Mattel Chooses 6502

Teach Your Atari To Talk

NCC And The APPLE III

COMPUTE.

$2.00
July/August, 1980
Vol. 1, Issue 5

The Journal for Progressive Computing™

The Resource Magazine For Apple, Atari, and Commodore

Commentary: Business Applications Analysis. . . The Missing Step

Atari's Marketing VP Profiles The Personal Computer Market

Plotting With The Commodore 2022 Printer

Assembly Language Programming With UCSD Pascal

TRENDCOM 200

High-Speed Intelligent Printer

40 characters-per-second

80 characters per line

Upper and lower case

Continuous graphics at 60 dots per inch

Microprocessor controlled

Bidirectional look-ahead printing

Automatic "wrap-around"

Quiet operation

The Trendcom 200 is a high speed thermal printer offering the combination of text printing at 80 characters per line and continuous graphics at 60 dots per inch. In the text mode, upper and lower case data are printed at 40 characters per second. The 5 x 7 characters provide clear readable copy on white paper; no hard to find, hard to read aluminized paper.

In the graphics mode, seven bits of each byte correspond to the seven dots in each of the 480 print positions per line. Since the computer driving the printer has full control over every print position, it can print graphs, bar charts, line drawings, even special and foreign language symbols. Despite its low cost, the Trendcom 200 is a true intelligent printer with full line buffering and bidirectional look-ahead printing. After one line has been printed left to right, the internal microprocessor examines the next line to choose the shortest print direction. The microprocessor also provides a built-in self-test mode for easy verification of proper operation.

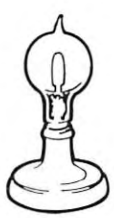
High reliability is designed in: The thick film thermal print head has a life expectancy of 100,000,000 characters. Two DC stepping motors provide positive control of the print head and the paper drive, the printer's only driven parts. The absence of gears and solenoids also makes the printer extremely quiet; the only noise is the rustling of the paper advancing.

——— **Skyles PAL-80 printer(s) complete** with 2½ foot interface cable to attach to my PET at **$675.00 each.*** (Plus $10.00 shipping and handling.) I also will receive a test and graphics demonstration tape at no additional charge and over 150 feet of 8½ inch wide black on white thermal paper.

——— rolls of **8½ inch wide** by 85 ft. long thermal paper (black ink) at $5.00 each; or cartons at 10@ $45.00.

Visa, Mastercharge orders call (800) 538-3083

California orders please call (408) 257-9140

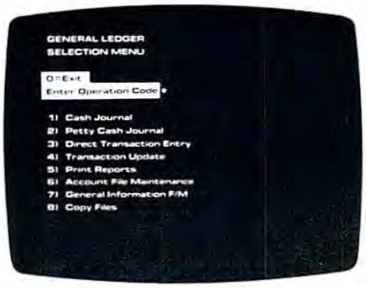


Skyles Electric Works

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

Professional Business Software

For The Commodore 32K Microcomputer System
With 2040 Dual Drive Disk & 2022 Tractor Feed Printer



General Ledger

- Holds Up To 300 Accounts.
- Accepts Up To 3000 Transactions Per Month.
- Cash Disbursements Journal, Cash Receipts Journal, and Petty Cash Journal for simplified data entry.
- Maintains Account Balances For Present Month, Present Quarter, Present Year, Three Previous Quarters, And Previous Year.
- Complete Financial Reports Including Trial Balance, Balance Sheet, Profit & Loss Statement, Cash Receipts Journal, Cash Disbursements Journal, Petty Cash Journal and more.
- Accepts Postings From External Sources Such As Accounts Payable, Accounts Receivable, Payroll, Etc. **$295.00**

Accounts Payable

- Interactive Data Entry With Verified Input And Complete Operator Prompting.
- Automatic Application Of Credit And Debit Memos.
- Maintains Complete Purchase Records For Up To 200 Vendors.
- Invoice File Accepts Up To 400 Invoices.
- Random Access File Organization Allows Fast Individual Record Updating.
- Multiple Reports Provide A Complete Audit Trail.
- Check Printing With Full Invoice Detail.
- Full Invoice Aging.
- Automatic Posting To General Ledger **$195.00**

Accounts Receivable

- Maintains Invoice File For Up To 300 Invoices.
- Accomodates Full Or Partial Invoice Payments.
- Customer File Maintains Purchase Information For Up To 1000 Customers.
- Allows For Automatic Progress Billing.
- Provides For Credit And Debit Memos As Well As Invoices.
- Prints Individualized Customer Statements.
- Interactive Data Entry With FullOperator Prompting.
- Complete Data Input Verification And Formating.
- Automatic Posting To General Ledger **$195.00**

Payroll

- Maintains Monthly, Quarterly, And Yearly Cumulative Totals For Each Employee.
- Payroll Check Printing With Full Deduction And Pay Detail.
- Sixteen Different Reports Including W2 And 941.
- Interactive Data Entry With Easy Correction Of Entry Errors.
- Automatic Data Verification.
- Complete Job Costing Option With Cumulative Totals And Overhead Calculations.
- Random Access File Organization For Fast Updating Of Individual Records.
- Automatic Posting To General Ledger $350.00

Structured around the time tested and reliability proven series of business software systems developed by Osborne and Associates, these programs have been designed to fill the need of a comprehensive accounting package for the new Commodore PET micro computer system. Each program can either stand alone, or be integrated with the others in a total software system.

Designed with the first time user in mind, these programs lead the operator through step by step, verified data entry. It is impossible to 'crash' a program due to operator error or invalid data input. Design consistency has been maintained from program to program to greatly increase operator familiarity and confidence.

Documentation, normally a problem for small systems users, is provided by the comprehensive series of Osborne and Associates user manuals. These three manuals together total over 800 pages of detailed step by step instructions written at three levels for DP Department Managers, Data Entry Operators, and Programmers. You don't have to worry about getting 'promises' instead of documentation because the documentation was written before the programs were developed. A second set of manuals details any changes required during conversion. Each program provided on disk with complete documentation. Packaged in a handsome three ring binder with pockets and twelve monthly dividers for convenient storage of reports.

See your nearest Commodore dealer for a demonstration of this outstanding business software system.

CMS Software Systems

5115 MENEFEE DRIVE • DALLAS, TX 75227 • 214-381-0690

Table of Contents

July/August 1980. Volume 1. Issue 5

Page 13

Page 16

Page 28

Page 41

Page 73

COMPUTE. The Journal for Progressive Computing is published six times each year by Small System Services, Inc., P.O. Box 5406, Greensboro, NC 27403 USA. Phone: (919) 272-4867. Editorial Offices are located at 900 Spring Garden Street, Greensboro, NC 27403.

Domestic Subscriptions: 6 issues, $9.00. Send subscription orders or change of address (P.O. Form 3575) to Circulation Dept., COMPUTE. Magazine, P.O. Box 5406, Greensboro, NC 27403. Controlled circulation postage paid at Greensboro, NC 27403. ISSN 0194-357X.

Robert C. Lock, Publisher/Editor
Joretta Klepfer, Associate Editor
Carol Holmquist Lock, Circulation Manager
J. Gary Dean, Art Direction/Production Assistance

COMPUTE receives continuing editorial assistance from the following persons:
Harvey Herman, University of North Carolina at Greensboro
Jim Butterfield, Toronto, Canada
Larry Isaacs, Raleigh, NC

The following writers contribute on a regular basis as Contributing Editors:
Al Baker, 2327 S. Westminster, Wheaton, IL 60187
Gene Beals, 115 E. Stump Road, Montgomeryville, PA 18936
Len Lindsay, 1929 Northport Drive #6, Madison, WI 53704
Roy O'Brien, P.O. Box 426, Beaumont, CA 92223

Subscription Information (6 Issue Year):
COMPUTE. Circulation Dept.
P.O. Box 5406
Greensboro, NC 27403 USA

U.S. $9.00
Canada $12.00 (U.S. funds)
Europe: Air Subscription, $22.50 (U.S. funds) if ordered direct, or available in local currency from the following distributors:

United Kingdom
Contact L. P. Enterprises,
8-11 Cambridge House
Cambridge Road
Barking, Essex
England IG1 18NT

Germany,
Switzerland,
Austria
Contact Ing. W. Hofacker GMBH
8 Munchen 75
Postfach 437
West Germany

Canadian Retail Dealers should contact:
Micron Distributing
409 Queen Street West
Toronto, Ontario M5V 2A5
(416) 363-6058

The Editor's Notes

Robert Lock

Interesting Months Ahead...

As the small computer market expands, the various manufacturers are lining up a host of new support products aimed at expanding their own coverage of the market.

