NO. 55 DECEMBER 1982

MICRO™

THE 6502/6809 JOURNAL

Commodore Feature

ATARI Graphics

APPLESOFT GOTO/GOSUB Checker

68000 Logic Instructions

0 74470 11690 12

THE PROWRITER COMETH.

(And It Cometh On Like Gangbusters.)

Evolution.

It's inevitable. An eternal verity.

Just when you think you've got it knocked, and you're resting on your laurels, somebody comes along and makes a dinosaur out of you.

Witness what happened to the Centronics printer when the Epson MX-80 came along in 1981.

And now, witness what's happening to the MX-80 as the ProWriter cometh to be the foremost printer of the decade.

SPEED

MX-80: 80 cps, for 46 full lines per minute throughput.
PROWRITER: 120 cps, for 63 full lines per minute throughput.

GRAPHICS

MX-80: Block graphics standard, fine for things like bar graphs.
PROWRITER: High-resolution graphics features, fine for bar graphs, smooth curves, thin lines, intricate details, etc.

PRINTING

MX-80: Dot matrix business quality.
PROWRITER: Dot matrix correspondence quality, with incremental printing capability standard.

FEED

MX-80: Tractor feed standard; optional friction-feed kit for about $75 extra.
PROWRITER: Both tractor and friction feed standard.

INTERFACE

MX-80: Parallel interface standard; optional serial interface for about $75 extra.
PROWRITER: Parallel and serial interface standard.

WARRANTY

MX-80: 90 days, from Epson.
PROWRITER: One full year, from Leading Edge.

PRICE

Heh, heh.

Distributed Exclusively by Leading Edge Products, Inc., 225 Turnpike Street, Canton, Massachusetts 02021. Call: toll-free 1-800-343-6833; or in Massachusetts call collect (617) 828-8150. Telex 951-624.

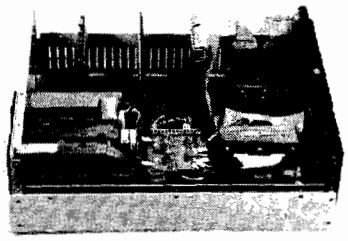
LEADING EDGE.™

For a free poster of "Ace" (Prowriter's pilot) doing his thing, please write us.

FLEX - OS-9 LEVEL ONE - UNIFLEX - OS-9 LEVEL TWO

ONLY GIMIX Systems can be configured to run any of these.

GIMIX systems utilize the most powerful 6809 operating systems: FLEX, UniFLEX, OS-9 LEVEL ONE and TWO -- the systems the PROs use. This means a wide selection of software to choose from as well the ability to develop sophisticated, multi-user/multi-tasking programs on your **GIMIX** System.



The GIMIX CLASSY CHASSIS™ consists of a heavy-weight aluminum mainframe cabinet which provides more than ample protection for the electronics and 1 or 2 optional 5¼" drives.

Backpanel connectors can be added for convenient connection of terminals, printers, drives and other peripherals.

A 3 position locking keyswitch enables users to disable the front panel reset button to prevent accidental or unauthorized tampering with the system.

The **GIMIX** system mother board provides fifteen 50 pin slots and eight 30 pin I/O slots -- the most room for expansion of any SS50 system available. The on board baud rate generator features 11 standard baud rates, 75 to 38.4K, for maximum versatility and compatibility with other systems. Extended address decoding allows the I/O block to be addressed anywhere in the 1 megabyte address space. All components feature Gold plated connectors for a lifetime of solid connections. All boards are fully buffered for maximum system expansion.

Each **GIMIX** Mainframe System is equipped with an industrial quality power supply featuring a **ferro-resonant constant voltage transformer** to insure against problems caused by adverse power input conditions such as A.C. line voltage fluctuations etc. The supply provides 8 volts at 30 amps and plus or minus 16 volts at 5 amps, more than enough capacity to power a fully loaded system and two internal drives.

The 2MHz GIMIX 6809 PLUS CPU board includes a time of day clock with battery back-up and 6840 programmable timer to provide the programmer with convenient, accurate time reference. Later addition of 9511 or 9512 arithmetic processors is provided for on the board. The unique **GIMIX** design enables software selection of either OS-9 or FLEX, both included in many complete **GIMIX** systems.

GIMIX STATIC RAM boards require no complicated refresh timing cycles or clocks for data retention. GIMIX memory boards are guaranteed for 2 MHz operation with no wait state or clock stretching required.

Our low power NMOS RAM requires less than 3/4 amp at 8V for a fully populated 64K board. For critical situations, our non-volatile 64K byte CMOS static RAM boards with built in battery back-up retain data even with system power removed. A fully charged battery will power this board for a minimum of 21 days. A write protect switch permits CMOS boards to be used for PROM/ROM emulation and software debugging.

The GIMIX DMA controller leaves the processor free to perform other tasks during disk transfers - an important feature for multi-user/multi-tasking systems where processor time allocation is critical. The DMA board will accomodate up to 4 drives 5¼" or 8" in any combination running single or double density single or double headed. Programmed I/O Disk Controllers are also available.

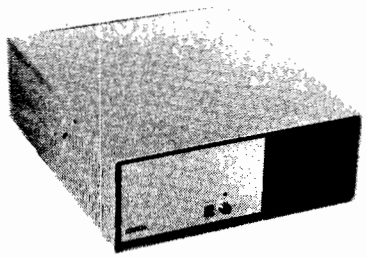
GIMIX systems are designed with ultimate **RELIABILITY** in mind. You can choose from the below featured systems or select from our wide variety of components to build a custom package to suit your needs.

GIMIX 2MHz 6809 System including: CLASSY CHASSIS, 6809 PLUS CPU BOARD, 56KB STATIC RAM, 2 SERIAL PORTS W/CABLES, GMXBUG MONITOR, FLEX, and OS-9 LEVEL 1 ..... **$3248.49**
FOR TWO 5¼" 40 TRACK DSDD DRIVES ADD ..... **$ 900.00**
GIMIX 128KB WINCHESTER SYSTEM including: CLASSY CHASSIS, 6809 PLUS CPU BOARD, 128KB STATIC RAM, 4 SERIAL PORTS W/CABLES, 5¼" 80 TRACK DSDD FLOPPY DISK DRIVE, 19MB 5¼" WINCHESTER HARD DISK, OS9 LEVEL 2, EDITOR AND ASSEMBLER **$8998.09**

50HZ Versions Available, 8" Drives Available — Contact GIMIX for Prices and Information.

The Sun Never Sets On A GIMIX!

GIMIX users are found on every continent, including Antarctica. 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, Meta Lab, Computerized Business Systems.



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

GIMIX® and GHOST® are registered trademarks of GIMIX Inc.
FLEX and UniFLEX are trademarks of Technical Systems Consultants Inc.
OS-9 is a trademark of Microware Inc.

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

GIMIX INC.

The Company that delivers

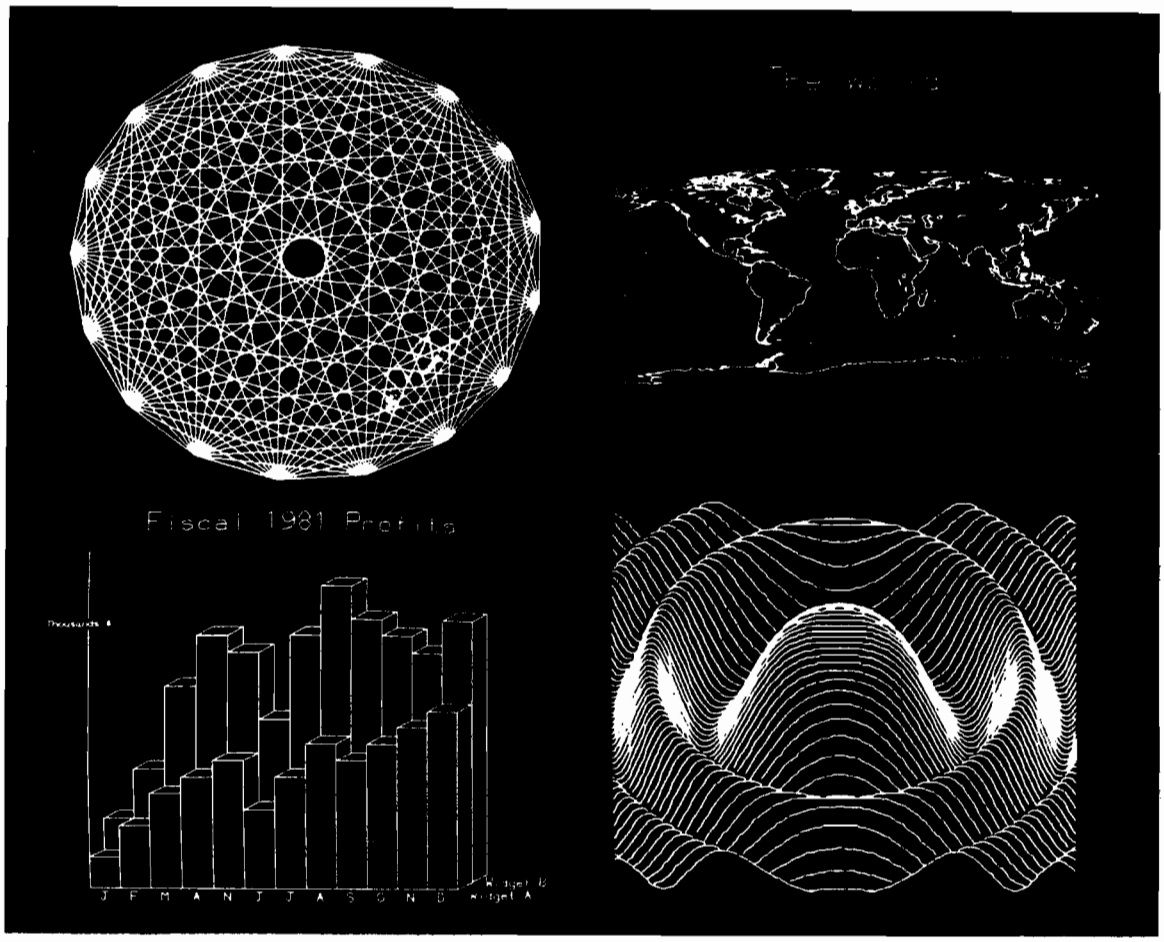
Quality Electronic products since 1975.

