSITION
1152 9D 00 D0           STA    HORZ,X        DIRECTLY SET HORIZONTAL REGISTER
1155 EA                 NOP                  TIME SYNCHRONIZATION WITH SLOWEST CASE (UP)
1156 EA                 NOP
1157 A9 01              LDA    #$01          SUCCESSFUL MOVE
1159 60                 RTS

115A A9 00      NORGT   LDA    #$00          NO MOVE
115C 60                 RTS

                ;       SET PARAMETERS FOR LEFT MOVE

115D BD 09 06   LEFT    LDA    POSX,X        GET CURRENT X POSITION

1160 C9 2D              CMP    #45           AT SCREEN LIMIT ???
1162 F0 0E              BEQ    NOLFT         YES, NO MOVE

1164 DE 09 06           DEC    POSX,X        NO, UPDATE TO NEW POSITION
1167 BD 09 06           LDA    POSX,X        USE NEW POSITION
116A 9D 00 D0           STA    HORZ,X        SET HORIZONTAL REGISTER
116D EA                 NOP
116E EA                 NOP
116F A9 01              LDA    #$01          SUCCESSFUL MOVE
1171 60                 RTS

1172 A9 00      NOLFT   LDA    #$00          NO MOVE
1174 60                 RTS
```

Software for the Apple II and Apple II Plus*

BENEATH APPLE DOS

A Technical Manual

By Don Worth and Pieter Lechner

Become an expert on the intricacies of Apple's DOS (Disk Operating System). BENEATH APPLE DOS is the perfect companion to Apple's DOS 3.3 Manual. Containing eight chapters, three appendices, a glossary, an index, and over 160 pages, this manual will serve to completely fill in the many gaps left by Apple's DOS 3.3 Manual. Written for Apple users with DOS 3.3, 3.2 or earlier versions, any Apple disk user would welcome having this carefully written manual at his fingertips.

LEARN . . .

- How DOS 3.3 differs from other DOS versions.
- How disks are protected.
- How to reconstruct a damaged diskette CATALOG.
- How tracks are formatted.
- How to use the disk directly, *without* DOS.
- How to call DOS's file manager.
- How *every* routine in DOS works.
- How to customize DOS to your needs.
- How to overcome DISK I/O ERRORS.
- About the "secret" file types — S and R.

INCLUDES . . .

- Large quantities of excellent diagrams and tables.
- Source listings of useful disk utilities.
- Glossary of over 150 technical terms.
- Exhaustive description of DOS program logic.
- Handy reference card.
- Useful patches to DOS.
- Many programming examples.

Book - $19.95

CROSS-REF by Jim Aalto

Applesoft programmers will be delighted to have this cross reference utility program in their 'tool kit' of software aids. What can CROSS-REF do to speed and facilitate your Applesoft program development? Consider these functions:

VARIABLE CROSS REFERENCE	LINE CROSS REFERENCE
FIND VARIABLE	FIND LINE NUMBER
REPLACE VARIABLE	VARIABLE ONLY LISTING
LINE ONLY LISTING	

Features that make CROSS-REF easy to use include:

- Written in machine language, occupies less than 3K.
- Resides passively in memory while DOS or Applesoft is active.
- Can be loaded with your Applesoft program already resident.
- Very fast — a VARIABLE CROSS REFERENCE for a 16K Applesoft program can start printing in 5 seconds.
- Contains printer format controls and headers for documentation.
- Prints English language error messages.

Cassette - $22.95 Diskette - $24.95

L I N K E R

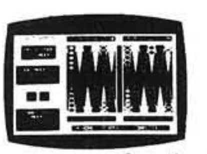
LINKER by Don Worth.

Turn your Apple II or Apple II Plus into a powerful and productive software development machine with this superb linking loader/editor package. LINKER does the following and much more:

- Dynamically loads and relocates suitably prepared machine language programs anywhere in RAM.
- Combines a main program with subroutines. You can assemble a subroutine once and then use it with as many main programs as you wish.
- Produces a map of all loaded routines, giving their location and the total length of the resulting module.
- Contains a library of subroutines including binary multiplication and division, print text strings, delay, tone generator, and random number generator.

Linker works with virtually any assembler for the Apple II. Requires 32K of RAM and one disk drive.

Diskette - $49.95
Manual Only - $19.95



FASTGAMMON™ By Bob Christiansen.

Sound, hi res, color, and musical cartoons have helped make this the most popular backgammon playing game for the Apple II. But don't let these entertaining features fool you — FASTGAMMON plays serious backgammon. Runs on any Apple II with at least 24K of RAM.

Cassette - $19.95 Diskette - $24.95

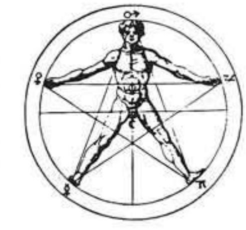
METEOROIDS IN SPACE™

By Bruce Wallace

We have taken our popular space game, formerly called Asteroids in Space, and made some important improvements. To accent these improvements we have given it a new name — METEOROIDS IN SPACE. Your space ship travels through a shower of deadly meteoroids. If your ship is hit, it will be destroyed, so you use your laser gun to blast the meteoroids. Big meteoroids shatter into smaller meteoroids when hit, and the smaller ones are usually faster and just as deadly. From time to time you will encounter an alien space ship whose mission is to destroy you, so you'd better destroy it first. All the action is displayed in fast, smooth, high resolution graphics, accompanied by sound effects. You now can control your ship using one of two options — the Apple game paddles or the keyboard. One of the game paddle buttons controls the laser fire. In METEOROIDS IN SPACE, the spaceship's velocity gradually decreases unless more thrust is applied, adding an element of control. Also new to this version is a hyperspace feature — translate instantly to another spot in the galaxy. The game is over when five of your ships have been destroyed. An additional ship is added for every 10,000 points you score. Runs on any Apple II with at least 32K of RAM and one disk drive.

Diskette - $19.95

ASTROAPPLE™ by Bob Male.



Your Apple computer becomes your astrologer, generating horoscopes and forecasts based on the computed positions of the heavenly bodies. This program offers a delightful and stimulating way to entertain friends. ASTROAPPLE produces natal horoscopes (birth charts) for each person based on his or her birth data. Any two people may be compared for physical, emotional, and intellectual compatibility. The program is written in Applesoft BASIC with machine language subroutines. It requires either RAM or ROM Applesoft and at least 32K of memory.

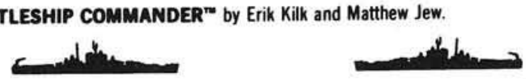
Cassette - $14.95 Diskette - $19.95

FRACAS™ by Stuart Smith.

A fantastic adventure game like no other! Up to eight players can participate in FRACAS at the same time. Journey in the land of FAROPH, searching for hidden treasure while warding off all sorts of unfriendly and dangerous creatures. You and your friends can compete with each other or you can join forces and gang up on the monsters. Your location is presented graphically and sound effects enliven the battles. Save your adventure on diskette or cassette and continue it at some other time. Both integer BASIC and Applesoft versions included. Requires at least 32K of RAM.

Cassette - $19.95 Diskette - $24.95

BATTLESHIP COMMANDER™ by Erik Kilk and Matthew Jew.



A game of strategy. You and the computer each start out by positioning five ships of different sizes on a ten by ten grid. Then the shooting starts. Place your volleys skillfully — a combination of logic and luck are required to beat the computer. Cartoons show the ships sinking and announce the winner. Sound effects and flashing lights also add to the enjoyment of the game. Both Applesoft and integer BASIC versions are included. Requires at least 32K of RAM.

Cassette - $14.95 Diskette - $19.95

Also by Don Worth . . .

BENEATH APPLE MANOR — Adventure. Uses Integer BASIC.

Cassette - $14.95 Diskette - $19.95

BABBLE - Fun with words, sound, and graphics.

Cassette - $19.95 Diskette - $24.95

QS QUALITY SOFTWARE

6660 Reseda Blvd., Suite 105, Reseda, CA 91335

(213) 344-6599

Now exclusive distributors for products from The Software Factory, Newhall, California

*Apple II and Apple II Plus are trademarks of Apple Computer, Inc.

WHERE TO GET IT: Call us at (213) 344-6599 for the name of the Quality Software dealer nearest you. If necessary you may order directly from us. Mastercard and Visa cardholders may place orders by telephone. Or mail your check or bankcard number to Quality Software, 6660 Reseda Blvd., Suite 105, Reseda, CA 91335. California residents add 6% sales tax. SHIPPING CHARGES: Within North America orders must include $1.50 for first class shipping and handling. Outside North America the charge for airmail shipping and handling is $5.00. Pay in U.S. currency.

Othello

This program simulates the popular board game Othello. Designed for two players, the program maintains the Othello board on the Apple Lo-Res graphics screen. Written in Applesoft BASIC, Othello should be easily modifiable to other dialects of BASIC.

Charles F. Taylor, Jr.
587F Sampson Lane
Monterey, California 93940

Most computer game programs are designed to be played by one person. The computer plays the role of opponent, scorekeeper, referee, and manager of the display. This results in a "man-against-machine" scenario. The objective is to "beat the computer" and thereby establish your intellectual superiority over silicon circuitry. (Never mind that you are really playing against an algorithm designed by another person.)

This game program, Othello, is designed to be played by two persons. The computer no longer is the opponent, but plays the role of slave, keeping track of the board position, checking for illegal moves, keeping score, and managing the display.

Background

I wrote this program for my ten-year-old son. Othello is a good game for interaction across the generation gap because it is more than challenging enough for me, but not too difficult for my son. He beats me more often than I care to admit!

Perhaps the best way to describe the game of Othello is to describe how it is played as a board game, without the aid of the computer. The playing board is 8 squares by 8 squares, much like a checker or chess board, except that all squares are usually the same color. The playing pieces are disks, black on one side and white on the other. Each player starts with 32 pieces; one player is designated "white" and the other "black."

The game begins with two pieces of each color in the center of the board in the configuration shown in figure 1. White has the first turn. He must place a white piece (a piece with the white side up) in such a manner as to "capture" a black piece. A piece is captured when it is "surrounded" by pieces of the opposite color, either horizontally, vertically, or diagonally. Captured pieces are turned over and become the color of the captor. More than one piece can be captured at a time.

Figure 2 illustrates the capture of two black pieces by a white piece. A move is not legal unless it accomplishes one or more captures. The game is won by either capturing all of your opponent's pieces, or by having more pieces than your opponent at the end of the game.

Implementation

The program was written in Applesoft BASIC on an Apple II Plus. Low-resolution graphics are used to display the game board, thus pieces are shown as square rather than round. The selection of colors is easily changed to suit your own display (see lines 280 - 300). I am currently using a "green screen" monitor and find it hard to judge colors as they might appear on another display.

The program is shown in listing 1. The coding is straightforward, but perhaps a few comments are in order. The board is represented internally by the array "BOARD." The function "FN M2(Q)" finds the modulus base 2 of a number (the remainder after integer division by 2) and is used to compute whose turn it is. The legality of each move is checked. The subroutine at 1430 searches for and executes all possible captures, beeping for each capture. The score is displayed after each move.

***Figure 1:* Initial Board Configuration**

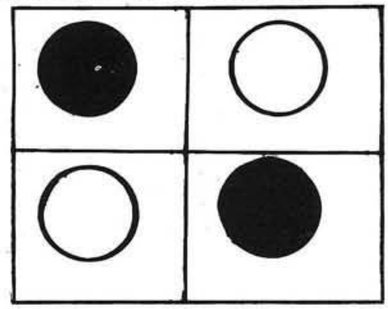
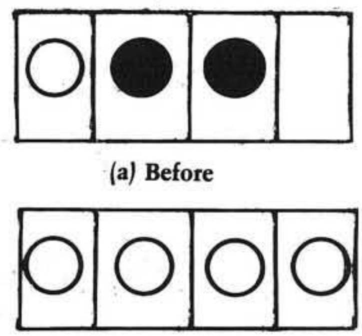


***Figure 2:* White Captures Two Black Pieces**



(a) Before

(b) After

Play

To move, a player types the row and column where he wants to place his piece. Columns are labeled A-H, left to right; rows are labeled 1-8, bottom to top. The lower left corner is then A1, the lower right corner H1, and so on. Should you ever find yourself in a position such that no legal moves are possible, type "P" for "Pass." Play tends to ebb and flow like the tides, but without any predictability. A player can be comfortably ahead at one moment and hopelessly behind the next. Ah, the changes of fortune! Closer analysis will reveal, however, that skill plays a much more significant role in the play than does fortune.

Charles Taylor is on the faculty at the Naval Postgraduate School in Monterey, California, where he teaches courses in Operations Research and Computer Science.

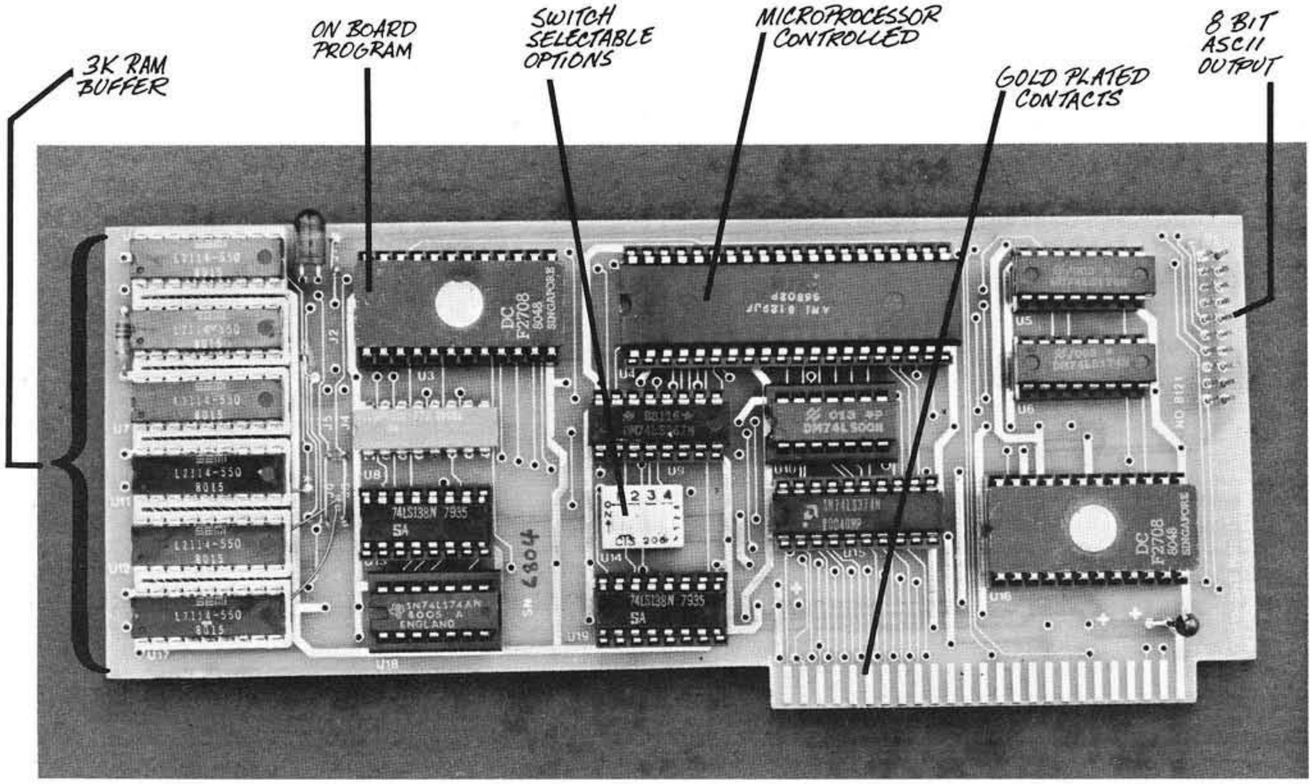
(Continued on next page)

Listing

```
100  REM  * THE GAME OF OTHELLO
110  REM  * BY C. F. TAYLOR, JR.
120  REM  * FOR NAT TAYLOR
130  REM  *
140  REM
150  REM  INITIALIZE
160  DIM BOARD(9,9)
170  DIM CC(2): REM  HOLDS CURRENT COLOR
180  DIM PROMPT$(2)
190  DIM SC(2)
200  DIM DX(8): DIM DY(8)
210  DEF  FN M2(Q) = Q -  INT (Q / 2) * 2
220 PROMPT$(1) = "INPUT WHITE MOVE:"
230 PROMPT$(2) = "INPUT BLACK MOVE:"
240 BLACK = 0
250 WHITE = 15
260 CC(1) = WHITE
270 CC(2) = BLACK
280 BC = 12: REM  BACKGROUND COLOR
290 TC = 13: REM   TITLE COLOR
300 DC = 4: REM  BORDER COLOR
310  DATA  0,1,1,1,0,-1,-1,-1
320  DATA  1,1,0,-1,-1,-1,0,1
330  FOR I = 1 TO 8: READ DX(I): NEXT I
340  FOR I = 1 TO 8: READ DY(I): NEXT I
350  FOR I = 0 TO 9
360  FOR J = 0 TO 9
370 BOARD(I,J) = 0
380  NEXT J,I
390  GOSUB 780
400  COLOR= WHITE
410 X = 5:Y = 5
420 BOARD(X,Y) = 1
430  GOSUB 1260: REM  CALL BLOT
440 X = 4:Y = 4
450 BOARD(X,Y) = 1
460  GOSUB 1260: REM  CALL BLOT
470 SC(1) = 2
480  COLOR= BLACK
490 X = 4:Y = 5
500 BOARD(X,Y) = 2
510  GOSUB 1260: REM  CALL BLOT
520 X = 5:Y = 4
530 BOARD(X,Y) = 2
540  GOSUB 1260: REM  CALL BLOT
550 SC(2) = 2
560 TURN = 2
570  REM  BEGIN MAIN LOOP
580  FOR Q = 1 TO 100
590 TURN =  FN M2(TURN) + 1
600  COLOR= CC(TURN)
610  PRINT "SCORE IS:  WHITE ";SC(1);"
     BLACK ";SC(2)
620  PRINT PROMPT$(TURN)
630  GOSUB 1330: REM  CALL GETMOVE
640  IF PASS THEN 700
650  IF BOARD(X,Y) <  > 0 THEN 620
660  GOSUB 1430: REM  CALL MOVES
670  IF FLAG = 0 THEN 620
680  IF ((SC(1) + SC(2)) = 64) THEN 710
690  IF ((SC(1) = 0) OR (SC(2) = 0))
     THEN 710
700  NEXT Q
710  IF SC(1) > SC(2) THEN  PRINT
     "WHITE WINS!": GOTO 740
720  IF SC(1) < SC(2) THEN  PRINT
     "BLACK WINS!": GOTO 740
730  PRINT "IT'S A TIE!!"
740  PRINT "FINAL SCORE:  WHITE ";SC(1);"
     BLACK ";SC(2)
750  INPUT "WOULD YOU LIKE TO PLAY AGAIN?
     ":A$
760  IF  LEFT$ (A$,1) = "Y" THEN 350
770  END
780  REM  SUBR TO DRAW OTHELLO BOARD
790  GR
800  COLOR= BC
810  FOR I = 0 TO 39
820  HLIN 1,39 AT I
830  NEXT I
840  COLOR= TC: REM  TITLE COLOR
850  REM  PLOT "OTHELLO"
860  REM  FIRST "O"
870  VLIN 1,5 AT 7
880  PLOT 8,1
890  PLOT 8,5
900  VLIN 1,5 AT 9
910  REM  NEXT "T"
920  HLIN 11,13 AT 1
930  VLIN 2,5 AT 12
940  REM  NEXT "H"
950  VLIN 1,5 AT 15
960  PLOT 16,3
970  VLIN 1,5 AT 17
980  REM  NEXT "E"
990  VLIN 1,5 AT 19
1000  HLIN 20,21 AT 1
1010  PLOT 20,3
1020  HLIN 20,21 AT 5
1030  REM  NEXT TWO "L"S
1040  VLIN 1,5 AT 23
1050  HLIN 24,25 AT 5
1060  VLIN 1,5 AT 27
1070  HLIN 28,29 AT 5
1080  REM  FINALLY ANOTHER "O"
1090  VLIN 1,5 AT 31
1100  PLOT 32,1
1110  PLOT 32,5
1120  VLIN 1,5 AT 33
1130  REM  NOW DO BOARD ITSELF
1140  COLOR= DC: REM  BORDER COLOR
1150  FOR I = 7 TO 39 STEP 4
1160  HLIN 4,36 AT I
1170  NEXT I
1180  FOR I = 4 TO 36 STEP 4
1190  VLIN 8,38 AT I
1200  NEXT I
1210  RETURN
1220  REM  SUBR MAP FINDS SCREEN COORDS
      (XS,YS) GIVEN BOARD COORDS (X,Y)
1230 XS = 1 + 4 * X
1240 YS = 40 - 4 * Y
1250  RETURN
1260  REM  SUBR BLOT FILLS IN A SQUARE
      WITH THE CURRENT COLOR
1270  GOSUB 1220
1280 X2 = XS + 2
1290  HLIN XS,X2 AT YS
1300  HLIN XS,X2 AT YS + 1
1310  HLIN XS,X2 AT YS + 2
1320  RETURN
1330  REM  SUBR GETMOVE
1340  INPUT MOVE$
1350 PASS = 0
1360  IF  LEFT$ (MOVE$,1) = "P" THEN
     PASS = 1: RETURN
1370  IF  LEN (MOVE$) <  > 2 THEN 1340
1380 X =  ASC ( LEFT$ (MOVE$,1)) - 64
1390  IF X < 1 OR X > 8 THEN 1340
```

```
1400 Y =  ASC ( RIGHT$ (MOVE$,1)) - 48
1410  IF Y < 1 OR Y > 8 THEN 1340
1420  RETURN
1430  REM  FIND AND EXECUTE MOVES
1440 FLAG = 0
1450 OP = 3 - TURN: REM COLOR OF OPPONENT
1460  FOR I = 1 TO 8
1470 NR = 0
1480 XN = X:YN = Y
1490 XN = XN + DX(I):YN = YN + DY(I)
1500  IF BOARD(XN,YN) = OP THEN NR =
      NR + 1: GOTO 1490
1510  IF (BOARD(XN,YN) = 0) OR (NR = 0)
      THEN 1700
1520  REM  IF WE GET HERE,
      CAPTURE IS POSSIBLE
1530 FLAG = 1
1540  COLOR= CC(TURN)
1550  IF BOARD(X,Y) <  > 0 THEN 1590
1560  GOSUB 1260: REM  CALL BLOT
1570 BOARD(X,Y) = TURN
1580 SC(TURN) = SC(TURN) + 1
1590  FOR J = 1 TO NR
1600 XN = XN - DX(I):YN = YN - DY(I)
1610 BOARD(XN,YN) = TURN
1620 XTEMP = X:YTEMP = Y
1630 X = XN:Y = YN
1640  GOSUB 1260: REM  CALL BLOT
1650 X = XTEMP:Y = YTEMP
1660  PRINT  CHR$ (7)
1670 SC(TURN) = SC(TURN) + 1
1680 SC(OP) = SC(OP) - 1
1690  NEXT J
1700  REM
1710  NEXT I
1720  RETURN
```

SMART II MEANS FAST.



SMART II MICROPROCESSOR CONTROLLED PARALLEL PRINTER INTERFACE

Be Smart! With the new **SMART II** parallel printer interface for your Apple II* Computer you can have print spooling, left and right margin control, and adjustable tab stops. The **SMART II** can buffer over three thousand characters before it signals the Apple to stop sending. This eliminates the start - stop problem created with conventional printer cards and will keep your printer printing (instead of waiting).

The **SMART II** is compatible with all known hardware and software including the Pascal Language System, Microsoft Z-80 Softcard*, and Hayes Micromodem II*.

FEATURES:

- Compatible with all Centronics-type Parallel printers including the Epson MX-70/80/100, Centronics 737/739/779, IDS 440/445/460/560, C. Itoh Starwriter, Anadex 8000/9000/9500, and similar printers.
- 3K Print Spooler which acts much larger when spooling text because of a unique compaction routine.
- On board software supports typewriter-like TAB Commands and has 16 software selectable TAB positions. Left and right margin commands are also software selectable to ease in the justification of reports and listings.
- Use with the Hayes Micromodem II* to prevent loss of characters while on line with a host computer.

AVAILABLE AT YOUR LOCAL APPLE DEALER

INTRODUCTORY RETAIL PRICE **$225.** (cable and connector included)

HARDWARE: 6800 type microprocessor
Two ROMs
Six static RAMs
Eight support ICs
4 ft printer cable and connector
High quality board with gold plated edge connector

OLENSKY BROS., INC.
COMPUTER SALES DIVISION
3763 AIRPORT BLVD.
MOBILE, AL 36608
TOLL FREE: 800-633-1636
DEALER INQUIRIES INVITED.

* Apple is a registered trademark of Apple Computer, Inc.
* Z-80 Softcard is a registered trademark of Microsoft.
* Micromodem II is a registered trademark of Hayes, Inc.

Ultimate Ping-Pong for PET

This version of the popular "Pong" game features CB2 sound and selectable paddle widths and speeds. In addition, the paddles can be moved in and out toward the net. The graphics are all handled in machine language for all 40-column PETs, and should serve as an example for other high-speed graphic applications.

Werner Kolbe
Hardstr. 77
CH 5432 Neuenhof
Switzerland

Soon after I purchased my CBM 3040 floppy disk drive and the Commodore Assembly Language Development package, I developed a Ping-Pong program, which existed already in a simpler monitor-written version. To modify it, I prepared a file for the editor using my symbolic disassembler (MICRO 32:23). Then I could insert, change and modify whatever I wanted without keeping in mind all the addresses and pointers. Many of the labels used (L31, J4, etc.) were created in the symbolic disassembly process.

Program Description

The program consists of two parts. One is in BASIC and contains mostly the description, and the other is in machine code and allows the fast graphics.

This is a two-player game. Each player has four keys to control the movement of the paddles in the four directions. Also, to put the ball into the game each player has a service key. The service is only allowed with the paddle at the end of the table, but after the serve it is possible to move near to the net. You can select different widths of the paddles and different speeds of the game. (The program uses the CB2 as sound output.)

The BASIC program first determines which player has the serve by the random number in variable A. The direction of the ball is set randomly in variable B. Then in line 15 we jump into machine language. We enter the main program, which is a set of subroutines that initiate the pointers. Next we draw the net and the table and then end in a closed loop that waits until the game is finished.

The game itself happens in the hardware interrupt cycle, which is initiated every 60th of a second. This has two advantages:

1. the timing is easily accomplished (but only in steps of 1/60 seconds);

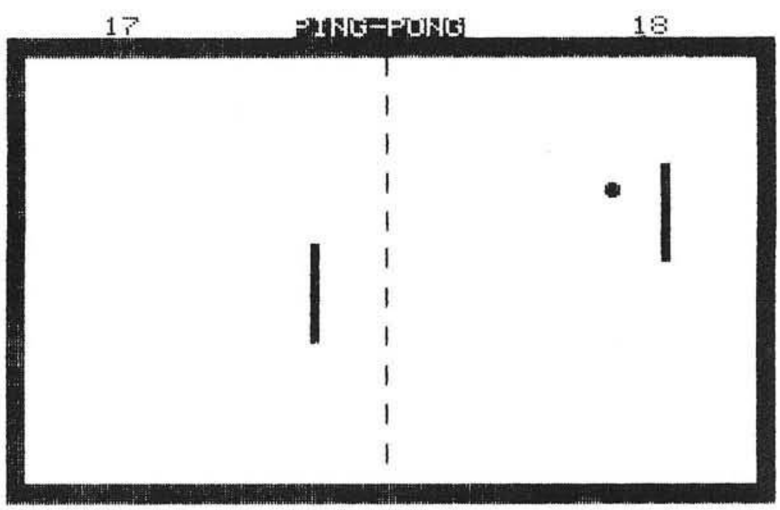
2. the "snow" on the screen is avoided as the interrupt is initiated with the retrace of the beam when it is dark.

In the interrupt the following subroutines are executed:

SCAN scans the keyboard and puts the results into RSLTS and following. I think that is a generally useful routine since which keys are sensed is determined by TABL3 (row) and TABL4 (column). The routine is made to store the result of a pair of keys into one byte that contains FF if the first is pressed, 01 if the second is pressed, and 00 if both are pressed.

RLEFT, RRIGHT move the paddles according to the keys that are pressed. By simply storing 0 into the place of RSLTS for the side movement, they are reset during service. The routines are designed to save everything that comes under the paddle, to make it possible, for example, to write text onto the table during the game. But I later omitted this for the clarity of the game.

SERVE does the service. It needs some calculation to let the ball start right in front of the racket, as this can have different sizes and different positions.



MVEBAL moves the ball and reflects it accordingly. When testing the game I found I had forgotten that there is also a reflection that is necessary if we hit the ball when moving the racket forward. (Otherwise we would just go over the ball.) For this purpose I inserted the PATCH that jumps into a side entry (J37) of MVEBAL. Not very elegant but it works. Every reflection produces a sound that is taken from the SNDS-table.

The details of the program are given in the remarks, as far as possible. If the ball is not returned from one side, the endflag becomes zero and after restoring the interrupt pointer we return to BASIC.

There the score is counted in ZR and ZL and the service is changed accordingly. If one side gets more than 20 points and is over 1 point higher than the other side, the game is finished.

Entering the Program

The best way to enter the program is to use an assembler. If you haven't got one you can use a monitor, preferably

Listing 1