Apple formally introduced the **Apple III** at the NCC in California. You'll find a full review on the "Appletivities" and the Apple III in this issue's Apple Gazette.

From **Commodore,** we can expect (in the next 30 days), initial deliveries of their **modem**/communications module. Hard on the heels of this new hardware comes **PASCAL** for the Commodore line. Not the UCSD version, this Pascal, developed in England by Commodore UK, requires a 32K machine. Price was not available at press time, but look for initial deliveries by the end of July.

A Host of New Peripherals

From Commodore, we can look forward to a whole group of new goodies during the summer and fall: A fifteen character per second, bi-directional **daisy wheel printer** in the $1500.00 price range.
Commercial Basic with print using and floating point decimal arithmetic.
A 10 megabyte **hard disk drive,** a family of 8-inch disk drives, and a lower cost single disk drive.
A monitor-less **color machine** for well under $1000.00 with such user oriented features as programmable sound and color function keys.
We've also heard that a **132 column printer** is on the way from Commodore... in short a host of new support products.

It seems that the **Atari** systems are starting to pick up some momentum in the consumer market, and Atari appears to be moving to increase their target area. Chief among these high end introductions are their 80 column printer and dual density, dual drive disk. Their I/O module, scheduled to sell for $219.95, is being released. It provides four serial I/O ports, and one parallel port. The new **80 column printer,** whose specifications are quite similar to the Centronics 737, lists at $995.95 and requires the parallel port on the I/O module. Among their other new or soon to be releases are the **326K dual density, dual drive disk** ($1495.00), the modem ($200.00), and the Atari light pen ($75.00). They're also showing off the **Accounting Package** now, and dealers should have it available by mid to late summer.

One other item of interest... the Atari 800 system has been reconfigured as of June 1, and incorporates these changes: The price is still $1080.00. The system now comes with 16K standard rather than 8K standard. It no longer includes the tape cassette, the Educational Master Cartridge, or the Introduction to Programming tape.

It May Be Months Away But...

Mattell is moving into the expanding computer market as well. Their Intellivision line is the vehicle chosen, and the microprocessor? None other than our favorite, the 6502. The unit will provide Microsoft Basic as well, another plus in COMPUTE's book of *How to Write General Articles that Help All Readers...* While the formal introduction is months away, we're mentioning it because it helps show the future strength of the 6502. The Dr. Chip Interview in this issue provides the only details we have as yet.

Speaking of Microsoft: Atari, by the way, is rumored to be bringing out a Microsoft Basic for their own machines. Ah, yes. COMPUTE votes for it.

All the Activity is Not on the Inside

We've reviewed **Visicalc** in this issue. While our version is the Apple one, you Commodore and Atari owners will be happy to know that Visicalc will be available for your machines this summer. See the box with the review for "press time" details.

Color For The Commodore

The Electric Crayon is coming! Regency Educational Systems (7407 Brantfield Drive, Dallas, TX 75248) has just introduced their **Regency Crayon** for the Commodore machines. It is said to run on old and new ROMs, etc. With $495.00 and a color TV or monitor, viola!... a color PET. The set-up comes with 6K of refresh memory, cables, and software that runs in less than 1K of memory. Users end up with 10 different graphics modes and eight colors. Resolution on the graphics modes ranges from 32 by 16 to 256 by 192. One is on the way to us and we'll have a full review next issue. $495's not cheap, but if you ever wanted a more colorful life for your PET, here you are.

Business Software Reviewers Needed

I realize that those of you using your machines for business applications are quite busy. We still need qualified practicing business people to review software. If you'd like to be on the list, send me a note with name, address, business applications/needs, and system configuration. For your time, you just might end up with a free review copy of some package you've been looking for.

This Didn't Come In Time for Dr. Chip but...

VIC was "market screened" by Commodore in a private booth at the Consumer Electronics Show in Chicago. Who's VIC? VIC, or Video Interface Computer, is a 6502 based self-contained computer that's small enough to fit into a briefcase, hook up to your TV or monitor, and is expected to sell for something between $300.00 and $400.00. It will not be available until sometime in the Fall (it's currently in prototyping), but here's my best guess at the final minimum configuration:

- 4K RAM
- 20 Columns by 25 Lines
- Small keyboard, similar to that of the early PET, but redesigned
- "Simple" PET Basic
- Extensive interfacing capability for joysticks, lightpens, and sound synthesis
- Two modes of graphics resolution

A color version is expected to be available for a higher price, with other options including 40 columns, 8K, full keyboard, and full BASIC. COMPUTE is happy to see the 6502 community grow!

One Last Update...

The new Panasonic machine is 6502 based. It comes with 1K (yes Jim, 1K), but has an expansion module for 12K. Pricing (unconfirmed at this point) is around $400.00. It's tiny... apparently not much larger than (for example) those language translators we've seen advertised lately. COMPUTE assumes it won't have Microsoft as standard equipment, but we can always hope! Next issue we'll shoot for a comprehensive 6502 new machine profile, complete with photos.

The Missing Columns

The Learning Lab, by Marlene Pratto, will return in the next issue. Pet Programs on Tape Exchange by Gene Beals will also return in the next issue. ©

Program Listings for COMPUTE

Cursor control characters will appear in source listings as shown below:

```
h=HOME         , ĥ=CLEAR SCREEN
↓=DOWN CURSOR  , ↑=UP CURSOR
→=RIGHT CURSOR, ←=LEFT CURSOR
r=REVERSE      , r̂=REVERSE OFF
```

Graphics (i.e. shifted) characters will appear as the unshifted alphanumeric character with an underline. This does not apply to the cursor control characters. The Spinwriter thimble doesn't have a backarrow symbol, so a "~" is used instead.

The "¬" is used to indicate the beginning of a continuation line. It is also used to indicate the end of a line which ends with a space. This prevents any spaces from being hidden. ©

The Reader's Feedback

Robert Lock

My thanks to all of you who responded to the first round of Editor's Feedback cards. Voted Best Article in Issue 4 was Elizabeth Deal's "Big Files on A Small Computer", followed closely by "Block Access Method Map" (Tom Conrad) and "PET Get With Flashing Cursor" (Gary Greenburg). The Editor's Feedback cards are here to stay. You'll find one bound into the center of the magazine. With this issue, we initiate a continuing forum through which I'll respond to your feedback cards. Keep them coming!

"I'm developing software for agricultural use. Would like nput from, and communication with others interested in this area. How about a reader response column?"

Okay. If you're currently working on software for a specific applications area, or looking for some, drop me a note.

"Please make longer programs available on tape..."

Done. At least for PET. Some of our longer programs will now be available from the PET Software on Tape Exchange (See the table of contents).

"... step by step machine/assembly language usage examples..."

"... simple machine language programs... every step shown, preferably the "how" and not "why"."

We're working on this one and hope to introduce a general purpose series in the next issue.

"More business applications reviews..."

"We are waiting for your review of Rich Daley's Mail List program. It blows up in three out of three systems; while it is very powerful, the original and... new version... don't seem to stand the test of time... We are waiting here for your review to see if you will plug his product because of his ad money or shoot it down."

Neither. At least 'till next issue. Apparently Richard ran afoul of various ROM changes (DOS problems) and thus the bugs. I've personally talked to a field test site that is up and running with the current version. They're engaged in extensive re-testing and debugging. This is the version Richard intends to release, along with an almost 70 page booklet of supporting documentation. You, and two other people, wrote about this. Richard has assured me that all owners of the new program will be "retrofitted" with the new version, and new documentation, at no charge.

We expect to review his new software next time, as well as that of several others recently introduced. Several of the newer mailing programs are designed to interact with Commodore's Word Pro III.

"I would like to know more about available commands for Atari. Included manual wasn't very complete and was hard to read."

Most of you should have the newer documentation by now. COMPUTE will continue to present as much information as we can generate to help. You new Atari owners should remember to send your programming articles and advice to help other readers as well.

At press time, we're still in the midst of collecting the results on the Z-80 software question posed on the last Editor's Feedback card. Many of you raised the question, "Why not develop a CP/M for the 6502?" We'll pass full feedback along to Microsoft Consumer Products, Commodore and you. Highlights of the Feedback card this time include Best Article and "Most Desired Peripheral". Thanks for taking the time (and the ten cent stamp) to respond.

And finally, among the "other suggestions":

"I like all of COMPUTE; Keep going! Make it monthly!"

We're considering it.

"Let's get a high-level compiler language for the PET..."

"Suggest more space on this card..."

"My COMPUTE to arrive on time..."