ANNOUNCING ElectroScreen™

only $595

the Superior Alternative

to the Traditional Alphanumeric Terminals



The ElectroScreen™ Intelligent Graphics Board Features:

Graphics

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

Intelligence

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

Terminal

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

Additional Features

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

See your dealer today!

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

The ElectroScreen has a 90 day warranty from purchase date.

Dealers, please contact us for our special introductory package.

Privac Inc (703) 671-3900

3711 S. George Mason Dr., Falls Church, Va. 22041

MICRO™

December Highlights

Commodore Machines Featured

This month we cover the full range of Commodore's machines: the PET, VIC, SuperPET, and the exciting new Commodore 64. Each machine has its own distinct features, but also shares characteristics with the other Commodore family members. CBM users will want to read all the Commodore related articles in this issue.

The second part of the University of Rochester's series (p. 59) discusses the use of an inexpensive device, the analog transducer, which can be applied to many problems outside the college teaching laboratory. The analog transducer makes it possible for your digital computer to deal with quantities measured on a continuous scale — light, voltages, densities.

Contributing Editor Jim Strasma starts on a six-part series (p. 37) that will help you write better program packages. In particular, it will cover CBM's powerful, yet poorly understood, relative record system. The first part, however, deals with designing a modular program package, setting things up, and passing parameters. Jim uses portions of the public domain program "Bennett's Mail List 4040" to illustrate his points.

We also offer a number of utilities for Commodore machines. Hans Hoogstraat's "BASIC Squeeze for PET" (p. 42) is a cassette buffer-sized program that can be saved with a fully expanded and commented BASIC program. When the program is run, it makes a call to the squeeze routine, which compresses the program to take less space and run faster. Troup and Strasma's "SOUP" (p. 52) is a compare program for machine-language routines saved on disk. Thomas Henry's "BASIC Line Delete for PET and VIC" (p. 47) adds the capability of deleting more than one BASIC program line at a time.

In our "Short Subjects" section (p. 97) we have two items of interest to users of Commodore machines. Terry Peterson explains the ASCII character set on the SuperPET and reveals some hidden features. "VIC Jitter Fixer," by Contributing Editor Dave Malmberg, can be added to your paddle, joystick, and light-pen programs to give you more reliable readings from these devices.

Finally, we feature the new Commodore 64 computer in both "PET Vet" and on our data sheet. Loren Wright's column (p. 54) reviews the graphic capabilities of this exciting new computer, and the data sheep (p. 109) provides a memory map, interfacing information, and lists of graphics and sound registers.

Expand Your Computer's Capabilities with New Hardware

The BSR X-10 allows you to control remotely a wide variety of electrical devices in your home. There are two versions available; one sends its signals using power lines as antennas, and another uses ultrasonic signals. Each light or appliance is connected to its own receiver module. John Krout's "Home Control Interface for C1P" (p. 77) shows how to add ultrasonic circuitry to your computer at a cost much less than the BSR ultrasonic option. David Hayes's "Atari Meets the BSR X-10" (p. 82) shows how to convert the unit for control from Atari's controller ports.

If you've ever looked at a 6502 programming manual, you might have noticed all the unused op codes. Now you can use those codes to execute your own machine-language routines. Curt Nelson and his associates ("Utilizing 6502's Undefined Operations," p. 93) present a circuit that causes the 6502 to execute your code, instead of crashing, when it encounters an unused op code.

In "Programmable Character Generator for OSI" Colin Macauley demonstrates how to define your own characters (p. 88). OSI readers shuld turn to our OSI book announcement on page 25.

Joe Hootman's in-depth coverage of the 68000's instruction set continues (p. 85) with a discussion of the logic instructions. As usual, convenient reference tables are included.

Apple and Atari

Paul Swanson concludes his three-part series on Atari's character graphics (p. 22) with a demonstration of patching into Atari's vertical blank interrupt routine. His "From Here to Atari" column (p. 32) covers a variety of topics, including Atari's new software acquisition centers and some technical tidbits.

Peter Meyer presents an "Applesoft GOTO/GOSUB Checking Routine" (p. 26) that displays all incorrect GOTO and GOSUB references. "ILISZT for Integer BASIC," by Leonard Anderson, is a follow up to a similar program he presented for Applesoft (p. 13). It produces an attractive, formatted listing of your Integer BASIC program, complete with indentation, paging, and other fancy features. Tim Osborn's "Apple Slices" (p. 65) presents a general-purpose binary search routine that can be called using the & vector.

MICRO™

NO. 55 DECEMBER 1982

MICRO™

THE **6502/6809** JOURNAL

DEPARTMENTS

COMMODORE FEATURE

BASIC AIDS

HARDWARE

Lyco Computer Marketing & Consultants

TO ORDER CALL US

TOLL FREE 800-233-8760

In PA 1-717-398-4079

December ATARI SPECIALS

810 Disk Drive ... $429.00
32K RAM ..... $ 79.00
400 32K RAM ...$349.00

800 48K...$609.00

ATARI®

A Warner Communications Company

PERCOM : In Stock

Single Drive	CALL
Dual Drive	CALL
(Read all Atari Disks)	

PRINTERS : In Stock

Epson Mx 80	$449.00
Epson Mx 80 FT III	$499.00
Okidata 82A	$479.00
Okidata 83A	$719.00
Okidata 84	$1089.00
Citoh	CALL
Prowriter I	$499.00
Prowriter II	CALL
SMITH CORONA TP-1	$625.00
NEC	CALL
(Interfacing Available)	

JOYSTICKS : In Stock

Atari CX-40	$18.00
LeStick	$34.00
Wico Command Control	$24.00
WICO RED BALL	$27.95
STICK STAND	$ 6.75

Computer Covers

800	$6.99
400	$6.99
810	$6.99

DISKETTES : In Stock

Maxell MD1 . . .(10)	$34.00
Maxell MD2 . . .(10)	$44.00
Elephant . . .(10)	$21.00

THIRD PARTY SOFTWARE ATARI PROGRAM EXCHANGE

Eastern Front 1941	$25.50
Avalanche	$15.50
Outlaw/Howitzer	$15.50
Dog Daze	$15.50
Wizard of War	$31.00
Gorf	$31.00
Frogger	$26.00

BUSINESS SOFTWARE : In Stock

Atari Word Processing	$109.00
Letter Perfect	$129.00
Text Wizzard	$ 89.00
Datasam/65	$125.00
Interlisp	$125.00
Monkey Wrench	$ 42.00
Utility Disk	$ 36.50
Ultimate Renumber	$ 15.50

ATARI HARDWARE

410 Cassette Recorder	$75.00
825 Printer	$585.00
830 Phone Modem	$149.00
850 Interface	$164.00

PACKAGES

CX481 Entertainer	$69.00
CX482 Educator	$125.00
CX483 Programmer	$49.00
CX494 Communicator	$325.00

SOFTWARE

CXL4012	MISSILE COMMAND	$28.75
CXL4013	ASTEROID	$28.75
CXL4020	CENTIPEDE	$32.75
CXL4022	PACMAN	$32.75
CXL4011	STAR RAIDER	$34.75
CXL4004	BASKETBALL	$26.75
CXL4006	SUPER BREAKOUT	$28.75
CXL4008	SPACE INVADER	$28.75
CX8130	CAVERNS OF MARS	$31.75
CX4106	HANGMAN	$12.75
CX4102	KINGDOM	$12.75
CX4112	STATES & CAPITALS	$12.75
CX4114	EUROPEAN COUNTRIES	$12.75
CX4109	GRAPHIT	$16.75
CX4121	ENERGY CZAR	$12.75
CX4123	SCRAM	$19.75
CX4101	PROGRAMMING I	$19.75
CX4106	PROGRAMMING II	$22.75
CX4117	PROGRAMMING III	$22.75
CXL4015	TELELINK	$21.75
CX4119	FRENCH	$39.75
CX4118	GERMAN	$39.75
CX4120	SPANISH	$39.75
CX4120	SPANISH	$39.75
CXL4007	MUSIC COMPOSER	$33.75
CXL4002	ATARI BASIC	$45.75
CX8126	MICROSOFT BASIC	$65.75
CXL4003	ASSEMBLER EDITOR	$45.75
CX8126	MACROASSEMBLER	$69.75
CXL4018	PILOT HOME	$65.75
CX405	PILOT EDUCATOR	$99.75
CX415	HOME FILING MANAGER	$41.75
CX414	BOOKEEPER	$119.75

NEW RELEASES

CHOP LIFTER	$27.75
APPLE PANIC	$23.75
PREPPIE	$19.95

THIRD PARTY SOFTWARE for atari 800 or 400

K-BYTE

KRAZY SHOOTOUT	$35.00
K-DOS	$65.00
K-STAR PATROL	$37.75
K-RAZY ANTICS	$37.75
K-RAZY KRITTERS	$37.75
Q-BALL JOYSTICK KIT	$6.75