```
1 GOSUB500:POKE1,144:POKE2,0
2 IFPEEK(537)=133THENPOKE1,25:POKE2,2
5 M=59464:A=255:IFRND(1)>.5THENA=1
10 POKE2531,A:PRINT"":GOSUB440
12 IFA=1THENPRINT"              SERVICE ON THE LEFT!":GOTO14
13 PRINT"              SERVICE ON THE RIGHT!"
14 B=39-176*(RND(1)>.5):IFA=1THENB=41-176*(RND(1)>.5)
15 POKE2513,B:SYS2304
30 B=PEEK(181):IFB=39ORB=215THENZR=ZR+1:GOTO40
35 ZL=ZL+1
40 GOSUB440:B=69:FORJ=1TO40:POKEM,B
50 FORI=0TO8:NEXT:B=162-B:POKEM,B:FORI=0TO8:NEXT:NEXT:POKEM,0
90 IFZR>20ORZL>20THEN120
100 IFINT((ZR+ZL)/5)=(ZR+ZL)/5THEN130
110 GOTO10
120 IFABS(ZR-ZL)>1THEN200
130 A=256-A:GOTO10
200 POKE59467,0:PRINT"ANOTHER GAME (Y/N) ?"
210 GETA$:IFA$<>"Y"ANDA$<>"N"THEN210
220 IFA$="Y"THENRUN5
230 PRINT"GOOD BYE!":END
440 PRINT"   "ZL:PRINT""SPC(15)"PING-PONG"SPC(10)ZR:RETURN
500 PRINT""SPC(8)"*** PING PONG ***"
510 PRINT"MOVE THE PADDLES :"
520 PRINT"ON THE LEFT   WITH '!' AND 'SHIFT' AND"
530 PRINT"              SIDEWAYS WITH 'A' AND 'D'"
540 PRINT"ON THE RIGHT  WITH 'DEL' AND '=' AND"
550 PRINT"              SIDEWAYS WITH '5' AND '*'
560 PRINT"SERVICE : ON THE LEFT WITH 'S'"
570 PRINT"           ON THE RIGHT WITH '6'.
580 PRINT"       HAVE A GOOD TIME"
590 PRINT"         WERNER KOLBE"
600 GETA$:IFA$=""THEN600
601 PRINT"HOOK A SPEAKER TO CB2 !"
610 INPUT"WIDTH  OF PADDLES (1-10)";B:IFB<1ORB>10THEN610
620 POKE2522,B
625 INPUT"SPEED (1-5)";B:IFB<1ORB>5THEN625
630 POKE2520,6-B
650 POKE2521,6-B
660 RETURN
```

Listing 2 A

```
                  ;*********************
                  ;*                   *
                  ;*     PING-PONG     *
                  ;*                   *
                  ;* BY WERNER KOLBE   *
                  ;* HARDSTRASSE 77    *
                  ;* CH 5432 NEUENHOF  *
                  ;* SWITZERLAND       *
                  ;*                   *
                  ;*********************
                  ;
                          .BA $900
                  ;
0900- 4C 33 0D    ENTRY   JMP MAIN
                  TX1     .DI ENTRY+3      ;FOR CONTENTS OF
                  TX2     .DI ENTRY+$69    ; THESE TABLES
                  TX3     .DI ENTRY+$7B    ; AND TEXT AREAS
                  TABL3   .DI ENTRY+$8B    ; SEE THE
                  TABL4   .DI ENTRY+$9B    ; SEPARATE
                  TBL1    .DI ENTRY+$AB    ; HEX DUMP
                  TBL2    .DI ENTRY+$B3    ; PROVIDED
                  SNDS    .DI ENTRY+$C3
                  TBL0    .DI ENTRY+$CB
                  ;
                  SAVIT   .DI TBL0+$20     ;UNDER THE PADDLES
                  ZERO    .DE $00          ;IND. JUMP
                  Z3      .DE $01          ;OLD INTERRUPT VECTOR
                  BALPOS  .DE $B1          ;BALL POSITION POINTER
                  TEMP1   .DE $B3
                  TEMP2   .DE $B4
                  INCR    .DE $B5          ;BALL DIRECTION
                  RRPOS   .DE $B6          ;POS. OF RIGHT PADDLE
                  RLPOS   .DE $B8          ;POS. OF LEFT PADDLE
                  PRVCHR  .DE $BB          ;UNDER THE BALL
                  DLYB    .DE $BC          ;DELAY OF BALL
                  WIDTH   .DE $BE          ;WIDTH OF PADDLES
                  RRYS    .DE $BF          ;HORIZ. SHIFT RIGHT
                  RLYS    .DE $C0          ;AND LEFT
                  RSLTS   .DI TBL1+$10     ;LEFT UP=FF, DOWN=01
                  W20     .DI RSLTS+1      ;LEFT <=FF, >=01
                  W23     .DI RSLTS+2      ;RIGHT UP=FF, DOWN=01
                  W24     .DI RSLTS+3      ;RIGHT <=FF, >=01
                  W29     .DI RSLTS+4      ;SERVICE RGT=01, LFT=FF
                          ;INITIAL VALUES
                  W4      .DI TBL0+2
                  W5      .DI TBL0+$0D     ;BALL DELAY
                  DLYR    .DI TBL0+$0E     ;RACK DELAY
                  DURTN   .DI TBL0+$12     ;DURATION OF SOUND
                  ENDFLG  .DI TBL0+$14     ;PLAY STOP FLAG
                  CNTRR   .DI TBL0+$15     ;DELAY COUNTER RIGHT
                  CNTRL   .DI TBL0+$16     ;DELAY COUNTER LEFT
                  CNTSND  .DI TBL0+$17     ;DELAY COUNTER SOUND
                  SDEFLG  .DI TBL0+$18     ;SERVE SIDEFLAG
                  W21     .DI SAVIT-1      ;AUXILLIARIES
                  W25     .DI SAVIT+$0F
                  SCR1    .DE $8004        ;SPECIFIC SCREEN LOCATIONS
                  SCR2    .DE $8023
                  SCRL1   .DE $8028
                  SCR3    .DE $8118
                  SCR4    .DE $8172
                  SCR5    .DE $823B
                  W40     .DE $8272
                  SCR6    .DE $82B5
                  W2      .DE $83C0
                  W15     .DE $E810        ;KEYBOARD SELECT ROW
                  W17     .DE $E812        ;SENSE PB
                  W9      .DE $E848        ;THE 6522 CB2 SOUND REGIST
                  W27     .DE $E84A
                  W26     .DE $E84B
                  ;
                          .BA TBL0+$45
                  ;
                  ;SOME ADVERTISING
0A10- A9 30       DISPLY  LDA #'0
0A12- CD 23 80            CMP SCR2
0A15- D0 4A               BNE L75

0A17- CD 04 8             CMP SCR1
0A1A- D0 45               BNE L75
0A1C- A2 00               LDX #$00
0A1E- BD 03 09    L77     LDA TX1.X
0A21- F0 06               BEQ L76
0A23- 9D 72 81            STA SCR4.X
0A26- E8                  INX
```

(Continued on next page)

one with a disassembler. After having entered BASIC, the easiest way to combine both is to save it from the monitor: S "1:PING-PONG",08,0400,0DA4.

For the tape you type 1 instead of 8. Before you run it you must reload the program from disk/tape in order to protect the machine code from being destroyed by the BASIC variables. But it is also good to have the BASIC part saved separately, because necessary changes may destroy machine code in the combination.

Editorial Note: This program has been tested by the MICRO staff on all three 40-column versions of the PET. It will run as is on 3.0 and 4.0 PETs, with as little as 4K RAM, but will require the following changes for 1.0 (old) PETs.

In the assembly language program, change the definitions of all addresses in the range $B1-$C0 to the corresponding addresses in the range $11-$20. In the BASIC program change line 30 to read:

30 B = PEEK(21):IFB = 39ORB = 215THENZR = ZR + 1:GOTO40

```
0A27- D0 F5              BNE L77
0A29- AA           L76   TAX
0A2A- BD 69 09     L79   LDA TX2,X
0A2D- F0 06              BEQ L78
0A2F- 9D 3B 82           STA SCR5,X
0A32- E8                 INX
0A33- D0 F5              BNE L79
0A35- AA           L78   TAX
0A36- BD 7B 09     L81   LDA TX3,X
0A39- F0 06              BEQ L80
0A3B- 9D B5 82           STA SCR6,X
0A3E- E8                 INX
0A3F- D0 F5              BNE L81
0A41- A0 0A        L80   LDY #$0A
0A43- 84 00              STY *ZERO
0A45- A0 00        L84   LDY #$00
0A47- A2 00        L83   LDX #$00
0A49- CA           L82   DEX
0A4A- D0 FD              BNE L82
0A4C- 88                 DEY
0A4D- D0 F8              BNE L83
0A4F- C6 00              DEC *ZERO
0A51- D0 F2              BNE L84
0A53- A9 20              LDA #$20
0A55- 9D 72 81     L85   STA SCR4,X
0A58- CA                 DEX
0A59- D0 FA              BNE L85
0A5B- 9D 72 82     L86   STA W40,X
0A5E- CA                 DEX
0A5F- D0 FA              BNE L86
0A61- 60           L75   RTS
                   ;
                   ;DRAW TABLE AND NET
                   ;
0A62- A9 A0        DRAW  LDA #$A0
0A64- A0 27              LDY #$27
0A66- 99 28 80     L1    STA SCRL1,Y
0A69- 99 C0 83           STA W2,Y
0A6C- 88                 DEY
0A6D- 10 F7              BPL L1
0A6F- A9 50              LDA #$50
0A71- 85 B3              STA *TEMP1
0A73- A9 80              LDA #$80
0A75- 85 B4              STA *TEMP2
0A77- A0 27              LDY #$27
0A79- A2 00              LDX #$00
0A7B- A9 E7        L3    LDA #$E7
0A7D- 81 B3              STA (TEMP1,X)
0A7F- A9 E5              LDA #$E5
0A81- 91 B3              STA (TEMP1),Y
0A83- 20 9E 0A           JSR ADLN
0A86- 90 F3              BCC L3
0A88- A9 50              LDA #$50
0A8A- 85 B3             STA *TEMP1
0A8C- A9 80              LDA #$80
0A8E- 85 B4              STA *TEMP2
0A90- A0 13              LDY #$13
0A92- A9 67        L5    LDA #$67
0A94- 91 B3              STA (TEMP1),Y
0A96- A9 50              LDA #$50
0A98- 20 A0 0A           JSR J4
0A9B- 90 F5              BCC L5
0A9D- 60                 RTS
0A9E- A9 28        ADLN  LDA #$28
0AA0- 18           J4    CLC
0AA1- 65 B3              ADC *TEMP1
0AA3- 85 B3              STA *TEMP1
0AA5- 90 02              BCC L6
0AA7- E6 B4              INC *TEMP2
0AA9- A5 B4        L6    LDA *TEMP2
0AAB- C9 83              CMP #$83
0AAD- D0 04              BNE L7
0AAF- A5 B3              LDA *TEMP1
0AB1- C9 C0              CMP #$C0
0AB3- 60           L7    RTS
                   ;
                   ;SWAP INTERRUPT POINTER
                   ;
0AB4- 78           SWAP  SEI
0AB5- A0 01              LDY #$01
0AB7- B1 01        L8    LDA (Z3),Y
0AB9- BE CB 09           LDX TBL0,Y
0ABC- 99 CB 09           STA TBL0,Y
0ABF- 8A                 TXA
0AC0- 91 01              STA (Z3),Y
0AC2- 88                 DEY
0AC3- 10 F2              BPL L8
0AC5- 58                 CLI
```

```
0AC6- 60                    RTS
                    ;
                    ;INIT. ZERO PAGE
                    ;
0AC7- A2 0F         INIT    LDX ##$0F
0AC9- BD CD 09      L9      LDA W4,X
0ACC- 95 B1                 STA *BALPOS,X
0ACE- CA                    DEX
0ACF- 10 F8                 BPL L9
0AD1- 60                    RTS
                    ;
                    ;MOVE THE BALL
                    ;
0AD2- C6 BC         MVEBAL  DEC *DLYB         ;DELAY
0AD4- D0 37                 BNE L10
0AD6- AD D8 09              LDA W5
0AD9- 85 BC                 STA *DLYB
0ADB- 18            L15     CLC
0ADC- A2 00                 LDX ##$00
0ADE- A5 B5                 LDA *INCR         ;GET THE INCREMENT
0AE0- 10 01                 BPL L11           ;THE HIGH BYTE MUST =
0AE2- CA                    DEX               ; $FF OR $00
0AE3- 65 B1         L11     ADC *BALPOS       ;ADD IT
0AE5- 85 B3                 STA *TEMP1
0AE7- 8A                    TXA
0AE8- 65 B2                 ADC *BALPOS+1
0AEA- 85 B4                 STA *TEMP2
0AEC- A0 00                 LDY ##$00
0AEE- B1 B3                 LDA (TEMP1),Y     ;GET NEW POSITION
0AF0- A2 05         J37     LDX ##$05
0AF2- DD AB 09      L13     CMP TBL1,X        ;IS THERE A
0AF5- F0 17                 BEQ RFLCT         ; REFLECTION?
0AF7- CA                    DEX
0AF8- 10 F8                 BPL L1
0AFA- A6 BB                 LDX *PRVCHR       ;RESTORE AREA
0AFC- 85 BB                 STA *PRVCHR       ; AD SAVE NEW CHR.
0AFE- 8A                    TXA

0AFF- 91 B1                 STA (BALPOS),Y
0B01- A9 51                 LDA ##$51         ;DRAW BALL
0B03- 91 B3                 STA (TEMP1),Y
0B05- A6 B3                 LDX *TEMP1        ;SAVE POINTER
0B07- A5 B4                 LDA *TEMP2
0B09- 86 B1                 STX *BALPOS
0B0B- 85 B2                 STA *BALPOS+1
0B0D- 60            L10     RTS
0B0E- BD B3 09      RFLCT   LDA TBL2,X        ;REFLECT
0B11- F0 13                 BEQ L14           ;IS IT OUTSIDE?
0B13- 45 B5                 EOR *INCR
0B15- 85 B5                 STA *INCR
0B17- BD C3 09              LDA SNDS,X        ;SET SOUND
0B1A- 8D 48 E8              STA W9
0B1D- AD DD 09              LDA DURTN         ;AND START IT
0B20- 8D E2 09              STA CNTSND
0B23- 38                    SEC
0B24- B0 B5                 BCS L15           ;FORCED
0B26- 8D DF 09      L14     STA ENDFLG        ;STOP GAME
0B29- 60                    RTS
                    ;
                    ;SCAN THE KEYBOARD
                    ;
0B2A- A0 09         SCAN    LDY ##$09
0B2C- A2 00         L19     LDX ##$00
0B2E- 20 49 0B              JSR GETKEY        ;UP / LEFT
0B31- D0 01                 BNE L17           ;0 = NOT PRESSED
0B33- CA                    DEX               ;FF = PRESSED
0B34- 88            L17     DEY
0B35- 20 49 0B              JSR GETKEY        ;DOWN / RIGHT
0B38- D0 01                 BNE L18
0B3A- E8                    INX               ;0 ALSO IF BOTH PRE-
0B3B- 98            L18     TYA
0B3C- 48                    PHA               ;SAVE Y
0B3D- 4A                    LSR A
0B3E- A8                    TAY               ;DIVIDE BY TWO
0B3F- 8A                    TXA
0B40- 99 BB 09              STA RSLTS,Y       ;STORE IT
0B43- 68                    PLA
0B44- A8                    TAY
0B45- 88                    DEY
0B46- 10 E4                 BPL L19           ;CONTINUE
0B48- 60                    RTS
0B49- B9 8B 09      GETKEY  LDA TABL3,Y
0B4C- 8D 10 E8              STA W15           ;SELECT ROW AT PA
0B4F- B9 9B 09              LDA TABL4,Y       ;GET MASK
0B52- 2D 12 E8              AND W17           ;SENSE COLUMN AT PB
0B55- 60            L20     RTS
                    ;
```

(Continued on next page)

```
                          ;MOVE RIGHT PADDLE
                          ;
0B56- CE E1 09            RLEFT   DEC CNTRL        ;DELAY
0B59- D0 FA                       BNE L20
0B5B- AD D9 09                    LDA DLYR
0B5E- 8D E1 09                    STA CNTRL
0B61- A2 00                       LDX #$00
0B63- AD BB 09                    LDA RSLTS        ;KEY UP/DOWN PRESSED?
0B66- F0 26                       BEQ L21
0B68- 10 01                       BPL L22          ;ADD 0028 OR FFD8
0B6A- CA                          DEX
0B6B- 29 F0               L22     AND #$F0
0B6D- 49 28                       EOR #$28
0B6F- 18                          CLC
0B70- 65 B8                       ADC *RLPOS
0B72- 85 B3                       STA *TEMP1
0B74- 8A                          TXA
0B75- 65 B9                       ADC *RLPOS+1
0B77- 85 B4                       STA *TEMP2
0B79- C9 80                       CMP #$80         ;CHECK POSITION
0B7B- D0 06                       BNE L23
0B7D- A5 B3                      LDA *TEMP1
0B7F- C9 51                       CMP #$51
0B81- 90 0B                       BCC L21
0B83- 20 EF 0B            L23     JSR ADWDTH       ;ADD *WIDTH
0B86- C0 83                       CPY #$83         ;CHECK BOUNDARY
0B88- D0 0C                       BNE L25
0B8A- C9 C3                       CMP #$C3
0B8C- 90 08                       BCC L25
0B8E- A5 B8               L21     LDA *RLPOS       ;DON'T MOVE
0B90- A4 B9                       LDY *RLPOS+1
0B92- 85 B3                       STA *TEMP1
0B94- 84 B4                       STY *TEMP2
0B96- A5 C0               L25     LDA *RLYS        ;GET HOR. OFFSET
0B98- A8                          TAY
0B99- 18                          CLC
0B9A- 6D BC 09                    ADC W20          ;ADD LEFT/RIGHT SHIFT
0B9D- C9 11                       CMP #$11         ;CHECK BOUNDARY
0B9F- 90 01                       BCC L26
0BA1- 98                          TYA              ;DON'T MOVE
0BA2- 48                  L26     PHA
0BA3- A6 BE                       LDX *WIDTH       ;RESTORE AREA
0BA5- BD EA 09            L28     LDA W21,X
0BA8- 91 B8                       STA (RLPOS),Y
0BAA- 20 E2 0B                    JSR INDEX1
0BAD- D0 F6                       BNE L28
0BAF- 68                          PLA
0BB0- A8                          TAY
0BB1- 85 C0                       STA *RLYS
0BB3- 20 D9 0B                    JSR SVPTRL
0BB6- A6 BE                       LDX *WIDTH       ;SAVE THE AREA
0BB8- B1 B8               L31     LDA (RLPOS),Y
0BBA- C9 51                       CMP #$51         ;CAUGHT THE BALL
0BBC- 20 FF 0B                    JSR PATCH
0BBF- 9D EA 09                    STA W21,X        ;STORE IT
0BC2- 20 E2 0B                    JSR INDEX1
0BC5- D0 F1                       BNE L31
0BC7- 20 D9 0B                    JSR SVPTRL
0BCA- A6 BE                       LDX *WIDTH       ;NOW DRAW THE PADDLE
0BCC- A9 61               L32     LDA #$61
0BCE- 91 B8                       STA (RLPOS),Y
0BD0- 20 E2 0B                    JSR INDEX1
0BD3- D0 F7                       BNE L32
0BD5- 20 D9 0B                    JSR SVPTRL
0BD8- 60                          RTS              ;DONE
                          ;
0BD9- A5 B3               SVPTRL  LDA *TEMP1
0BDB- 85 B8                       STA *RLPOS
0BDD- A5 B4                       LDA *TEMP2
0BDF- 85 B9                       STA *RLPOS+1
0BE1- 60                          RTS
0BE2- 18                  INDEX1  CLC
0BE3- A5 B8                       LDA *RLPOS
0BE5- 69 28                       ADC #$28
0BE7- 85 B8                       STA *RLPOS
0BE9- 90 02                       BCC L33
0BEB- E6 B9                       INC *RLPOS+1
0BED- CA                  L33     DEX
0BEE- 60                          RTS
0BEF- A4 B4               ADWDTH  LDY *TEMP2
0BF1- A6 BE                       LDX *WIDTH

0BF3- A5 B3                       LDA *TEMP1
0BF5- 18                  L35     CLC
0BF6- 69 28                      ADC #$28
0BF8- 90 01                       BCC L34
0BFA- C8                          INY
0BFB- CA                  L34     DEX
```

```
0BFC- D0 F7              BNE L35
0BFE- 60                 RTS
                   ;
                   ;HANDLES REFLECTION IF BALL IS
                   ;OVERTAKEN BY MOVING PADDLE
                   ;
0BFF- D0 23        PATCH  BNE L36
0C01- A5 BB              LDA *PRVCHR
0C03- 48                 PHA
0C04- 8A                 TXA            ;SAVE EVERYTHING ON STACK
0C05- 48                 PHA
0C06- 98                 TYA
0C07- 48                 PHA
0C08- A5 B3              LDA *TEMP1
0C0A- 48                 PHA
0C0B- A5 B4              LDA *TEMP2
0C0D- 48                 PHA
0C0E- A9 61              LDA #$61
0C10- A0 00              LDY #$00
0C12- 20 F0 0A           JSR J37        ;AND MOVE BALL
0C15- 68                 PLA
0C16- 85 B4              STA *TEMP2     ;RESTORE NOW
0C18- 68                 PLA
0C19- 85 B3              STA *TEMP1
0C1B- A9 01              LDA #$01       ;ACCELERATE BALL
0C1D- 85 BC              STA *DLYB
0C1F- 68                 PLA
0C20- A8                 TAY
0C21- 68                 PLA
0C22- AA                 TAX
0C23- 68                 PLA
0C24- 60           L36    RTS
                   ;
                   ;MOVE THE RIGHT PADDLE
                   ;
0C25- CE E0 09     RRIGHT DEC CNTRR     ;DELAY
0C28- D0 FA              BNE L36
0C2A- AD D9 09           LDA DLYR
0C2D- 8D E0 09           STA CNTRR
0C30- A2 00              LDX #$00
0C32- AD BD 09           LDA W23        ;UP/DOWN PRESSED?
0C35- F0 26              BEQ L38
0C37- 10 01              BPL L39        ;ADD 00F8 OR FFD8 RESP
0C39- CA                 DEX
0C3A- 29 F0        L39    AND #$F0
0C3C- 49 28              EOR #$28
0C3E- 18                 CLC
0C3F-65 B6              ADC *RRPOS
0C41- 85 B3              STA *TEMP1
0C43- 8A                 TXA
0C44- 65 B7              ADC *RRPOS+1
0C46- 85 B4              STA *TEMP2
0C48- C9 80              CMP #$80       ;CHECK BOUNDARY
0C4A- D0 06              BNE L40
0C4C- A5 B3              LDA *TEMP1
0C4E- C9 66              CMP #$66
0C50- 90 0B              BCC L38
0C52- 20 EF 0B     L40    JSR ADWDTH    ;ADD WIDTH
0C55- C0 83             CPY #$83       ;AND TEST BOTTOM
0C57- D0 0C              BNE L41
0C59- C9 D7              CMP #$D7
0C5B- 90 08              BCC L41        ;IT'S OK
0C5D- A5 B6        L38    LDA *RRPOS    ;DON'T MOVE
0C5F- A4 B7              LDY *RRPOS+1
0C61- 85 B3              STA *TEMP1
0C63- 84 B4              STY *TEMP2
0C65- A5 BF        L41    LDA *RRYS     ;ADD HOR. OFFSET
0C67- A8                 TAY
0C68- 18                 CLC
0C69- 6D BE 09           ADC W24        ;HORIZONTAL SHIFT
0C6C- C9 11              CMP #$11       ;IN LIMITS?
0C6E- 90 01              BCC L42
0C70- 98                 TYA            ;NO, DON'T MOVE
0C71- 48           L42    PHA           ; SAVE FOR A MOMENT
0C72- A6 BE              LDX *WIDTH
0C74- BD FA 09     L44    LDA W25,X     ;RESTORE UNDERNEATH
0C77- 91 B6              STA (RRPOS),Y
0C79- 20 B0 0C           JSR INDEX2
0C7C- D0 F6              BNE L44
0C7E- 68                 PLA            ;SAVE NEW HOR. SHIFT
0C7F- A8                 TAY
0C80- 85 BF              ST *RRYS
0C82- 20 A7 0           JSR SVPTRR     ;SAVE POINTERS
0C85- A6 BE              LDX *WIDTH
0C87- B1 B6        L46    LDA (RRPOS),Y ;SAVE UNDERNEATH
0C89- C9 51              CMP #$51       ;DID WE HIT BALL?
0C8B- 20 FF 0B           JSR PATCH
```

(Continued on next page)

Listing 1 *(Continued)*

```
0C8E- 9D FA 09          STA W25,X        ;SAVE IT
0C91- 20 B0 0C          JSR INDEX2
0C94- D0 F1             BNE L46
0C96- 20 A7 0C          JSR SVPTRR
0C99- A6 BE             LDX *WIDTH       ;DRAW PADDLE
0C9B- A9 E1    L47      LDA #$E1
0C9D- 91 B6             STA (RRPOS),Y
0C9F- 20 B0 0C          JSR INDEX2
0CA2- D0 F7             BNE L47
0CA4- 20 A7 0C          JSR SVPTRR
0CA7- A5 B3    SVPTRR   LDA *TEMP1
0CA9- 85 B6             STA *RRPOS
0CAB- A5 B4             LDA *TEMP2
0CAD- 85 B7             STA *RRPOS+1
0CAF- 60                RTS
0CB0- 18       INDEX2   CLC
0CB1- A5 B6             LDA *RRPOS
0CB3- 69 28             ADC #$28
0CB5- 85 B6             STA *RRPOS
0CB7- 90 02             BCC L48
0CB9- E6 B7             INC *RRPOS+1
0CBB- CA       L48      DEX
0CBC- 60                RTS
0CBD- A9 10    INITSD   LDA #$10         ;INIT SOUND
0CBF- 8D 4B E8          SA W26
0CC2- A9 0F             LDA #$0F
0CC4- 8D 4A E8          STA W27
0CC7- A9 00             LDA #$00
0CC9- 8D 48 E8          STA W9
0CCC- 60                RTS
               ;
               ;SERVICE
               ;
0CCD- AC E3 09 SERVE    LDY SDEFLG
0CD0- F0 60             BEQ L49          ;SERVICE?
0CD2- CC BF 09          CPY W29          ;KEY PRESSED?
0CD5- D0 5B             BNE L49
0CD7- A5 BE             LDA *WIDTH       ;YES
0CD9- 4A                LSR A            ;CALCULATE MID OF PADDLE
0CDA- AA                TAX              ;SAVE IT
0CDB- 98                TYA              ;WHICH SIDE?
0CDC- 10 09             BPL L50
0CDE- A5 B7             LDA *RRPOS+1     ;RIGHT
0CE0- 85 B2             STA *BALPOS+1
0CE2- A5 B6             LDA *RRPOS
0CE4- 18                CLC
0CE5- 90 06             BCC L51          ;FORCED
0CE7- A5 B9    L50      LDA *RLPOS+1
0CE9- 85 B2             STA *BALPOS+1
0CEB- A5 B8             LDA *RLPOS
0CED- E0 00    L51      CPX #$00         ;WIDTH WAS 1?
0CEF- F0 0A             BEQ L52
0CF1- 18       L54      CLC              ;POSITIONING OF BALL
0CF2- 69 28             ADC #$28
0CF4- 90 02             BCC L3
0CF6- E6 B2             INC *BALPOS+1
0CF8- CA       L53      DEX
0CF9- D0 F6             BNE L54
0CFB- 85 B1    L52      STA *BALPOS
0CFD- 98                TYA              ;WHICH SIDE?
0CFE- 10 01             BPL L55
0D00- CA                DEX              ;AT RIGHT
0D01- 18       L55      CLC
0D02- 65 B1             ADC *BALPOS      ;ADD OR SUBTRACT 1
0D04- 85 B1             STA *BALPOS
0D06- 8A                TXA
0D07- 65 B2             ADC *BALPOS+1
0D09- 85 B2             STA *BALPOS+1
0D0B- A6 C0             LDX *RLYS
0D0D- 98                TYA              ;NOW ADD HORIZ. SHIFT
0D0E- 10 02             BPL L56
0D10- A6 BF             LDX *RRYS
0D12- 8A       L56      TXA
0D13- 18                CLC
0D14- 65 B1             ADC *BALPOS
0D16- 85 B1             STA *BALPOS
0D18- 90 02             BCC L57
0D1A- E6 B2             INC *BALPOS+1
0D1C- A0 00    L57      LDY #$00         ;SAVE THINGS UNDER
0D1E- B1 B1             LDA (BALPOS),Y   ; THE BALL
0D20- 85 BB             STA *PRVCHR
0D22- A9 51             LDA #$51
0D24- 91 B1             STA (BALPOS),Y
```