More next time... Robert ©

Coming In Issue 6 (September/October):

Loading APPLESOFT Programs on Your PET

Telecommunications Overview

Waterloo Structured BASIC for the PET

Atari Disk I/O

A User's Guide To Visicalc: Apple, Atari and Pet

The Educational Gazette

Business Software Review: The Osborne/McGraw Hill Series for Apple and Commodore

And a whole lot more!

COMPUTER BOOKS THAT TALK YOUR LANGUAGE... FROM BEGINNER TO EXPERT.

Howard W. Sams is the source for the best and latest in computer technology. The world's leading authors. Clear, concise, easy-to-understand text. Fully illustrated for better comprehension. Actual programs for today,s personal computers. Informative and entertaining. Titles for everyone...from the novice to the hobbyist to the expert.

THE STARTERS

Entry-level books for the beginner can bring the novice up to a high level of knowledge quickly.

☐ **THE HOWARD W. SAMS CRASH COURSE IN MICROCOMPUTERS**
The short-cut to understand microcomputers. Spend a few hours with this self-instructional course and gain a working knowledge of all aspects of microcomputers from fundamentals to operating systems, programming, peripheral equipment and much, much more. 264 pages. No. 21634. By Frenzel. $17.50.

☐ **MICROCOMPUTERS FOR BUSINESS APPLICATIONS**
A microcomputer expert discusses the benefits of microcomputer systems for small business owners. Explains types of microcomputers available, points out pitfalls to avoid, and defines computer-related terms in easy-to-understand language. 256 pages. By Barden. No. 21583. $8.95.

☐ **A GUIDEBOOK TO SMALL COMPUTERS**
If you are contemplating buying a small computer system for home or business, this book can save you time and trouble. Covers fundamentals of small computers including hardware and software. Includes information on manufacturers' warranties, service and a handy directory of small computer manufacturers. By Barden. No. 21698. $5.95.

☐ **GETTING ACQUAINTED WITH MICROCOMPUTERS**
Provides a complete working knowledge of microcomputer organization, operation, and programming. 288 pages. By Frenzel. No. 21486. $8.95

☐ **FUNDAMENTALS OF DIGITAL COMPUTERS (2nd ED.)**
Unravels the mysteries of computers and programming. 320 pages. By Spencer. No. 21534. $9.95.

☐ **INTRODUCTION TO MICROCOMPUTERS FOR THE HAM SHACK**
Gives the radio amateur an opportunity to be in the forefront of utilizing and developing techniques in "computercations." 96 pages. By Helms. No. 21681. $4.95.

LEARN ABOUT MICROCOMPUTERS, SAVE 10%...AND GET A FREE GIFT!

☐ **THE COMPUTER PRIMER LIBRARY**
This 3-volume "starter" library was selected for the novice who wants to learn about microcomputers. Concepts. Programming. Computer graphics. Languages. Games. By Mitchell Waite and Michael Pardee—two of the leading authors in the field of computers.

MICROCOMPUTER PRIMER—368 pages of up-to-date facts on microcomputers and computer concept.

COMPUTER GRAPHICS PRIMER—Over 20 programs for creating your own video graphics (complex drawings, plans, maps, and schematics) on the new 6502-based personal computers.

BASIC PROGRAMMING PRIMER—Provides a working knowledge of BASIC, the most popular computer language today. Explains fundamentals of BASIC, program control, organization, additional functions, variations, and includes a game program.

3-VOLUME LIBRARY PACKAGE ONLY $30.47—includes a FREE COPY of YOUR OWN COMPUTER—an introduction to home computers. No. 21706

THE PROGRAMMERS

A computer is only as good as the software. Sams offers several books that explain programming and tell you how to develop, debug and test programs for your equipment.

☐ **PROGRAMMING & INTERFACING THE 6502, WITH EXPERIMENTS**
Dr. De Jong has compiled 14 chapters of information and programs for 6502-based microcomputer systems. Experiments and examples are written so that a KIM, AIM or SYM system may be used to reinforce the material in each chapter. 448 pages. By De Jong. No. 21651. $13.95.

☐ **HOW TO PROGRAM AND INTERFACE THE 6800**
Discusses the internal structure, instruction set, and programming techniques for the 6800. Experiments demonstrate "real world" applications. By Staugaard. 432 pages. No. 21684. $13.95.

☐ **DBUG: AN 8080 INTERPRETIVE DEBUGGER**
Covers program operation and how its applied to program development and testing. 112 pages. By Titus and Titus. No. 21536. $4.95.

☐ **6502 SOFTWARE DESIGN**
Tells how to program for the 6502 assembly language. 288 pages. By Scanlon. No. 21656. $10.50.

☐ **8080/8085 SOFTWARE DESIGN—2 VOLUMES**
☐ VOLUME 1 gives you an introduction to assembly language programming. 336 pages. By Titus, Larsen, Rony, & Titus. No. 21541. $9.50

☐ VOLUME 2 is a unique, one-of-a-kind computer science book for the design engineer. Written in Intel machine code. 352 pages. By Titus, Larsen, & Titus. No. 21615. $9.95.

☐ TWO-VOLUME SET No. 21659 ONLY $17.50

☐ **TEA: AN 8080/8085 CO-RESIDENT EDITOR-ASSEMBLER.**
256 pages. No. 21628. By Titus. $8.95.

THE COMPUTER TECHNOLOGY LEADERS

Even if you're an avid computer buff, rapidly changing technology makes it difficult to stay current. Sams has a complete line of books and reference publications to help you keep abreast of the current state-of-the-art.

☐ **MICROCOMPUTER-ANALOG CONVERTER SOFTWARE & HARDWARE INTERFACING.**
Concepts and techniques of interfacing digital computers to analog devices. 288 pages. By Titus, Titus, Rony, & Larsen. No. 21540. $9.50.

☐ **TRS-80 INTERFACING.**
192 pages. No. 21633. By Titus. $8.95.

☐ **INTERFACING & SCIENTIFIC DATA COMMUNICATIONS EXPERIMENTS.**
160 pages. By Rony, Larsen, Titus, &Titus. No. 21546. $5.95.

☐ **MICROCOMPUTER INTERFACING WITH THE 8255 PPI CHIP.**
224 pages. By Goldsbrough. No. 21614. $8.95.

☐ **Z-80 MICROPROCESSOR PROGRAMMING & INTERFACING—2 VOLUMES.**
800 pages. By Nichols, Nichols, & Rony. No. 21611. $21.95.

☐ **INTRODUCTORY EXPERIMENTS IN DIGITAL ELECTRONICS AND 8080A MICROCOMPUTER PROGRAMMING AND INTERFACING—2 VOLUMES.**
912 pages. By Rony, Larsen, & Titus. No. 21552. $20.95.

A BEST-SELLER... A TREMENDOUS VALUE!

This is the most comprehensive reference available on all phases of computers and their applications. Contains over 22,000 definitions, acronyms, and abbreviations. 14 appendices. Hardbound. 944 pages. By Sippl & Sippl. No. 21632. $29.95.

☐ **COMPUTER DICTIONARY & HANDBOOK (3RD EDITION)**

THE LOGIC COOKBOOKS

☐ **LANCASTER'S COOKBOOK LIBRARY**
A famous resource for all electonics buffs who want to know what makes TTLs, CMOS, and active filters cook—what they are, how they work and how to use them. By Don Lancaster, one of the most popular authors in the electronics industry.

TTL Cookbook No. 21035	$ 9.50
CMOS Cookbook No. 21398	$10.50
Active-Filter Cookbook No. 21168	$14.95
TOTAL LIST PRICE	$34.95
Less 15% Discount	5.24
YOU PAY ONLY	$29.71

☐ **ORDER SPECIAL LIBRARY PACKAGE NO. 21707 AND SAVE!**

ORDER FORM

HOWARD W. SAMS & CO., INC.
4300 West 62nd Street, P.O. Box 7092
Indianapolis, Indiana 46206
(317) 298-5400

Indicate quantity in boxes above and complete ordering information below.

Sub Total__________

Add local sales tax where applicable__________

GRAND TOTAL__________
☐ PAYMENT ENCLOSED
☐ CHECK ☐ MONEY ORDER
☐ MASTER CHARGE ☐ BANK AMERICARD/VISA

Exp. Date__________

Account No.__________

Interbank No. (Master Charge only)__________
Minimum Credit card purchase $10 AD045

Name (print)__________

Signature__________

Address__________

City_____ State_____ Zip_____

Prices subject to change without notice. All books available from Sams Distributors, Bookstores, and Computer Stores. Offer good in U.S. only. In Canada, contact Lenbrook Industries Ltd., Scarborough, M1H 1H5, Ontario, Canada.

Howard W. Sams & Co., Inc.