AUTOMATIED SIMULATIONS

Star Warrior	$28.00
Crush, Crumble & Chomp	$23.00

WE CARRY MANY OTHER THIRD PARTY PRODUCTS YOU CAN CALL FOR PRICES ON AND ASK FOR YOUR FREE ATARI PRODUCT CATALOG.

commodore

VIC-20 ..... $189.00

VIC1530 DATASSETTE	$67.00
VIC1540 DISK DRIVE	$499.00
VIC1515 PRINTER	$355.00
VIC1210 3K RAM	$35.00
VIC1110 8K RAM	$52.00
VIC1211A SUPER EXPANDER	$53.00

VIC-20 SOFTWARE

VIC1212 PROGRAMMER AID	$45.00
VIC1213 VICMON	$45.00
VIC1906 SUPER ALIEN	$23.00
VIC1914 ADVENTURE LAND ADVENTURE	$35.00
VIC1915 PRIVATE COVE ADVENTURE	$35.00
VIC1916 MISSION IMPOSSIBLE	$35.00
VIC1917 THE COUNT ADVENTURE	$35.00
VIC1919 SARGON II CHESS	$35.00

THIRD PARTY SOFTWARE

ALIEN BLITZ	$21.00
Omega Race	$35.00
Gorf	$32.00
16K RAM/ROM	$99.00
AMOK	$21.00
SUPER HANGMAN	$16.00
SPIDERS OF MARS	$45.00

POLICY

In-Stock items shipped within 24 hours of order. Personal checks require four weeks clearance before shipping. PA residents add sales tax. All products subject to availability and price change. Add 4 % for Mastercard and Visa.

TO ORDER CALL TOLL FREE

800-233-8760

In PA 1-717-398-4079
or send order to
Lyco Computer
P.O. Box 5088
Jersey Shore, PA 17740

About the Cover

```
100 Fröliche Weinachten
200 Joyeux Noël
300 Felices Pascuas
400 Buon Natale
500 Happy Holidays
```

This month MICRO is taking a holiday from presenting a graphic with a computer theme on our cover. Instead, we want to offer our warmest greetings — in five languages. The colorful lights in the picture belong to the city of Frankfurt, Germany and symbolize the festive glow of the holiday season. Froliche Weinachten!

Cover photo by Phil Daley

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

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to

MICRO INK
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

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

MICRO™

Editorial

Getting to Know You

"It's more useful than my Swiss army knife." Now that's what we like to hear about MICRO and that's what one of you said in response to our reader survey. But we did the survey for more than a pat on the back.

We did the survey to find out just as much as we can about who you are and what kind of information, both in editorial content and advertising, you need and want.

We discovered that you are an extremely well-educated, affluent, gainfully employed bunch of people with a great deal of technical computer knowledge at your command — and you want more.

33% of you have advanced degrees
70% have incomes over $25,000
60% are programmer/analysts, engineers, or technicians, and
90% of you have intermediate to advanced knowledge of software and 80% of hardware.

No wonder only 6% of our readers consider MICRO too technical. Your biggest beef? Not enough information on your own system — whatever that may be. Too much Apple, not enough Apple, not enough Atari, not enough OSI. Now we know that that is going to be something of a problem in a publication that covers more than one system, or more than one chip, but we think it's important to cross-fertilize, to generalize, to bring you knowledge and information that is transferable. Our goal is to make at least half of the magazine non-system specific, while dividing the other half in much the way our readers are divided — about half Apple and the other half heavily weighted toward OSI, Commodore, Atari, and 6809 systems. Interest in the 6809 and 68000 remains high, especially among users who are adding boards and processors to 6502 machines.

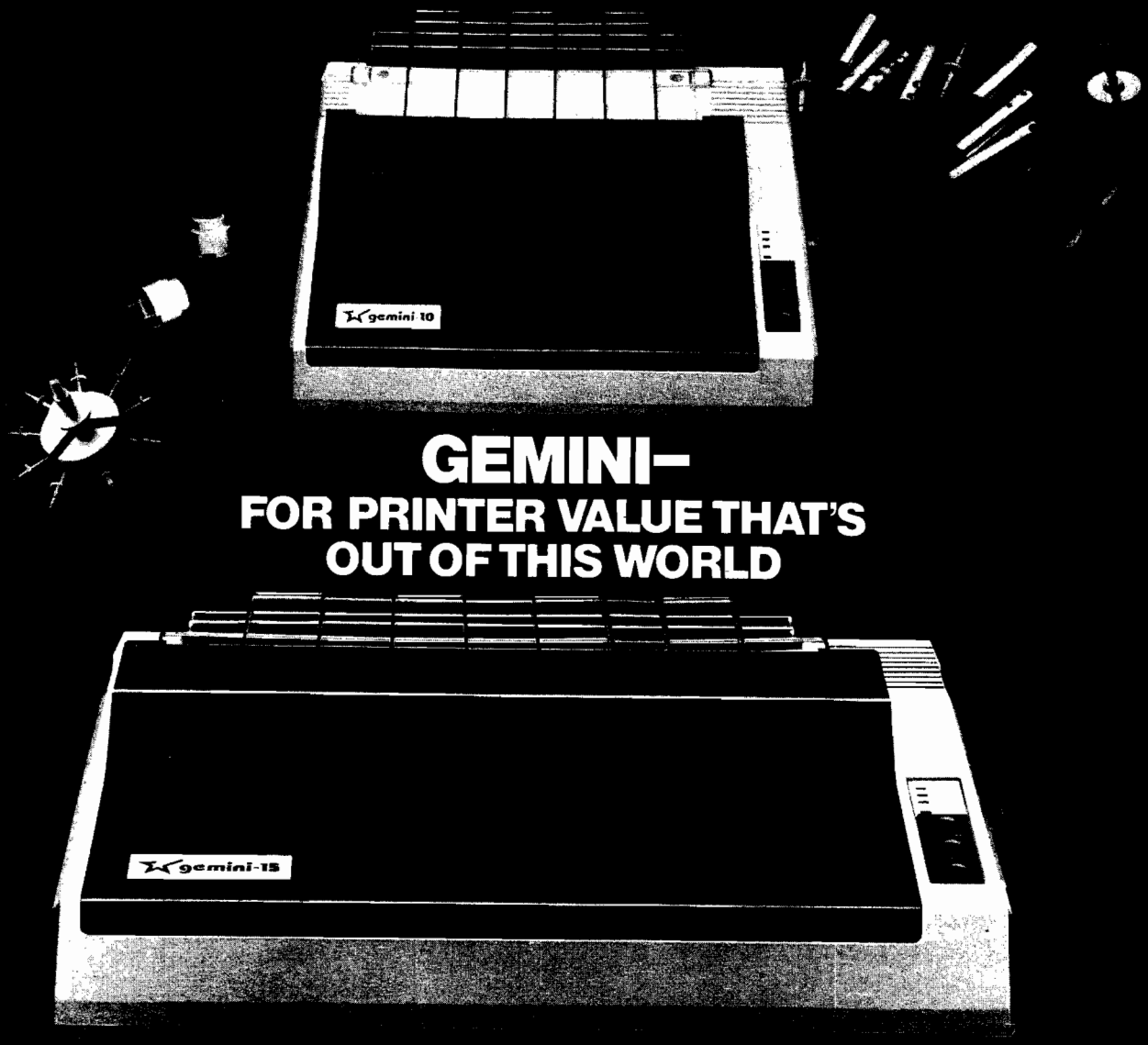
A great many of you (62%) use more than one kind of system and 46% have systems both at home and at work; nearly all of you plan to spend money adding more equipment during the coming year. We trust that the reviews, hardware and software catalogs, and advertisements are helping you make those purchases.

There is a great proliferation of system-specific publications and more and more information for the beginning computer user. We are trying not to clutter up the magazine with information you already have — you've learned a lot over the last few years and we want to help you build on that knowledge. You've matured, the market has matured, and MICRO is growing along with you. The system-specific magazines are a great place to get hints, corrections, fixes, and details about your own equipment — the kind of material it made sense for us to publish back in 1977 when no one else coverd the 6502. But now that manufacturers are doing a better job of providing documentation and there are lots of publications for beginners, we want to concentrate on more advanced issues that cut across machine and processor lines, that keep you abreast of new developments and stretch your knowledge into new areas.

MICRO's editorial schedule for the next year reflects that concern. This is the last system-specific feature we'll be running. Upcoming issues will feature various kinds of peripherals, languages, operating systems, communications. With your strong engineering background you'll want to know what new processors are being developed and how they can be used even before they're available in complete systems. There are new programming languages being developed — we will look at what they are, which ones are worth pursuing for what purposes, etc. We will provide information in the form of data sheets and information sheets on a variety of products and issues. And most interesting of all we will explore new modes of computer use: e.g., networks, communications, automated offices, and industrial control systems.

We think that advanced computer expertise is best imparted in a journal that doesn't limit itself to one system or one chip or one operating system. After all, the whole industry is moving toward compatibility and we think that is a step in the right direction. In light of that fact, and as a result of all we've learned about you and your interests from the survey, as of next month (i.e., with the January 1983 issue), we will change MICRO's subtitle to ***"Advancing Computer Knowledge."*** We are in no way abandoning the 6502 or the 6809 or any of the specific systems we've been covering. We are, instead, making a statement about your technical expertise, your maturity and the industry's, and our desire to move toward ever increasing compatibility and wider proliferation of advanced information and knowledge. You — the sophisticated user — need your own publication; we hope it's MICRO.