```
0D26- 8C E3 09           STY SDEFLG   ;SERVICE IS DONE
0D29- A9 20              LDA #$20     ;CLEAR MESSAGE
0D2B- AA                 TAX
0D2C- 9D 18 81   L58     STA SCR3,X
0D2F- CA                 DEX
0D30- D0 FA              BNE L58
0D32- 60         L49     RTS
                 ;
                 ;MAIN PROGRAM
                 ;
0D33- 20 10 0A   MAIN    JSR DISPLY   ;GRAPHICS
0D36- 20 62 0A           JSR DRAW
0D39- 20 C7 0A           JSR INIT     ;INITIALIZATION
0D3C- 20 BD 0C           JSR INITSD
0D3F- A9 FF              LDA #$FF
0D41- 8D DF 09           STA ENDFLG
0D44- 20 B4 0A           JSR SWAP     ;INTERRUPT POINTER
0D47- 58                 CLI
0D48- A9 6C              LDA #$6C     ;SET AN
0D4A- 85 00              STA *ZERO    INDIRECT JUMP
0D4C- A9 20              LDA #$20
0D4E- AA                 TAX
0D4F- 9D EB 09   L64     STA SAVIT,X  ;CLEARING
0D52- CA                 DEX
0D53- 10 FA              BPL L64
0D55- AD DF 09   L65     LDA ENDFLG   ;CLOSED LOOP
0D58- D0 FB              BNE L65      ; UNTIL GAME
0D5A- 60                 RTS          ; IS FINISED
                 ;
                 ;INTERRUPT LOOP
                 ;
0D5B- AD DF 09   ITRP    LDA ENDFLG   ;GAME FINISHED?
0D5E- F0 38              BEQ L66
0D60- 20 2A 0B           JSR SCAN     ;SCAN KEYBOARD
0D63- AD E3 09           LDA SDEFLG   SERVIC?
0D66- F0 08              BEQ L68
0D68- A9 00              LDA #$00     ;YES,
                                       RESET PADDLES
0D6A- 8D BC 09           STA W20
0D6D- 8D BE 09           STA W24
0D70- 20 56 0B   L68     JSR RLEFT    ;MOVE PADDLES
0D73- AD DF 09           LDA ENDFLG
0D76- F0 20              BEQ L66
0D78- 20 25 0C           JSR RRIGHT
0D7B- AD DF 09           LDA ENDFLG
0D7E- F0 18              BEQ L66
0D80- 20 CD 0C           JSR SERVE    ;SERVICE
0D83- AD E3 09           LDA SDEFLG   ;PLAY?
0D86- D0 03              BNE L72
0D88- 20 D2 0A           JSR MVEBAL
0D8B- CE E2 09   L72     DEC CNTSND   ;TIMING OF SOUND
0D8E- D0 05              BNE L74
0D90- A9 00              LDA #$00
0D92- 8D 48 E8           STA W9
0D95- 6C CB 09   L74     JMP (TBL0)   ;FINISH INTERRUPT
0D98- A9 5D      L66     LDA #$5D
0D9A- 8D 48 E8           STA W9
0D9D- 20 B4 0A           JSR SWAP     ;RESTORE INTERRUPT
0DA0- 4C 00 00          JMP $0000    ;COMPLETE IT
```

Listing 2 B

```
.:  0900 4C 33 0D 55 43 43 43 43
.:  0908 43 43 43 43 43 43 43 43
.:  0910 43 43 43 43 43 43 43 49
.:  0918 20 20 20 20 20 20 20 20
.:  0920 20 20 20 20 20 20 20. 20
.:  0928 20 20 20 42 20 10 20 89
.:  0930 20 0E 20 87 20 2D 20 90
.:  0938 20 0F 20 8E 20 07 20 5D
.:  0940 20 20 20 20 20 20 20 20
.:  0948 20 20 20 20 20 20 20 20
.:  0950 20 20 20 4A 40 40 40 40
.:  0958 40 40 40 40 40 40 40 40
.:  0960 40 40 40 40 40 40 40 4B
.:  0968 00 11 15 01 0C 09 14 19
.:  0970 20 20 13 0F 06 14 17 01
.:  0978 12 05 00 02 19 20 17 05
.:  0980 12 0E 05 12 20 0B 0F 0C
.:  0988 02 05 00 08 00 04 04 09
.:  0990 01 05 05 05 04 AA AA AA
.:  0998 AA AA AA 01 01 02 01 80
.:  09A0 80 80 40 01 80 AA AA AA
.:  09A8 AA AA AA A0 A0 61 E1 E5
.:  09B0 E7 00 00 F0 F0 0E 0E 00
.:  09B8 00 00 00 00 00 00 00 00
.:  09C0 AA AA AA 40 60 80 40 AA
.:  09C8 AA AA AA 5B 0D 00 81 00
.:  09D0 81 D7 56 81 41 81 20 20
.:  09D8 04 04 02 10 00 03 FF 00
.:  09E0 02 02 E2 00 00 AA AA AA
```

MICRO™

MICRO
PET Vet

By Loren Wright

Since this issue is centered around games, I probably shouldn't let it pass without saying something about the PET and games. The PET lends itself nicely to games. The combination of a wide variety of graphic characters easily available from the keyboard and programmable cursor controls makes it easy for someone with little experience to do some sophisticated game programming. Animation is a simple matter, whereas on the Apple it takes more skill to manipulate the high-resolution graphics.

Commercial game program availability started out pretty big, then diminished, and only recently has begun to pick up again. Many software houses have withdrawn from the Commodore market in favor of the bigger and more lucrative TRS-80, Apple, and now Atari markets. The PET's sudden ROM switches, lack of a color display, and difficulties in program protection have all contributed to the dearth of game programs. Nevertheless, there is still a fair amount available, and now that Commodore seems to be showing some consistency in its approach to the market, that amount should increase.

CURSOR (Box 550, Goleta, CA 93116 — $18/year), a quarterly cassette magazine for the PET, has been around since 1978 and has established a reputation for technical excellence. Games and novelty programs have always been a significant part of CURSOR's offering.

In spite of MICRO's previous "hands-off" policy toward games, a few — notably *Life* — have appeared. The most recent version of *Life* for the PET by Werner Kolbe (MICRO 19:45 and *Best of MICRO III*, p. 249) presented a technique to use the PET's screen as a movable window into a much larger playing area. Other articles, such as John Girard's "Horizontal Screen Scrolling" (MICRO 37:81) and Peter Coyle's "PET Interface to Bit Pad" (38:83), present techniques which have obvious applications to games.

In my August "PET Vet" column I reviewed "VIGIL," a game-oriented language from Abacus Software. For VIC users, there was an excellent article by David Malmberg on light pens in last month's issue. My "Substitute Characters" article (also in October) presents some food for thought for both PET and VIC game programmers. As you can see, MICRO is a good source of ideas, information, and techniques for game programmers.

Kolbe's "Ping-Pong" has been reassembled and thoroughly tested by the MICRO staff. (The champion is Associate Editor Mary Ann Curtis!) It will run on 3.0 and 4.0 PETs as is, but needs a few (conceptually) simple changes for 1.0. I hope "Ping-Pong" will serve not only as an entertaining game, but also as an example for your own high-speed graphic programs. Some of the other games in this issue should be modifiable for the PET.

Captain Kirk and the PET

Commodore has announced what it calls its "biggest ad campaign ever," featuring actor William Shatner, who is "known throughout the world — and beyond — as Captain James Kirk, commander of the *Starship Enterprise* on *Star Trek*." The campaign will promote the entire line of Commodore computer products, from the VIC to the new "SuperPET." What difference does this make to those of us who already own PETs? Well, the more people who own PETs, the bigger the market gets, and the more support the products get — not only from Commodore, but also from independent companies.

MICRO™

Microbes and Updates

Larry P. Gonzalez caught an error in his article "Disassembling to Memory with AIM 65," which appeared in MICRO (39:25):

The listing of a sample run of the program was typeset with several errors. Here is the corrected version.

Sample Program Run

```
<*>=0E00
<G>/
TO=0C00
EDITOR END=0D00
FROM=EA46
/10
 EA46 48 PHA
 EA47 4A LSR .A
 EA48 4A LSR .A
 EA49 4A LSR .A
 EA4A 4A LSR .A
 EA4B 20 JSR EA51
 EA4E 68 PLA
 EA4F 29 AND #0F
 EA51 18 CLC
 EA52 69 ADC #30
MORE?Y/04
 EA54 C9 CMP #3A
 EA56 90 BCC EA5A
 EA58 69 ADC #06
 EA5A 4C JMP E9BC
MORE?N
<T>
*=$EA46
=<L>
/
OUT=P
*=$EA46
PHA
LSR A
LSR A
LSR A
LSR A
JSR $EA51
PLA
AND #$0F
CLC
ADC #$30
CMP #$3A
BCC $EA5A
ADC #$06
JMP $E9BC
 END
```

Please note this correction:

The Commodore PET User Group Newsletter as listed in the Resource Update (37:104) should be:

$15/6 issues
Commodore Interface
681 Moore Road
King of Prussia, PA 19406

M.J. Keryan of Tallmadge, Ohio, offered these corrections to his article "An Inexpensive Printer for your Computer," (MICRO 39:61), listing 1:

The listing on page 61 will not work properly; the two lines of code should read as below:

```
LOCATION
804C  BD F5 80   LDA ROMTAB-1,X
804F  95 E5      STA TABLEA-1,X
```

One of our readers called in with these corrections to Mark Bernstein's article "Jumps and the 6502," (40:08):

On page 8, the last lines in the second column should read: *"next instruction following the JSR command − 1."*

On page 11, on the bottom of the first column, the lines beginning with LDA should read:

```
LDA #L,MONITOR -1
LDA #H,MONITOR -1
```

Fred Boness of Kenosha, Wisconsin sent us this update:

Two of the letters I have received about my article on memory expansion for the Superboard (37:79) have shown me that it was hopelessly out of date by the time it was published. Earl Morris has told me that OSI stopped selling bare boards more than a year ago. I bought mine from an OSI distributor early in the summer of 1980. I probably have one of the last boards available.

The second letter is from Mr. William H. Conrad who states that OSI does not sell bare printed circuit boards. I have to believe him; Mr. Conrad is the field support manager for OSI.

Both men have mentioned that other companies are making products for OSI machines. Because of that, the expansion possibilities are much better now than when I started to modify my Superboard.

Ian Pawson in Leicester, England, sent these revisions:

The following modifications to David L. Rosenberg's excellent double barrelled disassembler (MICRO 38:33) will enable it to give the correct output with the Apple High Speed Serial card.

Alter line 22 to VID = $7F9
Alter line 94 to LDA #$01
Alter line 95 to STA VID
Move the label from line 98 to 99
Delete lines 98 and 101

These mods enable the screen display for correct tab positioning. It assumes that the card is in Slot 1.

Alex Bamp of Carmel, Indiana, revised an Apple program — now it runs on his OSI C1P.

I noticed the article in your September 1981 issue of MICRO, "Dollars and Sense Revisited" (40:66). I attempted to use the Apple program on my OSI C1P and it failed. This was caused by the OSI form of handling numbers with a space before and after the number, mixing up the MID$ statement at the end of line 120 in listing 1, or line 20 in the text. For OSI machines, the program works perfectly if you type in the following substituting line:

```
20 N$ = STR$(SGN(N)          *
   INT(ABS(N)))              +
   MID$(N$,3,3)
```

The only change is in the last statement, MID$(N$,3,3), instead of MID$(N$,2,3) for the Apple. This is a *great* algorithm, and I compliment David Delli Quadri for a job well done.

Robert N. Bolster of Alexandria, Virginia sent this update:

Here is an addition to complete Scott Schram's useful Applesoft

Variable Dump program in MICRO (36:23). Between lines starting at 40B3 and 40B5, insert

```
      AND  #$E0   DETECT CONTROL
                  CHRS.
      BNE CONT4   NOT CONTROL
      LDA #$7E    YES, MARK WITH
                  SYMBOL $FE
      JSR OUTDO
      LDA (SPL),Y
      CLC
      ADC #$40    CONTROL TO
                  NORMAL
      JMP CONT5
CONT4 LDA (SPL),Y
```

and label the next line 'CONT5':

```
CONT5 JSR OUTDO
```

Strings which are (or contain) control characters will now be fully displayed, with a symbol (>on the screen, ~ on a printer) before each control character, i.e. " D $ >D" for control-D.

Earl Morris sent this update:

Since I claim to be lazier than Les Cain (MICRO 37:33), I have converted his program to create the "READ" and "POKE" statements as well as the DATA.

Morris Listing

```
10 PRINT"DATA TAPE MAKER"
20 A$="DATA"
30 LN=5000:IX=2
40 PRINT:INPUT"BEGINING HEX ";N$:GOSUB1000:ST=D
50 PRINT:INPUT"ENDING HEX ";N$:GOSUB1000:FI=D
60 PRINT:INPUT"TURN ON RECORDER";N$:SAVE:PRINT
70 PRINTLN;"FOR X="ST"TO"FI":READJ:POKEX,J:NEXT"
80 LN=LN+IX
90 FORI=ST TO FI STEP10
100 PRINTLN;A$;:LN=LN+IX
120 FORJ=I TO I+8
125 IFJ=FITHEN170
130 X$=STR$(PEEK(J))
140 T=LEN(X$)
150 PRINTRIGHT$(X$,T-1)",";
155 NEXT
160 PRINTPEEK(J):NEXT:GOTO999
170 PRINTPEEK(J)
999 LOAD:END
1000 FORI=1TO4
1010 D(I)=ASC(MID$(N$,I))-48
1020 IFD(I))9 THEND(I)=D(I)-7
1030 NEXT
1040 D=4096*D(1)+256*D(2)+16*D(3)+D(4)
1050 RETURN
```

Sample Run

```
5000 FOR X= 28672 TO 28693 :READJ:POKEX,J:NEXT
5002 DATA0,1,2,3,4,5,6,7,8, 9
5004 DATA10,11,12,13,14,15,16,17,18, 19
5006 DATA20,21,22,23,24,25, 26
```

MICRO

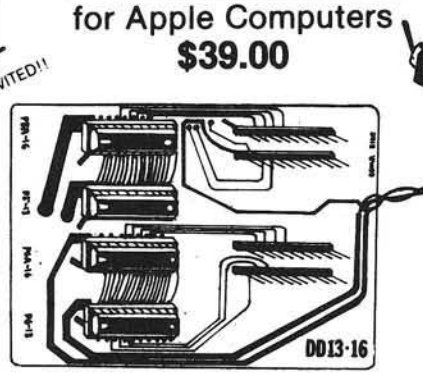
NEW

DEALER INQUIRIES INVITED!!

DOUBLE DOS PLUS

for Apple Computers

$39.00



201-839-3478

MasterCard

VISA

DOUBLE DOS Plus—a piggyback board that plugs into the disk-controller card so that you can switch select between DOS 3.2 and DOS 3.3. Works with the language system eliminating the need in many cases to boot the BASICs disk. Also eliminates the chore of converting all of your 3.2 disks to 3.3

NOTE: APPLE is a registered trademark of APPLE Computer, Inc., Cupertino, CA.

WHY IS DOUBLE DOS PLus better?

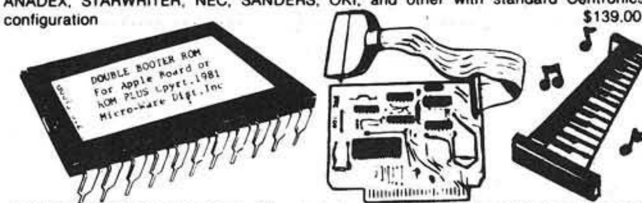
- Nothing needs to be soldered, just plug in and go.
- Since all four ROMS are used, all software will work, even early 3.1 DOS.
- Because the ROMS fit on the back of the board, it has the thinnest configuration allowing full use of slot #7
- One set of ROMS is powered up at a time, thus saving power. DOUBLE DOS Plus requires APPLE DOS ROMS
- Full 90-day warranty from TYMAC.

MICRO-WARE DISTRIBUTING INC.
P.O. BOX 113
POMPTON PLAINS, N.J. 07444

OTHER UNIQUE PRODUCTS FROM MICRO-WARE DISTRIBUTING INC.

THE APPLE CARD—Two sided 100% plastic reference card for the Apple computer. Loaded with information of interest to all Apple owners $3.98

PARALLEL PRINTER CARD—PPC-100—A Universal Centronics type parallel printer board complete with cable and connector. This unique board allows you to turn on and off the high bit so that you can access additional features in many printers. Use with EPSON, ANADEX, STARWRITER, NEC, SANDERS, OKI, and other with standard Centronics configuration $139.00.



THE DOUBLE BOOTER ROM—Plugs into the empty D8 Socket on the Apple motherboard or the Integer ROM Card to provide a 13 sector boot without using the BASICs Disk. DoubleBooter may also be used in the MOUNTAIN HARDWARE ROM PLUS board. This chip will not work in a plus machine unless it contains an Integer board or a ROM Plus board $29.00

DISK STIX—Contains 10 dozen diskette labels with either 3.3 or 3.2 designation. Room for program names and type also $3.98

★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ **SOFTWARE** ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★

SUPER SEA WAR—Hires battleship type simulation $13.95

ULTIMATE XFER—A telephone software transfer program, uses DC Hayes Assoc. micromodem $25.00

ROAD RALLYE—Hires driving game with 5 different full screen tracks $15.00

MISSILE CHALLENGER—Hires arcade type game where you defend your cities from falling missiles. 8 levels & writes name & high score to disk $19.95

SUPER PIX—Hires screen dump for the EPSON MX-80, inverse or normal, larger than full page graphics in 2 orientations. Needs Tymac PPC-100 Printer board or we will upgrade your EPSON board for $25 $39.95

GRAPH-FIT—A hires graphing program that produces bar charts, pie charts and line graphs. Has auto scaling feature too $25.00

STILL MORE APPLE GOODIES

APPLE KEYBOARD SYNTHESIZER—49 note (C to C) AGO Keyboard with 3 sawtooth sq wave shapers, 3 audio oscilators, 3 low pass filters, 4-64 point shape controllers, 2 envelope generators. Complete system $995

KEYBOARD ONLY with Apple Interface $649

GRAPHIC NOTEWRITER—Hires note write for synthesizer system $99

SUPER PIX OKI—Hires screen dump for OKI Microline 80, 82, 83 Printers. Same features as super pix. Needs Tymac PPC-100 Board $24.95

NIBBLES AWAY—The best disk back up program to date. Allows you to make backups of most every disk ever produced for the Apple. Over 10 user defined parameters including synchronized & unsynchronized copies as well as automatic half tracking and raw date transfer $59

Call 201-839-3478 for Dealer & Distributor Inquiries.

EVER WONDER HOW YOUR APPLE II WORKS?

QUICKTRACE will show you! And it can show you WHY when it doesn't!

This relocatable program traces and displays the actual machine operations, *while* it is running *and* without interfering with those operations. Look at these *FEATURES:*

Single-Step *mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

Trace *mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

Background *mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

QUICKTRACE *allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

Two optional display formats *can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

QUICKTRACE *is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

QUICKTRACE *is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

QUICKTRACE *is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while* ***QUICKTRACE*** *is alive.*

QUICKTRACE *is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

Price: $50

QUICKTRACE requires 3548 ($E00) bytes (14 pages) of memory and some knowledge of machine language programming. It will run on any Apple II or Apple II Plus computer and can be loaded from disk or tape. It is supplied on disk with DOS 3.3.

QUICKTRACE was written by John Rogers.
QUICKTRACE is a trademark of Aurora Systems, Inc.

FLIPPER

This long overdue device will switch any electrical signals, either from two inputs into a single output, or from a single input into either of two outputs.

The **FLIPPER** is usually used to switch between 40 and 80 column video displays. (Our word processor, The Executive Secretary, supports it automatically.)

The **FLIPPER** can switch your monitor between the Apple display and video tape, disc, or tuner. (Great enhancement for sorting routines!)

The **FLIPPER** mounts on the 'game bus', yet leaves it free.

The **FLIPPER** is available with the Apple shift key modification already in place. It requires *no soldering*, does not void your warranty, and mounts in seconds.

price, without shift key mod:	$50
with shift key mod:	$55

OMNISCAN T.M.

The interface that provides the most revolutionary means of information retrieval since the printing press by combining these important technologies:

1) *the Apple II computer,*
2) *the Pioneer VP-1000 Laser Video Disc,*
3) *and the Color Television.*

The **OMNISCAN** *interface is used to control the Pioneer LaserDisc player in an interactive way, with software running on the Apple II computer. The system can display information with color, motion, and stereo or bilingual sound* **under program control**. *It can teach, review, test, and grade material while allowing for individual learning rates. The branching capability of the computer gives unlimited flexibility in programing a learning sequence.*

Price: $ 250

Also from Aurora...

Versacalc *(a Visicalc enhancement)*

The Executive Secretary
(word processor extraordinaire)

The Rental Manager
(rental property management)

The Performance Manager
(an assessment of your work)

Hebrew II ( ]] עברית)

Educational Programs
(with emphasis on high school)

aurora systems, inc.

2040 East Washington Ave.
Madison, WI 53704
(608) 249-5875

QuickTrace, Omniscan, and The Performance Manager are trademarks of Aurora Systems, Inc. The Rental Manager is a trademark of Money Tree Systems, Inc. The Executive Secretary is a trademark of Personal Business Systems, Inc. Visicalc is a trademark of Personal Software, Inc. Flipper is a trademark of Instrument Interfaces, Inc. Apple II is a trademark of Apple Computer, Inc. VP-1000 LaserDisc is a trademark of Pioneer USA, Inc.

OS-9 and the 6809: Revolutionary Tools

The Motorola MC6809 microprocessor incorporates advanced architectural design features that make it a highly powerful machine. The Microware OS-9 operating system is an advanced software package designed to fully exploit the powerful features of the 6809. This article describes the highlights of OS-9, its concepts, its features, and its supporting software systems.

Brian Capouch
RR #2, Box 525
Wheatfield, Indiana 46392

On the Evolution of Tools

Programmers who exclusively use a high-level language tend not to care about the characteristics of the microprocessor or operating system they use. This promotes situations where superior products suffer from apathy in the marketplace. For this reason I implore student programmers to try to look at their jobs from an automobile mechanic's viewpoint. Even though a computer lacks levers and knobs that invite intuitve understanding, coming to grips with "what makes that thing tick" leads to a deeper understanding of the process of programming, and hence to better programs.

From the earliest days of mankind we have been distinguished as "the animal that uses tools." A history of civilization is a telling of those tools we have developed, and the influence that they have had on the people who use them. Computers are tools that have the potential to drastically change our everyday lives. Plus, computers are changing and growing in time. In the world of 8-bit microprocessors, the current vanguard lies in the Motorola 6809.

A History of 6809 and OS-9

The 6809 is called a third generation microprocessor, and as such is one of those that has been designed after the beginning of the microelectronics revolution. The engineers who designed it began by drawing up a list of functions that they considered to be the next major advances in computer architecture. Then, they proceeded to perform an intensive analysis of code written for their second generation processor, the 6800, to see which instructions and which types of instructions were used most. Then they interviewed a wide cross-section of 6800 programmers to see what features they considered desirable in an advanced microprocessor. Only after this footwork was completed did they actually specify the 6809.

Programming in 1978 was beginning to show signs of radical change. Along with the ascendency of the microprocessor came the realization that programming methods needed to be undertaken scientifically. Pascal became widely known, BNF notation was presented to the programming public, and the unconditional branch fell into generally loud disfavor. All of these happenings implied a movement towards *structure*. And structure implies an opposition to entropy, which any physicist will tell you requires work. Even beginning programmers were exhorted to refrain from dashing off code helter-skelter, and those who did soon realized the price when they later attempted to modify or correct their work. All of this came at an opportune time for the designers of the 6809. They were able to incorporate these trends into the specifications for their new processor, and in fact design a processor around modern techniques.

At the same time that Motorola product engineers were laying out the 6809, they wisely looked ahead into the software arena and contracted, with the Microware Corporation, to develop software for the new processor. Because of this, introduction of software products that utilized the advanced features of the 6809 began concurrently with the introduction of the chip itself.

Now I am fully aware that almost everyone considers the operating system that resides on his own computer to be the cat's meow — the hottest and most efficient system in the world. I call this the "emperor's new clothes" syndrome, and it is a powerful factor resisting change in the micro world. But our little world is growing. Hardware is now capable of performing tasks that would have heavily loaded a minicomputer of fairly recent vintage. Expecting archaic operating systems designed in the hobbyist days to take full advantage of modern hardware is a pipe dream of awesome proportions. I have worked with five operating systems in the past four years, two of them on sixteen-bit hardware. I feel that my software output has been greater in the six months I have used OS-9 than it was in the three and one-half previous years combined. It has a rakish logical simplicity that nearly defies description — it almost always does just what you think it will do, even when trying something for the first time.

What follows here is a brief introduction to the features of Microware's OS-9 operating system, and a listing of high-level languages and processors designed to run on it.

Major Features of OS-9

Motorola bills the 6809 as a "programmer's dream machine," and I bill OS-9 as a "programmer's dream operating system." Imagine for a moment never having to bother with memory mapping, or with two programs that need to run in the same RAM space. Or imagine a 64K machine allowing five users to simultaneously run separate programs, yet also access common subroutines and data files. Or perhaps having a program write data out to a line printer during debugging that will be written into a file in its final version,

Figure 1 : General Memory Module Format

$00 $01	Sync Bytes ($87CD)	
$02 $03	Module Size (bytes)	
$04 $05	Module Name Offset	header parity
$06	Type ! Language	
$07	Attributes ! Revision	
$08	Header Parity Check	
	(Execution Parameters and optional extensions are located from here upward)	

without having to change any code. All of these situations, and literally hundreds more, are possible with this advanced operating system.

OS-9 is based almost exactly on the functional specifications of UNIX, the multiuser operating system developed by the Bell Laboratories. UNIX is expected to take the 16-bit world by storm when machines designed around the newer processors become widely available. A favorite quote of mine is this one from a software engineer quoted in *Electronics* magazine, who says that using UNIX is "like sitting behind the wheel of a well-tuned sports car — when you press the gas, it goes, and when you hit the brakes, it stops. It's the ultimate in responsiveness, and yet all the time you are riding in comfort."

One of the prime features of OS-9 is that all code, be it machine code, operating system parameters, text, etc., must reside in what is called a memory module. Each module is self-relative, or relocatable, which is to say that it can be loaded into any sufficient area of memory that is currently available. Modules are preceded, both in memory and in files, by a module header (see figure 1), which tells the operating system about certain features of that module, such as its name, size, intended use, and so forth.

A particularly powerful feature is the Attributes-Revision byte. The left nibble of this byte controls the types of access that may be made legally to the module. The right nibble is a revision number. Whenever a load is performed from peripheral storage, the operating system first checks to see if that module is already in memory. If it is, revision numbers are then checked. The module with the highest revision is then allowed to stay in memory. This means that code located in ROM can be overlaid by modified code without having to resort to reprogramming the ROM. It is very handy for customizing operating system modules, or for interim fixes for bugs. It should be noted that the module format is the *only* way that memory can be managed by OS-9, and that self-modifying or non-relocatable code is not permitted.

The OS-9 user interacts with the operating system through a program called "Shell." This works exactly like UNIX, with the Shell being a command interpreter that orders up operating system functions as required. It minimizes the knowledge the user must possess of the inner workings of the more complex system capabilities. For instance, any program may be run in one of two basic modes: sequential or concurrent. While executing programs sequentially, the Shell waits for one program to finish before beginning another. When a system command is suffixed with an ampersand ("&") the Shell interprets this as a request to run that program concurrently, or in the background. In this case the Shell returns almost immediately for another command, while the concurrent program runs simultaneously until it finishes.

Shell commands can call machine code modules, high-level language modules, or groups of Shell commands (procedure files). If the module language is not object code, the proper high-level language or processor is automatically loaded to run the module.

Figure 2 : Explanation of Header Values

Module Offset	Description
$00,01	Sync Bytes These unused 6809 opcodes are used by OS-9 to automatically locate modules.
$02,03	Module Size Overall size of module in bytes, including all header and CRC values.
$04,05	Offset to Module Name Address of module name relative to the start (first sync byte) of module. The name may exist anywhere within the module and is made up of a string of ASCII characters.
$06	Module/Language Type Values of Module Type Nibble: $1 - Program Module $2 - Subroutine Module $3 - Multi-module $4 - Data Module $5/$B - User-defined $C - OS9 System Module $D - OS9 File Manager Module $E - OS9 Device Driver Module $F - OS9 Device Descriptor Module Values of Language Nibble: 0 - Data 1 - 6809 Object Code 2 - Basic09 I-code 3 - Pascal P-code 4 - COBOL I-code 5-15 - For future use
$07	Attributes/Revision Byte Value of upper four bits of this byte indicate usage attributes, at this time only bit 7 (m.s.) is defined — when set indicates module is "sharable" (re-entrant code) Lower four bits indicates revision number from 0-15

Interfacing with the World

Input and output operations performed on OS-9 files are simplified by a pair of conventions: all devices look to the operating system exactly like files, and all files look to the operating system like a stream of bytes. This is a good example of logical simplicity in action. Thus a program written to use a disk data file may be debugged by replacing the file name with the name of a line printer, or conversely, output destined for a printer can be spooled to a disk file, which may later be "listed" on the physical printer.

The "stream of bytes" convention means that all structure imposed on data must be done by programs, and thus programs need to be aware of how data is structured. There is no difference to the operating system between random and sequential files. The operating system maintains a moveable pointer to the next byte to be read or written, all other record manipulation is left to programs.

Programs running on OS-9 are assumed to use "standard" data paths. In the default case input is expected to be provided by a terminal keyboard, and output is performed to its CRT display. The Shell allows a set of "pseudo-commands" to redirect these standard paths to any file or device. (Remember that they are the same to the operating system!) This means that programs can be written to use these standard data paths, and the paths can be redirected, *at run time*, to any file or device. Path redirection enables a single program to function both as an interactive processor using terminal input and output, or as a batch processor driven by disk-bound input command files. As an example, to see a listing of the text file containing this article on the CRT screen, I would type:

```
list micro_article
```

If, instead, I wish the listing to appear on the line printer instead of the terminal display, I type:

```
list micro article >/printer
```

where "/printer" is a *pathlist* that describes the printer that I wish to use to the operating system, and ">" tells the operating system to redirect the standard output path for that program. A disk file could also be used as the object of this redirection. The slash character (/) is used to delineate elements of a path so that the operating system can access the desired data.

Figure 3: Example of Redirected Output

Directory of /d1/work 23:38:06

Owner	Last Modified	Attributes	Sector	Byte-count	Name
0	81/07/26 2336	d-ewrewr	18	700	work_bak
0	81/07/25 1626	------wr	38	5758	micro_safe
0	81/07/22 2253	----r-wr	BC	300	copy_all
0	81/07/22 2255	----r-wr	C0	170	thisdir
0	81/07/22 2341	----r-wr	C3	62F	many_copy
0	81/07/22 2352	----r-wr	C9	10E	check_dir
0	81/07/22 2352	----r-wr	CC	185	reader
0	81/07/24 2251	----r-wr	10D	9BD	many_copy_com
0	81/07/24 2320	----r-wr	118	3A0	check_dir_com
0	81/07/26 2318	------wr	12C	57C6	micro_article
0	81/07/26 2320	------wr	1A0	4A4	memmod
0	81/07/26 2323	------wr	1A6	4DE	mod_explanation
0	81/07/26 2333	------wr	1AC	420	bio

Example of Redirected Output

This is the extended output of the operating system "dir" utility. Normally this information would be displayed on a CRT terminal. In this case, however, it is redirected to a line printer, and in the example Basic09 procedure, it is redirected to a disk file, which is then processed by the "copy" utility.

File structure under OS-9 consists of a hierarchical system of directories, again much like UNIX. OS-9 recognizes only one special file type, and that is directory. There is a bit in an attribute byte in each file that marks a file as a directory. Entries in a directory file can themselves be directories, ad infinitum. Paths to data can pass through an unlimited number of directory files.

Rather than spend several pages trying to describe this concept, it would be better to illustrate it with an example, and at the same time illustrate the use of several system utility programs. In this example I will create a backup data directory as a "child" directory of the main data directory. I will then copy my article-file into the backup directory from the main directory. Let's call the original data directory "work." First, I must use the system utility "makdir" to create a new directory:

```
makdir /work/work_bak
```

Then I "copy" my data into it:

```
copy /work/micro_article
/work/work_bak/micro_article
```

Note here that most often these names would be prefaced by a name signifying a physical disk drive unit. For instance,

```
copy /d1/work/micro_article
/d2/work_bak/micro
```

copies the file located in the directory "work" located on disk drive one into a directory named "work_bak" located on disk drive two. At the same time, the name is changed from "micro_article" to "micro."

Any user at a given moment is assigned to two directories. One, the *execution directory*, contains operating system utilities, command files, and so forth, that operate on the user's data. Most of the time all users will share a single execution directory. The other, the *data directory*, contains data files, program source code, and so forth. These assignments can be changed on the fly by a shell pseudo-command, or by most of the high-level processors. Generally each user will be assigned to his own data directory. This allows several users to maintain files using the same names without crashing the operating system. Hierarchical data structuring is a powerful tool that corresponds with intuitive understanding of the real world.

Sundry Information

There is no limit set by the operating system on the number of programs (called "processes") that may run concurrently. Of course, it will never be possible to use more memory than that available at any given instant, so this, in fact, does set a limit on the number of simultaneous processes. Also, repeatedly loading and unloading small modules may cause memory to become discontiguous. For instance, if a user loads the directory list "dir" command, then another user loads the free memory "mfree" command, when the first user unloads his program there will be a "hole" left in memory. That hole may or may not be closed when the second user finishes with "mfree," depending on what has happened in the meantime. Because memory used to hold a module must be contiguous, decay of the free memory area sometimes requires re-booting the system. This limitation only holds true in the Level One version of OS-9; the extended Level Two version automatically manages memory to avoid this situation.

Any process may spawn another process at any time. A Basic09 program, for instance, may send a listing file to a print spooler, requesting that it be run as a concurrent process. If sufficient memory exists, the program can go right along with its business, with the spooler program running at the same time. It is also possible for users to assign priorities to their processes, which are used by the operating system's scheduler to determine the frequency and duration of the time slice that each is given. Thus, processes that are interfaced with a user who must wait on the system may be given a higher priority than those which can run in the background, allowing for optimization of human time spent interacting with the machine.

"Type-ahead" is fully supported by OS-9. What this amounts to is a logical separation of the keyboard and display functions on a typical terminal. When interacting with the system, it is possible to enter commands as fast as they can be typed. While a given line is being processed, OS-9's input handling routines save input from the keyboard in an invisible buffer, which acts as a queue. This queue is then acted upon, one line at a time, as programs request input. The display is unaffected by the keyboard during the time a program is acting on a command. Other operating systems which boast of having type-ahead do not manage the screen display in the same way, and even though commands can be entered rapidly, often the display becomes virtually unintelligible since each character is displayed as it is entered. Type-ahead lets you rapidly input a series of commands, then sneak a cup of coffee while they are being processed. It is a very powerful capability in the hands of a competent typist.

This quick overview does not give justice to the true power of OS-9, which must be experienced to be understood. The documentation provided by Microware is voluminous, consisting of an 88-page user's guide, and a 156-page *System Programmer's Guide*. The user's guide serves as a tutorial introduction to the system, while the *System Programmer's Guide* explains in great detail the inner workings of the system to allow for customization. Included in the *System Programmer's Guide* are source listings for a variety of system modules, both to increase understanding of the system and to illustrate programming conventions (such as the module concept), that allow for OS-9's unique power. Microware also maintains a hotline that is manned by a staff programmer during all business hours.

An operating system by itself, no matter how powerful, would be a weak tool indeed. In this respect, Microware has introduced an entire family of integrated programs that serve as a complete tool kit for the modern programmer. Foremost amongst them is Basic09, which Microware calls a "high-level programming system," instead of a BASIC interpreter or compiler. This is because Basic09 consists of an integrated package of programs that include an interactive text editor, a line-at-a-time or batch compiler, a run-time interpreter that executes optimized "I-code," and an integral debugger that allows for program tracing, inspection of variables, etc.

I've included here a teaser source code listing of a pair of Basic09 "procedures." These programs extend the operating system "copy" utility to enable it to copy all the data files in a given directory into any other directory. Note: the hex numbers at the left in the listing are not line numbers, they are "I-code" addresses that show where a given line compiles into memory, relative to the beginning of the procedure. Also, procedures may call other procedures and pass parameters to them; this is what is done in calling "check_dir." Also, dimensioning of all variables is not required, but is generally good programming practice.

Other programs in the Microware line include a traditional Macro Text Editor, an Assembler that produces both OS-9 compatible modules and MIKBUG compatible object code, a high-level debugger, the Stylograph text processing system, and a Pascal compiler. In the works, and scheduled for release in the near future, are both COBOL and C compilers. Several large applications software houses are supporting OS-9, and a number of excellent applications programs are now appearing on the market.

No system, no matter how advanced, is without its faults, and it is only fair that I relate a few reservations along with the praise. First, in the past, Microware has had a tendency to be overly optimistic when scheduling release dates for their software products. The original Level One OS-9 was announced to be available in June of 1980 and was, in fact, first delivered sometime just before January 1, 1981. Because of this they are now refusing to promise exact dates for the delivery of upcoming products.

Also, I have personally had quite a time coming to grips with the redirected I/O feature of the Shell processor. Let me forewarn users that when redirecting I/O, the system looks for all input to come from the redirected input file, and writes all output to the redirected output file. If several concurrent processes come to be "reporting live" into a single user's terminal, watch out! If I had it to do over, I would have spent a lot more time experimenting with these features before proceeding to develop applications programs.

Sometimes the error messages generated do not describe the error that has occurred. I appreciate the fact, however, that an exhaustively descriptive list of errors could run well into the hundreds, and that lines must be drawn somewhere if software is to be released.

Another positive aspect of OS-9 that bears mentioning is the wide range of hardware that it has been designed to support. Right now it is running on no

Listing 1

```
PROCEDURE directory_copy
 0000   (*
 0003   (*
 0006   (* Source Listing of Full Directory Copy Utility
 0036   (*
 0039   (* OS-9 Programmer's Tool Kit Program No. 3
 0064   (* Version 1.2
 0072   (*
 0075   (*
 0078   (* All commercial rights reserved 1981
 009E   (* Oikos Systems Company
 00B6   (*
 00B9   (*
 00BC   (* Allocate Variable Data Storage
 00DD   (*
 00E0   DIM data_line:STRING[120]; argument:STRING[80]
 00F7   DIM from_name,to_name,from_dir,to_dir:STRING[30]
 010F   DIM clear:STRING[2]; inkey:STRING[1]; clearchar:STRING[3]
 0131   DIM dpath,index1:INTEGER
 013C   DIM verify,exists,OK:BOOLEAN
 014B   (*
 014E   (* Next two lines control Soroc IQ-120
 0174   (*
 0177   clear=CHR$(27)+"+"
 0183   clearchar=CHR$(8)+CHR$(32)+CHR$(8)
 0193   (*
 0196   (* Softstart Point; Initialize Default Variables
 01C6   (* NOTE: Basic09 does NOT initialize variables!!!
 01F7   (*
 01FA 1 verify=FALSE
 0203   (*
 0206   (* Print Banner
 0215   (*
 0218   PRINT clear
 021D   PRINT  \ PRINT  \ PRINT
 0223   PRINT "F U L L   D I R E C T O R Y   C O P Y"
 024C   PRINT
 024E   (*
 0251   (* --Get & Verify Parameters
 026D   (*
 0270   (* ----Source Directory
 0287   (*
 028A   INPUT "Enter pathlist of desired source directory ",from_dir
 02BD   RUN check_dir(from_dir,exists)
 02CC   IF exists=FALSE THEN 1
 02DA   PRINT
 02DC   (*
 02DF   (* ----Output Directory
 02F6   (*
 02F9   INPUT "Enter pathlist of destination directory ",to_dir
 0329   RUN check_dir(to_dir,exists)
 0338   IF exists=FALSE THEN 1
 0346   PRINT
 0348   (*
 034B   (* ----Shall we verify???
 0364   (*
 0367   PRINT "RETURN=No verify-ANY OTHER KEY=Verify ";
 0392   GET #0,inkey
 039B   PRINT clearchar
 03A0   PRINT
 03A2   IF ASC(inkey)=$0D THEN
 03B0     verify=FALSE
 03B6   ELSE
 03BA     verify=TRUE
 03C0   ENDIF
 03C2   (*
 03C5   (* --Write source directory into list file on drive 0
 03FA   (*
 03FD   argument="dir e "+from_dir+">/d0/dirlist"
 041D   SHELL argument
 0422   (*
 0425   (* --Open list file
 0438   (*
 043B   OPEN #dpath,"/d0/dirlist":READ
 0451   (*
 0454   FOR index1=1 TO 2
 0464     (* --Skip header lines
 047A     READ #dpath,data_line
 0484   NEXT index1
 048F   (*
 0492   FOR index1=1 TO 100
 04A2     (* --Copy up to arbitrary limit of 100 files
 04CE     READ #dpath,data_line
```

```
04D8 EXITIF data_line="" THEN
04E4    (* --Last line of file is blank, leave loop
050F ENDEXIT
0513   IF MID$(data_line,22,1)="-" THEN
0525     (* --Dash in position 22 means non-directory file,
          therefore copyable
056A     (* --Filename begins at position 49
058D     from_name=RIGHT$(data_line,LEN(data_line)-48)
059D     IF verify=TRUE THEN
05A8       (* --See if operator wants to copy
05CA       PRINT clear
05CF       PRINT
05D1       PRINT "Ready to copy "; from_name
05E7       PRINT "RETURN=Proceed with copy-ANY OTHER KEY=No copy "
           ;
061B       GET #0,inkey
0624       PRINT clearchar
0629       PRINT
062B       IF ASC(inkey)=$0D THEN
0639         (* --Copy if a carriage return
0657         OK=TRUE
065D       ELSE
0661         OK=FALSE
0667       ENDIF
0669     ENDIF
066B     (*
066E     IF verify=FALSE OR verify=TRUE AND OK=TRUE THEN
0685       (* --If everything is cool then copy, else skip to next
            line
06C1        PRINT "Copying "; from_name; " to "; to_dir
06DC        argument="copy "+from_dir+"/"+from_name+" "+to_dir+"/"
             +from_name
0704        SHELL argument
0709     ENDIF
070B   ENDIF
070D NEXT index1
0718 (*
071B (* --Close up shop
072D (*
0730 CLOSE #dpath
0736 SHELL "del /d0/dirlist"
0749 END
```

fewer than five different hardware configurations, and several different bus designs. Many hardware development people tell me that they are working furiously to implement OS-9 on their particular configurations, so I assume that the list of supported hardware will be increasing substantially in the future. This is enhanced by Microware's willingness to sell source code for almost the entire system. At first I thought that this was a crazy notion. Now that I see Tandy introducing a 6809 processor in the Color Computer, and several outfits working to produce a 6809 board for the Apple, the logic becomes easier to understand. If OS-9 is going to reach its fullest potential, it is going to have to be used by a very large number of programmers. This would not be possible if Microware had to implement each and every customization itself. Plus, advanced users feel far more comfortable with systems that are supported by source code than by the virtual "black boxes" that other vendors are supplying, and this can only lend to the support that OS-9 will gain.

It does not take too long a perusal of the literature to discover that experts feel we are suffering a "software crisis." Problems of machine and memory location dependence require the reworking of most software even when transporting among systems using the same processor. Heartache caused by slapdash programming, while more nebulous, consumes untold hours of programmer time when software must be adapted or repaired. The Motorola MC6809 microprocessor takes a giant step forward in solving this crisis, and it is used to its fullest in conjunction with Microware's revolutionary systems software.

Brian Capouch runs a farm consulting business in northern Indiana. He works on applications software for farms and small businesses, writes for newspapers and magazines, and teaches high school data processing.

Listing 2

```
PROCEDURE check_dir
 0000      (*
 0003      (*
 0006      (* Source listing of Valid Directory Verify Routine
 0039      (*
 003C      (* OS-9 Programmer's Tool Kit Procedure No. 3
 0069      (* Version 1.0
 0077      (* NOTE: THIS VERSION NOT VALID IN MULTIUSER CONFIGURATIONS
 00B2      (*
 00B5      (* All Commercial Rights Reserved 1981
 00DB      (* Oikos Systems Company
 00F3      (*
 00F6      (* -Allocate Variable Data Storage
 0118      PARAM directory_name:STRING[30]; exists:BOOLEAN
 012A      DIM dpath:INTEGER
 0131      (*
 0134      (* -Set error routine vector
 0150      (*
 0153      ON ERROR GOTO 10
 0159      (*
 015C      (* Try to open file
 016F      (* -Note:This operation will always result in error
 01A3      (* -      if file is a directory
 01C2      (*
 01C5      OPEN #dpath,directory_name:READ
 01D1      CLOSE #dpath
 01D7      (*
 01DA      (* If we got this far, it's a data file--not directory!!
 0212      (*
 0215      exists=FALSE
 021B      END
 021D 10   IF ERR=214 THEN
 022A        (* -This error is returned if file is directory type
 025E        exists=TRUE
 0264        END
 0266      ENDIF
 0268      IF ERR=216 THEN
 0272        (* -This error is returned if there is no such file
 02A5        exists=FALSE
 02AB        END
 02AD      ENDIF
 02AF      (*
 02B2      (* -Next line is a precautionary measure
 02DA      (*
 02DD      exists=FALSE
 02E3      END
```

MICRO

Apple II Digital Storage Oscilloscope

Use minimal hardware, Applesoft BASIC, and 6502 machine language to convert an Apple II into a triggered digital "storage oscilloscope."

Ellis Cooper
200 West 14th St.
New York City, New York 10011

This article gives complete details on building a peripheral card that fits in slot #7 of the Apple II computer. The card has an audio frequency buffer and attenuation circuit, plus a very easy-to-use analog-to-digital converter which is capable of sampling the signal every 25 microseconds. The software is a combination program of Applesoft BASIC and 6502 machine language. It prompts you to initiate an audio input, then waits for it. It has a trigger threshhold. When the signal level exceeds the threshhold, the circuit takes 19,600 samples spaced about 40 μS apart, storing each one in upper RAM (starting at $7370 in a 48K space). Then a high-resolution plot of the first 280 samples — one screen — is displayed.

The caption beneath the screen-plot gives start and stop times of that screen in milliseconds. It also prompts you to push certain keys to look at other screen-plots. There are 70 screen-plots altogether, each covering roughly 12 milliseconds. Thus, the original signal is sampled for about .84 seconds. See figure 1 for an example of a screen-plot. The circuitry and code are fully operational, but the information here should really be taken as a suggestive guide for customizing the ideas to your own needs.

Physically, the circuit is connected together by point-to-point soldering of wire-wrap type wire on a California Computer Systems model 7500 prototype board (see figure 2). (This board costs $21, and the labels on both sides

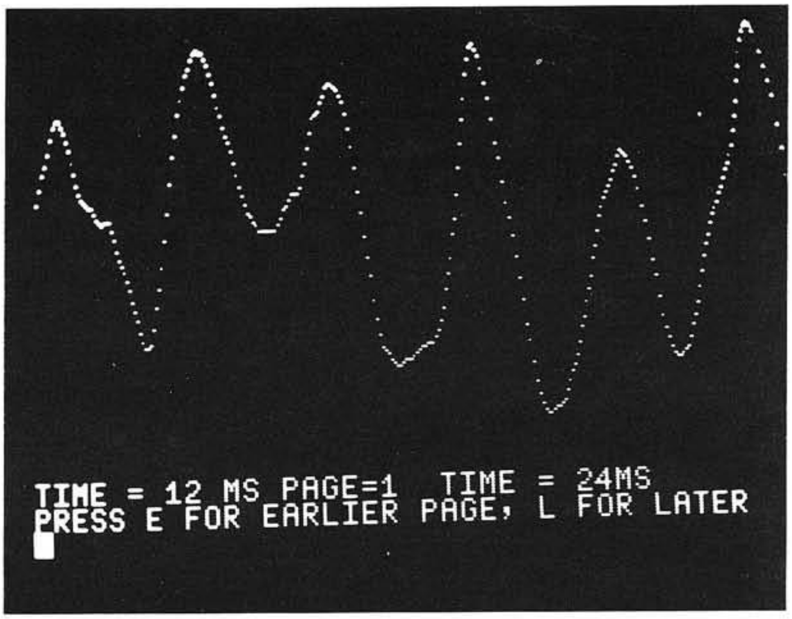


Figure 1: Sample plot of waveform produced by a conga drum.

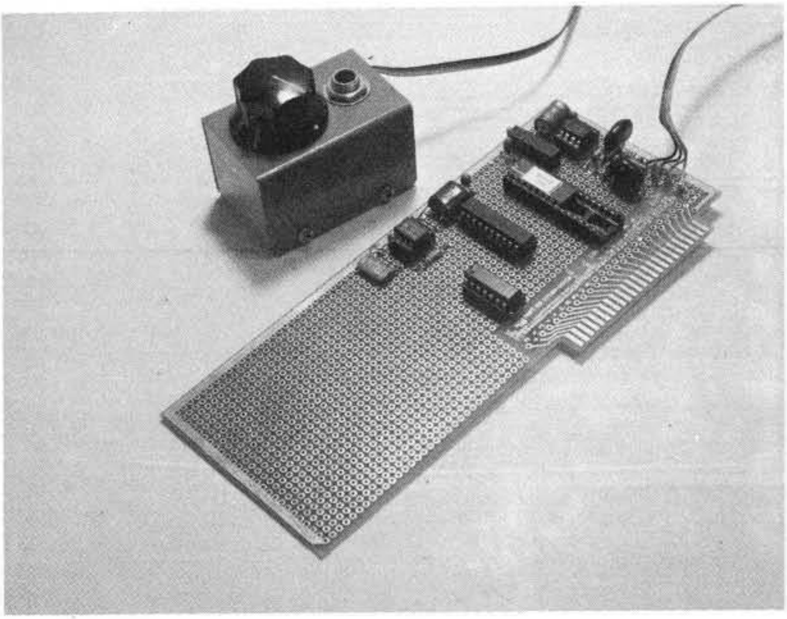


Figure 2: Component side of the ADC board.

for all edge-connector terminals are very helpful for wiring and checkout.) There is a two-wire ribbon cable from the card to a little metal box with a phone jack for plugging in the signal, and a potentiometer for attenuating the input signal.

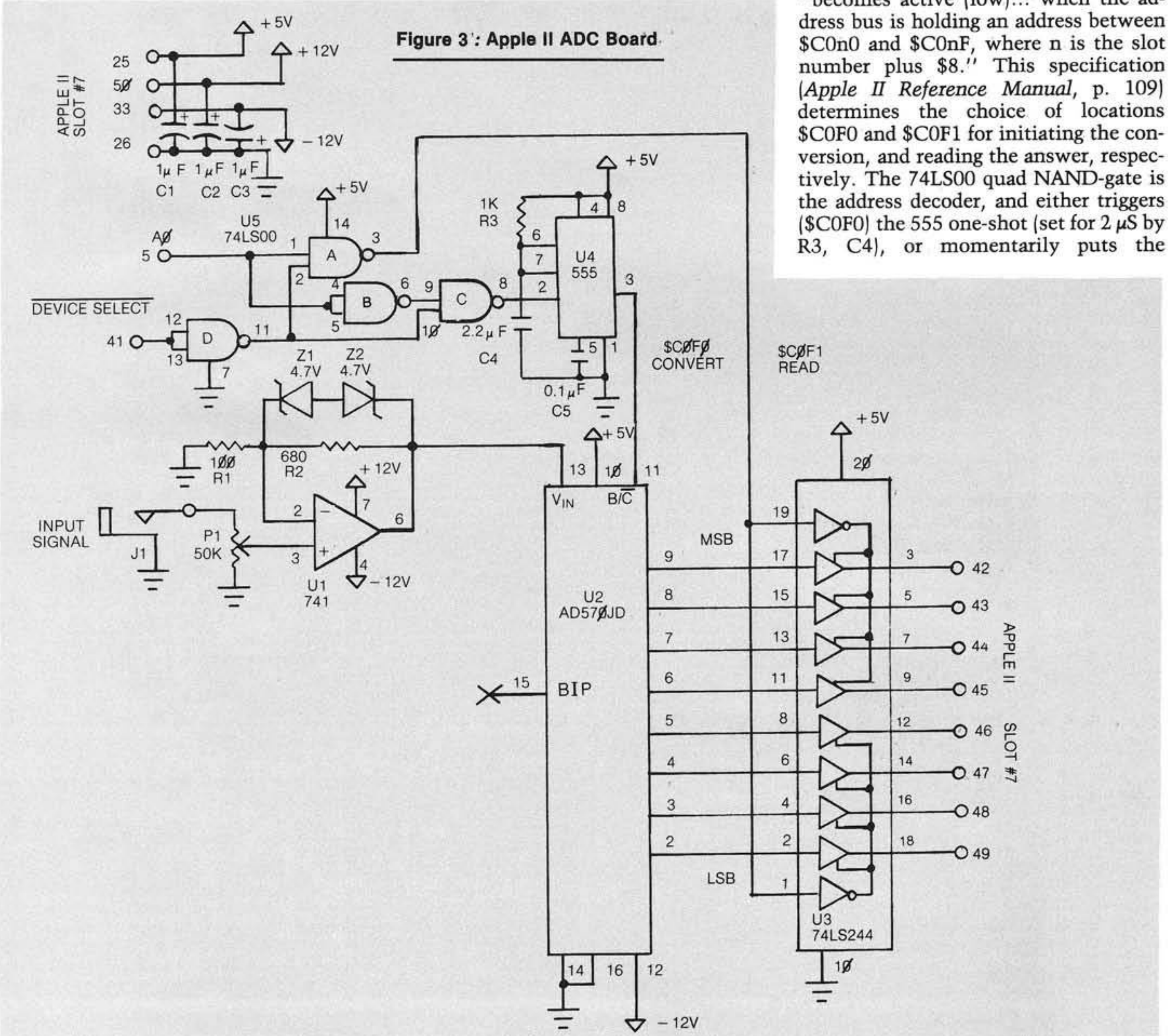
You do well if you find an integrated circuit as friendly as the Analog Devices AD570JD. This costs $22 (in singles) but is a completely self-contained, successive-approximation, analog-to-digital converter. It has one analog input (choose unipolar 0-10V input by grounding the BIPOLAR CONTROL pin, or leave the pin floating for bipolar + −5V input), and 8 digital outputs. There is one input control line called BLANK/$\overline{\text{CONVERT}}$ (B/$\overline{\text{C}}$), and one output acknowledgement line, $\overline{\text{DATA READY}}$ ($\overline{\text{DR}}$). In BLANK mode the digital output lines are floated (tri-state, high impedance condition). Nothing happens until B/$\overline{\text{C}}$ is brought low, thereby switching the unit into CONVERT mode. After grinding out the answer in 25 µS, the result appears latched onto the eight output lines, and DR is brought low. That is it. Bring the unit back into BLANK, and start over. It has to stay in BLANK a minimum of 2 µS before another conversion is initiated. Also, in bipolar mode the output is offset binary (zero signal gives output 128). At this point you can foresee that the software loop for filling RAM area with samples of audio signal will have the following steps:

1. initiate a conversion by bringing B/$\overline{\text{C}}$ from low to high for 2 µS;
2. wait at least 25 µS, and use the time to check for done and to compute the address of the next location to stick the answer;
3. read the answer;
4. stick it in RAM;
5. go back to 1.

Here is how the circuit works (figure 3). The audio signal is applied at phonejack J1 to a non-inverting gain stage based on a 741-type operational amplifier. Potentiometer P1 adjusts input signal level, and is mounted remotely with J1. The gain is equal to one plus the ratio of R2 to R1, and with the values shown, the gain is 7.8. The back-to-back Zener diodes Z1, Z2 clip the signal to + −5V peak-to-peak, which is the analog input range of the AD570JD.

I needed a 2 µS positive-going pulse from some pin of slot #7. That was not available, but the DEVICE SELECT line at pin 41 of the peripheral connector "becomes active (low)... when the address bus is holding an address between $C0n0 and $C0nF, where n is the slot number plus $8." This specification (*Apple II Reference Manual*, p. 109) determines the choice of locations $C0F0 and $C0F1 for initiating the conversion, and reading the answer, respectively. The 74LS00 quad NAND-gate is the address decoder, and either triggers ($C0F0) the 555 one-shot (set for 2 µS by R3, C4), or momentarily puts the

Figure 3: Apple II ADC Board



Listing 1