Computers and Society

David D. Thornburg and Betty J. Burr
Innovision
P.O. Box 1317, Los Altos, CA 94022

In his poem, "All Watched Over by Machines of Loving Grace", Richard Brautigan says:

"I like to think (and the sooner the better!) of a cybernetic meadow where mammals and computers live together in mutually programming harmony like pure water touching clear sky."

Well, Richard, I think we did it!

Photographs courtesy Louise Burton

**for information on the Peninsula School Computer Project, write to:*
Katie Thornburg
Peninsula School Computer Project
Peninsula School
Peninsula Way
Menlo Park, CA 94025

***for more information on ComputerTown, USA!, write to:*
Pat Cleland
People's Computer Company
P.O. Box E
Menlo Park, CA 94025

Computers at the Learning Fair . . .

Each year the Peninsula School in Menlo Park, California conducts a "Learning Fair" for children, parents, teachers and neighbors. This year the fair was held on May 4 and was attended by more than a thousand people who came to enjoy the food and music, learn about solar energy, make kites and candles, and to take part in myriad other activities including playing with computers. While the computer activity was a small part of the overall event, it was an enjoyable, if exhausting, task for those of us who put it together.

The computer activity was planned by the Peninsula School Computer Project* (Katie Thornburg), ComputerTown, USA!** (Pat Cleland) and Innovision (your faithful scribe, DT). We intended to present a computer activity which was smoothly integrated, in spirit, with the other activities at the fair, including leather work, face painting, and events naturally suited to a fair conducted in a semi-rural environment on a beautiful spring day. The goal of the computer activity was to provide an opportunity for people of all ages to learn about computers and play games without feeling intimidated by these machines. This activity was to be used by everyone from computer experts to people who had never before seen a computer of any kind.

There were three major tasks which faced us as planners of this event. First, we needed to have enough hardware available to give every interested person a chance to use the machines. Second, we needed software which was appealing to boys and girls, young and old alike. Third, we needed enough volunteers to help people with the computers and to keep everything in working order.

Since we could only provide eight computers ourselves, we sought additional support from the outside. Through the generosity of Commodore, Atari, and Radio Shack we were able to have more than twenty-five computers running at one time. This allowed us to run one program per machine - a blessing when one considers the time spent just loading tapes.

The software included stimulating games of many types, with the exception that no "arcade" type games were used. Our reluctance to use arcade games was based on several factors. First, the majority of arcade games with which we were familiar were sufficiently "addictive" to certain children (primarily boys) that we could envision problems in providing easy access to the computers. More importantly, these games generally require very little cognitive activity on the part of the player, and were thus not considered appropriate for incorporation in a "learning" fair. The most compelling argument against the use of these games, however, is that we were able to find a tremendous number of games which were fun, mentally stimulating, and appealing to people with little or no previous exposure to computers. This software included adventure games like TAIPAN and QUEST, logic games like BUTTON BUTTON, BRAIN BUSTER, QWERT, and MOTIE, simulations like LEMONADE, word games like WORDSEARCH, and creative environments like DRAW, just to mention a few of the programs which were in use that day.

We received tremendous assistance from the many volunteers who kept everything running smoothly. Volunteers included interested parents and several local computers professionals who enjoy working with children. This support allowed us to set aside a few machines for visitors who wanted to learn a little about computer programming. We also set aside two computers for the exclusive use of pre-schoolers.

In addition to the mixture of Commodore, Atari, Apple, and Radio Shack computers, we had several "outdoor" computer activities as well. Visitors to the computer area were treated to the beautiful sounds from two Alpha Syntauri computer music systems which were set up on Apples out under the arching oak trees. Nearby, children played with several "Big Trak" programmable toys which were set up on special roadways outside the computer rooms. Several of us have found that these toys (which behave similarly to a LOGO "turtle") are very well suited for teaching programming concepts to youngsters (In case you wondered, Big Trak is a motorized device with a keyboard that accepts programs up to sixteen steps long. Once the program is entered, the user presses GO and watches the machine move along a path defined by the program.)

Next to an information booth, we had contiuous showings of "Don't Bother Me, I'm Learning", the film on the use of computers in schools which appeared on PBS earlier this year.

The smooth mixture of all these activities blended in with the leather work, crafts, music, food, and other activities of the Learning Fair to create a truly magical environment.

Coming next month . . .

By the time you read this, the second Semi-Annual Consumer Electronics Show will have been held in Chicago, we will have finished reading Alvin Toffler's new book, The Third Wave, and we should have our hands on two pieces of federal legislation which are pertinent to the world of personal computers. Any or all of these topics will be discussed in our next column. Most importantly, we want to hear from you. Write us at the address shown at the beginning of the column and let us know what you think about the social impact of computers. ©

An Interview With Dr. Chip

Robert Lock, Editor

As past readers know, Dr. Chip is Professor of 6502 science at Figment U., a little known but widely respected branch of academia that's devoted to the study of various happenings in our industry. From time to time he contributes to these pages:

RCL: Dr. Chip, you sounded awfully excited when you called at three this morning.

Chip: I'll say. My West Coast sources have just uncovered a major manufacturer planning to enter the 6502 market. That always gets me excited.

RCL: Can you share the details?

Chip: They're incomplete, but I'll tell you what I know so far. Mattel Electronics Division, the manufacturers of the Intellivision, are adding what's been billed as an "intelligent" keyboard to their unit. It's scheduled to be released in the Fall, with full production by early 1981.

RCL: That's certainly news Dr. Chip, but they've been showing off the intelligent keyboard for several months.

Chip: True enough, but what they haven't been showing off is that the unit is 6502 based, and is adding Microsoft Basic to the Intellivision.

RCL: I see why you're excited. Any specifics?

Chip: Reliable sources indicate the unit will be supplied with 8K Microsoft in ROM, and 16K of random access memory. Final details on pricing and delivery aren't yet available.

RCL: Will you keep us posted Dr. Chip?

Chip: I expect to have a complete set of information for a future issue. We've placed the Santa Clara chapter of the Figment U. 6502 Users Group on full alert, and they're out scrounging for news now.

RCL: Thank you Dr. Chip. We'll look forward to your further comments next time.

Chip: I'm not finished.

RCL: Excuse me. . .

Chip: Mattel's not the only company that's been active lately. Atari is gearing up for some new hardware and software introductions as well. By the time your readers see this, Atari will have introduced at least one new piece of hardware at the summer Consumer Electronics Show in Chicago. Rumor has it they'll be showing off the dual drive, dual density disk for their machine, as well as some new software, including a general accounting package.

RCL: That sounds exciting. What have Commodore and Apple been up to?

Chip: Well, you know that Apple introduced the Apple 3 in the NCC. *(Editor's note: See the review and commentary this issue...)* They certainly are going after the loyalty of their users and dealers in a big way. They even rented Disneyland for one whole evening and passed out free tickets at their booth!

RCL: And Commodore?

Chip: Ah well. They're my personal specialty you know. Lots has been happening there. I've been struggling to keep up. Since last issue, they've been through some major changes. And let me emphasize major. They've completely revamped their marketing structure in the US. The whole US sales/marketing organization has now been relocated to Norristown, PA from Santa Clara, CA. That's a big move, but even bigger is the fact they they've revamped and replaced most of their marketing team. All the way up to a new General Manager for US Sales and Marketing. They're setting up regional warehouses/distribution centers all over the country, and have indicated a whole new range of services and commitment to their dealers.

RCL: Frankly Dr. Chip, that sounds familiar.

Chip: I know, I know... but dealers who've been involved with the new team are telling me it's really true.

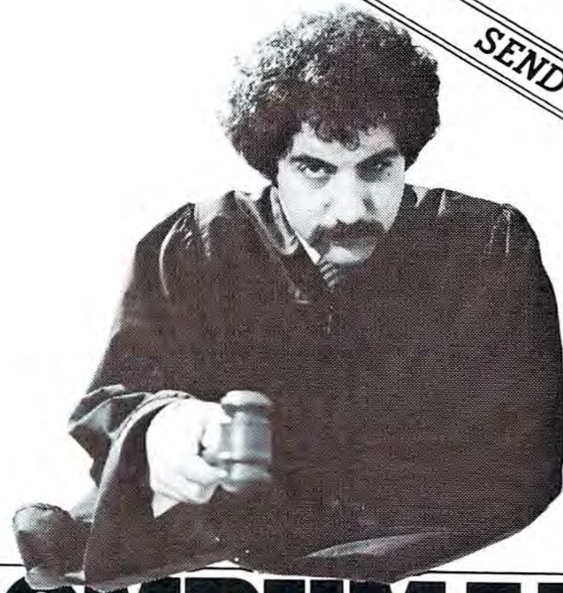
RCL: I've been hearing some beefs at this end.