Mary Grace Smith



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

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

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

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

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

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

star micronics • inc

MAKING A NAME FOR OURSELVES

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

Letterbox

Back and FORTH

Dear Editor:

I was quite pleased with the two articles on FORTH in the June issue of MICRO. Regarding the benchmark comparisons of BASIC, FORTH, and RPL (page 63), I would have to say that Mr. Stryker is apparently somewhat biased in his viewpoint, since he is the father of RPL. What he appears to have done is take perfectly readable FORTH and translate it into hieroglyphics. Surely, the FORTH word DUP is more meaningful as a stack operator than "#", and who would ever guess what ";", ".", and "%" have to do with anything? Single-character words are very useful for lazy typists, but they do tend to produce "write-only" code for those who need to determine what a program is doing.

Every FORTH implementation I have ever seen has a machine-language primitive to handle block moves on a character basis. Why do we go through the gyrations of listing 1B when the word CMOVE would do just as well (actually better!)? Even without using CMOVE, the word BLKM would execute faster and with fewer FORTH words if it were written:

```
: BLKM  OVER + SWAP
        DO DUP C@ I C! 1+
        LOOP DROP ;
```

This word expects a slightly different order of things to be on the stack than originally specified: FROM TO and COUNT (634 826 150 using his numbers). This is the same order that CMOVE would expect them also. I am sure that this arrangement would be of benefit for RPL as well.

Regarding the SHUFFLER benchmark; first of all, it appears there is a typographical error of omission in line 8 of listing 2B, since the word MOD referred to in the text is not there. Even so, however, the way the routine was implemented can do nothing but slow it down.

Finally, regarding the Falling-Tone benchmark, I certainly feel the author's comments on page 68 regarding how hard it was to come up with a FORTH implementation, show a decided lack of understanding of structured programming! Listing 3A shows the same lack of structure that can be no way blamed on BASIC itself. After analyzing what the program is supposed to do, the following structured code would have been much clearer:

```
1010 DC=20:FOR Z = 20 TO 255
1020 DC=DC-Z
1030 IF DC >= 0 THEN 1020
1040 POKE 59464,Z
1050 DC=DC+256
1060 NEXT
1070 POKE 59467,0:POKE
     59466,0:RETURN
```

The same code written in FORTH looks like this:

```
: TONE 0 59464 C! 16 59467 C!
170 59466 C! 20 256 OVER DO
    BEGIN I - DUP 0 <
    UNTIL
    I 59464 C! 256 +
        LOOP DROP 0 59466 ! ;
```

Notice that we use 0 59466 ! to reset both 59466 and 59467 to zero, since FORTH inherently works with 16-bit numbers and uses 8-bit numbers only occasionally. I would probably do the same thing at the beginning of TONE to set up 59466 and 59467 initially, assuming this is a PIA register address of some sort. At any rate, the structure is there and can also be used in the RPL version, I'm sure.

Edward B. Beach
5112 Williamsburg Blvd.
Arlington, VA 22207

Dear Editor:

In "BASIC, FORTH, and RPL" (MICRO 49:63), three different computer languages are compared in terms of speed and memory economy using three benchmark programs. However, within the text of the article there were some comments made about FORTH by the author, Timothy Stryker, which require rebuttal.

Mr. Stryker states that program modules in RPL do not execute directly but rather place their address on the stack where a second call operator (&) actually executes this address. As correctly noted, this is in contrast to FORTH where the defined word directly executes; it does not need a second execute operator. This allows *all* FORTH definitions to be treated as syntactically equal. Programmers may freely mix FORTH language words with their own new definitions — indeed, there is no difference in the internal dictionary structure between these two parts.

On the other hand, RPL forces us to use (&) for execution of all new words while pre-existing ones are immune to this rule and execute directly, creating an inconsistent syntax. That this is memory efficient is doubtful. The higher level definitions of any non-trivial application program can consist of a large proportion of user-defined operators, each one of which would require the addition of this execute operator in RPL. This probably consumes some memory in the compiled form and it certainly and unnecessarily clutters up the source code. With FORTH, the address of any definition can be placed on the stack with an additional operator when it is desired, although this function is seldom needed.

It is true that FORTH handles symbols differently depending on whether they are variables, constants, or executing subroutine names. This is part of the beauty of the language, not a weakness. Each type of symbol has a different function. Subroutine names execute, constants leave their value on the stack, and variables leave their address so we can suffix them with load or store operators. Nothing could be simpler or more efficient: uniformity of function by means of inconsistent internal operation. RPL reverses this, giving us consistent internal operation while forsaking clarity of function at the programmer's level. This forces us

to be even more aware of what each definition does — something I would prefer to be left up to my compiler.

As Mr. Stryker correctly states, the FORTH string literal print word (.") and the numeric print words never leave their output string on the stack. This is seldom needed and would possibly slow down the system. Besides, the stack may not be large enough to safely handle this, since on the 6502 the FORTH stack is placed in page zero (shared with a few other FORTH locations and probably some used by the host computer for disk or terminal I/O). If we need to alter the string in numeric conversion and printing, FORTH has some primitives available for inserting additional characters in the string. With a minor effort we can add print using to an application program or make it a permanent part of the FORTH we use each day. Other than the string literal defining word (."), there are no other string operators defined in the FORTH standards, but these are not difficult to add to such an easily extensible language.

Some additional points: The modulo primitive in the fig-FORTH 6502 model takes 1.2 milliseconds to execute. No random-number generator is defined by the Group, so the poor speed of this word in Mr. Stryker's unnamed FORTH version was not optimized for speed by whomever wrote it.

Language experimentation and comparison is certainly needed to fuel the evolutionary process of computer technology. But it should best be done with the full understanding of each language involved.

Raymond Weisling
Jalan Citropuran No. 23
Solo, Jawa Tengah
Indonesia

Dear MICRO:

Thanks very much for the chance to respond to Mr. Beach and Mr. Weisling in regard to their letters concerning my recent article.

First of all, I take exception to the contention in both of these letters that I unjustly biased the benchmarks and the conclusions drawn therefrom in favor of RPL. In fact, precisely *because* I knew that this objection might be raised, I bent over backward to give the benefit of every doubt to FORTH. This may not be immediately apparent in the article because I did not make a point of saying so, but, for example, wherever my measured execution times varied slightly from one run to the next, I uniformly presented FORTH's fastest time, and RPL's slowest; for another, I specifically excluded from consideration any benchmarks involving manipulation of character strings, stack-resident arrays, finite-state automata, and other operations that RPL handles much more naturally than FORTH. Further evidence of this concern will become apparent below.

First I'll address Mr. Beach and his comments on the use of single-character operator-tokens. I do agree that RPL source must look like hieroglyphics to a person versed in FORTH — but perhaps you remember what FORTH (or any computer language) looked like before you became fluent in it. Experienced RPL users have as little difficulty reading RPL source as you do

reading FORTH. The advantages of single-character operator-tokens are three: 1. as you acknowledge, they cut down on typing time; 2. they cut down on the physical size of the source, so that more source can be fit into memory at once when undertaking nontrivial applications; and 3. they speed up compilation by cutting down on the operator-token search time.

Thank you for pointing out a better method of doing block moves in both FORTH and RPL. In writing the benchmarks, I was primarily concerned about making sure that the FORTH and RPL versions were as close to identical in approach as possible, so I missed seeing that the block móve could be done more efficiently in the way you suggest. You may be interested to know, though, that the FORTH source you show for this routine yields an execution jiffy-count of 717, considerably in excess of the 591 given for FORTH in the article. The reason? Your use of the composite "1+" operator in the innermost loop. When the sequence "1 +" is substituted for this, the execution time falls to 584 jiffies. Spaces, as you note in your letter *are* important in FORTH — one might even say, alarmingly so. They make no difference in RPL. Unfortunately, the use of even the sped-up form of your block-move algorithm does not change the standings. FORTH requires 84 program bytes to do it in 584 jiffies, whereas the following RPL equivalent:

```
BLKM: ; + 1 - % FOR # PEEK FN
POKE 1 + NEXT . RETURN
```

requires only 52 bytes to do it in 508, a "merit ratio" of 1.85 to 1.

Now, there seems to be some confusion in your letter regarding various aspects of the SHUFFLER benchmark. To begin with, there are no typos anywhere in the article. The MOD routine is, as stated, internal to the RND routine I used. This RND routine, modeled after that available under MMSFORTH, expects an integer passed to it on the stack, and returns a random number in the range from 0 up to that integer minus 1 — hence, the MOD.

Moving on to your comments regarding the third benchmark: you are right. There was no need for me to introduce unstructured code in this case. The new FORTH TONE routine you exhibit takes only 3465 jiffies, and requires only 130 bytes of program space. The corresponding RPL routine is:

```
TONE: 0 59464 POKE 16 59467 POKE
170 59466 POKE 20 256 ; FOR
LOOP: FN - # 0 < IF
FN 59464 POKE 256 +
THEN LOOP GOTO END
NEXT . 0 59466 ! RETURN
```

which requires 83 bytes of storage and executes in 3338 jiffies. The resulting merit ratio of 1.62 to 1 represents a considerable improvement. You were right, incidentally, not to condense the leading POKEs of 59467 and 59466 into a single store — the order of the POKEs into those 6522 VIA registers makes a big difference.

On to Mr. Weisling's letter. Programmers who are bothered by the necessity of suffixing their subroutine references with an ampersand in RPL are free to eliminate the space separating the two and thereby regard the composite "SUBRNAME&" as just a one-keystroke-longer method of invoking the routine. You doubt that this is memory efficient. Please find out for certain by way of the following procedure: take any nontrivial FORTH application program to which you have access and count up the number of occurrences of (A) invocations of the thirty or forty real low-level FORTH "primitives" such as DUP, "=", IF, DO, "@", and things of that nature (including ";" but not including ":"); (B) references to literal numeric quantities, whether CONSTANTs or not, it does not matter, which fall in the range from 0 to 63; (C) references to literal numeric quantities greater than 63 but less than 32768, plus all references to VARIABLEs, CVARIABLEs, and whatnot; (D) all references to literal numeric quantities not covered under B or C; and (E) all routine-invocations (other than ":") not covered under A. Be sure, if you count a routine-invocation under E, that you also consider the body of that routine part of the program source. Now form the sum A + B + 2*C + 3*D + 3*E. This is a rough approximation of the number of object program bytes that would be required, were the program translated, absolutely mechanically from FORTH into RPL. Multiply this by about 0.8 to arrive at the memory size of the equivalent program, had it been designed in RPL to begin with.