```
100  HOME
110  REM ---DEFINE UPPER LIMIT OF BASIC AREA;
120  REM ---LOWER LIMIT OF DATA BUFFER;
130  REM ---K IS A CONVERSION CONSTANT
140  REM ---FOR PLOTTING IN HI-RES.
150  HIMEM: 8191:BASE = 29552:W = 159 / 255
160  REM ---PROMPT.
170  PRINT "PLAY A FEW NOTES AFTER HEARING TONE"
180  REM ---BRIEF DELAY.
190  FOR I = 1 TO 500: NEXT
200  REM ---BRIEF TONE.
210  FOR I = 1 TO 200:E =  PEEK (49200): NEXT
220  REM ---DEFINE JMP $0300 FOR USR FUNCTION.
230  POKE 10,76: POKE 11,0: POKE 12,3
240  REM ---CALL DATA COLLECTION ROUTINE.
250 ANS =  USR (X)
260  REM ---INITIALIZE FOR SCREEN-PLOT ("PAGE") CAPTION.
270 TIME = 12:PAGE = 0
280  REM ---INITIALIZE FOR HI-RES:
290 SW = 49232
300  REM ---MIXED,
310  POKE SW + 3,0
320  REM ---PAGE-ONE,
330  POKE SW + 4,0
340  REM ---HIGH-RESOLUTION,
350  POKE SW + 7,0
360  REM ---GRAPHICS.
370  POKE SW,0
380  REM :---ADJUST CURSOR FOR CAPTION.
390  HOME : VTAB 21: POKE 36,0
400  REM ---DISPLAY CURRENT PAGE.
410  GOSUB 560
420  REM ---COMPUTE AND DISPLAY CAPTION.
430 LFT = PAGE * TIME
440 RIT = LFT + TIME
450  PRINT "TIME = "LFT" MS";
460  PRINT  TAB( 14)"PAGE="PAGE;
470  PRINT  TAB( 22)"TIME = "RIT"MS"
480  REM ---INTERPRET PRESSED KEY.
490  PRINT "PRESS E FOR EARLIER PAGE, L FOR LATER"
500  GET K$: IF K$ <  > "E" AND K$ <  > "L" THEN  GOTO 500
510  IF K$ = "E" AND PAGE > 0 THEN PAGE = PAGE - 1: GOTO 390
520  IF K$ = "L" AND PAGE < 30 THEN PAGE = PAGE + 1: GOTO 390
530  GOTO 500
540  REM ---HI-RES SCREEN-PLOT:
550  REM ---INITIALIZE BEGIN AND END BYTES.
560 LO = BASE + PAGE * 280
570 HI = LO + 279
580  REM ---CLEAR PREVIOUS PLOT.
585  HGR
590  REM ---PLOT 280 DATA POINTS FROM BUFFER.
600  FOR I = LO TO HI
610 Y =  PEEK (I)
620 Y =  INT (Y * W)
630  HPLOT I - LO,159 - Y
640  NEXT
650  RETURN
```

latched data at the converter output ($C0F1) onto the system data bus. That is how the circuit works.

While assembling the unit, I was concerned that the 12V would get through a wiring error onto some TTL line — either onto the +5V supply, or onto an address or data line. So I advise you to do at least as much double-checking and testing of the circuit as I did, even before plugging it in without chips. First, perform continuity tests to insure that there are no paths from −12V to +5V; second, see if all pins have "correct" resistance to ground. Third, leave all chips out, and install the board in slot #7. Turn on the Apple II and if it behaves normally, measure voltages at all pins of all sockets on the new unit. Fourth, when you are satisfied again that all is well, turn off the power, pull the board, and install U4, U5.

Fifth, put the board back, re-apply power, check voltages at the installed chips, and if all is well, try the following test program from the machine language monitor:

```
0C00  STA $C0F0
      STA $C0F1
      JMP $0C00
```

Or, for more fun, try the slightly more elaborate

```
100 A=49392:B=A+1:X=
    40/255
110 POKE A,0
120 Y=PEEK(B):P=INT(X*Y)
130 PRINT TAB(P)"*"
140 GOTO 110
```

Be sure to try out the attenuator, P1.

(Continued)

Listing 2

```
                  ;*******************
                  ;*                 *
                  ;* DIGITAL STORAGE *
                  ;*  OSCILLOSCOPE   *
                  ;*                 *
                  ;*   ELLIS COOPER  *
                  ;*                 *
                  ;*******************
                  ;
                  SCOPE  ORG $300

                  ;
0300 48           START  PHA             ;SAVE PROCESSOR
0301 8A                  TXA             ; STATUS ON STACK...
0302 48                  PHA
0303 98                  TYA
0304 48                  PHA
0305 08                  PHP
0306              ;
0306 A96F                LDA #$6F        ;INITIALIZE TWO CONSECUTIVE
                                          BYTES,
0308 854B                STA $4B         ;$4C $4B, IN PAGE ZERO AS
030A A973                LDA #$73        ;POINTER TO DATA BUFFER..
030C 854C                STA $4C
030E 8DFOCO       CNV1   STA $COFO       ;START A CONVERSION.
0311 A207                LDX #$07        ;DELAY UNTIL END
0313 CA           DELAY  DEX             ; OF CONVERSION.
0314 DOFD                BNE DELAY
0316              ;
0316 ADF1CO              LDA $COF1       ;READ PULSE AMPLITUDE.
0319 C98C                CMP #$8C        ;IS THRESHHOLD EXCEEDED?
031B 30F1                BMI CNV1
031D              ;
031D A000                LDY #$00        ;YES; CLEAR Y FOR
                                          INDIRECT-INDEXED MODE
031F 8DFOCO       CNV2   STA $COFO       ;START A CONVERSION
0322 A54C                LDA $4C         ;LAST PAGE OF BUFFER?
0324 C994                CMP #$94
0326 D006                BNE INCR        ;NO.
0328 A54B                LDA $4B         ;YES; LAST BYTE OF LAST PAGE OF
032A C900                CMP #$00        ;OF BUFFER?
032C F015                BEQ EXIT        ;YES.
032E              ;
032E 18           INCR   CLC             ;NO; CLEAR CARRY FOR ADDITION
032F A54B                LDA $4B
0331 6901                ADC #$01        ;INCREMENT THE POINTER...
0333 854B                STA $4B
0335 A54C                LDA $4C
0337 6900                ADC #$00
0339 854C                STA $4C
033B              ;
033B ADF1CO              LDA $COF1       ;READ PULSE AMPLITUDE
033E 914B                STA ($4B),Y     ;STORE IN DATA BUFFER
0340 4C1F03              JMP CNV2        ;GO BACK FOR MORE
0343              ;
0343 28           EXIT   PLP             ;RESTORE PROCESSOR STATUS
0344 68                  PLA             ;FROM STACK
0345 A8                  TAY
0346 68                  PLA
0347 AA                  TAX
0348 68                  PLA
0349              ;
0349 60                  RTS             ;RETURN TO BASIC.
                         END
```

Using an oscilloscope you should see 2 μS positive pulses at U2(11) and 0.5 μS active-low pulses at U3(1) and U3(19). Now, sixth, install the remaining chips and repeat all tests.

If everything appears OK at this point, you may be confident that your board is working, but there is nothing like a conversion to convince you. Seventh, plug in an electric guitar or a microphone, or even just a speaker which has a big magnet, and try a program in BASIC:

```
100 POKE 49392,0
110 PRINT PEEK(49393)" ";
120 GOTO 100
```

If these little tests do not turn up any surprises, it is time to put in the main program of this article (listing 1 and listing 2). Be sure to save both parts before running. Be warned, you must have 48K RAM to use this software exactly as written. Like I said, though, you should use these listings only as a starting point, if at all, to carry out your own ideas.

One refinement of the software would be to display only every nth sample, or to sample less frequently but for a longer duration. Another idea is to swap back and forth between two high-resolution pages, achieving an "animated" display of the waveform. As for me, it is time to bone up on algorithms for extracting significant information from the stored data, e.g., pitch periods, envelopes, and so forth.

References

1. "The Piecewise-Linear Technique of Electronic Music Synthesis," E.D. Cooper and A.D. Bernstein, *Journal of the Audio Engineering Society*, V. 24, No. 6, July/August, 1976, pp. 446-454.

2. "Circuits for the Piecewise-Linear Technique of Electronic Music Synthesis," E.D. Cooper, *Electronotes Newsletter of the Musical Engineering Group*, V. 8, No. 69, September, 1976, pp. 8-13.

3. "Program Performs Harmonic Analysis," E.D. Cooper, *μComputerist Corner*, EDN, February 5, 1980, pp. 80-85.

Ellis Cooper owns an Apple II Plus microcomputer with a single disk drive, NEC 12" video monitor, and Centronics 737 printer. He is employed as a research mathematician by an international gold bullion dealer.

MICRO

Function Generator and Library Manager

This program builds Applesoft subroutines to handle keyboard input, display output, file input/output/update, sorts and PRINT USING. Input/output functions are customized, based on a file description in DATA statements.

Ray Cadmus
600 W. Lee
Moberly, Missouri 65270

Program Writer (Almost)

Have you ever considered how nice it would be to write the main routine for a program and then have all the subroutines it calls just magically appear? Sound farfetched? Stay with me. I don't really have any magic, just the next best thing — a program to generate the routines you need.

Here is a quick example to illustrate what I mean. I want a program to read and display the first 10 records in a direct access file. I'm assuming the record numbers are 1 to 10.

```
10 FOR R = 1 TO 10
20 GOSUB 1000 : REM READ A
   RECORD
30 GOSUB 2000 : REM DISPLAY
   THE RECORD
40 PRINT:PRINT "ANY KEY
   WHEN READY ":GET Z$:
   PRINT Z$
50 NEXT
60 END
```

Nothing to that part. It's those subroutines that take the time. It's even worse working with direct files (as I do 99% of the time). With direct files you want fixed length fields within a fixed length record. That calls for a lot of packing and unpacking of data. No more! Instead we run "FDGEN."

FDGEN (file descriptor generation) grew out of my frustration with the situation just described. I found myself spending 10% of my time writing the guts of a program and the other 90% coding the routines to make it work. Since that time, FDGEN has grown to include keyboard input and update routines, sort routines, etc., as well as disk I/O, including the pack and unpack to handle fixed length fields and records.

FDGEN works from a file description in DATA statements that you add to the program as needed. This could be a separate file instead, but I've found it handy to keep all my file definitions together here. Anyway, RUN FDGEN. It asks if the file definition exists. If not, it terminates to allow you to add them.

The format is:

```
nnnn DATA filename,length
nnnn DATA field-id,length,title...
```

If you answer yes, that the file description does exist, then you get a menu asking what routine to generate. As you select routines, they are tailored to the file description and written to disk. When you have selected all the necessary routines, then select the end option to close out the disk file.

Now, save FDGEN with your file description, clear the computer (NEW), and EXEC your file name.FD. Your subroutines should now be in memory, just as if you had coded them yourself. Add the statements for your main routine and you have a program.

A couple random comments: You may have any number of file definitions as DATA statements. The program searches by file name. Caution: the total of the field lengths must add up to the record length. Use a dummy field, if necessary, to pad out the record and allow for expansion room. The file handling routines are designed for fixed length records, hence you must make some provision for getting the correct record number "R" before executing the read or write routine.

This program belongs to a class of programs that I consider programming tools. Before I get into descriptions of individual routines built by FDGEN, I would like to mention a couple other tools that I consider indispensable. One is RENUMBER from Apple, the other is PLE or Program Line Editor from *Call A.P.P.L.E.*, written by Neil Konzen. These programs are complementary and reside together in memory at all times when I'm programming. The reason that I mention these programs here is to point out that the routines built by FDGEN may not always be just what is needed. Feel free to modify and do your own thing with them. PLE makes the chore much easier.

Now for the routines themselves. I'll start with the I/O routines since they are the reason that this program exists at all. As I mentioned earlier, these routines deal with fixed length fields and records. Perhaps it's my big systems background, but I feel more comfortable with fixed length records. Also Apple DOS requires them for random access. The I/O routines are similar in that each starts with a file open, if the file is not already open. You must close the file in your own code. Remember, at that time, to set F1 = 0 so the routine knows the file is closed.

Random files also require a record number. The routines expect it in "R." As you build a file you may either assign it sequentially, or you could "hash it" from some field in the record to achieve random access. By hash it I mean develop a unique number based upon something not necessarily numeric. As an example, you could add up the numeric "ASC" equivalents for each position in a name field and use the sum as the record number. There is any number of possibilities, so let your imagination go.

The keyboard input routine serves a dual purpose. If you execute it with UD = 1 (UD is the update switch), it will display existing values and give you a chance to change them. If UD = 0, then it accepts input normally. Here again feel free to modify. Since the input routine reads as a string initially, it will accept anything without error, and will give you a chance to insert any editing or check routines that you desire. If you do no editing and alpha data is input to a numeric field, the resulting value will be 0.

Once all fields for a record are entered, the routine will ask if all are OK. If reply is "N," then the cursor is repositioned at the first entry position. Change data if you wish, or leave it by hitting "RETURN."

Another way to use this routine is during entry of data, where some of the data is common to a group. If you assign a new record number, then enter the input routine with the update switch on (UD=1). Then the last record will be displayed and "RETURN" will duplicate any field. New data may be keyed over the old as required.

The sort routines are straightforward Shell-Metzner sorts. The numeric routine will sort the ARRAY "SR" from the first position to the number in "SY." The string sort does basically the same thing, only sorting array "SR$." One special feature of the string sort is the use of the string swap routine written by Randy Wiggington and published in *Call A.P.P.L.E.* This speeds up the process and eliminates pauses for "garbage collection" by BASIC. To sort on particular fields in a record with the string sort, change the comparison statements to be "MID$" type, rather than compare the whole record.

The PRINT USING routine is a quick and dirty one for money fields only. (See Arnold Edelstein's article in this issue for a more elaborate routine.) To use it, place the numeric field to be printed into "P." The length you want the printed or displayed data to take up on output goes in "PL," then GOSUB nnnn:PRINT P$. There is one deficiency in the routine. A value of −.01 through −.09 will print as .−n rather than −.0n as it should. That's the price for simplicity.

That's about it. The easiest way to get a good feel for the program is to look over the examples, then use the routine. If you come up with any favorite routines worth adding to FDGEN, I'd like to hear about them.

Ray Cadmus has been in data processing since the late 50's and programming since the early 60's. Most of his work has been with business applications on large scale IBM equipment. He started programming microcomputers because it gave him the opportunity to write what he wanted, rather than what business pressure dictated. Now, though he still works with micros for fun, he is expanding his consulting activities into the area of small business computers and hopes to someday make that his primary occupation.

***Figure 1:* Sample File Structure**

```
2001  DATA   TEST,20
2002  DATA    A$,10,FLD1,B,5,FLD2,C$,5,FLD3
2003  DATA   ADDR,74
2004  DATA  NA$,20,NAME, AD$,20,ADDRESS
2005  DATA  CT$,15,CITY
2006  DATA  ST$,2,STATE
2007  DATA  ZP$,5,ZIP
2008  DATA  PH$,12,PHONE
```

***Figure 2:* Sample Run**

```
 FDGEN


 THIS PROGRAM BUILDS A TEXT FILE THAT MAY BE EXECUTED  INTO
AN A-SOFT PROGRAM TO OPEN - READ - WRITE - PACK - AND UNPACK
A DATA FILE
THE ROUTINE IS BUILT FROM DATA STATEMENTS
YOU ADD TO THIS PROGRAM



THE DATA STATEMENT FORMATS ARE:

2NNN DATA FILENAME,RECORDLENGTH
2NNN DATA ID,LENGTH,TITLE...


DOES FILE DATA ALREADY EXIST? Y


ENTER FILE NAME ADDR
SELECT ONE OF


1 - BUILD FILE OUTPUT
2 - BUILD FILE INPUT
3 - BUILD KEYBOARD INPUT
4 - BUILD LIST ROUTINE
5 - BUILD STRING SORT
6 - BUILD NUMERIC SORT
7 - PRINT USING ROUTINE
9 - END

?3
STARTING LINE NO 1000
1000REM

 INPUT ROUTINE


1000HOME
1001N=1
1002N=N+1:VTAB N:HTAB 1:PRINT"NAME";
1003IF UD THEN HTAB 15:PRINTNA$
1004N=N+1:VTAB N:HTAB 1:PRINT"ADDRESS";
1005IF UD THEN HTAB 15:PRINTAD$
1006N=N+1:VTAB N:HTAB 1:PRINT"CITY";
1007IF UD THEN HTAB 15:PRINTCT$
1008N=N+1:VTAB N:HTAB 1:PRINT"STATE";
1009IF UD THEN HTAB 15:PRINTST$
1010N=N+1:VTAB N:HTAB 1:PRINT"ZIP";
1011IF UD THEN HTAB 15:PRINTZP$
1012N=N+1:VTAB N:HTAB 1:PRINT"PHONE";
1013IF UD THEN HTAB 15:PRINTPH$
1014N=1
1015N=N+1:VTAB N:HTAB 15:INPUT"";Z$
1016IF LEN(Z$)=0 AND NOT UD THENNA$=""
1017IF LEN(Z$)=0 AND UD THEN VTAB N:HTAB15:PRINTNA$
```

(continued)

***Figure 2:* Sample Run** *(continued)*

```
1018IF LEN(Z$)>0THENNA$=Z$
1019N=N+1:VTAB N:HTAB 15:INPUT"";Z$
1020IF LEN(Z$)=0 AND NOT UD THENAD$=""
1021IF LEN(Z$)=0 AND UD THEN VTAB N:HTAB15:PRINTAD$
1022IF LEN(Z$)>0THENAD$=Z$
1023N=N+1:VTAB N:HTAB 15:INPUT"";Z$
1024IF LEN(Z$)=0 AND NOT UD THENCT$=""
1025IF LEN(Z$)=0 AND UD THEN VTAB N:HTAB15:PRINTCT$
1026IF LEN(Z$)>0THENCT$=Z$
1027N=N+1:VTAB N:HTAB 15:INPUT"";Z$
1028IF LEN(Z$)=0 AND NOT UD THENST$=""
1029IF LEN(Z$)=0 AND UD THEN VTAB N:HTAB15:PRINTST$
1030IF LEN(Z$)>0THENST$=Z$
1031N=N+1:VTAB N:HTAB 15:INPUT"";Z$
1032IF LEN(Z$)=0 AND NOT UD THENZP$=""
1033IF LEN(Z$)=0 AND UD THEN VTAB N:HTAB15:PRINTZP$
1034IF LEN(Z$)>0THENZP$=Z$
1035N=N+1:VTAB N:HTAB 15:INPUT"";Z$
1036IF LEN(Z$)=0 AND NOT UD THENPH$=""
1037IF LEN(Z$)=0 AND UD THEN VTAB N:HTAB15:PRINTPH$
1038IF LEN(Z$)>0THENPH$=Z$
1039?:?"OK < Y,N >";:GETZ$:?Z$:IF Z$="N"
       THEN UD=1:GOTO 1000
1040RETURN



1041:
SELECT ONE OF


1 - BUILD FILE OUTPUT
2 - BUILD FILE INPUT
3 - BUILD KEYBOARD INPUT
4 - BUILD LIST ROUTINE
5 - BUILD STRING SORT
6 - BUILD NUMERIC SORT
7 - PRINT USING ROUTINE
9 - END




?1
STARTING LINE NO 2000
2000REM

FILE I/O


2001IF NOT F1 THEN F1=1:PRINTD$"OPEN ADDR,L75"
2002R$="":B$="                    "
2003R$=R$+LEFT$(NA$+B$,20)
2004R$=R$+LEFT$(AD$+B$,20)
2005R$=R$+LEFT$(CT$+B$,15)
2006R$=R$+LEFT$(ST$+B$,2)
2007R$=R$+LEFT$(ZP$+B$,5)
2008R$=R$+LEFT$(PH$+B$,12)
2009PRINT D$"WRITE ADDR,R"R
2010PRINT R$
2011PRINT D$
2012RETURN
SELECT ONE OF


1 - BUILD FILE OUTPUT
2 - BUILD FILE INPUT
3 - BUILD KEYBOARD INPUT
4 - BUILD LIST ROUTINE
5 - BUILD STRING SORT
6 - BUILD NUMERIC SORT
7 - PRINT USING ROUTINE
9 - END

?9
```

Program Listing

```
10  REM   *** FDGEN 8/18/80 ***
20  REM  *** RAY CADMUS - MOBERLY, MO. ***
30  REM  *** INITIALIZE ***
34 Q$ =  CHR$ (34)
40  GOSUB 850: REM  - FIND FILE TO PROCESS
50 FF$ = F$ + ".FD"
60 CD$ = "": REM   CTL-D
70 LN = 60200
80  PRINT CD$"MON O"
90  PRINT CD$"OPEN "FF$
100  GOTO 120
110  PRINT CD$"WRITE"FF$: RETURN
120  REM *** MAIN ROUTINE ***
130  PRINT CD$: HOME
140  PRINT "SELECT ONE OF": PRINT : PRINT
150  PRINT "1 - BUILD FILE OUTPUT"
160  PRINT "2 - BUILD FILE INPUT"
170  PRINT "3 - BUILD KEYBOARD INPUT"
175  PRINT "4 - BUILD LIST ROUTINE"
176  PRINT "5 - BUILD STRING SORT"
177  PRINT "6 - BUILD NUMERIC SORT"
178  PRINT "7 - PRINT USING ROUTINE"
180  PRINT "9 - END": PRINT
190  INPUT N: IF N = 9 THEN  GOTO 230
200  IF N < 5 THEN  GOSUB 970: GOSUB 1150: GOSUB 110
210  ON N GOSUB 250,600,1000,1170,1300,1400,1500
220  GOTO 130
230  PRINT CD$"CLOSE"FF$: END
240  REM *** BUILD OUTPUT RTNS ***
250  GOSUB 330: REM   BUILD OPEN STATEMENT
260  READ D$,L,H$:LN = LN + 1
270  IF  RIGHT$ (D$,1) = "$" THEN  GOSUB 460: GOTO 290
280  GOSUB 420
290 RL = RL + L
300  IF RL = R THEN  GOTO 500
310  GOTO 260
320 :
330  REM *** OUTPUT OPEN ***
350  PRINT LN"REM

                  FILE I/O
                          "
360 LN = LN + 1
370  PRINT LN"IF NOT F1 THEN F1=1:PRINTD$"Q$"OPEN "F$"
    ,L"R + 1;Q$
380 LN = LN + 1
390  PRINT LN"R$="Q$;Q$;":B$="Q$"                    "Q$
400  RETURN
410 :
420  REM  *** PACK NUMERIC ***
430  PRINT LN"R$=R$+LEFT$(STR$("D$")+B$,"L")"
440  RETURN
450 :
460  REM  *** PACK ALPHA *****
470  PRINT LN"R$=R$+LEFT$("D$"+B$,"L")"
480  RETURN
490 :
500  REM *** OUTPUT CLOSE - CLOSE ***
510 LN = LN + 1
520  PRINT LN"PRINT D$"Q$"WRITE "F$;",R";Q$;"R"
530 LN = LN + 1
540  PRINT LN"PRINT R$"
550 LN = LN + 1
560  PRINT LN"PRINT D$"
570 LN = LN + 1
580  PRINT LN"RETURN"
590  RETURN : REM  - END BUILD WRITE
600  REM  ** GEN READ & UNPACK ***
610  GOSUB 330: REM  BUILD OPEN STATEMENT
620 RL = 0
630  GOSUB 970: REM   - FIND FILE IN DATA STATEMENTS
640 LN = LN + 1
650  PRINT LN"PRINT D$"Q$"READ "F$;",R";Q$;"R"
660 LN = LN + 1
670  PRINT LN"INPUT R$"
680 LN = LN + 1
690  PRINT LN"PRINT D$"
```

(continued)

```
700  READ D$,L,H$
710 P1 = RL + 1
720 LN = LN + 1
730  IF  RIGHT$ (D$,1) = "$" THEN  GOSUB 780: GOTO 750
740  GOSUB 810
750 RL = RL + L
760  IF RL = R THEN LN = LN + 1: PRINT LN"RETURN": RETURN
770  GOTO 700
780  REM  *** UNPACK ALPHA ***
790  PRINT LN;D$"=MID$(R$,"P1","L")"
800  RETURN
810  REM  *** UNPACK NUM ***
820  PRINT LN;D$"=VAL(MID$(R$,"P1","L"))"
830  RETURN
840  REM *** FIND FILE DATA ***
850  HOME : PRINT : PRINT "FDGEN "
860  PRINT : PRINT : PRINT "THIS PROGRAM BUILDS A TEXT FILE TH
AT MAY BE EXECUTED  INTO AN A-SOFT PROGRAM"
870  PRINT "TO OPEN - READ - WRITE - PACK - AND UNPACK A DATA
FILE"
880  PRINT "THE ROUTINE IS BUILT FROM DATA STATEMENTS YOU ADD
TO THIS PROGRAM": PRINT : PRINT
890  PRINT "THE DATA STATEMENT FORMATS ARE:": PRINT
900  PRINT "2NNN DATA FILENAME,RECORDLENGTH"
910  PRINT "2NNN DATA ID,LENGTH,TITLE..."
920  PRINT : PRINT : INPUT "DOES FILE DATA ALREADY EXIST? ";QQ
$
930  IF  LEFT$ (QQ$,1) < > "Y" THEN  PRINT "ENTER DATA STATEM
ENTS - THEN RESTART": END
940  PRINT : PRINT : INPUT "ENTER FILE NAME ";N$
950  GOSUB 970: RETURN
960  REM *** SET PTR TO START OF FILE ***
970  RESTORE :RL = 0
980  READ F$: IF F$ = N$ THEN  READ R: RETURN
990  GOTO 980
1000  REM *** BUILD INPUT RTN ***

1003  GOSUB 960: REM    - FIND FILE
1006  PRINT LN"REM     INPUT ROUTINE
                              "
1009 LS = LN
1012  PRINT LN"HOME"
1015 LN = LN + 1
1018  PRINT LN"N=1"
1021 LN = LN + 1
1024  READ D$,L,H$
1027  PRINT LN"N=N+1:VTAB N:HTAB 1:PRINT"Q$H$Q$";"
1030 LN = LN + 1
1033  PRINT LN"IF UD THEN HTAB 15:PRINT"D$
1036 RL = RL + L
1039  IF RL = R THEN  GOTO 1045
1042  GOTO 1021
1045 LN = LN + 1
1048  GOSUB 970: PRINT LN"N=1"
1051 LN = LN + 1
1054  READ D$,L,H$
1057  PRINT LN"N=N+1:VTAB N:HTAB 15:INPUT"Q$Q$";Z$"
1060 LN = LN + 1
1063  PRINT LN"IF LEN(Z$)=0 AND NOT UD THEN"D$"="Q$Q$
1066 LN = LN + 1
1069  PRINT LN"IF LEN(Z$)=0 AND UD THEN VTAB N:HTAB15:PRINT"D$
1072 LN = LN + 1
1075  IF  RIGHT$ (D$,1) = "$" THEN  PRINT LN"IF LEN(Z$)>0THEN"D$"=Z$"
1076  IF  RIGHT$ (D$,1) < > "$" THEN  PRINT LN"IF LEN(Z$)>0THEN"D$"=VAL(Z$)"
1078 RL = RL + L
1081  IF RL = R THEN  GOTO 1087
1084  GOTO 1051
1087 LN = LN + 1
1090  PRINT LN"?:?"Q$"OK < Y,N >"Q$";:GETZ$:?Z$:IF Z$="Q$"N"Q$"THEN UD=1:GOTO "LS
1093 LN = LN + 1
1096  PRINT LN"RETURN"
1099 LN = LN + 1
1102  PRINT LN":"
1105  RETURN
1150  REM   *** GET LINE NUMBER ***