Chip: That's the kicker. In the midst of all these new changes... and I'll have to say I think it looks good, there's a growing controversy over changes in the boards. You see, when Commodore introduced their new standard keyboard machines, they used essentially the same primary board for all three machines: the 8K, the 16K and the 32K. End users and dealers quickly discovered that one could easily expand and upgrade a machine, making use of the plated-thru (but soldered over) holes in the board, by adding RAM chips. The units are designed for expansion by plugging in additional memory boards on the memory expansion port, and Commodore is taking the position that this is the approved method of expansion. To reinforce that approach, or perhaps I should say to "enforce" that approach, they've begun drilling through the in-place traces in the board that were previously used by dealers and users to expand their machines. The bottom line is that all 8K and 16K units shipped for the past several weeks have had four 3/8 inch holes drilled through the second row of traces in the CPU boards.

There appear to be two current schools of thought on this activity. Commodore defends themselves with the position that they've always sold three distinct models of machine with a memory expansion port provided for upgrading 8K and 16K machines. They also maintain that they've never condoned or encouraged the alternate, cheaper method of expansion.

The alternate view of the process is that Commodore is multilating perfectly good boards to prevent the more inexpensive method of upgrading 8K and 16K machines by adding RAM. It appears that a part of Commodore's motivation was the result of a group

SEND FOR OUR FREE CATALOG



COMPUMART LETS YOU PUT 'EM ON TRIAL

WITH 10 DAY FREE RETURN PLUS THESE SPECIALS

APPLE

COMPUMART EXCLUSIVE/ Order any combination of Apple Computers and Accessories from CompuMart totalling $3,500 and we will send you your choice of the following:

1. A Zenith Color Monitor (Value $499.)
2. A Centronics P1 Printer (Value $470.)

Call for details on other great Apple specials.

WE STOCK THE ENTIRE APPLE LINE.

COMMODORE

NEW! From Commodore. The CBM 8000 Business Computer is here, and CompuMart has both the 8016, and 8032 in stock for IMMEDIATE DELIVERY.

$100 in FREE Commodore Accessories with purchase of a 16K or 32K computer.

WE STOCK THE ENTIRE COMMODORE LINE.

Buy direct from the Largest Commodore dealer in the country, and the very first Commodore distributor in the U.S. Buy from the experts — buy from CompuMart

ATARI

SUPER SALE on the Atari 800 Computer System. CompuMart Price $879 (List Price $995)

CompuMart also offers the following Atari specials (you MUST buy an Atari 800 to take advantage):

1) Double your memory for free. Buy 8K of additional memory and receive 16K/ Buy 16K and receive 32K.
2) Buy the Atari 800 and take $100 off the price of the Atari 810 Disc Drive or the Atari 820 Printer.

Call for details and prices.

270 THIRD ST. DEPT 307 **COMPUMART** CAMBRIDGE, MA 02142

TOLL FREE 1-800-343-5504 **IN MASS 617-491-2700**

of dealers who began ordering only small machines and then upgrading them for sale as 16K or 32K machines. They considered this essentially dishonest, with some justification I guess. The people who bear the greater impact of this change are those dealers and end users who sold or bought machines on the basis of an expectation of future internal expandability. Not with any intent to cheat Commodore or their dealers but rather with a perfectly reasonable desire to economize and build on their system(s) as money and time permitted. I think it's a matter that will take some time to resolve. And it points out one of the major problems with Commodore in the past... a failure to act from an informed perspective. The drilled out boards simply started showing up at dealers. I assume some of the larger companies may have known about the change, but the smaller ones certainly didn't It was obvious from the phone calls we received that Commodore had not bothered to inform their dealers or distributors of the pending change.

RCL: That's the worst part of the whole thing. I got a lot of complaints from dealers as they realized what was going on. I've written to Commodore and talked to their new General Sales and Marketing Manager, Dick Powers, and I think he's in the unfortunate position of landing in his new job just in time to meet the proverbial frying pan. He defends the decision as a necessary one, and one clearly consistent with the original intent and goals of the product line. But he candidly admits the failure in the process... not seeking dealer input; not, at the least, giving dealers 60 to 90 days notice of the change, and so on. He comes from a background of national sales force management and appears to clearly understand the problems and confusion caused by "unilateral" happenings from on high. My opinion is that Commodore should rethink the whole thing, but short of that, should clarify the happenings for their dealer and end-user network. It appears that this will be the future course of Dick Powers, and I wish him well in building that network of dealer trust and support. He inherited a situation not ideal for a new beginning, but seems determined to build new strength and loyalty at the dealer level. Time will tell, of course. You dealers can start the process rolling by writing to him at this address:

Mr. Dick Powers
Commodore Business Machines, Inc.
Computer Sales Division
950 Rittenhouse Road
Norristown, PA 19401

Continue to keep me posted on your progress. RCL ©

Guest Commentary

BUSINESS APPLICATIONS ANALYSIS--THE MISSING STEP

Hal Wadleigh

Business applications analysis seems to be the most neglected element of the microcomputer industry today. The shame of it is that the principles of business analysis affect almost every phase of the use of microcomputers in the business environment--from the initial choice of equipment to evaluating programs in use. The root of the problem appears to lie in the history of microcomputer software.

A short time ago, there was little or no business software available for the smaller microcomputer systems. The software market was flooded with games, but programs that do anything useful for businesses were few and far between. When business programs could be found, they were unfortunately lacking in the qualities that make "good" software distinct from "bad" software. Now that the systems have been out for a while, the quantity of business packages available is greatly improved. The bad news is that the quality of this software (with a few notable exceptions) is as poor as ever.

Both of these situations--the plethora of games and the low quality of business software--seem to be related to the way in which most microcomputer programs are developed. The programmer gets an idea and sits down to start coding. This approach is ideal for games because any interesting oddities that occur during this rather non-objective procedure can be incorporated into the game to make it more interesting. This is also the worst procedure possible for business programming.

The nature of games is that they don't have to do anything in particular (except hold the player's interest) and the job itself can be redefined to accommodate any discoveries made during the programming process. In this case, the program is more important than the job it is supposed to do!

Business programs, however, are the exact opposite--the job is everything and elegant programming is almost meaningless. A good business program is one that does the job well. A bad business program is one that does the job poorly. The elegance and sophistication of the program does not matter. Successful games are usually programs that continually surprise and amaze the user. Business programs had better NOT surprise and amaze the user.

The principles of business applications analysis are really quite simple. It does not take a great deal of intelligence or education--just a little control. It is a five step process:

STEP #1: DEFINE THE JOB

It is not too unusual to hear a small businessman say something like "I bought one of those little computers last year. What do you think I ought to do with it." It's a rather amazing statement when you think about it. The man has a tool and would like to know what kind of job to do with it. The proper procedure is to buy the tool that best fits the job that needs to be done--it doesn't matter if we're talking about hammers or computers.

Any computer is a tool for processing information. Defining the job for a computer is usually a simple matter of completing the sentence "I want to get. . ." with a detailed description of what will be the output of the system.

This step is often called the *OUTPUT SPECIFICATION* phase.

STEP #2: DEFINE THE INFORMATION NECESSARY TO DO THE JOB

No computer will create new information. A computer will, however, change the form of information that is available to it into a more useful form. For example, a file available to the computer might have a lot of records on items in a business' inventory. Each one of these items has the information on what the value of the items are individually and a count of how many of these items are in stock. The computer can, whenever necessary, take these individual items of information and produce that information in the form of a statement that, "Current inventory is worth $9875.42" on a display. This is not really a matter of producing new information--since the information is already contained in the individual inventory items. The computer has simply changed the form of that information into something more desirable.

Since we have already defined the job we want the computer to do, we now have to define the information that the computer will need to do that job. This often involves a bit of research. The person who does this part of the anlysis has to know how to do the job itself. It also usually involves finding out the exact form the information is in when it becomes available to the people who will be operating the computer.

This step is often called the *INPUT SPECIFICATION* phase.

STEP #3: DEFINE THE INFORMATION TO BE STORED

Some of the information necessary to do the job will be needed over and over again. It is silly and wasteful to require operators to enter this information every time it is needed. Sometimes the job itself is simple data retrieval--looking at stored information. This is the step where the information that should be stored is defined. In this step, you decide the number of data files, the form of each data record in the file, and even the size of the file.

This step is often called the *FILE SPECIFICATION* phase.

STEP #4: DETERMINE THE PHYSICAL FLOW OF THE INFORMATION

Business applications are a matter of getting the right information to the right place at the right time. If the computer is going to be printing reports in the accounting office and the information is needed at the loading dock, then the system specifications have to include a means of getting that printed report to the loading dock. This step will be almost meaningless in some applications--but it will be the most critical step in others. In either case, it cannot be ignored--even if it seems to be unimportant at first glance.