Next, a discussion on symbol handling. The fact that RPL is more efficient has been demonstrated already. That it is simpler may be difficult to appreciate second-hand like this, but RPL "gives us consistent internal operation" *without* forsaking "clarity of function at the programmer's level." The question of how aware the programmer needs to be as to "what each definition does" has nothing to do with it.

The ability to manipulate character strings conveniently is fundamental to most user-oriented software development. Indeed, your remark about the size and location of the FORTH stack points up the fact that this is one area in which FORTH's extensibility does it little good. RPL locates both stacks in page one: the parameter stack is the hardware stack, and the return stack is an indexed sort of affair down below it. Stack-resident strings up to 60 characters long or so can be manipulated freely without fear of crashing the machine — and execution is brought to a controlled halt if the 64-word stack entry limit is exceeded.

And on your last point: under my version of FORTH, a public-domain version identifying itself simply as "fig-FORTH 1.0" (which, however, includes such exotic facilities as double-precision and floating-point math, IEEE-488 I/O, etc.), the following routine, as timed with an actual watch, takes 2 minutes and 40 seconds to execute:

```
: TEST 30000 0 DO 6543 52 MOD DROP
LOOP ;
```

When the MOD is replaced with another DROP, it takes 14 seconds. I leave you to draw your own conclusions.

Timothy Stryker
Samurai Software
P.O. Box 2902
Pompano Beach, FL 33062

MICRO™

Your opinions, comments, and criticisms can be aired in MICRO too. Send mail to Letterbox, MICRO, P.O. Box 6502, Chelmsford, MA 01824.

CHRISTMAS SEASON SPECIALS!

Let ARK COMPUTING Make This Your Best Christmas Ever!

Super Fan II by R.H. Electronics **59.95/79.95**

Applicard, a high performance Z-80 card with 64K Ram, complete with CP/M
4 mhz **324.95**/445.00
6 mhz **395.00**/595.00

Microsoft Z-80 card with CP/M and Microsoft Basic
2 mhz **269.95**/395.00

Microtek Parallel Printer Interface complete with centronic compatible connector
64.95/79.95

Lazer Lower Case +Plus with Character Set +Plus **49.95**/84.90
Lower Case +Plus alone **39.95**/59.95

Lazer Graphics +Plus **99.95**/159.95
Graphics +Plus and Lower Case +Plus **134.95**/219.90

Computer Stop 16K Ram Board **69.95**/149.95

Computer Stop Omnivision 80 Column board
129.95/295.00

Videx Video-term with softswitch, inverse character set and 80 column Visicalc preboot
295.00/450.00

Wizard BPO 16K buffered printer interface (expandable to 32K) **134.95**/179.95

Wizard 80, 80 column board **195.00**/295.00

Lazer Pascal **29.95**/39.95

Anix 1.0 **34.95**/49.95

Lazer Forth **44.95**/59.95

D Tack 68000 board for the Apple II with 4K Ram **895.00**

Lazer Model/32 (16032 board for the Apple II)
CALL!

Lisa	**59.95**/79.95
Lisa Educational Pak	**79.95**/119.95
Alien Ambush	**19.95**/29.95
Bandits	**19.95**/29.95
Cannonball Blitz	**24.95**/34.95
County Fair	**19.95**/29.95
Cranston Manor	**24.95**/34.95
Cyclod	**19.95**/29.95
David's Midnight Magic	**24.95**/34.95
Dosource 3.3	**24.95**/39.95
Dueling Digits	**19.95**/29.95
Falcons	**21.95**/29.95
Firebird	**21.95**/29.95
Foosball	**19.95**/29.95
Horizon V	**25.95**/34.95
Genetic Drift	**19.95**/29.95
Kabul Spy	**24.95**/34.95
Jelly Fish	**19.95**/29.95
Lemmings	**19.95**/29.95
Labyrinth	**19.95**/29.95
Mouskattack	**24.95**/34.95
Outpost	**19.95**/29.95
Red Alert	**19.95**/29.95
Pig Pen	**24.95**/34.95
Russki Duck	**25.95**/34.95
Minator	**24.95**/34.95
Track Attack	**19.95**/29.95
Thief	**17.95**/29.95
Space Quarks	**19.95**/29.95
Snack Attack	**19.95**/29.95
Swash Buckler	**24.95**/34.95
Gin Rummy	**24.95**/34.95
The Dictionary	**69.95**/99.95
General Manager	**99.95**/149.95
4 Ft. Disk Cable	**19.95**/29.95
Visicalc	**179.95**/250.00
Using 6502 Assembly Language Book	**14.95**/19.95
Kids and The Apple Computer Book	**15.95**/19.95
Apple Panic	**19.95**/29.95
Kraft Joystick	**49.95**/69.95

714735-2250
P.O. Box 2025
Corona, CA 91720

APPLE ILISZT for Integer BASIC Programs

by Leonard Anderson

ILISZT prints an Integer BASIC program in a clear, structured format with the ability to detect embedded or attached BINARY code.

ILISZT
requires:
Apple II with both
Integer and Applesoft
Disk drive
Printer

The purchase of several disks at the end of 1981 added a number of Integer BASIC programs to my Apple II library. No listings were available and I decided to print all of them.[1] Several had embedded binary code, a condition that caused much "nonsense" display on both screen and printer. "LISZT" was already up and running (MICRO 48:37), so it seemed logical to modify this Applesoft program to format Integer listings. The ILISZT result kept the original format and added the ability to find exact binary code addresses.

ILISZTER is the formatting and printing program, run by EXEC file ILISZT. ILISZTER is Applesoft rather than Integer. While an Integer program might seem better, many Apple II owners possess ROM or RAM cards for language duality and ILISZTER seems more compact in Applesoft due to string-handling capability. Another advantage is that ILISZTER can be re-run without disk operations or loss of Integer source code.

ILISZTER retains the original features such as separation of concatenated statements, indenting, and remark highlighting. Multiple-iterator NEXT statement handling for restoring FOR-NEXT loop indents is an improvement. The added binary code determination and restoration routine is useful for listing certain utilities.[2] Since Integer BASIC differs from Applesoft, a brief review of Integer structure will help provide an understanding of ILISZTER.

Integer BASIC Source Code

Figure 1 shows one line number of source code in Integer. The first byte contains the number of bytes per line with the next two bytes having the line number in binary. End-of-line is signified by the end byte having a value of one.

Each entered line is immediately checked for syntax. Line numbers are limited to 32767 but may be modified by utilities. Numeric constants are converted to binary on entry, an advantage for program execution time.

All function words are stored as one-byte "tokens" in the range of zero to 127 decimal. Punctuation, arithmetic, and logical operators are also tokens. Eight tokens are unused and three others are used only with keyboard entries. ASCII characters have the high bit set to use the decimal range of 128 and 255. Token and character values are opposite that of Applesoft.

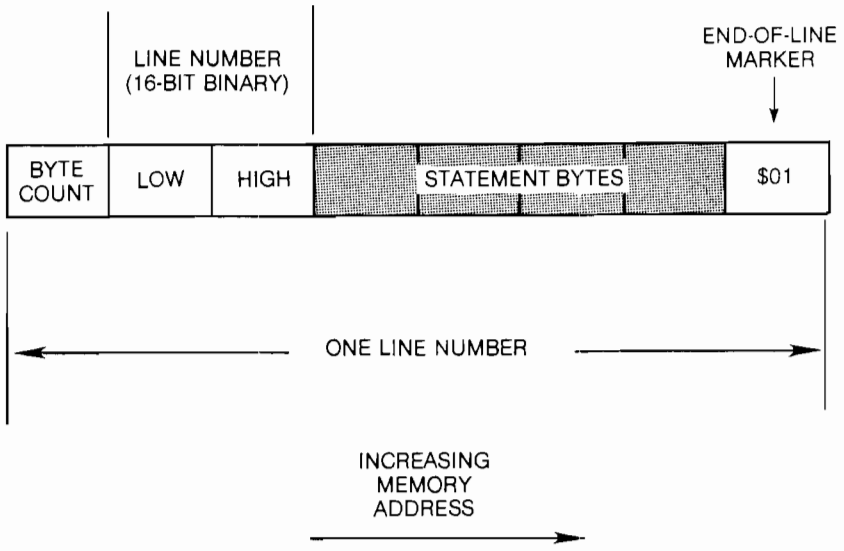
A major difference also exists in handling numeric constants within Integer. Certain functions permit a following numeric constant or variable name. Distinction of a numeric constant is done by making the first byte following an ASCII number ($B0 to $B9, not allowed as first letter of a variable) with the next two bytes containing the numeric constant in binary.

Integer BASIC is located just below the highest free memory address. Integer does not need the three-null end of program marker required by Applesoft. Other details may be found in earlier publications.[3, 4, 5]

An EXEC File for Glue

If an Integer program exists in memory, loading an Applesoft program will not destroy the Integer source code. Loading does change the Integer start-of-program pointer at $CB, $CA (203, 202). Integer end-of-program, or HIMEM at $4D, $4C (77, 76) remains unchanged.