1160  HOME : INPUT "STARTING LINE NO ";LN: RETURN
1170  REM   *** LIST ROUTINE ***

1180  PRINT LN"HOME"
1190 LN = LN + 1
```

(continued)

```
1200  READ D$,L,H$
1210  PRINT LN"PRINT"Q$H$Q$","D$
1220 RL = RL + L
1230  IF RL = R THEN  GOTO 1250
1240  GOTO 1190
1250 LN = LN + 1
1260  PRINT LN"RETURN"
1270  RETURN
1300  REM *** BUILD STRING SORT ***
1315  PRINT CD$
1320  HOME : PRINT "NO LINE NO CHOICE": PRINT
1325  FOR Z = 1 TO 1000: NEXT
1327  POKE 33,33
1330  PRINT CD$"WRITE "FF$
1340  LIST 6000 - 6100
1350  PRINT CD$
1355  TEXT
1360  RETURN
1400  REM *** BUILD NUMERIC SORT ***
1415  PRINT CD$
1420  HOME : PRINT "NO LINE NO CHOICE": PRINT
1425  FOR Z = 1 TO 1000: NEXT
1427  POKE 33,33
1430  PRINT CD$"WRITE "FF$
1440  LIST 7000 - 7100
1450  PRINT CD$
1455  TEXT
1460  RETURN
1500  REM *** BUILD PRINT USING ***
1515  PRINT CD$
1520  HOME : PRINT "NO LINE NO CHOICE": PRINT
1525  FOR Z = 1 TO 1000: NEXT
1527  POKE 33,33
1530  PRINT CD$"WRITE "FF$
1540  LIST 8000 - 8100
1550  PRINT CD$
1555  TEXT
1560  RETURN
2000  REM *** DATA FOLLOWS ***
2001  DATA   TEST,20
2002  DATA   A$,10,FLD1,B,5,FLD2,C$,5,FLD3
2003  DATA   ADDR,74
2004  DATA  NA$,20,NAME, AD$,20,ADDRESS
2005  DATA  CT$,15,CITY
2006  DATA  ST$,2,STATE
2007  DATA  ZP$,5,ZIP
2008  DATA  PH$,12,PHONE
2009 : *** DATA FOLLOWS ***
2010  DATA  CHECK,37
2011  DATA  N1,4,NUMBER
2012  DATA  T1$,20,CHECK TO
2013  DATA  C1$,5,CATAGORY
2014  DATA  A1,8,AMOUNT
6000  REM  *** STRING SORT ***
6001  REM  SORTS ARRAY SR$ FROM 1 TO SY

6002  IF S1 GOTO 6009: REM  BYPASS SWAP SETUP
6003 S1 = 1: REM   SET BYPASS SWITCH
6004  REM      STRING SWAP
6005  REM      BY RANDY WIGGINGTON
6006 HEX$ = "3B0:20 E3 DF 85 85 84 86 20 BE DE 20 E3 DF A0 02
B1 85 48 B1 83 91 85 68 91 83 88 10 F3 60 N D823G": REM  ASSY

6007  FOR I = 1 TO  LEN (HEX$): POKE 511 + I, ASC ( MID$ (HEX$
,I,1)) + 128: NEXT : POKE 72,0: CALL  - 144:

      REM  DUMP TO MEMO R Y

6008  POKE 1013,76: POKE 1014,176: POKE 1015,3:
      REM  SET & VECTOR
6009  REM  ** SORT **
6010 SM = SY
6011 SM =  INT (SM / 2)
6012  IF SM = 0 THEN  RETURN
6013 SK = SY - SM
6014 SJ = 1
6015 SI = SJ
6016 SL = SI + SM
6017  IF SR$(SI) <  = SR$(SL) THEN  GOTO 6021
6018  & SR$(SI),SR$(SL)
6019 SI = SI - SM
6020  IF SI >  = 1 THEN  GOTO 6016
6021 SJ = SJ + 1
6022  IF SJ > SK THEN  GOTO 6011
6023  GOTO 6015
7000  REM  *** SORT ROUTINE ***

7001  REM SORTS NUMERIC ARRAY "R"
7002  REM    FROM 1 TO SY
7003 SM = SY
7004 SM =  INT (SM / 2)
7005  IF SM = 0 THEN  RETURN
7006 SK = SY - SM
7007 SJ = 1
7008 SI = SJ
7009 SL = SI + SM
7010  IF SR(SI) <  = SR(SL) THEN  GOTO 7016
7011 ST = SR(SI)
7012 SR(SI) = SR(SL)
7013 SR(SL) = ST
7014 SI = SI - SM
7015  IF SI >  = 1 THEN  GOTO 7009
7016 SJ = SJ + 1
7017  IF SJ > SK THEN  GOTO 7004
7018  GOTO 7008
7019  REM    ** END OF SORT RTN **

8000  REM
          *** PRINT USING ***

8001  REM FOR $ AND CENTS FORMAT
8002  REM    P= VALUE, PL= LENGTH
8003  REM   P$=FIELD ACTUALLY PRINTED
8004 P$ =  STR$ ( INT ((P + .005) * 100))
8005  IF  LEN (P$) < 3 THEN P$ =  LEFT$ ("000",
      (3 -  LEN (P$))) + P$
8006 P$ =  LEFT$ (P$,( LEN (P$) - 2)) + "." +  RIGHT$
      (P$,2)
8007 P$ =  RIGHT$ ("            " + P$,PL)
8008  RETURN
```

ASCII Dump for the Apple

This article presents an assembly language program that extends the "Examine Memory" routine in the Apple monitor. The use of the program is identical to that of the monitor routine except that memory contents are displayed as ASCII characters as well as hex numbers.

Robert F. Zant
P.O. Box 13006
Denton, Texas 76203

The Apple II monitor contains very handy routines for printing the contents of RAM locations. The 'Examine Memory' routine along with the monitor's disassembler makes it possible to view the contents of main storage, respectively, as hex numbers and as instructions. The missing capability is the ability to display the contents as ASCII characters. The following assembly language program augments the monitor routines to display the hex and ASCII representation of storage contents.

The routine was assembled with the S-C Assembler to load into the top position of the line input area ($240-2FC). When run, the routine responds with the right parenthesis, ')', as the prompt character. The range of locations to be displayed is specified in the same manner as with the monitor routine (i.e., hex addresses separated by a period). For example, 801.10CF would display the locations of 801 to 10CF inclusive. The routine is exited by entering a CTRL-@.

Editor's note: The .n labels (n is a digit) are local labels. If your assembler does not support them, use distinct and unique lables instead.

Robert Zant is a Professor of Information Systems at North Texas State University. He has experience in the computer industry as a programmer, analyst, teacher, and consultant.

```
                    *       ASCII DUMP
                    *
                    *       ROBERT F ZANT
                    *
                    *---------------------------------
                            .OR $0240
0240- 20 3A FF      START   JSR $FF3A     BELL
0243- A9 A9                 LDA #$A9      ')' PROMPT
0245- 85 33                 STA $33
0247- 20 67 FD      READ    JSR $FD67     READ LINE
024A- 20 C7 FF              JSR $FFC7     CLEAR MODE, Y=0
024D- 20 A7 FF      NEXT    JSR $FFA7     GET ITEM
0250- 84 34                 STY $34
0252- C9 B9                 CMP #$B9      CTRL-@?
0254- F0 20                 BEQ .9
0256- C9 99                 CMP #$99      BLANK?
0258- F0 0E                 BEQ .1
025A- C9 A7                 CMP #$A7      '.'?
025C- F0 12                 BEQ .3
025E- C9 C6                 CMP #$C6      CR?
0260- D0 DE                 BNE START
0262- 20 79 02              JSR EXAM
0265- 4C 47 02              JMP READ      NEXT LINE
                    *---------------------------------
0268- 20 79 02      .1      JSR EXAM      PRINT THEM
026B- A4 34         .2      LDY $34
026D- 4C 4D 02              JMP NEXT
0270- 20 18 FE      .3      JSR $FE18     SET MODE
0273- 4C 6B 02              JMP .2
0276- 4C D0 03      .9      JMP $03D0     RETURN TO BASIC
                    *---------------------------------
                    *---------------------------------
0279- A2 00         EXAM    LDX #$00      CLEAR MODE
027B- 86 31                 STX $31
027D- 88                    DEY           JUST A BLANK?
027E- D0 0A                 BNE PRTADR    NO
0280- A5 3C                 LDA $3C       YES
0282- 09 07                 ORA #$07      PRINT NEXT
0284- 85 3E                 STA $3E         EIGHT BYTES
0286- A5 3D                 LDA $3D
0288- 85 3F                 STA $3F
                    *---------------------------------
028A- A4 3D         PRTADR  LDY $3D
028C- A6 3C                 LDX $3C
028E- 86 40                 STX $40
0290- 84 41                 STY $41
0292- 20 8E FD              JSR $FD8E     CR OUT
0295- 20 40 F9              JSR $F940     PRINT X,Y
0298- A0 00                 LDY #$00
029A- A9 AD                 LDA #$AD      '-'
029C- 20 ED FD              JSR $FDED
029F- A9 A0         PRTHEX  LDA #$A0
02A1- 20 ED FD              JSR $FDED     PRINT BLANK
02A4- B1 3C                 LDA ($3C),Y
02A6- 20 DA FD              JSR $FDDA     PRINT HEX
02A9- 20 BA FC              JSR $FCBA     INC 3C.3D
02AC- B0 0C                 BCS .9        < 3E.3F?
02AE- A5 3C                 LDA $3C       PRINTED ALL
02B0- 29 07                 AND #$07        EIGHT?
02B2- D0 EB                 BNE PRTHEX    NO!
02B4- 20 BE 02              JSR ASCII       YES
```

```
02B7- 4C 8A 02          JMP PRTADR
02BA- 20 BE 02 .9       JSR ASCII
02BD- 60        RTN     RTS
                *-----------------------------
                *-----------------------------
02BE- A9 A0     ASCII   LDA #$A0
02C0- 20 ED FD          JSR $FDED       PRINT BLANK
02C3- A9 1E             LDA #$1E
02C5- C5 24             CMP $24         COL 32?
02C7- B0 F5             BCS ASCII
02C9- A0 00             LDY #$00
02CB- A9 07             LDA #$07
02CD- 25 40             AND $40         CALCULATE
02CF- 49 FF             EOR #$FF          COUNT
02D1- 18                CLC
02D2- 69 08             ADC #$08
02D4- 48        .0      PHA             SAVE COUNT
02D5- B1 40             LDA ($40),Y     GET CHAR
02D7- 29 7F             AND #$7F        CLEAR HIGH BIT
02D9- C9 20             CMP #$20        SPECIAL CHAR?
02DB- 90 04             BCC .1            YES
02DD- 09 80             ORA #$80        CAPS
02DF- D0 02             BNE .2
02E1- A9 A0     .1      LDA #$A0        BLANK IT!
02E3- 20 ED FD .2       JSR $FDED       PRINT IT
02E6- A5 40             LDA $40         NEXT ADDR
02E8- C5 3E             CMP $3E
02EA- A5 41             LDA $41
02EC- E5 3F             SBC $3F
02EE- E6 40             INC $40
02F0- D0 02             BNE .3
02F2- E6 41             INC $41
02F4- 68        .3      PLA             GET COUNT
02F5- B0 05             BCS .9
02F7- 38                SEC
02F8- E9 01             SBC #$01        PRINTED ALL 8?
02FA- 10 D8             BPL .0            NO
02FC- 60        .9      RTS               YES
```

Apple Bits, Part 3

In this third and final part of the series, the author presents some applications, including giant letters and animation.

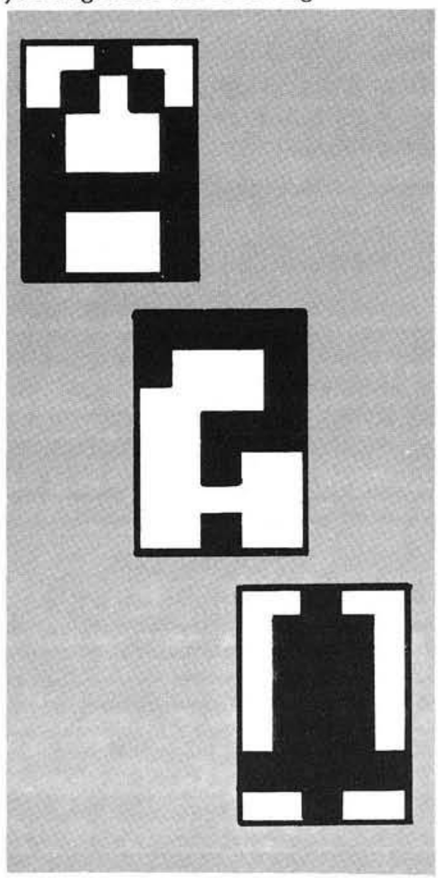
Richard Vile, Jr.
3467 Yellowstone Drive
Ann Arbor, Michigan 48105

This article discusses the use of the machine language driver program (Part 1 of the series — September 1981) and the Pattern Maker program (Part 2 of the series — October 1981) in the creation of "animations" for the low-resolution screen. The major example considered is a program for converting the Lo-Res screen into a terminal that displays "giant" letters and other patterns. (Note: the information displayed is *not* passed on as commands to BASIC, although with some effort that could be accomplished.)

Giant Letters — The Patterns

The first step in creating any Apple Bits application is to design a set of patterns. In this case, the patterns will be letters and other characters that can be plotted on the screen when their associated keys are struck. The pattern size that works with the Integer BASIC program to be presented is 5 × 7. By suitable modifications to that program (left as an exercise to the reader), other character pattern sizes can be used as well.

To design your character set, run the Pattern Maker program. Following the instructions given in Part 2, create patterns for each character on the Apple keyboard. You can also create patterns for keys which do not produce displayable graphics (control keys). The Pattern Maker will accept control keys as well as normal keys. For example, for the keys "A," "?," and "↑G" (Control-G), you might use the following:



When you are satisfied with your results, stop the Pattern Maker by typing "Q" or "QUIT" and then BSAVE your patterns. This takes a little calculating. Suppose your pattern table was started at location 3072 (decimal, or $C00 hex) and the patterns are, of course, 5 × 7 in size. To store the patterns for the characters Control-A through Z, you would consume 5 × 96, or 480 bytes. Thus,

```
BSAVE LETTERS,A3072,L480
```

would do. Since I'm lazy and don't like to figure out exactly how much I need, I normally just reserve all the space from $C00 to $FFF for patterns — that is more than enough, even for 96 patterns of 8 × 8 characters. I simply use the command

```
BSAVE LETTERS,A$C00,L$3FF
```

Once you have created your patterns, the program to "drive" the screen is shown in listing 1. Don't forget to set LOMEM:

```
> LOMEM: 4096
```

There are some generally useful points to note in this program. You may be able to make use of them in other programs of your own.

In lines 10 and 15:

```
10 GR : POKE -16302,0 :
   COLOR=0

15 FOR I=40 TO 47 : HLIN 0,39
   AT I : NEXT I
```

The POKE statement selects FULL SCREEN graphics. This causes any information already displayed on the bottom four lines of the screen to suddenly change to "living color." Line 15 blackens the bottom four lines again.

In line 12:

```
12 POKE 32,0 : POKE 33,40 :
   POKE 34,0 : POKE 35,24
```

These statements set the "text window" back to the full screen. But why do that? This is a graphics program, right? Yes it is, but it is also a text program as well — the letters are just a bit larger than usual! So when our screen fills with our maxi-alphabetics, how do we make room for more? The answer is simple: scroll! But, you say, you can't scroll the graphics screen. Want to bet? Look at line 60:

```
60 FOR J=1 TO 4 : CALL -912
   : COLOR=0 : HLIN 0,39 AT
   47 : NEXT J
```

The routine at −912 is the normal monitor routine for text scrolling. It uses the settings of the window variables in locations 32 - 35 to determine what portion of the screen to scroll. The GR statement sets these variables so that only the bottom four lines will scroll. Our POKEs in line 12 have fooled the monitor into thinking that the whole screen should be scrolled. The Apple will then scroll the graphics display, without a whimper. Since the lines which appear at the bottom during the scrolling process will be WHITE, we use the HLIN statement to re-blacken them.

If you study the listing further, you will discover that the left and right arrow keys will function in a manner similar to their normal text interpretation. In addition, the ENTER key will cause the display to proceed to the beginning of the next "line." The ESC key functions as a "Clear Screen" key. It also causes the next character to appear at the upper left hand corner of the display. I leave it to you to dig out the details of these points.

A Random Walk

The program of listing 2 presents an animation. It causes a "little" man to walk across the screen from the lower right corner to the upper left corner. The actual path taken is different each time, consisting of a random pattern of moves to the left and/or up.

The data for the patterns of program 2 is presented in listing 3.

Computer Choo-Choo

Listing 4 moves a locomotive across the screen from right to left. The train gives off "smoke" as it goes and periodically toots its whistle. The whistle is produced by calling a routine in the Apple Programmer's Aid ROM. If you do not have this installed in your Apple, you will have to locate and remove the CALL statements in the program. They could be replaced by CALLs to your own tone-producing routine.

The data for the locomotive progam is presented in listing 5.

Notes on Implementing Animations

In both the random walk program and the locomotive program, only a small number of patterns was needed. Notice that the pattern selected for display by the programs at any given time is specified by a small positive number. For example, examine lines 535 to 540 of listing 2. The way that the

***Listing 1:* Large Letters Driver**

```
  1 KBD=-16384:CLR=-16368
  5 POKE 2048,5: POKE 2049,7
 10 GR : POKE -16302,0: COLOR=0
 12 POKE 32,0: POKE 33,40: POKE 34,0: POKE 35,24
 15 FOR I=40 TO 47: HLIN 0,39 AT I: NEXT I
 20 ROW=0:COL=0
 22 COLOR= RND (15)+1
 25 GOSUB 700
 30 POKE 36,COL: POKE 37,ROW
 35 POKE 60,(3072+5*K1) MOD 256
 40 POKE 61,(3072+5*K1)/256
 42 COLOR= RND (15)+1
 45 CALL 2058
 50 COL=COL+6: IF COL<36 THEN 25
 55 COL=0:ROW=ROW+8: IF ROW<=40 THEN 25
 60 FOR J=1 TO 4: CALL -912: COLOR=0: HLIN 0,39 AT 46:
      HLIN 0,39 AT 47: NEXT J
 65 COLOR= RND (15)+1
 70 ROW=40:COL=0: GOTO 25
700 KEY= PEEK (KBD): IF KEY<128 THEN 700
705 POKE CLR,0
710 K1=KEY-128
712 IF K1#27 THEN 718
713 COLOR=0: FOR I=0 TO 47: HLIN 0,39 AT I: NEXT I:
      COLOR= RND ( 15)+1
715 ROW=0:COL=0: GOTO 700
718 IF K1=13 THEN 785
719 IF K1=7 THEN 775
720 IF (K1#8 AND K1#21) THEN RETURN
722 IF K1#21 THEN 725
723 K1=32: RETURN
725 COL=COL-6: IF COL>=0 THEN 750
730 COL=30:ROW=ROW-8: IF ROW>=0 THEN 750
735 ROW=0:COL=0
750 COLOR=0
755 FOR J=0 TO 7
760 HLIN COL,COL+5 AT ROW+J
765 NEXT J
770 COLOR= RND (15)+1: GOTO 700
775 PRINT "";: RETURN
785 ROW=ROW+8: IF ROW>=48 THEN 790
787 COL=0: GOTO 700
790 COLOR=0
792 FOR J=1 TO 4: CALL -912
793 HLIN 0,39 AT 46: HLIN 0,39 AT 47
794 NEXT J
799 ROW=40:COL=0: COLOR= RND (15)+1: GOTO 700
```

patterns came to be associated with these numbers involves the Pattern Maker program. The control keys correspond to the numbers 1 through 26. Thus, when you use the Pattern Maker to create a set of patterns and record a particular one using, say, Control-E, then that pattern becomes the 5th pattern in the table.

To set up the address of this pattern (so the machine language driver knows which one to display), the statements in lines 536 and 537 of listing 2 would be used. These are similar to the statements appearing in lines 60 and 65 of the Fireworks Animation presented in Part 1 of the series.

Let's review the general form of the set-up instructions:

POKE 60, (TABLE + OFFSET)
MOD 256

POKE 61,(TABLE + OFFSET)
/256

where,

TABLE — represents the address in Apple II RAM of the very beginning of the Pattern Table. In all of our examples this has been 3072, decimal. However, it could be other values as well.

Note: The numbering of the entries in the table actually begins at 0. The 0th entry is inaccessible, since the Pattern Maker cannot accept a key whose character code is 0. Also, the entry in the table which corresponds to the Control-C key (number 3) will always contain "garbage." This is the reason for the IF test in line 535 of listing 2.

OFFSET —represents the distance (in bytes) from the beginning of the pattern table at which a given pattern may be found. This offset may be calculated using the formula:

OFFSET = WIDTH * KEY

where,

WIDTH — is the width of the patterns in the table.

KEY — is the number of the pattern you wish to retrieve.

(Continued on next page)

***Listing 2:* Random Walk**

```
  5 MOVE=500
 10 GR : POKE -16302,0: COLOR=0
 15 FOR I=40 TO 47: HLIN 0,39 AT I: NEXT I
 21 POKE 2048,8: POKE 2049,8
 32 POKE 36, RND (32): POKE 37, RND (40)
 35 COLOR= RND (15)+1
 40 D= RND (2)
 45 IF D#0 THEN 55
 50 DX=0:DY=-1: GOSUB MOVE: GOTO 35
 55 IF D#1 THEN 65
 60 DX=-1:DY=0: GOSUB MOVE: GOTO 35
 65 IF D#2 THEN 75
 70 DX=1:DY=0: GOSUB MOVE: GOTO 35
 75 DX=-1:DY=0: GOSUB MOVE: GOTO 35
500 COL= PEEK (36):ROW= PEEK (37)
505 COL=COL+DX: IF COL<32 THEN 510: GOSUB 600:COL=0
510 IF COL>0 THEN 515: GOSUB 600:COL=32
515 ROW=ROW+DY: IF ROW<40 THEN 520: GOSUB 600:ROW=0
520 IF ROW>0 THEN 530: GOSUB 600:ROW=40
530 POKE 36,COL: POKE 37,ROW
535 KEY= RND (5)+1: IF KEY=3 THEN 535
536 POKE 61,(3072+8*KEY)/256
537 POKE 60,(3072+8*KEY) MOD 256
540 CALL 2058
545 FOR TIME=1 TO 25: NEXT TIME
555 COLOR=0
560 HLIN COL,COL+7 AT ROW+7
562 VLIN ROW,ROW+7 AT COL+7
570 RETURN
600 COLOR=0: FOR I=0 TO 7: HLIN COL,COL+7 AT ROW+I: NEXT I
610 RETURN
```

***Listing 3:* Little Men**

```
0C00- FF FF FF 15 1F 7E 7C 78
0C08- 84 48 2B 3F 4B 88 10 00
0C10- 00 98 4B 3F 2B 48 84 00
0C18- 48 77 41 5D 41 77 78 3C
0C20- 00 98 CB 3F 6B C8 04 00
0C28- 00 10 88 6B 1F 2B CC 80
0C30- 5D 7F 08 1C 2A 49 08 01
0C38- 0F 08 78 40 6C 64 7C 64
0C40- 6C 78 48 7F 09 0F 7F 41
0C48- 49 41 7F 59 49 6B 49 4D
0C50- 7F 49 6B 49 7F 7F 49 7F
0C58- 49 7F 77 41 77 41 77 7F
0C60- 49 00 49 7F 22 55 49 55
0C68- 22 10 18 1C 18 10 41 63
0C70- 77 63 41 7F 3E 1C 08 00
0C78- 00 08 1C 3E 7F 08 1C 3E
```

***Listing 4:* Locomotive Program**

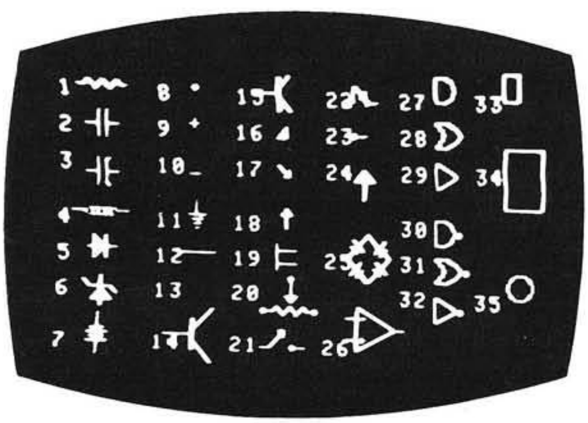
```
  1 MUSIC=-10473
  2 POKE 767,40: POKE 766,30: POKE 765,32
  5 MOVE=500:SMOKE=22
 10 GR : POKE -16302,0: COLOR=0
 15 FOR I=40 TO 47: HLIN 0,39 AT I: NEXT I
 21 POKE 2048,8: POKE 2049,8
 32 POKE 36,20: POKE 37,24
 33 CC= RND (15)+1
 35 COLOR=CC
 40 D=1
 50 DX=-1:DY=0: GOSUB MOVE
 55 GOTO 35
500 COL= PEEK (36):ROW= PEEK (37)
505 COL=COL+DX: IF COL<32 THEN 510: GOSUB 600:COL=0
510 IF COL>0 THEN 515: GOSUB 600:COL=32:CC= RND (15)+1
515 REM
530 POKE 36,COL: POKE 37,ROW
535 KEY=1
536 POKE 61,(3072+8*KEY)/256
537 POKE 60,(3072+8*KEY) MOD 256
540 CALL 2058
542 GOSUB 800
545 FOR TIME=1 TO 25: NEXT TIME
550 IF RND (25)=0 THEN GOSUB 700
555 COLOR=0
560 HLIN COL,COL+7 AT ROW+7
562 VLIN ROW,ROW+7 AT COL+7
570 RETURN
600 COLOR=0: FOR I=0 TO 7: HLIN COL,COL+7 AT ROW+I: NEXT I
610 RETURN
700 CALL MUSIC
705 POKE 766,100: FOR I=1 TO 50: NEXT I
710 CALL MUSIC: POKE 766,30: RETURN
800 PLOT COL+1,SMOKE
810 COLOR=0: PLOT COL+2,SMOKE+1
815 IF SMOKE=22 THEN PLOT COL+2,1
818 IF COL=32 THEN PLOT 2,SMOKE+1
820 SMOKE=SMOKE-1
830 IF SMOKE=0 THEN SMOKE=22
840 RETURN
```

***Listing 5:* Train**