This step is often called the *WORKFLOW SPECIFICATION* phase.

STEP #5: DEFINE THE TIME CONSTRAINTS OF THE OPERATION

Since we are dealing with a system that has to get the right information to the right place at the right time, we need to make some rather exact definitions of the tolerable delays for each step of the job. It would be silly to define a system that has to sort large files in many different ways without allowing enough time for these sorting operations. It would also be silly to try to function without such sorting operations if they are critical to the operation itself. This final step is often called the *RESPONSE TIME SPECIFICATION PHASE.*

This constraints defined in this stage may show that the previous steps have resulted in a system design that simply cannot work fast enough to do the job. This could necessitate doing one or more of the earlier steps over until all five steps conclude with a acceptable applications design.

THE FINAL RESULT--SYSTEM SPECIFICATIONS

Now that you have completed these five steps, you have some idea of what you are looking for. You still haven't chosen any equipment and you haven't even designed any programs--but you DO have a complete definition of the exact job to be done--and that is the most critical point:

YOU CANNOT BUY AND PROGRAM A COMPUTER TO DO A JOB UNLESS YOU KNOW EXACTLY WHAT THE JOB IS!!!!

Unless you have gone through this process, you don't really know what the job is and you can't really make any informed decisions about equipment or programming. The end results are all too often either comical or tragic.

The general impression of many computer professionals is that micro systems are toys and that micro software is limited to games and junk. There is an uncomfortable amount of truth to that view--due to the haphazard way in which micros have been used. If people in the microcomputer industry begin using their tools properly, that attitude will change. It will soon become obvious that mainframe systems are needlessly expensive behemoths and that mainframe software is archaic and oversensitive to small errors.

The *real* microcomputer revolution will begin when microcomputers are used properly--and defining the job to be done is *always* the first step to proper use. ©

An Introduction to Small Business Software for the PET*. II.

Can DR. DALEY's offer a better Mailing List Maintenance System?

You've seen them all! Every software supplier offers a mailing list system of some sort or another. Each of them has some advantages and some disadvantages over the others.

So when DR. DALEY's decided to offer a mailing list we felt that it had to offer some other advantages over all of the others. We have offered—and sold some—mailing list systems before, but these offer few things that makes them unique.

SERIOUS BUSINESS

When you wish to purchase a software system for any business purpose you need to give it serious and thorough consideration. What do you wish to accomplish with the software? What are your needs? How can a computer assist you in filling these needs? We have asked these questions numerous times to people who do mailings with lists in the size range of 500 to 15,000 names. The result was unanimous: everyone has different information needs. This, of course, means that everyone who buys a mailing list system, or any other business software, must find a program that comes closest to his needs. This is a time consuming, expensive task. We've talked with businessmen who have become frustrated with this process and are ready to throw in the towel. Another option is to hire a programmer to write the software for you or to write your own. This can cost more than the cost of the computer.

The last option is to find prepackaged software which each individual user can easily configure to his own needs. This would allow each business to customize its own computer maintained mailing list files to, as closely as is possible, parallel the current mailing list operation. Until now, this option has been virtually impossible to fulfill, from any software publisher.

IMPLEMENTATION

Our computerized mailing list system is designed to be easy for you, the user, to be able to easily configure your files to contain information in much the same way as you currently are doing. This means less of the pain and anguish that frequently accompanies computerization.

During the programming the author was in frequent contact with potential end users. The main thought during the development phase was to make the operation easy to understand, yet powerful enough to handle the job. Give the user as many options as is feasible, with the flexibility to make the greatest possible use of the file information. Finally, be sure that the capacity of the system is sufficient to allow most any business to make use of it.

The final version will allow records of 117 USABLE characters in length with a maximum of 15 fields within each record. It also allows reasonably large capacity with multiple diskette (maximum of 100 diskettes on a 32K PET or CBM) files and up to 1340 records per diskette.

WHAT ABOUT SORTING?

We hear this question most frequently from you. This is because sorting is the operation that divides the MAILING LIST system from any mailing list system. Why sorting? Well it is the way that the user can do such things as selective mailings to groups with common characteristics. This could include regional mailings, mailings to customers of a particular product, mailings to purchasers or to prospective customers, etc., etc. Or you might wish to make any possible combination of these categories.

Try to do this on most ordinary mailing list programs. You simply can't do it with most of the offerings on the market today.

This sorting is done by a "wild card" type of sort. This means that you can specify the contents of any portion of a field for a match and the computer will take any match for the rest of the field. This type of sort is best illustrated with the following examples:

A sort key can be: **R**1
Matches with FORT#1
and T4R321
and %/R@31

Our system allows this type of sorting using up to three fields within each record. Thus you should be able to retrieve almost any conceivable subset of the files.

File organization is done using two of the fields as sort keys. This again is user selectable. You could, for example, specify that you wish the file to be in ZIP CODE sequence or in alphabetical sequence and all records within the file will be sequenced with that field. There is also a second sort field which is used to sequence the file where the first field is the same.

WHAT ABOUT LABELS?

We hear this one almost as often as the sorting. Well, here this is up to you. You can, at the time you print labels, choose the layout of the labels, you can also choose the number of labels per line. If you wish to have a four line address and printed four records wide you can do it.

WHAT ABOUT EDITING?

Editing is accomplished at several points in the program. These are at the time of entry, before saving the records to the file and from the disk file. You can easily modify any record at any of these points.

This does not really cover all of the operations on the files. Space simply does not allow a more complete description of the user oriented approach of the program.

We asked the question: Can we offer a better mailing list system? You bet we can! It's here now.

HARDWARE REQUIREMENTS

At present this requires a Commodore PET or CBM computer with a dual disk drive and a printer. It is set up to work with the Commodore printer or with most any other printer. Watch for these programs to be introduced for use with other types of popular microcomputers. The APPLE II version will be available about June 1, 1980. Watch for it!

ORDERING

At the present time many Commodore dealers do not carry our software. Thus you will most likely need to either persuade them to order for you or calling us directly at (616) 471-5514 anytime between noon and 9 p.m. Eastern time Sunday through Thursday. For only $99.95 plus four percent tax in the state of Michigan, you get this powerful, field tested, fully documented program packaged in a convenient three-ringed binder.

INVENTORY

We must add this note. There is too little space to allow us to describe the INVENTORY system adequately here. It offers the same flexibility as does the MAILING LIST described above, but we can't tell you much more. Write or call for details. It also is priced at $99.95.

Charge to your MC/VISA

master charge THE INTERBANK CARD

VISA

DR. DALEY's Software
425 Grove Ave., Berrien Springs, MI 49103
Phone (616) 471-5514
Sun.-Thurs. noon to 9 p.m., Eastern Time

*Watch for it on the APPLE II.

Atari's Marketing Vice President Profiles The Personal Computer Market

Conrad Jutson

Michael S. Tomczyk

Atari doesn't especially like my nickname for their 400/800 personal computer -- "the pop-top computer" -- but it's a fact the computer has a "pop top" where the plug-in RAM/ROM cartridges fit, part of their innovative user-proof system which also includes interchangeable cords for the computer's various peripherals. Atari also has a growing array of educational and game software, including the most sophisticated real-time simulation game (STAR RAIDERS) in the galaxy ... a long way from "Pong," the game that started it all.

Atari's competitors in the personal computer market chuckle at what they see as the company's attempt to develop the "home" computer market, in the face of extensive market research that says the home market won't "happen" for another 4-5 years. Does that mean Atari is wasting its resources? Are they really going after the home market? Or are they laying the groundwork for a broader marketing program?

To answer some of these questions, I interviewed Atari's new Vice President-Sales & Marketing for Personal Computers -- he's Conrad Jutson, who came to Atari in November 1979 with a scant background in computers but over 20 years experience in consumer electronics at G.E. (12 yrs.), Toshiba (6 yrs.) and Texas Instruments (3 yrs.).

Jutson began by describing what he sees as the outlook for the personal computer market: "**Small business** in the short run will account for fifty percent of the personal computer business, dollar wise," he predicted, defining small businesses as those with less than $1 million in annual gross revenues, employing 10-15 people, and usually involved in manufacturing or a service-oriented industry. Typically, they do their bookkeeping by hand through a full or part time employee, or have it done by a local service. The key to reaching this market, Juttson explained, is being able to show them that a microcomputer will increase their productivity and make the investment worthwhile.

The second broad market segment is the **consumer market** which, he said, consists of hundreds of subsets.