Figure 1: **Source code structure on one line number in Integer**



HIMEM will restore to the end of free memory on re-loading an Integer program; the mechanism is unknown but confirmed through experiments.

EXEC file ILISZT is executed after loading the Integer program to be listed. The first two POKEs in ILISZT generator MAKE ILISZT will move the Integer HIMEM pointer into the LOMEM space at $4B, $4A (75, 74). LOMEM also restores on Integer reload. The last two POKEs move the start-of-program into the space normally used for Integer HIMEM.

Running ILISZTER will automatically switch over to Applesoft without disturbing the new Integer start and end addresses. MAKE ILISZT can be deleted when EXEC text file ILISZT is generated.

Starting ILISZTER

The first line resets Applesoft high memory to prevent string operations from overwriting the Integer source. Token words are initialized at line 91. Since quotes are tokens if not in a remark, the DATA declaration uses an "&" symbol with conversion *via* the IF and CHR$(34) statement.

A token evaluation array is generated in V at line 96. The V array is used in line parsing to test unused tokens and tokens that may have following numeric constants. Unused tokens (V=2) may be nulls or single spaces; spaces were written just in case the binary-insert routine crashed.

The choice of lower-case characters in token words is up to the user. Mixed-case token words give distinction from normal upper-case variables. Available utilities can edit upper-case source code by adding hexadecimal $20 to each desired lower-case letter.[6]

Initial display at line 98 is optional but it does indicate proper location and operation. The "DIFFERENT START ADDRESS" prompt allows listing to begin *after* an embedded binary; binary addresses will appear in normal printouts. ILISZTER can be RUN after any RESET or list completion without disturbing Integer source code.

Printer control in lines 107 to 110 should be set to your particular printer and interface. Subroutines at lines 17 and 18 can be changed to other runtime control. Source code control characters are converted to letters before output.

Lines that Parse in the Right

A source code line parse begins at

ILISZTER

```
0 PS =  PEEK (77) * 256 +  PEEK (76) - 1: HIMEM: PS: GOTO 82
1  REM "GET BYTE" SUBROUTINE   *
2 P = P + 1:B =  PEEK (P): RETURN
3  REM "BLANK LINE PRINT" SUBROUTINE   *
4 D = 0: GOSUB 6: PRINT S$: RETURN
5  REM "TEST PAGE SUBROUTINE   *
6 LC = LC + 1: IF LC =  < LP THEN  RETURN : REM NOT A NEW PAGE
7  GOSUB 17:LC = 6:PC = PC + 1: PRINT S$: PRINT BB$;LB$;"<continued>"
8  REM A FORM-FEED FOR TOP OF NEXT PAGE; ALLOWS VARIATION FOR DIFFERENT P
     RINTERS.
9  FOR K = 1 TO 4: PRINT S$: NEXT
10  REM PRINT THE HEADER
11 H$(4) = "Integer   Page " +  STR$ (PC): FOR K = 1 TO 4:E =  INT ((LL -
     LEN (H$(K))) / 2) + 1: PRINT M$; LEFT$ (BB$,E);H$(K): NEXT :K =  FRE
     (0): PRINT S$: IF  NOT D THEN  RETURN
12  REM PUT LINE NUMBER IN BRACKETS AS A STATEMENT IDENTIFICATION ON NEXT
     PRINT PAGE
13 N$ =  STR$ ( VAL (N$)):K =  LEN (N$): REM N$ IS NOW WITHOUT SPACES; BR
    ACKET N$ AND ATTACH TO STATEMENT CHARACTERS
14 C$ =  RIGHT$ (( LEFT$ (LB$,(6 - K)) +  CHR$ (91) + N$ +  CHR$ (93) + S
    $),8) +  RIGHT$ (C$,( LEN (C$) - 8)):K =  FRE (0): RETURN
15  REM *   *   MX-80 STANDARD/ITALICS SUBROUTINES   *   *   *
16  REM "GRAFTRAX" Only.  Single-character-set printers should DELETE the
    se calls throughout if not used for other print functions.
17  PRINT  CHR$ (27)"5";: RETURN : REM ESC-5 IS STANDARD SET
18  GOSUB 17: IF RF THEN  PRINT  CHR$ (27)"4";: REM ESC-4 IS ITALICS SET
19  RETURN
20  REM HEXADECIMAL CONVERT SUBROUTINE   *
21 A$ = "": REM ENTER WITH 'L' AS DECIMAL NUMBER, RETURN IN 'A$'
22  FOR K = 1 TO 4:D =  INT (L / 16):E =  INT ((L - (D * 16)) + 1):L = D:
    A$ =  MID$ (X$,E,1) + A$: NEXT : REM PREFIX THE "$" HEX NOTATION
23 A$ = "$" + A$:K =  FRE (0): RETURN
24  REM BEGIN A NEW LINE NUMBER WITH TEST OF NUMBER OF BYTES IN LINE FROM
     FIRST BYTE, THEN CONVERT BINARY LINE NUMBER TO DECIMAL
25  GOSUB 2: IF P =  > PE GOTO 123: REM POINTER EQUAL TO OR BEYOND END OF
     INTEGER PROGRAM
26 LA = P:BC = B: IF B > 127 GOTO 114: REM BYTE COUNT TOO LARGE, PROBABLE
     ATTACHED BINARY
27 TN = TN + 1: REM BUMP LINE NUMBERS, THEN MAKE LINE NUMBER STRING
28  GOSUB 2:L = B: GOSUB 2:L = B * 256 + L:B =  LEN ( STR$ (L)):N$ =  RIGHT$
    (( LEFT$ (LB$,(7 - B)) +  STR$ (L) + "  "),8)
29  REM BEGIN STATEMENT LINE PARSING WITH FIRST-BYTE DECISION
30 D = 0: GOSUB 2: IF B = 93 AND  NOT RF THEN  GOSUB 4: GOTO 34: REM SEPA
    RATE REM-GROUPS BY BLANK LINES
31  IF B = 93 AND RF GOTO 34
32  IF RF THEN RF = 0: GOSUB 4
33  REM RE-ENTRY POINT FOR NEXT BYTE IN STATEMENT DECISION
34  IF B < 128 GOTO 39: REM BYTE IS A TOKEN
35  IF B = 255 THEN B = 159: REM RUBOUT ($FF) BECOMES UNDERLINE BETWEEN B
    ARS
36 B = B - 128: IF B < 32 THEN B = B + 64:G$ = G$ +  CHR$ (124) +  CHR$ (
    B):B = 124: REM PUT CONTROL CHARACTERS BETWEEN BARS
37 G$ = G$ +  CHR$ (B): GOSUB 2: GOTO 34
38  REM    TOKENS
39  IF V(B) > 1 THEN G$ = "": GOTO 114: REM UNUSED TOKEN, PROBABLE BINARY
     PROGRAM ATTACHED SO GATHERING IS NULLED
40  IF B = 1 OR B = 3 THEN G$ = G$ + S$: GOTO 57: REM FORCE A NEW PRINT L
    INE ON E-O-L OR A COLON DELIMITER; SPACE ATTACHED TO PREVENT PRINT-L
    INE CRASH
41  IF B = 93 THEN TR = TR + 1:RF = 1:RS = 1: REM A "REM"
42  IF B = 37 AND  PEEK (P + 1) = 85 THEN G$ = G$ + T$(B):CF = 1: GOTO 57
    : REM FORCE A NEW LINE ON "THEN" FOLLOWED BY "FOR", SET CONDITIONAL
    FLAG
43  IF B = 85 THEN FF = 1: REM A "FOR"
44  IF B <  > 89 GOTO 51: REM SKIP AROUND A "NEXT"
45 FS = FS - 1:PT = P + 1: IF CF THEN FS = FS - 1: REM DECREMENT "FOR" SP
    ACER ON "IF" FLAG SET, BEGIN SCANNING AHEAD FOR 2 OR MORE ITERATORS
46 BT =  PEEK (PT): IF BT = 1 OR BT = 3 GOTO 49: REM NO OTHER ITERATOR
47  IF BT = 90 THEN FS = FS - 1: REM COMMA FOUND, DECREMENT "FOR" SPACER
48 PT = PT + 1: IF PT <  = (LA + BC) GOTO 46: REM CHECK AGAIN FOR ANOTHER
     COMMA WITHIN LINE
49  IF FS < 0 THEN FS = 0
50  REM GATHER TOKEN THEN TEST FOR A FOLLOWING 3-BYTE NUMBER GROUP
51 G$ = G$ + T$(B):L = B: GOSUB 2: IF V(L) = 0 GOTO 34: REM NO NUMBER SHO
    ULD FOLLOW
52  IF B < 176 OR B > 185 GOTO 34: REM THE $B0-$B9 FIRST-BYTE NOT THERE S
    O NO NUMBER FOLLOWS.  FALL-THROUGH IGNORES FIRST-BYTE AND DOES DECIM
    AL STRING CONVERSION
53  GOSUB 2:L = B: GOSUB 2:L = B * 256 + L:G$ = G$ +  STR$ (L): GOSUB 2: GOTO
    34
54  REM ADD EXTRA INDENT EACH SPLIT LINE, LIMITING ON "REM" STATEMENTS
55 TS = TS - 1:SF = 0:RS = RS + 1: IF RS > 2 THEN RS = 2
56  REM FIRST ENTRY TO PRINT-LINE BUILD, GET TOTAL INDENT SPACES PLUS SPL
    IT-POINT LOW LIMIT 'E'
57 TS = TS + 1:K = IM * (FS + RS):E = K + 13: IF K > 0 THEN G$ =  LEFT$ (
    BB$,K) + G$
```