```
0C00- FF FF FF 15 1F 7E 7C 78
0C08- FC BF FC 3C FF B9 F9 1F
0C10- 7D FD F0 78 70 FE F2 3E
0C18- 48 77 41 5D 41 77 78 3C
```

MICRO™

VersaWriter



What is VersaWriter?

- VersaWriter is an inexpensive drawing tablet for the APPLE II that lets you trace a picture and have it appear on TV display.
- VersaWriter is a comprehensive software drawing package which lets you color in drawings with over **100** different colors.
- VersaWriter is a shape compiler that converts anything on the screen automatically into a standard shape table.
- VersaWriter is a text writer for labeling pictures with text in six colors and five sizes. Use English or Greek, upper or lower case letters.
- VersaWriter is much more! Draw with brush, create schematic drawings, compute area and distance, edit pictures, save, recall and more.

VersaWriter requires ROM APPLESOFT and 48K memory.

$299 Suggested Retail

UNIQUE OFFER

Send us YOUR disk and $1. We will promptly return the disk with a slide package of 10 color pictures drawn with VersaWriter.

☐ Enclosed is $1 and my disk. Send me the slide package.

☐ Send more information including VersaWriter dealers in my area.

DEALER INQUIRIES INVITED.

NAME

ADDRESS

CITY STATE ZIP

Send To: Versa Computing, Inc. • 887 Conestoga Circle • Newbury Park, CA 91320 • (805) 498-1956

MICRO Hardware Catalog

Name: **Dynamic Memory Module, CMS 6505**
Memory: 65K × 9 Dynamic RAM
Description: 65K × 9 dynamic memory module, addressable in segments (4K increments), parity generation and check, on-board refresh, write protect, over voltage and reverse polarity protection, selectable speeds, directly compatible with 6500/6800 families. 6″ × 9.75″ module, uses only 5 watts power.
Price: $526 in single piece quantity
Available: General Micro Systems
1320 Chaffey Ct.
Ontario, CA 91762
(714) 621-7532

Name: **Model Q160**
Memory: Standard 2K buffer memory; 4K option
Description: The Model Q160 is the printing mechanism used in Computer Devices' Series 2000 portable computer terminals and printers. So lightweight — weighing only 3.5 lbs. — it's ideal for OEM use. Its special 1 × 11 dot printhead, developed from the most advanced thin film technology, generates 5 × 9 dot matrix characters with true upper/lower case letters and the underscore/overscore. The Q160 can give true 120 cps throughput because it's designed to print bidirectionally at 160 cps. It also has an 80/132-column selectability built in.
Price: $995.00
Available: Computer Devices Inc.
(Early 4th quarter; 60 days ARO)
25 North Avenue
Burlington, MA 01803
(800) 225-1230

Name: **EP12 Interface for MX-80 and Apple**
System: Apple II, Apple III with SOS
Memory: Any size
Language: BASIC, Assembler, Pascal, others
Hardware: Epson MX80, MX100 printers
Description: The EP12 interface features a wide variety of text options plus ability to print anything you see on the screen — text or graphics. HalfTone™ mode lets you print in shades of gray. SPECIAL CHARACTER™ mode lets you create your own print symbols. SuperRes™ graphics gives you 960 × 792 point plotting.
Price: $165.00 includes cable, manual, print sample disk.
Available: Interactive Structures, Inc.
112 Bala Avenue
P.O. Box 404
Bala Cynwyd, PA 19004
or your Apple dealer

Name: **RS-232 Interface Kit**
System: Ohio Scientific Superboard II or C1P
Description: This kit, offered by Dee Products, contains all the hardware needed to add RS-232 input and output capabilities to either the Superboard II or C1P. When these computers were produced, Ohio Scientific included the printed etches for a RS-232 interface, but not the components to use it. Adding this modification is simple and straightforward by following our well-illustrated step-by-step instructions. Also included at this price are the details for hooking up the popular Radio Shack Quick-Printer II, and interfacing with modems. Printed circuit quality solder included.
Price: $9.95 ppd.
Available: Dee Products
150 Birchwood Road
Lake Marian, IL 60110

Name: **Soundchaser**
System: Apple II Plus
Memory: 48K
Language: Applesoft, Assembly
Hardware: 3 Voice Synthesizer Cards, Music Keyboard
Description: *Soundchaser* transforms Apple II into a dynamic polyphonic synthesizer and sequencer. The music keyboard allows live entry of compositions which can be recorded with one sound and accompanied live with another sound. Sounds can be constructed by drawing waveforms and envelopes on the CRT with game paddles or a joy stick. The Voice cards incorporate 3 state of the art analog filters, oscillators and amplifiers for dynamic natural sounds. All sounds and sequences can be stored on disk for future use. The system is menu driven for quick and easy access to the subsystems.
Price: $1350.00 for 6-voice Soundchaser, keyboard and software
Available: Passport Designs, Inc.
785 Main Street, Suite E
Half Moon Bay, CA 94019

Name: **Super Music Synthesizer**
System: Apple
Description: Complete 16-voice music synthesizer on one card. Program music with our "Compose" software. Our manual shows you how, step by step. The Hi-Res screen shows what you've entered in standard sheet music format. Four white noise generators (great for sound effects). Plays music in true stereo as well as true discrete quadrophonic. Envelop control (volume). Will play songs written for Alf synthesizer. (Alf software will not take advantage of all the features of this board. Their software sounds the same on our synthesizer.)
Price: $159.00 — Texas residents add 5% sales tax
Available: Applied Engineering
P.O. Box 470301
Dallas, Texas 75247
(214) 492-2027

Name: **W7AAY RAE to KMMM Interface**
System: Synertek SYM-1 with 1 or 2 floppy disks
Software: Synertek's RAE and Wilserv Industries' KMMM disk operating system.
Description: Interfaces RAE to the KMMM DOS and provides the following commands for use from within RAE: Save file, Update file, Load file, Append file, Delete file, display disk volume contents, and exit to KMMM monitor. Supports assemblies continued on disk. Fully documented RAE source code supplied on 5¼″ disk or cassette tape. Specify which.
Price: $15.00 ppd. USA
Available: John M. Blalock
Blalock & Associates
P.O. Box 39356
Phoenix, AZ 85069

MICRO

GET FREE SOFTWARE FOR YOUR APPLE!

HOW? JUST ORDER ONY OF THE ITEMS BELOW, AND SELECT YOUR FREE SOFTWARE FROM THE BONUS SOFTWARE SECTION, USING THE FOLLOWING RULE: FOR THE FIRST $100.00 WORTH OF MERCHANDISE ORDERED TAKE 1 ITEM; FOR THE NEXT $200.00 WORTH OF MERCHANDISE ORDERED TAKE ANOTHER ITEM; FOR THE NEXT $300.00 TAKE A THIRD ITEM, ETC. ALL AT NO COST.

HARDWARE BY APPLE

APPLE II PLUS, 48k	1199
DISK DRIVE+CONTROLLER (3.3)	535
DISK DRIVE only	455
Language System w. Pascal	397
Silentype Printer & Interface	360
Integer or Applesoft Firmware Card	159
Graphics Tablet	645
Parallel Printer Interface Card	149
Hi-Speed Serial Card	155
Centronics Parallel Intfce.	175

HARDWARE BY OTHERS

HAYES MICROMODEM II	300
HAYES SMART MODEM	239
HAYES S100 MODEM	339
VIDEX VIDEOTERM 80 W. GRAPHICS	275
MICROSOFT Z80 SOFTCARD	299
MICROSOFT 16K RAMCARD	159
CORVUS 10MB HARD DISK	4750
SSM A10 SERIAL/PARALLEL A&T	189
MICRO-SCI Disk & Controller	495
TYMAC DOUBLE DOS 3.2/3.3	35

VIDEO MONITORS

Leedex-Video-100 12" B&W w/Cable	139
Leedex 12" Green w/Cable	165
Leedex 13" COLOR MONITOR & Cable	399
SUP-R-TERM RF MODULATOR	29

HARDWARE BY MOUNTAIN COMPUTER

Clock/Calendar Card	239
A/D & D/A Interface	319
Expansion Chassis	555
ROMplus Card	135
Mark Sense Card Reader	995
CPS Multifunction Bd.	239

SOFTWARE FOR APPLE

APPLE FORTRAN	159
APPLE PILOT	125
DOS 3.3	50
DOS TOOL KIT	65
APPLE PLOT	59
D. J. REPORTER	45
D. J. NEWS	45
PORTFOLIO	45
SHELL GAMES	25
ELEMENTARY DEAR APPLE	25

SOFTWARE BY OTHERS

APPLE FORTRAN by MICROSOFT	159
APPLE BASIC COMPILER by MICROSOFT	315
APPLE COBOL by MICROSOFT	599
VISICALC 3.3	169
VISIPLOT	155
VISIPLOT/VISITREND	199
VISIDEX	169
CCA DATA MGT.	79
DB MASTER by STONEWARE	189
DATACAPTURE 4.0	55
Z-TERM	65
ON-LINE APPLESOFT COMPILER	89

SOFTWARE BY PEACHTREE

GEN. LEDGER	219
A/R	219
A/P	219
PAYROLL	219
INVENTORY	219
MAIL LIST	219

WORD PROCESSING SOFTWARE FOR APPLE

PEN-ULTIMATE	235
WORD STAR	245
EZ WRITER Prof. Sys.	195
EZ WRITER	89
MUSE SUPER TEXT II	139
APPLE-WRITER	69
PROGRAMMA APPLE PIE 2.0	110
MAGIC WAND	345
WORDPOWER	50

EPSON PRINTERS

MX - 70 w/Graftrax	415
MX - 80	515
MX - 80 FT	615
MX - 80 w. GRAFTRAX	575
MX - 80 FT w. GRAFTRAX	675
MX - 100 FT w. GRAFTRAX	775
APPLE PAR. INTFCE (for all Epson)	75
MX-70/80 FRICTION FEED Adaptor	75

OTHER PRINTERS

IDS 445 w. GRAPHICS + 2K Buff.	750
IDS 460 w. GRAPHICS	899
IDS 460	825
IDS 560	1099
IDS 560 w. GRAPHICS	1150
CENTRONICS 737	699
CENTRONICS 739	799
CCS Centronics Par. Intfce & Cable	135
NEC SPINWRITER 5510 RO	2795
C. ITOH 25 CPS DAISYWHEEL	1750
C. ITOH 45 CPS DAISY WHEEL	2025
WATANABE MI-PLOT PLOTTER	1150
DIABLO 630 DAISY w. P. Wheel & Rib.	2350

BONUS SOFTWARE SECTION!

Let us acquaint you with MESSAGE-MAKING SOFTWARE. Just place the disk in the APPLE, enter the text, and colorful, dynamic messages appear on the screens of TV sets connected to the computer. Use the software to broadcast messages on TV screens in schools, hospitals, factories, store window, exhibit booths, etc. The following program is our latest release:

SUPER MESSAGE: Creates messages in full-page "chunks". Each message allows statements of mixed typestyles, typesizes and colors, in mixed upper and lower case. Styles range from regular APPLE characters, up to double-size, double-width characters with a heavy, bold font. Six colors may be used for each different typestyle. Vertical and horizontal centering are available, and word-wrap is automatic. Users can chain pages together to make multi-page messages. Pages can be advanced manually or automatically. Multi-page messages can be stored to disc or recalled instantly.
REQUIRES 48K & ROM APPLESOFT .... $50.

APPLE PLOTS YOUR DATA & KEEPS YOUR RECORDS TOO! APPLE DATA GRAPH 2.1: Plots up to 3 superimposed curves on the Hi-res Screen both X & Y axes dimensioned. Each curve consists of up to 120 pieces of data. Graphs can be stored to disc and recalled immediately for updating. Up to 100 graphs can be stored on the same disc. Great for Stock-market Charting. Business Management, and Classroom instruction!
REQUIRES 48K & ROM APPLESOFT $36

APPLE RECORD MANAGER: Allows complete files to be brought into memory so that record searches and manipulations are instantaneous. Records within any file can contain up to 20 fields, with user-defined headings. Information can be string or numeric. User's can browse thru files using page-forward, page-backward or random-search commands. Records can easily be searched, altered or sorted at will. Files can be stored on the same drive as the master program, or on another, if a second drive is available. Records of files can be printed, if desired. Additional modules coming are a STATISTICS INTERFACE, CHECKBOOK, MAILING LIST & DATA-ENTRY.
REQUIRES 48K & ROM APPLESOFT $40

APPLE LITERATURE DATABASE: allows rapid retrieval (via keywords) of references from total APPLE literature thru 1980, on 5.25" disk. Each entry in the data base consists of the article, author-name, periodical-name, date of issue, & page nos. The database is intended to support large magazine files which would require lengthy manual searching to recover information. Annual updates will be available.
REQUIRES 48K, ROM APPLESOFT $60

WORDPOWER: is a simple, powerful, low cost, line-oriented word-processor program. It offers a fast machine language FIND & REPLACE. Text can be listed to screen or printer, with or without line-numbers. Lower-case adaptors are supported. You can merge files, move groups of lines, and easily add, change, or delete lines. WORDPOWER can be used to create and maintain EXEC files. It can also be used as a rapid, unstructured, information-storage and retrieval system via its rapid search capabilities.
REQUIRES: 48K, ROM APPLESOFT $50

LABELMAKER: allows users to quickly create address labels. A given label may be generated in any quantity from 1 to 32767. Space is allowed on labels for a personal and company name, but the space is automatically closed up if only a personal name is entered. Space is also allowed for foreign countries. The program can also generate labels for price-tags, part numbers and mail-messages such as "RUSH", "FRAGILE", etc. A self-incrementing feature allows theatre-tickets to be produced, with a date, and numbers running from a000 to z999. An editor is provided for editing labels prior to printing. All labels may be saved to disk for instant recall.
REQUIRES 48K & ROM APPLESOFT $35

TO ORDER: Use phone or mail. We accept VISA, MASTERCARD, COD's, personal checks & money orders. Add 4% for credit card. Customer pays handling on COD orders, Foreign orders must be in American Dollars & include 10% for handling. Connecticut residents add 7.5% sales tax.

BANKAMERICARD VISA

master charge

CONNECTICUT INFORMATION SYSTEMS CO.
218 Huntington Road, Bridgeport, CT 06608 (203) 579-0472

MICRO Software Catalog

Name: **Engineering Software Library**
System: Apple II+, DOS 3.3
Memory: 48K
Language: Applesoft BASIC
Hardware: Disk drive, printer optional
Description: The *Engineering Software Library* is composed of a number of programs for the working engineer and is aimed at a void that exists in software for micros. Examples of major programs are: Truss and Linkage Analysis; Beam Analysis with Diagrams; Linear Natural Frequencies; Torsional Natural Frequencies; Rubber Element Design; Bolted Joint Design. Each of the above is available separately. This is a continually growing library developed as part of our consulting practice. All programs are well prompted and written in easily modified BASIC.
Price: $40.00 per program (disk or listing only), includes disk and documentation
Author: James R. Sturges & Assoc.
Available: Engineering Software
104 E, Queenwood Rd.
Suite 2
Morton, IL 61550

Name: **Utilities Disk for OS 65D**
System: OSI Challenger (C2 and C3 series)
Memory: 32K or 48K
Language: BASIC under OS 65D
Hardware: Disk drive, CRT, optional printer
Description: Contains three useful programs. 1. A re-sequencer which renumbers all or part of BASIC programs, correcting all references to statement numbers; 2. A disassembler of machine code written in BASIC, with the ability to disassemble machine code linked to BASIC programs or portions of BASIC itself; and 3. A number converter handling decimal, hexadecimal, octal, binary, and ASCII conversions.
Price: $30.00 for 8" disk and documentation, ppd.
Author: Mike Anderson
Available: Responsive Computer Technology, Inc.
P.O. Box 719
Silver Spring, MD 20901

Name: **Versacalc™**
System: Apple II or II Plus
Memory: 32K
Hardware: One disk drive and Visicalc™
Description: This enhancement to Visicalc gives you vast expansion of Visicalc's capabilities by allowing sorting of Visicalc display screens. It also shows how to do conditional testing, to display an error message if a value does not conform to requirements, to automatically execute a string of up to 255 Visicalc commands with only 4 keystrokes, to change columns to rows and vice-versa, and to blank large areas of the screen to ready it for new data while preserving previously computed results. Versacalc will display a complete catalog of your data disk on the screen. Versacalc is especially useful for developing interactive commercial programs and training programs for new users, and finally 'protecting' those programs.
Price: $100.00
Available: Aurora Systems, Inc.
2040 E. Washington Ave.
Madison, Wisconsin
53704

Name: **Action Sounds and Hi-Res Scrolling**
System: Apple II
Memory: 48K
Language: Applesoft, Machine Language, and Textfiles of Assembly Language EXECable by LISA
Hardware: Apple II Plus, Disk II
Description: Contains 31 sound effects in both binary files and LISA assembly-textfiles, mostly for space or combat games. Gives modification instructions. Contains 6 Hi-Res scrolling programs, either side with/without wraparound, up 8 or 64 lines. Includes our dynamic *Superfont* program with 9 sizes and 8 styles of large typeable keyboard characters. Saves. Complete instructions to use any machine language sound effect in your programs. Unique! This disk is totally *full*.
Price: $15.95 includes disk with instructions.
Author: Avant-Garde Creations
Available: Avant-Garde Creations
P.O. Box 30161
Dept. MC
Eugene, Oregon 97403

Name: **New General Ledger**
System: Apple II
Memory: 48K
Language: Applesoft or Language System
Hardware: Dual 5" drives, any 80-column printer
Description: Based on our standard G/L, this new system can be used alone or integrated with other accounting software. It features extensive error checking and data entry prompting, departmentalizing, budgeting, and thorough audit trails. User has complete freedom in formatting reports and defining chart of accounts. Sensitive data is protected from unauthorized personnel and operator error. Clear, concise documentation included. This highly recommended system combines flexibility, efficiency, and smooth performance.
Author: David McFarling
Available: Small Business Computer Systems
4140 Greenwood
Lincoln, Nebraska 68504
(402) 467-1878

Name: **Star Warrior**
System: Atari 400 or 800
Memory: 32K
Language: BASIC
Hardware: Atari 400/800, cassette or disk drive
Description: The player must take on an entire planetary force of storm troopers of the Stellar Union, armed with nine types of military vehicles — alone. He can walk, jump, or even fly over swamps, forests and mountains. In addition to several suits of armor, he has a choice of two scenarios, 19 command options, and five levels of skill, combined with six different sounds and a revolutionary graphics display.
Price: $39.95 includes disk or cassette, rule book, command summary card, and special instructions.
Author: Automated Simulations, Incorporated
Available: Automated Simulations, Incorporated
P.O. Box 4247
Mountain View, CA
94040
or local computer stores

Name: **Chart-Master Business Graphics Software**
System: Apple II or III
Memory: 48K of RAM
Language: Applesoft BASIC
Hardware: Hewlett-Packard H-P7225A/B Plotter
Description: Chart-Master allows the Apple II or III to drive a Hewlett-Packard plotter to produce bar, line, pie, and scatter charts in up to 10

colors. Charts are easily and quickly created, edited, stored, and plotted through this interactive, menu-driven program. Variety of options include 9 hatching patterns, fastplot selection, and ability to interface with Visicalc.

Price: $375 suggested retail price includes two diskettes and a User's Manual.
Author: Sean O'Connor, Jon Siegel
Available: Decision Resources
44 White Birch Road
Weston, CT 06883

Name: **Ultra Hi-Res Graphics**
System: Apple II or Apple II +
Memory: 48K
Hardware: Paper Tiger (IDS) 460G/560G or 440G/445G printer, 1 drive, 3.3
Description: A plotting program designed to take full advantage of the high-resolution capabilities of the IDS printers. The program first writes to the disk and dumps from disk to printer without being restricted by Apple's 280 × 192 resolution. Results in smoother curves and diagonal lines plus a larger picture.
Price: $49.95 includes disk and full documentation
Available: Computer Station
11610 Page Service Dr.
St. Louis, MO 63141
(314) 432-7019

Name: **Tabula RASA**
System: 6809 with Flex
Memory: 56K
Language: TSC Extended BASIC
Hardware: 6809
Description: Electronic spreadsheet system similar to desktop/plan, with full screen, menu-driven editing capabilities.
Price: $100.00 includes all source on disk and manual
Author: Bud Pass
Available: Computer Systems Consultants
1454 Latta Lane
Conyers, GA 30207
(404) 483-1717/4570

Name: **ZAPT**
System: Apple II or Apple II Plus
Memory: 32K
Language: Machine
Hardware: Apple II and Disk II
Description: ZAPT is a versatile utility program for displaying and altering the data on a disk. Data may be displayed or altered by specifying a track, sector, and offset, or by specifying a file name and offset. For binary files you may specify on offset or the actual assembled address. Three display modes are available: Hex and ASCII representation of the data; ASCII only; or disassembled 6502 code. Output may be directed to the screen or to a printer. Display data a line at a time, page at a time, or continuously. Copy any sector or range of sectors to any location on the same or different disk. Works with DOS 3.2 or 3.3. A valuable aid for problem diagnosis and resolution.
Price: $19.95 includes disk and documentation
Author: Andy Tuxen
Available: Andy Tuxen
4539 Andrew Street
Oshkosh, WI 54901

Name: **Transit™**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: One disk drive
Description: A versatile utility program which will convert almost any Apple II data file into an Information Master file. It lets you use data files from other software packages such as Personal Software's VisiCalc™ and High Technology's The Store Manager™ among many others. Once a file has been "TRANSITed" to Information Master, it can be sorted, searched, calculated, and printed in custom-designed reports.
Price: $50.00 alone or $189.00 packaged with Information Master
Author: Steve Williams
Available: High Technology Software Products, Inc.
P.O. Box 14665
Oklahoma City, OK 73113

Name: **DOW2000**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: Disk 3.3/3.2 with printer option
Description: Stock Market Analysis will determine price projections based on a stock's BETA coefficient or Relative Strength number and the Dow Jones Average. Projections are made as you vary the DOW (what if...) on one stock or entire portfolio with single scan, quick scan, or variable scan of values. Included is the booklet "The Art of Timing Your Stock's Next Move." Author in market 17 years and former registered investment advisor with S.E.C.
Price: $29.00 with booklet (booklet alone $6.00)
Author: [IA]:Calabrese
Available: Bit'n Pieces Series
P.O. Box 7035
Erie, PA 16510

Name: **PET Arcade**
System: Any PET/CBM
Memory: 8K
Language: BASIC and machine
Hardware: PET/CBM
Description: *Astroidz* and *Munchman* are now available for the 8K PET/CBM and will run on both old and new ROMs. *Astroidz* is based on the popular arcade game. There are huge astroidz invading the galaxy and your mission is to destroy them before they destroy you. Four levels of play from novice to expert. *Munchman* is based on the popular arcade game *Packman*. You must work your way through the computer maze as fast as you can and try to avoid Zip and Zap. Bonus points, time bonus and different levels of play.
Price: $9.95 each includes tape cassette
Author: Cliff and Nic Dudzik
Available: Computermat
Box 1664
2984 Daytona
Lake Havasu, AZ 86403

Name: **Lower Case Character Generator**
System: Apple II Plus Rev 7 or Apple II Rev 7
Memory: 16K
Description: LCCG plugs into the Apple and enables the user to generate a full lower case character set, with two-dot true descenders. This EPROM is compatible with all word processing packages that need a lower case set, including *Letter Perfect* from LJK.
Price: $34.95 includes installation manual
Author: Ken Leonhardi
Available: LJK Enterprises
P.O. Box 10827
St. Louis, MO 63129

Name: **Gorgon**
System: Apple II, Apple II Plus
Memory: 48K
Language: Machine
Hardware: Disk drive, 13- or 16-sector controller
Description: The Earth has entered a time warp... and the battle has just begun. Strange creatures are appearing and some have been reported stealing people from the surface of the Earth. As a fighter pilot you must defend the planet by destroying these creatures and saving the people who are being carried away. *Gorgon* has several different levels of play, incredible high-resolution color graphics and many other features. Keyboard control only.
Price: $39.95 includes disk and documentation
Author: NASIR (Presented by Sirius Software, Inc.)
Available: Your local Apple dealer or software store

MICRO

OSI COMPATIBLE PRODUCTS

56K 2-MHz Ultra Low Power CMOS Static Memory Board MEM-56K $850

Partially Populated Boards (Specify address locations required) **MEM-48K $750**
MEM Board uses the new 2K-Byte Wide Static RAM chips which are 2716 EPROM compatible. Any 2K byte memory segment can be populated with RAM or EPROM (or left empty for use of Address Space by another board). Fully expandable to any memory size you will ever need. No special addressing requirements, just solder in extra sockets and add memory. Also has space for a 1.75K Monitor ROM at $F800 (FC).

	MEM-32K $550
	MEM-24K $450
	MEM-16K $350
	MEM- 8K $250
	MEM- 4K $200
Extra 2K RAM Memory Chip	**$24**
Optional Parallel Printer Port	**-P $120**
Optional Calendar/Clock (Software available in EPROM)	**-T $ 25**
Both options (Disk software mods provided for use of 6522 VIA on printer).	**-PT $125**

EXAMPLE USES:

C4P & C8P: Expansion to **40K RAM** of Basic workspace.
Parallel Printer Port — Reserve Serial Port for MODEM
Space for 5.75K of **Enhanced Svstem Monitor EPROMS.**

All of this on 1 Board, using only one of your precious slots. Software for Enhanced System Monitor capabilities is continuously being developed and improved. As new EPROM Monitors are available, you may upgrade to them for any price differential plus a nominal $10 exchange fee. Another possibility is to fill any portion of the memory with Basic Programs in EPROM for **Power-on Instant Action.** This custom EPROM programming service is available at $25 per 2716 (includes EPROM). Extra copies at $15 for each EPROM.

C4P-MF & C8P-DF: Memory expansion to **48K.**
Add 4K**Memory** at $E000 for special software requirements.
Parallel Printer Interface and/or **Displaying Calendar/Clock.**
Add 1.75K **Enhanced System Monitor ROM.**

C3: Up to **56K of Memory** Expansion — can be addressed for **Multiuser.**
(Optionally, each user can have his own **Dedicated Printer Port).**
Add **Enhanced Monitor ROM** with Calendar/Clock software, warm start and Hard Disk Boot.

IEEE-488 INTERFACES AND SOFTWARE:

The General Purpose Instrumentation Bus (GPIB) Controller interface is available for all OSI Computers. Machine code GPIB Drivers are linked to Basic to provide easy control of IEEE-488 instruments which is equal to the best of Hewlett-Packard Controllers and far superior to most others. Basic Commands for Serial Poll, Parallel Poll, IFC Clear, full Local/Remote Control, Respond to SRQ Interrupts, Send Trigger, do Formatted Input/Output, Direct Memory Input/Output and MORE. Interface includes IEEE-488 Ribbon Cable/Connector.

GPIB Controller Interface for C2, C3, C4 and C8 Systems .... **GPIB 4-488 $395**
GPIB Software for **OS-65D** (Add -8 for 8" or -5 for 5") .... **GPIB 488-D $ 70**
GPIB Software for **OS-65U** .... **GPIB 488-U $100**
GPIB Software on two 2716 EPROMS for **ROM** based systems ..... **GPIB 488-R $100**
Add Optional **Parallel Printer Interface** to GPIB 4-488 .... **-P $120**
Add Optional **Calendar/Clock** to GPIB 4-488 .... **-T $ 25**
Add **2K RAM** to GPIB 4-488 (Specify location, $4000-$BFFF & $DOOO-$EFFF available) **-M $ 25**
Software for **EPROM Programming, Reading, Verifying,** and **Erased Check;** fully Integrated with **Assembler, Editor** and **Extended Monitor.** Can be used with many types of EPROM up to 8K. Requires Optimal Technology Model Ep-2A-79 EPROM Programmer and the GPIB 4-488 Board. Specify 8" or 5" Disk .... **EPROM MI-EP $180**
GPIB Controller for C1P, Includes Software, Clock & space for 6K EPROM .... **GPIB 6-488R $395**
Add Optional **Parallel Printer Interface** to GPIB 6-488R........................ **-P $120**

EPROMS: (Check with your Dealer for newest EPROM Products).

C1P ROM with 48 Col Display, Smart Terminal, Edit & More for Series II .... **ROM-TERM II $59.95**
C1P ROM with 24 Col Display, Other ROM-TERM II Features & Disk Boot .... **ROM-TERM $59.95**
Just flip switch for Serial or Video System with Corrected Keyboard .... **SYNKEY $39.95**

ENHANCED MONITOR ROMS FOR USE ON GPIB 4-488 & MEM BOARDS:

Expanded Support for **C4P & C8P** Featuring Calendar/Clock, Line Edit, Smart Terminal, Memory Files, Parallel Printer Control & More **MI48P1 $59.95**
Disk Support with Calendar/Clock, Warm Start and Corrected Keyboard **MI48D1 $59.95**
Expanded C3 Monitor with Calendar/Clock Software, Hard Disk Boot, Line Edit and Warm Start **MIC3-1 $59.95**
C1-P Series II Computer with ROM-TERM II Smart Terminal Monitor (Order Direct) ... **$549.00**

Check with your local Dealer or Order Direct. Phone orders accepted.
TERMS: Check/Money Order/Master Charge/VISA Sent POSTPAID ON PREPAID ORDERS.
Foreign Orders: Prepaid only. Add 5% for handling/shipping.

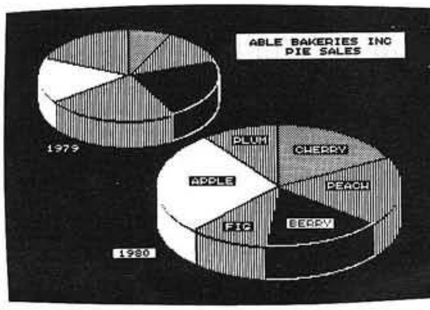
MICRO-INTERFACE
3111 SO. VALLEY VIEW BLVD., SUITE I-101
LAS VEGAS, NEVADA 89102
Telephone: (702) 871-3263

MORE

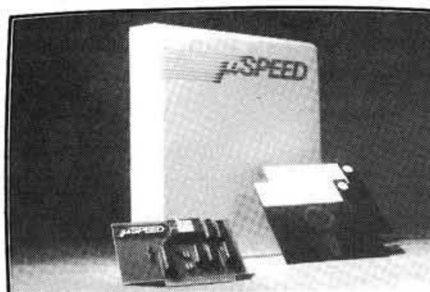
POWER
FOR YOUR APPLE

μSPEED][AND][+
LANGUAGE SYSTEMS

APPLESOFT: 30.3 MIN.
MICROSPEED][: 3.9 MIN.
MICROSPEED][+: 2.4 MIN.



FASTEST: UP TO 100 TIMES FASTER THAN APPLESOFT
MOST POWERFUL: MORE POWER THAN BASIC PASCAL OR FORTRAN
EXPANDABLE: LANGUAGE BASED ON FORTH
CREATIVE: GROW YOUR OWN LANGUAGE
USER-FRIENDLY: EASIEST FOR YOU TO LEARN



REQUIRES APPLE, SINGLE DISK
μ SPEED][USES 2mHz PROCESSOR
μ SPEED][+ USES 4mHz PROCESSOR

SEE YOUR DEALER OR CONTACT:

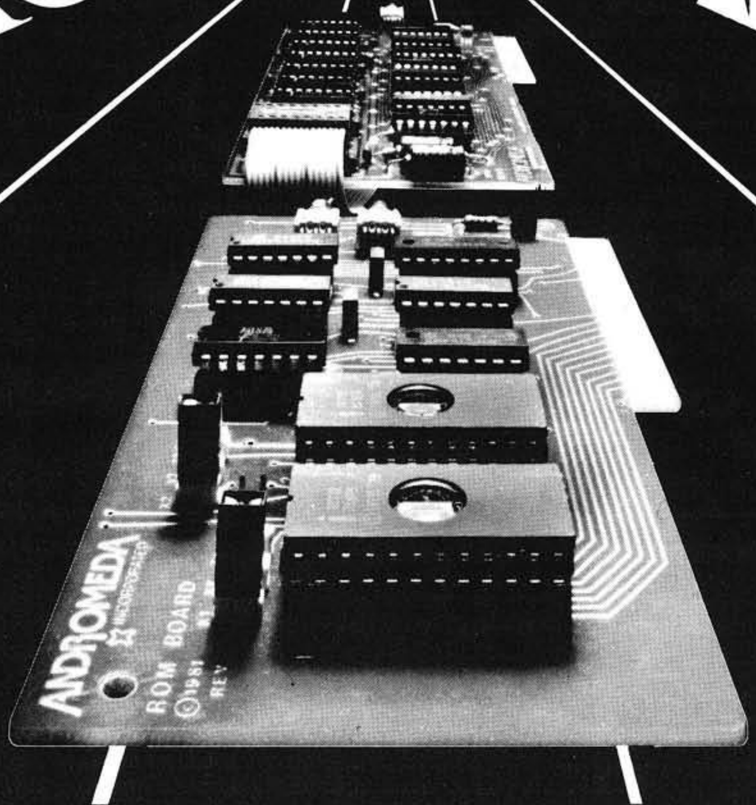
applied analytics incorporated
8910 Brookridge Dr., Suite 804, Upper Marlboro, Md. 20870
(301) 627-6650

I'm Interested: Please Send
☐ μ SPEED][$495. ☐ 160 page Manual $35.
☐ μ SPEED][+ $645. ☐ Detailed Information

Name ____________
Address ____________
City ____________
State ____________ Zip ____________

A BRIGHT NEW STAR FROM ANDROMEDA!

ROM ☆ RAM



NEW ROM BOARD FOR THE APPLE II*
$125.00 WITH UTILITY ROM.

With Andromeda's new ROM Board, you can plug many useful utility programs into your Apple II. Because ROM memory never forgets, you can access these utilities instantly without having to load them from disk.

The ROM Board comes with the utility ROM, which gives you five powerful options to apply to your Applesoft* programs. With the Utility ROM, you can do automatic line numbering, control a program list with a page mode, restore a crashed Applesoft* program in memory, alphabetize a disk catalogue and create a disk without DOS, giving you an extra 8K on your disk. Any of Soft Control Systems' other ROMS can be used, such as the Dual DOS in ROM, and 'Your'ple ROM.

You can install 2K PROMS, 4K PROMS, or even 2K RAM chips in each of the two memory sockets. So you can even have the Read - Write capability of RAM to develop PROM Programs yourself, or just have an extra 2K RAM for your machine - Language programs. Two 2732 PROMS allow a total of 8K of memory on the Board.

Now with One Year Warranty.

Don't forget the Andromeda 16K RAM Expansion Board $195.00

ANDROMEDA

INCORPORATED
P.O. Box 19144
Greensboro, NC. 27410
919 852-1482

Distributed By:

COMPUTER DATA SERVICES
P.O. Box 696
Amherst, NH. 03031
603 673-7375

*Apple II and Applesoft are trademarks.

MICRO

Dr. William R. Dial
438 Roslyn Avenue
Akron, Ohio 44320

6502 Bibliography: Part XXXVIII

1095. Call -A.P.P.L.E. 4, No. 2 (February, 1981)

DeGroat, Ron, "Seeing Double with Pascal Graphics," pg. 30-33.
Some routines that allow you to select which Apple Hi-Res page you want to draw on.

Heinonen, Jani and Kotivuori, Ilmo, "Utility EXEC," pg. 34-36.
Techniques for using control level instructions in Apple Pascal.

Bronstein, Neil, "Moving Signs," pg. 40-41.
Horizontal and Vertical moving sign routines for the Apple.

Golding, Val J., "Control Character Finder," pg. 43.
A program to find and display control characters imbedded in a disk catalog or BASIC program listing.

Horsfall, Richard C., "Readscrn," pg 45.
A program to read a character at an X and Y coordinate and print it again using 10-res Apple graphics functions.

1096. OSIO Newsletter 3, No. 3 (March, 1981)

Rowlett, Tom, "Some POKEs for C4P's and C8P's," pg. 1-2.
POKEs for OSI micros useful for polled keyboard/video system users; POKEing arguments into machine language routines directly.

Kirshner, Joe, "OS-65D Notes," pg. 1-3.
More on null strings, listing files, etc.

Anderson, Walter I, "Notes on Right Justify," pg. 4.
A tip for OSI users.

1097. The Harvest 2, No. 7 (March, 1981)

Russ, John, "Why Apple Pascal," pg. 1-3.
An article comparing the virtues of several languages for the Apple.

Breyfogle, Louis D. and Quinn, Jack D., "The 13/16 Sector Problem," pg. 13.
Hardware mod to switch the Apple Controller card between DOS 3.2 and 3.3.

1098. Personal Computing 5, No. 3 (March, 1981)

Perry, Robert L., "Inventory Control Programs," pg. 25-35.
A comparison of programs available for micros including the Apple, PET, OSI, Atari, TRS-80, CP/M, etc.

Veit, Stanley S., "Everything You Wanted to Know About Printers," pg. 58-69.
A comparison of over 80 printers for microcomputers.

1099. Washington Apple Pi 3, No. 1 (January, 1981)

Lefkowitz, Howard, "Using the Smarterm for Pascal," pg. 8-9.
Information useful to Apple-Pascal users running the 80-column modification.

Mahoney, John, "Graph of a Trigonometric Function," pg. 9.
Apple Hi-Res graphics program to generate a solid by rotating the graph of a cosine function.

Moon, John L., "A Questionnaire Subroutine," pg. 10-11.
An instructional article on designing an interactive routine for the Apple using the example of questions in ABBS programs.

Rose, Jim, "Fast Walsh-Hadamard," pg. 16-21.
A faster Fast Fourier Transform for the Apple. Machine language routine and Integer BASIC listing.

Chambers, Burton S., "Flavors — Little Tidbits," pg. 24-25.
Tips on using Apple/Pascal.

1100. The C.I.D.E.R. Press 3, No. 1 (January/February, 1981)

Titlebaum, Ed., "Apple Hi-Res Graphics: Math Applications," pg. 7-9.
Part I of a series includes a description of a plotting package called Quadplot.

Myers, Thomas, "Tuning Up Your Applesoft Programs," pg. 11.
Tips on programming efficiently.

1101. Washington Apple Pi 3, No. 2 (February, 1981)

Mesztenyi, C.K., "Passing Argument Values to Machine-Language Subroutines in Applesoft," pg. 5.
A short interface routine for the Apple.

Francis, Walton, "The Search for the Almost-Perfect, Low-Cost Apple Word Processor," pg. 6-9.
Considerations on the selection of a word processor for the Apple.

Kelly, Jim, "Pseudo Data Statements for Integer BASIC," pg. 13.
Several examples of implementing data statements on the Apple using Integer BASIC.

Lefkowitz, Howard, "The Mysterious Modem," pg. 14.
How to filter noise out of the phone lines coming into your Apple/Hayes Micromodem system.

Chambers, Burton S., "Flavors: Little Tidbits," pg. 16-17.
Hints and kinks for the Apple.

1102. Stems from Apple 4, Issue 1 (January, 1981)

Contreras, Warren, "Double Fun," pg. 7.
Add two joysticks to your Apple.

Allison, Gene, "Catalog to Epson Printer," pg. 8.
A program to make a hard copy of your disk catalog on the Epson MX-80.

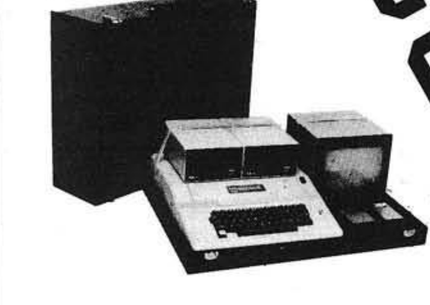
Allison, Gene, "Epson Printer Demo," pg. 11.
An Applesoft program to put your new Epson printer through its paces and provide a good tutorial to boot.

1103. Appleseed Newsletter 2, No. 6 (February, 1981)

Pump, Mark, "Apple II DOS Internals," pg. 7-11.
This installment includes Disk II Device Select addresses; DOS errata; miscellaneous goodies; sector skew factor, etc.

1104. Stems from Apple 4, Issue 2 (February, 1981)

Jochumson, Christopher, "One Liners," pg. 3.



A couple of one liners: a graphics program and a routine to alter the cursor of the Apple.

Smith, Eric, "FORTH Corner," pg. 6-7.

Benchmark Tests comparing FORTH, BASIC and Pascal.

Greenfield, Dave and Stein, Dick, "Unit PEEKPOKE," pg. 8-9.

A routine for the Apple/Pascal system and a test program.

Jochumson, Christopher, "Apple's Input and Output Registers, "pg. 10-13.

How the input and output registers are used in the Apple.

Jochumson, Christopher, "MX-80 Printer Utility."

A utility for the Apple/Epson MX-80 combination to provide a form feed at the bottom of a program listing to effectively skip over the perforation in folded paper.

1105. MICRO No. 34 (March, 1981)

Carlson, Edward H., "A 6502 Assembler in BASIC," pg. 7-9.

A 6502 assembler written in BASIC and tuned up for an OSI C2-4P computer.

Green, Len, "SYM-ple Sym-on," pg. 15-16.

A game for the SYM.

Reich, L.S., "Rapid Bubble Sort of Numerical Elements Using BASIC/ASL," pg. 21-22.

A BASIC sort program using a machine language subroutine for comparisons and swapping during sorting on the Apple.

Evans, Mel, "A Relocating Loader for AIM Tape," pg. 25-27.

With this routine you can assemble a program at one location and load it at another.

Strasma, James, "Unassembler for PET," pg. 29-32.

Here's a way to convert your machine-language programs into a form your assembler understands.

Hoyt, Sherwood, "Encryption with RND and USR," pg. 35-37.

A simple text-encoding scheme using Microsoft BASIC for 6502 micros.

Hyde, Randy, "The 6502 Dream Machine," pg. 67-74.

A proposal for a super 6502 as imagined by an expert.

Froelich, Jerry W., "A Second Cassette for PET," pg. 81-82.

A "how-to" article to modify a standard cassette recorder to function as a second cassette for the PET.

Brady, Joe, "Reset Protection for the Apple II," pg. 89-90.

A hardware modification for the Apple Reset.

1106. AppleGram 3, No. 2 (February, 1981)

Sander-Cederlof, Bob, "Hex/Dec Calculator," pg. 5.

A program for the Apple which simulates the TI Programmer calculator, operating in Reverse Polish like H-P units.

Sander-Cederlof, Bob, "Re-READing DATA Statements," pg. 8.

How to re-read data statements on the Apple without using the Restore statement.

1107. Rubber Apple Users Group Newsletter 4, No. 1 (January/February, 1981)

Park, John and Park, Lloyd, "Machine Language Routine Link to Applesoft Programs," pg. 5.

An Apple program to imbed a machine language routine into an Applesoft program.

1108. Apple-Dayton 2, No. 2 (March, 1981)

Anon., "Pascal I/O Error Reporting," pg. 4.
A procedure which can be used as an include file under the Apple/Pascal UCSD operating system.

1109. Atari Computer Enthusiasts 2, Issue 3 (March, 1981)

Bannister, Ray, "Memory Dump," pg. 5.
A program for the Atari which dumps the memory, using the special graphics characters as ASC hex equivalents.

Ness, Ron, "Music Without the Composer Cartridge," pg. 6-7.
An Atari music program and song table for Starwars.

Goff, Stacy, "Fileindx — Miracle Cure for All Problems," pg. 8-9.
An index to disk files with options to sort, save to disk, load back, display or print.

Hitz, Larry, "Modem to Disk," pg. 11.
An Atari program to download from Micronet, etc. to your disk.

1110. Apple Assembly Line 1, Issue 6 (March, 1981)

Bernard, Robert H., "A Beautiful Dump," pg. 2-5.
An Apple II relocatable memory dump program in assembly language.

Wellman, Chuck, "EDITASM & COPY on the Language Card," pg. 12-14.
A utility for the Apple II.

1111. Mini'App'Les 4, No. 3 (March, 1981)

Allen, Earl, "Applesoft Fast File Accessing," pg. 3-5.
A faster alternative for handling massive amounts of data on the Apple.

Hammond, Daryl, "Pondering Pascal: Zapping the BIOS," pg. 6-7.
This program modifies the Apple/Pascal BIOS file SYSTEM.APPLE so that lower case characters are displayed as true lower case.

1112. Softalk 1, No. 7 (March, 1981)

Wagner, Roger, "Assembly Lines," pg. 20-23.
Part 6 in this guide to assembly language covers the Zero and Carry flags of the status register of the 6502.

Mazur, Jeffrey, "The Look with Character: 80-Column Boards for the Apple," pg. 26-31.
A review and evaluation of 80-column boards for the Apple.

1113. Poke—Apple 3, No. 2 (March, 1981)

Watson, Dale, "Expanding Applesoft Programs," pg. 5-9.
How to add lines of code to a program while running.

Averill, Bonnie Kaufman, "Elementary Programming: A Basic Budget," pg. 10-11.
The second in a series of articles on developing a budget program.

McKee, Dan, "Pascal by Trial and Error," pg. 13.
A brief tutorial on elementary Pascal.

Neff, Thomas M., "Apple Notes," pg. 14-15.
How to add a binary subroutine to an Applesoft program.

Hill, Alan G., "Filename Track/Sector Lister," pg. 15-18.
This Apple program will print the Track/Sector list for each filename in the catalog.

MICRO

OSI TRS-80 COLOR-80 OSI

GALAXIAN - 4K - One of the fastest and finest arcade games ever written for the OSI, this one features rows of hard-hitting evasive dogfighting aliens thirsty for your blood. For those who loved (and tired of) Alien Invaders. Specify system — A bargain at $9.95 OSI

LABYRINTH - 8K - This has a display background similar to MINOS as the action takes place in a realistic maze seen from ground level. This is, however, a real time monster hunt as you track down and shoot mobile monsters on foot. Checking out and testing this one was the most fun I've had in years! — $13.95. OSI

THE AARDVARK JOURNAL

FOR OSI USERS — This is a bi-monthly tutorial journal running only articles about OSI systems. Every issue contains programs customized for OSI, tutorials on how to use and modify the system, and reviews of OSI related products. In the last two years we have run articles like these!

1) A tutorial on Machine Code for BASIC programmers.
2) Complete listings of two word processors for BASIC IN ROM machines.
3) Moving the Directory off track 12.
4) Listings for 20 game programs for the OSI.
5) How to write high speed BASIC — and lots more —

Vol. 1 (1980) 6 back issues - $9.00
Vol. 2 (1981) 4 back issues and subscription for 2 additional issues - $9.00.

ADVENTURES!!!

For OSI, TRS-80, and COLOR-80. 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 8K on an OSI and 16K on COLOR-80 and TRS-80. They sell for $14.95 each.

ESCAPE FROM MARS (by Rodger Olsen)
This ADVENTURE takes place on the RED PLANT. 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!

TREK ADVENTURE (by Bob Retelle)
This one takes place aboard a familiar starship. The crew has left for good reasons - but they forgot to take you, and now you are in deep trouble.

DEATH SHIP (by Rodger Olsen)
Our first and original ADVENTURE, this one takes place aboard a cruise ship - but it ain't the Love Boat.

VAMPIRE CASTLE (by Mike Bassman)
This is a contest between you and old Drac - and it's getting a little dark outside. $14.95 each.

OSI NEW-NEW-NEW OSI
TINY COMPILER

The easy way to speed in your programs. The tiny compiler lets you write and debug your program in Basic and then automatically compiles a Machine Code version that runs from 50-150 times faster. The tiny compiler generates relocatable, native, transportable machine code that can be run on any 6502 system.

It does have some limitations. It is memory hungry — 8K is the minimum sized system that can run the Compiler. It also handles only a limited subset of Basic — about 20 keywords including FOR, NEXT, IF THEN, GOSUB, GOTO, RETURN, END, STOP, USR(X), PEEK, POKE, -, =, *, /, , <,> , Variable names A-Z, and Integer Numbers from 0-64K.

TINY COMPILER is written in Basic. It can be modified and augmented by the user. It comes with a 20 page manual.

TINY COMPILER — $19.95 on tape or disk OSI

SUPERDISK II

This disk contains a new BEXEC* that boots up with a numbered directory and which allows creation, deletion and renaming of files without calling other programs. It also contains a slight modification to BASIC to allow 14 character file names.

The disk contains a disk manager that contains a disk packer, a hex/dec calculator and several other utilities.

It also has a full screen editor (in machine code on C2P/C4)) that makes corrections a snap. We'll also toss in renumbering and program search programs — and sell the whole thing for —

SUPERDISK II $29.95 (5¼") OSI

BARE BOARDS FOR OSI C1P

MEMORY BOARDS!!! — for the C1P — and they contain parallel ports!

Aardvarks new memory board supports 8K of 2114's and has provision for a PIA to give a parallel ports! It sells as a bare board for $29.95. When assembled, the board plugs into the expansion connector on the 600 board. Available now!

PROM BURNER FOR THE C1P — Burns single supply 2716's. Bare board — $24.95.

MOTHER BOARD — Expand your expansion connector from one to five connectors or use it to adapt our C1P boards to your C4/8P. - $14.95.

16K RAM BOARD FOR C1P — This one does not have a parallel port, but it does support 16K of 2114's. Bare Board $39.95.

WORD PROCESSING THE EASY WAY— WITH MAXI-PROS

This is a line-oriented word processor designed for the office that doesn't want to send every new girl out for training in how to type a letter.

It has automatic right and left margin justification and lets you vary the width and margins during printing. It has automatic pagination and automatic page numbering. It will print any text single, double or triple spaced and has text centering commands. It will make any number of multiple copies or chain files together to print an entire disk of data at one time.

MAXI-PROS has both global and line edit capability and the polled keyboard versions contain a corrected keyboard routine that make the OSI keyboard decode as a standard typewriter keyboard.

MAXI-PROS also has sophisticated file capabibilities. It can access a file for names and addresses, stop for inputs, and print form letters. It has file merging capabilities so that it can store and combine paragraphs and pages in any order.

Best of all, it is in BASIC (0S65D 51/4" or 8" disk) so that it can be easily adapted to any printer or printing job and so that it can be sold for a measly price.

MAXI-PROS — $39.95. Specify 5¼ or 8" disk.

SUPPORT ROMS FOR BASIC IN ROM MACHINES — C1S/C2S. This ROM adds line edit functions, software selectable scroll windows, bell support, choice of OSI or standard keyboard routines, two callable screen clears, and software support for 32-64 characters per line video. Has one character command to switch model 2 C1P from 24 to 48 character line. When installed in C2 or C4 (C2S) requires installation of additional chip. C1P requires only a jumper change. — $39.95

C1E/C2E similar to above but with extended machine code monitor. — $59.95 OSI

ARCADE GAMES FOR OSI, COLOR-80 AND TRS-80 (8K OSI, 16K TRS-80 AND COLOR-80)

TIMETREK - A REAL TIME, REAL GRAPHICS STARTRECK. See your torpedoes hit and watch your instruments work in real time. No more unrealistic scrolling displays! $14.95.

STARFIGHTER - This one man space war game pits you against spacecruisers, battlewagons, and one man fighters, you have the view from your cockpit window, a real time working instrument panel, and your wits. Another real time goody. $9.95

BATTLEFLEET - This grown up version of Battleship is the toughest thinking game available on OSI or 80 computers. There is no luck involved as you seek out the computers hidden fleet. A topographical toughie. $9.95

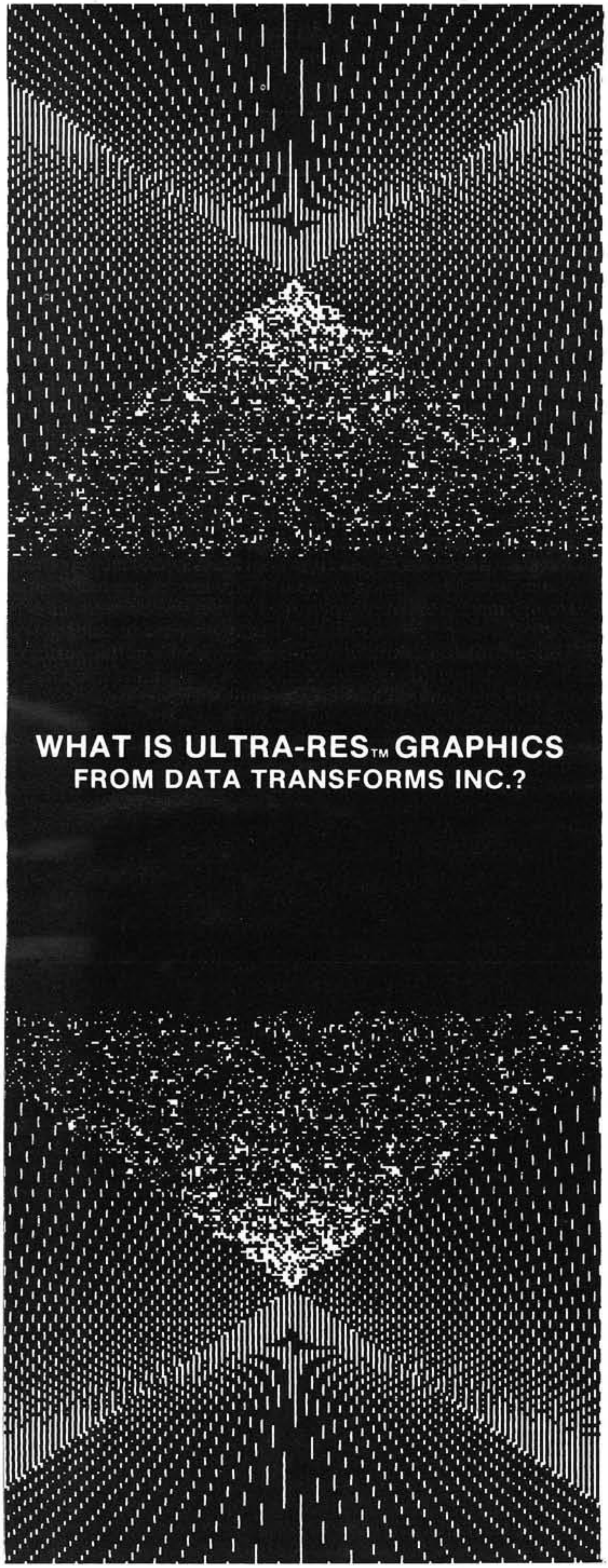
QUEST - A NEW IDEA IN ADVENTURE GAMES! Different from all the others, Quest is played on a computer generated mape of Alesia. Your job is to gather men and supplies by combbat, bargaining, exploration of ruins and temples and outright banditry. When your force is strong enough, you attack the Citadel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

Please specify system on all orders

This is only a partial listing of what we have to offer. We offer over 120 games, ROMS, and data sheets for OSI systems and many games and utilities for COLOR-80 and TRS-80. Send $1.00 for our catalog.

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

OSI COLOR-80



WHAT IS ULTRA-RES™ GRAPHICS
FROM DATA TRANSFORMS INC.?

Advertiser's Index

Next Month in MICRO

Double Apple Bonus: Applesoft and Programming Languages

- **Plotting Figures from Applesoft —** This program demonstrates how large and complex figures may be put onto the Hi-Res screen by plotting piecewise approximations of their edges.
- **Apple Memory Map Display —** MEMAP is a short, exec file utility which creates memory maps of Applesoft programs without altering the memory contents.
- **Applesoft Variable Dump —** The ability to dump the values of all variables can be immensely helpful in Applesoft program development. The Applesoft Variable Lister provides this ability and can be used with any program, located anywhere in memory.
- **Sweet 16 Revisited —** The Apple II's Integer BASIC ROM supports a powerful and seldom used pseudo machine known as Sweet 16. In this article, the Sweet 16 instruction set is described and programming hints, using a macro-assembler, are presented.
- **Applesoft Line Finder Routine —** This 55-byte machine language program will display the bytes constituting a specified line in an Applesoft program. Also demonstrated: the use of subroutines available in Applesoft and the Apple Monitor.

Other December Articles

Flags and Boolean Algebra in Microsoft BASICs; Shorthand for Cursor Control; The AIM 65 as a High-Speed Recorder; Recursive Use of GOSUB in Microsoft BASIC; OSI Symbolic Disassembler.

Plus Our Regular Departments

PET Vet
From Here to Atari
Software Catalog
Hardware Catalog
... and more

ADVENTURE has gone GOLD!
AND WE WANT YOU TO CELEBRATE WITH US.*

The 12 Scott Adams' Adventures are presented in our **Limited Gold Edition.** Each tape and disk is individually numbered, and guaranteed until July 10th, 2001. Each package contains a certificate of authenticity, a registration card and an autographed, (rather interesting) photo of the author.
The 12 Adventures normally retail for $239.40 individually on tape and $159.80 for 4 triple-pack disks.

The **Limited Gold Edition** is yours . . . **forever** . . . for $100.00 tape or disk, value for value.
To Order: The **Limited Gold Edition** is available in very limited quantity, from interested Software Retailers. Ask your local dealer. If he does not have The **Limited Gold Edition** . . . Then call toll free 1-800-421-5770. In California 1-800-262-4242 (Local 213-670-9461) . . . And we will direct you to a dealer who does have the **Limited Gold Edition.** ***Supply Is Limited!***

The **Limited Gold Edition** is available in the following different configurations.

- Apple 2 Disk with TRS-80 Disk in same package
- Atari Tape
- Atari Disk
- TRS-80 Tape

$100.00 Each

*In keeping with the spirit of **Adventure,** $1,000.00 in Gold Coins is hidden within The **Limited Gold Edition.** Its waiting for **you.**

The **Limited Gold Edition** from Adventure International is distributed exclusively by: SOFTSEL

8295 SO. LA CIENEGA BLVD. • INGLEWOOD, CA 90301

HUNTINGTON COMPUTING

ONE OF THE WORLD'S LARGEST INVENTORIES

Limited Time Only

Visidex

List 200.00 **Now 139[00]**

(Please mention this ad when ordering) ***Expires Dec. 31, 1981***

Personal Software, Brand New, Shrink Wrapped. Immediate Delivery. $2.00 Shipping (for total Software order). No Hidden Charges. No Gimmicks.

Also

Atari®

Pet®

TRS-80®

Huge Inventory

The World's Most Remote Computer Store! This is the Huntington Computing gang in the cotton field across the street from our 3300 square foot store in Corcoran, CA. We may be in the country but we believe we have the world's largest selection of microcomputer software. And, we accept School Purchase Orders.

SPECIALS

Visicalc List 200.00 ....Now **$149.00**

Pegasus II List $29.95 ....NOW **$22.99**

S-C Assembler II List $55.00 ....NOW **$46.69**

Microsoft Z-80 Card List $395 ....NOW **$299.00**

D.C. Hayes Micromodem II List $375 NOW **$299.00**

NEC 12" Green/Black Monitor
List $285, NOW $239.00

Andromeda or Microsoft 16K Expansion Board
List $195, NOW $169.00

All Educational Software
15% off list

Call Toll-Free 800-344-4111 (Outside California)

HUNTINGTON COMPUTING, Dept. MC-1
Post Office Box 787
Corcoran, California 93212

Order by Phone 800-344-4111
In California (209) 992-5411

SUPER DISCOUNTS

apple® SOFTWARE

VISA

master charge

We take MasterCard or VISA (Include card # and expiration date). California residents add 6% tax. Include $2.00 for postage. Foreign and hardware extra. Send for free catalog. Prices subject to change.