"If we were to profile the personal computer buyer in the early 80's, it would be a male or female head of household, most likely in a managerial, administrative or professional position, typically earning over $25,000 per year and falling into the 25 to 50 age bracket. Most likely, this person is already familiar with what a computer can do and can, in the home environment, identify a need for computing to address various problems and functions.

"There are several millions of these households in the U.S. that fit into the demographics I've described," he continued. "I don't believe personal computers will ever be an 'impulse item' off the shelf, partly because of the expense, so the logical question becomes, 'Why should I buy a personal computer and what will it do for me?'"

Jutson's answers to that question -- what will a computer do for me -- provided an interesting way of categorizing the personal computer market in terms of **function.** His list of personal computer uses included. . .

1) Planning and Record Keeping:
"I believe this type of managerial/administrative consumer does not pay enough attention to his own finances -- this is confirmed by the rapid growth of financial planning services. With the rapid inflation of the past few years, projected to continue through the 1980's, many consumers have found themselves in higher tax brackets with a higher cost of living that has made their lives more and more complex and difficult to manage. They've had to cope with budget planning, financial investments, mortgages, loan payments, credit unions, payroll stock plans, taxes and pensions. In this new complex environment, consumers have to organize their home record systems like they do at work - on a daily, year-round basis instead of just once a year at tax time. They have to look at their gross income, their investment tradeoffs, and I believe this type of consumer can justify the purchase of a personal computer with the appropriate

software to meet these various needs. . .given that the typical first purchase of a personal computer is around $2000-2200.''

2) Home Education:
The next category of purchase that adds value to the computer is home education. Jutson noted that a majority of schools and colleges are requiring some hands-on computer experience and more and more schools are bringing computers into the classroom as instructional aids. There is already an enormous investment in home education being made by the American family -- cutting across all demographic strata -- in home courseware, from encyclopedias to books. As a supplement to classroom education, this home courseware can be made much more exciting and ''fun'' through visual display and interaction with a computer, Jutson explained.

3) Personal Development & Interest:
There is also, he said, a huge market in how-to books, all the way from how to fix your appliances to learning foreign languages. Literally hundreds of topics are addressed. Personal computers provide for active hands-on demonstration for all age brackets and interests, and speed the learning process.

4) Interactive Entertainment:
Having purchased a personal computer, we're all challenged by interactive entertainment, he said, whether the entertainment is one of skill or of strategy. The sale of strategic board games (chess, backgammon) never seems to let up and in the skill area, the video arcades are doing extremely well. So entertainment accounts for a good deal of software sales.

5) Home Information/Communications:
If we move away from computation and hook up an interface and telephone modem, we now have the capability to hook up to a timesharing service. Using the computer as a terminal provides a capability for dialing up and subscribing to a variety of evolving services. Some, like Micronet and The source already have a fairly long menu. Atari has defined an information and communications strategy -- obviously it will leverage our installed base of hardware to help our users gain access and may involve a wholly owned subsidiary like Warner Amex Cable. Some of the future uses of this home information system which we can envision includes news, stock data and other services which will cut down driving time, mailing time, and minimize the hassle of shopping and bill paying. It's a question now of ''getting the players together,'' he said, and making it happen.

6) Home Monitor & Control:
The decade of the 1980's will witness a growth of consumer electronic products deriving in large part from introduction of smart electronics into the home. The personal computer is the ''leading edge'' of these products. By the mid-1980's, he expects to see dedicated smart electronics -- CPU devices which interact with the electronic environment -- in the home. It's unlikely that we'll see one massive all-purpose CPU controlling everything in the home. It will happen step by step, beginning with stand alone appliances containing their own microprocessors and other smart electronics.

These then are some of the major uses which Jutson foresees for personal computers, now and in the future.

He goes on to say that the Atari product was designed for the consumer, to be easy to use-by-consumers, easy to access, easily loaded (cartridges), and easily connected (modular cords).

''Does the end user care about the architecture of the machine?'' he asked rhetorically. ''The answer is no. What will it do for me? That's his major concern. We in the consumer electronics business are concerned with leveraging technology and bringing that technology to the consumer for his or her benefit, so why try to scare the consumer off by making it so he or she has to have a double E or be a computer programmer to utilize the full capabilities of a personal computer?''

He drew a parallel between the personal computer industry and the home stereo industry, pointing out that 15 years ago there were 1500 hi-fi salons in the United States and now there are about 15,000 outlets in the U.S. He feels that computer stores will become to the computer market what hi-fi specialty shops were originally to the hi-fi industry, and predicted that a number of stores will proliferate and become strong chains. A parallel development, he said, is the entry of general merchandisers such as J.C. Penney Department Stores into the personal computer distribution scheme.

He emphasized that Atari only started shipping late in the fourth quarter of 1979 and is just getting into the market with its 400/800 computers. Heavy advertising is planned for the second and third quarters of 1980, including a full dealer support program.

''Having just come out of the gate we have to and will continue to have a lot of things to do to strengthen our position in the industry,'' he said. ''Atari is a young company that has already, in a few years, achieved significant growth in consumer electronics products. We have a vertically integrated manufacturing capability, a marketing staff that understands marketing, distribution, sales and sales promotion; and a large blend of research and development and engineering expertise.''

''We believe that the Atari computers are different because from word one they were developed to take away whatever apprehensions a first time user might have and help him or her feel good about interfacing with our product. With Atari computers, you don't have to stop and think before you use them. Of course, more and more of the younger generation are learning to program and work with more sophisticated applications, and they will have the capability of doing so with our product.'' ©

INTRODUCING
THE NEW IMPROVED

BUSINESS ENHANCEMENTS COMPUSERVICE BUSINESS SOFTWARE
FOR
COMMODORE
AND
APPLE COMPUTER SYSTEMS

Micro Mini Computer World Inc. is an exclusive distributor for the BUSINESS SOFTWARE developed by Business Enhancements Compuservice of Escondido, California.

If you are selling or using the COMMODORE BUSINESS MACHINES or the APPLE computer systems, then you should provide yourself and your customers with the MOST COST EFFECTIVE and COMPREHENSIVE business software for a business computer system.

CURRENT B.E.C. SOFTWARE

- General Ledger--Master File 1000 Accounts and Journal File 4400 Entries
- Accounts Receivable--Master File 1170 Accounts and Invoice File 1430 Entries
- Accounts Payable--Master File 1170 and Invoice File 1430 Entries
- Payroll--440 Employees
- Job Costing--1100 Items Per Disk
- Inventory--1100 Items Per Disk
- Mail List/Customer Information--1000 Entries Per Disk

Above figures apply to CBM 2001 computer system with 32K CPU and 2040 dual disk. With the new CBM 8050 Megabyte disk the volumes will be increased significantly.

B.E.C. SOFTWARE FEATURES:

- Complete and total documentation
- Step by step walk through on every program operation, with examples
- Each package is MENU driven and uses dynamic load and overlay once the initial menu is loaded.
- Examples are provided for all reports and other printed forms. All forms are available from New England Business Services Inc. (NEBS).
- All input/output operations use random access
- Sorts are machine language sorts
- Programs are interactive with the General Ledger and update the GL automatically.

B.E.C. VALUE ADDED BENEFITS

- Total commitment to the development of excellent business software for the COMMODORE and APPLE computer systems.
- At reasonable rates Micro Mini Computer World Inc. will provide software modifications to meet customer requirements. (Call MMCWI for further information)
- EXTENDED WARRANTY which entitles users to any enhancements to accounting software during the year of coverage. (Cost is $100 per year)

Dealers and Interested Parties may obtain a copy of the B.E.C. software documentation for $25. If after review you are not interested, send the documentation back, in re-saleable condition, for a full refund or apply the $25 toward your first software purchase.

The NEW B.E.C. BUSINESS SOFTWARE requires a special ROM chip for proper operation.

Suggested Retail Prices are:

1. Rom chip $40 (required on any software package)
2. Individual software package $150
3. All seven software packages $995 (save $55)

DEALER INQUIRIES ARE INVITED

micro mini COMPUTER WORLD Inc.
TOWN & COUNTRY SHOPPING CENTER

74 ROBINWOOD AVE. COLUMBUS, OHIO 43213
(614) 235-5813 (614) 235-6058

VISICALC: A Software Review

Joseph H. Budge

There are very few single programs good enough to sell computers. Visicalc is one such program. Every Visicalc user knows of someone who purchased an Apple just to be able to use Visicalc. One user wrote down the software specifications for Visicalc and took them to IBM, asking "Can you do this for me?" IBM bid at the job: Cost would be $30,000 and it would take 3 years to complete the software. Yet Visicalc is available in any computer store today for $150 and runs on $2,000 worth of hardware. More computers have been sold on its account than with any other single software product. What is Visicalc, anyway, and why is it so good?