(continued)

```
58  REM BUILD TOTAL PRINT-LINE STRING
59  IF  NOT D THEN C$ = N$ + G$
60  IF D THEN C$ = LB$ + G$
61  REM TEST FOR LONG LINE, SPLIT IF NECESSARY
62 K =  LEN (C$) - LL: IF K < 1 GOTO 74: REM NOT A SPLIT LINE
63 G$ =  RIGHT$ (C$,K):C$ =  LEFT$ (C$,LL):SF = 1
64  REM BEGIN SPLITTING WITH SEARCH FOR A SPACE
65 D = LL
66  IF  MID$ (C$,D,1) = S$ GOTO 72
67 D = D - 1: IF D > E GOTO 66
68 D = LL: REM SPLIT NEXT AT ARITHMETIC OPERATOR OR COMMA
69 K =  ASC ( MID$ (C$,D,1)): IF K = 42 OR K = 43 OR K = 44 OR K = 45 OR
    K = 47 OR K = 124 GOTO 72
70 D = D - 1: IF D > E GOTO 69: REM FALL-THROUGH IS NO SPLIT
71  GOTO 74: REM NEXT LINE IS SPLITTING INSTRUCTION
72 K = LL - D: IF K > 0 THEN G$ =  RIGHT$ (C$,K) + G$:C$ =  LEFT$ (C$,D)
73  REM TEST PAGE LINE-COUNT, INSERT SPACES AS REQUIRED, THEN PRINT
74  GOSUB 6:K =  LEN (C$): IF SF = 0 OR K < 2 OR RF THEN 77: REM FORGET M
    ARKING UNDERLINING ON "REM"S
75  IF  MID$ (C$,K,1) = S$ THEN C$ =  LEFT$ (C$,(K - 1)) +  CHR$ (95): REM
     PUT A TRAILING UNDERLINE AT LAST SPACE AS A MARKER FOR THE LEFT-HAND
     STRING
76  IF  LEN (G$) > 2 AND  LEFT$ (G$,1) = S$ THEN G$ =  CHR$ (95) +  RIGHT$
    (G$,( LEN (G$) - 1)): REM PUT A LEADING UNDERLINE AT FIRST SPACE OF
    RIGHT-HAND STRING AS A MARKER
77  GOSUB 17:K =  LEN (C$): PRINT M$; LEFT$ (C$,8);: GOSUB 18: PRINT  RIGHT$
    (C$,(K - 8)):K =  FRE (0): IF SF THEN D = 1: GOTO 55: REM PRINT REST
     OF A SPLIT LINE
78 RS = 0: IF FF THEN FS = FS + 1:FF = 0
79 D = 0:SF = 0:G$ = "": IF B = 1 GOTO 25: REM GET ANOTHER LINE NUMBER IF
     E-O-L, ELSE FALL THROUGH AND GET ANOTHER STATEMENT
80  GOSUB 2:D = 1: GOTO 34
81  REM INITIALIZATION OF VARIABLES
82  DIM T$(127),H$(4),V(127)
83  REM INITIAL VARIABLE SETTING HAS AN 80-CHARACTER WIDE PRINT LINE AND
    82-LINE PAGE LENGTH (INCLUDING HEADER, EXCLUDING 'CONTINUED' INDICAT
    OR); CHANGE LL AND LP AS DESIRED FOR OTHER FORMAT SIZE.
84 PE =  PEEK (75) * 256 +  PEEK (74) - 1:P = PS: REM PS = INTEGER PROGRA
    M START ADDRESS MINUS ONE, PE = INTEGER PROGRAM STOP ADDRESS MINUS O
    NE
85 B = 0:LL = 80:LP = 82:IM = 4:TN = 0:TS = 0:TR = 0:S$ = " ":X$ = "01234
    56789ABCDEF":C$ = "":G$ = "":N$ = "":M$ = "":LB$ = "          ":BB$ =
    "                                                  "
86  REM 'T$' ARRAY STRING CONSTANTS FOR PRINTING TOKENS
87  DATA " ","<<"," ",":","Load ","Save ","Con","Run ","Run","Del ",","," 
    New","Clear","Auto ",","Man ","Himem : ","Lomem : ","+","-","*","/
    "," = "," # "," >= "," > "," <= "," <> "," < "," And "," Or "," Mod
    ","^"," ","(",","," Then "
88  DATA " Then ",",","","","&","&","(","  "," ","(","Peek","Rnd","Sgn","Abs
    ","Pdl"," ","(","+","-","Not ","(","  = "," # ","Len(","Asc(","Scrn("
    ,",","","(","$","  ","(",",","","","","","","","","","","Text","Gr","Ca
    ll ","Dim ","Dim ","Tab "
89  DATA "End","Input ","Input ","Input ","For "," = "," To "," Step ","N
    ext ",","","Return"," Gosub ","*    "," "," GoTo ","If ","Print ","Pri
    nt ","Print","Poke ",","," Color = ","Plot ",","," HLin ",","," At ","
    VLin ",","," At ","VTab "
90  DATA " = "," = ",")"," ","List ",","," List","Pop","NoDsp ","NoDsp ","
    NoTrace","Dsp ","Dsp ","Trace","Pr # ","In # "
91  FOR K = 0 TO 127: READ T$(K): IF T$(K) = "&" THEN T$(K) =  CHR$ (34):
     REM ONE WAY TO GET A DOUBLE QUOTE INTO A STRING
92  NEXT
93  REM 'V' ARRAY CONSTANTS FOR TOKEN TESTING
94  DATA 2,0,2,0,0,0,0,1,0,1,1,0,0,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,2,1,1,1,0,0,0,0,0,0,1,2,2,1,0,0,0,0,0,0,2,1,1,1,1,1,0,0,0,0,1,1,1,0,2,
    1,1,1,0,1,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,1,1,1,0,0,1,0,0,1,1,0,1,0
95  DATA 1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,2,1,1,0,0,0,0,0,0,0,0,1,1
96  FOR K = 0 TO 127: READ V(K): NEXT
97  REM SCREEN PROMPTS AND OPERATOR ALTERNATES
98  HOME : TEXT : VTAB 2: HTAB 12: INVERSE : PRINT " ILISZTER ACTIVE ": NORMAL
     : VTAB 4:L = PS + 1: GOSUB 21: PRINT "START OF INTEGER PROGRAM: ";A$
    :L = PE: GOSUB 21: PRINT "  END OF INTEGER PROGRAM: ";A$: REM OPTION
    AL TO CHECK APPROXIMATE ADDRESS LOCATION
99  PRINT : INPUT "PROGRAM NAME: ";H$(1): INPUT "  PROGRAMMER: ";H$(2): INPUT
     "        DATE: ";H$(3): REM REQUIRED FOR HEADER ON EACH PAGE
100  PRINT : PRINT "WANT DIFFERENT START ADDRESS ?": GET A$: IF A$ <  > "
    Y" GOTO 103
101  INPUT "   START ADDRESS (HEX): ";A$:D = 1:BT = 0: FOR K =  LEN (A$) TO
    1 STEP  - 1: FOR E = 1 TO 16: IF  MID$ (A$,K,1) =  MID$ (X$,E,1) THEN
    BT = D * (E - 1) + BT
102  NEXT E:D = D * 16: NEXT K: PRINT :P = BT - 1:L = BT: GOSUB 21: PRINT
     "   HEX ADDRESS = ";A$;"   CHANGE ?": GET A$: IF A$ = "Y" GOTO 101
103  PRINT : PRINT "NO LEFT MARGIN, WANT ONE ?": GET A$: IF A$ = "Y" THEN
     INPUT "  MARGIN SPACES: ";K: IF K > 0 AND K < 49 THEN M$ =  LEFT$ (
    BB$,K):LL = LL - K
104  REM REMINDER FOR PRINTER SET-UP
```

(continued)

line 25. Integer does not allow a byte count larger than 127. (The actual number is 255. The 127-byte limit (line 26) is for print-line reconstruction, usually longer than source-code line length.) A byte count that is too large will jump to the binary-insert routine at line 114. Line numbers up to 65535 will output whether they are actual line numbers or a chance byte-pair in binary. A test of number magnitude was included in an earlier version but then disregarded due to the large number of starting prompts.

Remark checking in lines 30 to 32 is part of the blank-line separation for REMs. Removing separation would delete all but the "D=0" statement; D must remain for line number printing.

Statements begin parsing in line 34. ASCII characters are restored for printing but control characters are uppercase between vertical bars. Source code rubouts are included to fill out lines in certain programs.[2]

Token parsing begins at line 39 with a test for unused tokens. The added space to the gather string at line 40 prevents a crash during a binary code test; a rare condition, but it was found in two listings.

Three programs were found with a FOR loop starting on an IF-true condition. Line 42 solves indenting and restoration on this rare case. Integer normally executes only one IF-true condition but, apparently, a FOR loop will execute until completed.

Two or More Iterators

The printout indent restoration of statements such as "NEXT J,K" is solved by the search routine in lines 45 to 49. Of several comma tokens, only decimal value 90 is the comma in a multiple-variable NEXT statement. This search and find will restore global indenting of FOR loops. It can also be patched into the original LISZTER to solve an oversight.[7]

Numbers Following You?

Some tokens allow following numeric constants. Integer BASIC flags a numeric constant with a $B0 to $B9 prefix (ASCII numbers 0 to 9). The test in lines 51 and 52 check for token and prefix, ignoring the prefix if it exists.

Line 53 builds the numeric constant string and gathers it in G$. Flow must return to line 34 afterwards. The next byte can be either a token or a char-

acter; variable names are ASCII characters.

The Final Print Line

Lines 55 to 80 form the output print line, splitting and indenting as in the original LISZTER. First-priority split is still a space, but second-priority split has a vertical bar added to line 69. Control characters seem to be used more in Integer. At this point they have been converted to upper-case letters between bars and will not upset printer control.

The complex print statement group in line 77 is solely for the italics capability of the Epson printer. A single-character-set printer can substitute a simple "PRINT M$; C$" for both GOSUBs and PRINTs.

Possible Binary?

An IF-true test at lines 26 or 39 indicates something is wrong with the Integer source code. More than likely it is due to embedding binary code with integer. The routine at lines 114 to 120 checks this condition.

Variable LA is made up of the address of each new source line number start. That address is converted to hexadecimal and printed with the "Possible Binary From" indicator. A search now begins for any byte group meeting the following: the group is below HIMEM, the group is less than 128 bytes long, and the end-of-line byte value is found from the first-byte address plus value. A successful search will print the byte group *last* address in hex to complete the indicator, then return to line 25 for a new source line number.

The indicator may be printed several times before a correct source line is found. The number of prints will be dependent on binary content but a correct Integer source line will always follow embedded binary.

A possibility is a bit error in memory that can yield another possible binary print line. An advantage is that a printout will show beginning and ending addresses for closer examination.

An "attached" binary program will terminate at highest available memory. The possible binary last print will indicate this as $95FF with standard DOS.

Alternatives

A purely Integer version of ILISZTER can be written by translation of the general structure. Page zero locations $69 through $6D can be used for

(continued)

```
105  HOME : INVERSE : PRINT " SET PAPER TO TOP OF FORM ": PRINT "
       THEN          ": PRINT "      TURN ON PRINTER     ": NORMAL : PRINT
     : GET A$
106  REM SET SCREEN WIDTH, TURN ON PROPER PORT
107  HOME : POKE 33,30: PR# 1
108  REM CONTROL CHARACTERS FOR MX-80 WITH "GRAPPLER" CARD.  CHR$(9)=CTRL
     -I, CHR$(27)=ESC
109  PRINT  CHR$ (9)"82N" CHR$ (27)"0" CHR$ (9)"I"
110  REM
111  REM    SET-UP TO START FIRST PRINT PAGE
112 LC = 6:PC = 1:D = 0: GOSUB 11: GOTO 25
113  REM POSSIBLE-BINARY INSERT/ADDITION ROUTINE
114 RF = 1: GOSUB 18:L = LA: GOSUB 21: GOSUB 6: PRINT M$;LB$;" >>> Possib
     le Binary from ";A$;" to ";
115  IF P > PE GOTO 121
116  IF B > 127 THEN  GOSUB 2: GOTO 115: REM BYTE-COUNT TOO LARGE
117 PT = P + B - 1:BT =  PEEK (PT): IF PT > PE GOTO 121
118  IF BT <  > 1 OR B < 5 THEN  GOSUB 2: GOTO 115: REM NO E-O-L OR BYTE-
     COUNT TOO SMALL
119  IF LA = (P - 1) THEN  GOSUB 2: GOTO 115: REM AVOID REPETITION; SOMEH
     OW THE POINTER DIDN'T ADVANCE
120 P = P - 1:L = P: GOSUB 21: PRINT A$:D = 0:G$ = "": GOTO 25: REM RETUR
     N TO LINE-NUMBER START
121 L = PE: GOSUB 21: PRINT A$
122  REM ENDING ROUTINE
123  GOSUB 4: GOSUB 17: PRINT M$;LB$;"End of Listing"
124  REM OPTIONAL STATISTICS
125  GOSUB 4: PRINT M$;"Program Length = ";(PE - PS);" Bytes,  Total of "
     ;TN;" Line Numbers": GOSUB 4: PRINT M$;(TS - TR);" Total Non-Rem Sta
     tements,  ";TR;" Total Remarks"
126  REM TURN OFF PRINTER, RESET SCREEN AND SHOW COMPLETION
127  PR# 0: POKE 33,40: HOME : VTAB 12: HTAB 10: INVERSE : PRINT " END OF
      ILISZTING ": NORMAL : END
128  REM  "ILISZTER" program to re-format INTEGER BASIC listing prints
129  REM    by Leonard H. Anderson     Version 2.8.8,  15 May 1982
130  REM        lower case and italics for MX-80 & "GRAFTRAX"
131  REM   Possible-Binary routines added to 2.8.1 (21 March 1982)
132  REM
133  REM DESCRIPTION OF VARIABLES:
134  REM
135  REM A$  TEMPORARY STRING, PARTLY FOR HEX CONVERSION
136  REM B   PROGRAM BYTE VALUE IN DECIMAL
137  REM BB$ 'BIG BLANK' STRING OF 48 SPACES
138  REM BC  BYTE-COUNT OF A LINE, DECIMAL
139  REM BT  TEMPORARY PROGRAM BYTE VALUE IN DECIMAL
140  REM CF  "IF" FLAG: SET ONLY ON "IF" FOLLOWED BY "FOR"
141  REM C$  CHARACTER AND TOKEN STRING TO BE PRINTED
142  REM D   TEMPORARY, PARTLY FOR 'DIRECTION'
143  REM E   TEMPORARY, PARTLY FOR SPLIT-LINE LIMITS
144  REM FF  "FOR" FLAG: 1 = "FOR" STARTED, 0 = NO "FOR"
145  REM FS  "FOR" INDENT SPACE COUNTER
146  REM G$  'GATHER' STRING TO BUILD A STATEMENT
147  REM H$  HEADER ARRAY FOR PRINT-PAGE TITLE
148  REM IM  INDENT SPACE MULTIPLIER
149  REM K   TEMPORARY
150  REM L   TEMPORARY, PARTLY FOR LOW-BYTE VALUE
151  REM LA  LINE NUMBER BEGINNING ADDRESS
152  REM LC  LINE COUNTER FOR PAGINATION
153  REM LL  LINE-LENGTH CONSTANT
154  REM LB$ 'LITTLE BLANK' STRING OF 8 SPACES
155  REM M$  LEFT MARGIN SPACING STRING
156  REM N$  LINE NUMBER STRING
157  REM P   POINTER TO PROGRAM BYTE, DECIMAL
158  REM PC  PAGE COUNTER FOR PRINT-PAGE HEADER
159  REM PE  INTEGER PROGRAM END ADDRESS, DECIMAL
160  REM PS  INTEGER PROGRAM START ADDRESS, DECIMAL
161  REM PT  TEMPORARY POINTER TO PROGRAM BYTE, DECIMAL
162  REM RF  "REM" FLAG: 1 = "REM" STARTED, 0 = NO "REM"
163  REM RS  "REM" INDENT SPACE COUNTER
164  REM SF  SPLIT-LINE FLAG: SET IF PRINT LINE MUST BE SPLIT
165  REM S$  SINGLE-SPACE STRING
166  REM TN  TOTAL LINE NUMBER COUNTER
167  REM TR  TOTAL REMARKS COUNTER
168  REM TS  TOTAL STATEMENTS COUNTER
169  REM T$  TOKEN STRING ARRAY
170  REM V   ARRAY FOR TOKEN EVALUATION:
171  REM       0 = NO BINARY NUMBER FOLLOWS TOKEN
172  REM       1 = A 3-BYTE BINARY NUMBER FOLLOWS
173  REM       2 = UNUSED/INTERNAL, DO NOT PRINT
174  REM X$  HEX CHARACTER STRING FOR CONVERSIONS
```

Make ILISZT

```
200      *   TEXT FILE GENERATOR FOR "ILISZT"
210      *   VERSION 3.0, 16 APRIL 1982  LHA

220  D$ = "|D|"
230  Print D$;"OPEN ILISZT"
240  Print D$;"WRITE ILISZT"

250      *   MAKE INTEGER LOMEM POINTER HOLD ENDING OF INTEGER PROGRAM

260  Print "POKE74,PEEK(76)"
270  Print "POKE75,PEEK(77)"

280      *   MAKE INTEGER HIMEM POINTER HOLD START OF INTEGER PROGRAM

290  Print "POKE76,PEEK(202)"
300  Print "POKE77,PEEK(203)"
310  Print "RUN ILISZTER"
320  Print D$;"CLOSE"
330  End
```

pointer re-arrangement as in the LISZT predecessor. Total code will probably exceed the 4.5K bytes of a "REM-less" ILISZTER in Applesoft. MAKE ILISZT can be either language; the created text file will be the same.

ILISZTER has successfully handled a 23K Integer program printout plus one program with two embedded binary code sections.

References

1. Apple Pugetsound Program Library Exchange "public domain" disks (members only). Printouts of 1057 programs fill three large loose-leaf notebooks; about a quarter are Integer.
2. "Higher Text" by Ron and Darrell Aldrich, *Call —A.P.P.L.E.* version. One Integer program has two binary embedments.
3. *MICRO on the Apple*, Volume 1, MICRO INK, pages 198-203.
4. *PEEKing at Call —A.P.P.L.E.*, Volume 2, pages 44-61, Apple Pugetsound Program Library Exchange, 1979.
5. *What's Where in the Apple!*, William F. Luebbert, MICRO INK. For address locations only.
6. "The Inspector," Omega Microware, Inc., is one example of a disk or memory byte-changer utility. Although the author has upper-/lower-case conversion on the keyboard, this utility was used to correct typos in ILISZTER's DATA statements.
7. "LISZT with Strings," Richard F. Searle, Don Cohen, Leonard H. Anderson, MICRO, May 1982, listing 2 on page 41. The easiest patch is a GOSUB in line 45 just after the "CF = 1" statement; the subroutine would look for a delimiter comma in ASCII, such as "BT = 44", to decrement the FOR spacer.

You may contact Mr. Anderson at 10048 Lanark St., Sun Valley, CA 91352.

MICRO™