In its simplest terms Visicalc is a numerical modelling program. The program divides computer memory into a matrix of up to 16,256 cells. Each cell can contain a string (label) or a number. So far so good; memory looks something like a multi-dimensional array in BASIC. But unlike BASIC each cell is displayed on the video monitor and can be related to any other cell by a mathematical formula. Thus cell A13 might be the sum of cells A1 through A12, while cell B13 represents the sum of the squares. Now change the value of one cell. Viola! All the other cells are instantly recomputed to reflect the change. Thus the user can set up models and interactively make changes to see the results.

The cell and formula system of memory organization is applicable to myriads of uses. Financial budgeting, with its columns and rows of numbers, fits in perfectly. Visicalc even uses 12-digit precision to allow accurate finance calculations. Loan terms, sales projections, and profit/loss statements can be successfully modelled or analyzed. But Visicalc isn't strictly a financial tool. It's capabilities are general enough to apply in many other fields. Demographers can model population trends, biologists can model biochemical systems, and physicists can model nuclear decay.

Visicalc is best known for simplifying complex business decisions. An acquaintance of mine just received a substantial raise for the forecasting work he started doing for his employer. What the employer doesn't know is that my friend takes the work home at night and solves the problems on Visicalc in 10 minutes. A large part of Visicalc's usefulness stems from its rather complex instruction set. Commands exist for inserting values and setting up formula, obviously. Other commands control screen display and split-screen displays. One part of the screen can be used to play with parameters while the other displays results. There are commands to allow cell and formulae replication as well as mass editing commands. And, naturally, models can be saved to diskette in the event that the user needs to back up a few steps.

Because of the complex instructions, Visicalc is certainly not for the faint-of-heart. All instructions consist of single or double keystrokes with minimal prompting. While this will slow down beginners, it allows experienced users to work very rapidly, Visicalc comes with excellent documentation which takes the beginner step by step through each instruction. After a week of study Visicalc operation should be second nature to almost everyone.

Personal Software has produced an excellent product in Visicalc. Unfortunately their idea of customer relations leaves a bit to be desired. Visicalc is sold on an uncopyable diskette with a 90 day warrantee. In effect one pays $150 to use a program whose performance after three months depends on luck. Three months is just about the expected useful life of a heavily used diskette. In this author's opinion the program should be good for a year or more. It should at least come with a backup so that business users don't have to suffer down-time while waiting for a replacement. Personal Software also touts their Visicalc Newsletter in their documentation. As far as I can tell no such thing exists.

There is a bug in some versions of Visicalc which users should be aware of. If you use version 1.35 or earlier to initialize a data diskette, your diskette will not initialize properly. The volume table of contents isn't set up correctly. When later data is written to the disk it will eventually overwrite the disk catalog. When this comes to pass all the data on that diskette will be lost. If you have an early version of Visicalc be sure to use Apple's regular initialization routines instead of Visicalc's.

Overall I would rate Visicalc as an excellent value for anyone with modelling to do. You will be surprised at how many applications are possible once the program is in your library. Be prepared to spend a lot of time at first studying the manual, and treat that diskette like gold. Visicalc is carried in almost every computer store, where excellent demonstrations are also available. ©

Editor's Update:

The PET/Commodore version of Visicalc is expected to be available in July. Price is unknown at this point. The Atari version is expected to be available sometime during third quarter, 1980.

Personal Software is currently reviewing their warranty policy. As Joe points out, the warranty presently provides free replacement of defective disks for 90 days. The warranty also provides for replacement of a defective disk at nominal charge ($15.00 at your local dealer, or $7.50 from Personal Software) during the first year of ownership. RCL

NEECO PROUDLY INTRODUCES

COMMODORE'S NEW 8000 SERIES (80 column) COMPUTERS

$1695 (available May/June '80)

CBM™ 8050 DUAL DRIVE FLOPPY DISK

The CBM 8050 Dual Drive Floppy Disk in an enhanced version of the intelligent CBM 2040 Disk Drive. The CBM 8050 has all of the features of the CBM 2040, and provides more powerful software capabilities, as well as nearly one megabyte of online storage capacity. The CBM 8050 supplies relative record files and automatic diskette initialization. It can copy all the files from one diskette to another without copying unused space. The CBM 8050 also offers improved error recovery and the ability to append to sequential files.

HARDWARE SPECIFICATIONS
Dual Drives
Two microprocessors
974K Bytes storage on two 5.25" diskettes (ss)
Tracks 70
Sectors 17-21
Soft sector format
IEEE-488 interface
Combination power (green) and error (red) indicator lights
Drive Activity indicator lights
Disk Operating System Firmware (12K ROM)
Disk Buffer (4K RAM)

FIRMWARE
DOS version 2.0
Sequential file manipulation
Sequential user files
Relative record files
Append to sequential files
Improved error recovery
Automatic diskette initialization
Automatic directory search
Command parser for syntax validation
Program load and save

CBM™ 8000 SERIES BUSINESS COMPUTERS

The new Commodore 8000 series computers offer a wide screen display to show you up to 80-character lines of information. Text editing and report formatting are faster and easier with the new wide-screen display. The 8000 series also provides a resident Operating System with expanded functional capabilities. You can use BASIC on the 8000 computers in both interactive and program modes, with expanded commands and functions for arithmetic, editing, and disk file management. The CBM 8000 series computers are ideally suited for the computing needs of the business marketplace.

SCREEN
2000 character display, organized into twenty-five 80-column lines
64 ASCII, 64 graphic characters
3 x 8 dot matrix characters
Green phosphor screen
Brightness control
Line spacing: 1½ in Text Mode
1 in Graphics Mode

KEYBOARD
73-key typewriter style keyboard with graphic capabilities
Repeat key functional with all keys

MEMORY
CBM 8016: 16K (15359 net) random access memory (RAM)
CBM 8032: 32K (31743 net) random access memory (RAM)

POWER REQUIREMENTS
Volts: 110V
Cycles: 60 Hz
Watts: 100

SCREEN EDITING CAPABILITIES
Full cursor control (up, down, right, left)
Character insert and delete
Reverse character fields
Overstriking
Return key sends entire line to CPU regardless of cursor position

INPUT/OUTPUT
Parallel port
IEEE-488 bus
2 cassette ports
Memory and I/O expansion connectors

FIRMWARE
24K or ROM contains:
BASIC (version 4.0) with direct (interactive) and indirect (program) modes
9-digit floating binary arithmetic
Tape and disk file handling software

The 8000 Series will be available May/June '80

Model 8016	Model 8032	2040 Dual Floppy
	Available Now	
$1495	$1795	$1295

commodore

CBM™ IEEE MODEM

SPECIFICATIONS
*Full or half duplex operation
*300 bits per second
*Standard IEEE 488 interface
*Switch selectable originate, off, answer-full duplex, test, half duplex
*Visible indicators are transmit data, receive data, carrier ready, test
*Frequency shifted modulation
*Bell 103/113 compatible

Available June/July

$395

"Exceptional performance - even on noisy phone lines"

VISICALC FOR CBM $150 (Available June)

*CBM is a registered trademark of Commodore. All prices and specifications are subject to change without notice.

NEECO
679 Highland Ave.
Needham, MA
02194

Mon-Fri 9:30-5:30
MasterCharge & Visa Accepted
(617) 449-1760
Telex: 951021

MICROAMERICA DISTRIBUTING
"Dealer and OEM inquiries invited"
21 Putnam Street
Needham, MA
02194
(617) 449-4310

Basically Useful BASIC

Robert Lock

The response to this page in Issue #4 was so good that we're making it a continuing feature. To accomplish that, I'll need your help. Send in short, general purpose programs to my attention. If they're of general interest and utility, and will run on various machines without changes, I'll use them in this column. We're especially interested in programs for home or office application.

Programs (continuation of the three financial programs last issue): Marvin L. DeJong, School of the Ozarks

```
10 REM PROGRAM IV-CALCULATES THE ANNUAL ¬
      ¬PERCENTAGE RATE USING AN INTERVAL
15 REM HALVING TECHNIQUE
20 PRINT "WHAT IS THE BALANCE OF THE ¬
      ¬LOAN?"
30 INPUT BAL0
40 PRINT "ENTER YOUR MONTHLY PAYMENT."
50 INPUT PMT
60 PRINT "ENTER THE NUMBER OF PAYMENTS."
70 INPUT N
80 IH = 50/1200
90 IL = 0
100 FOR J = 1 TO 20
110 I = (IH+IL)/2
120 Q1 = I * BAL0
130 Q2 = PMT*(1-(1+I)^(-N))
140 IF Q1>=Q2 THEN IH=I
150 IF Q1<Q2 THEN IL=I
160 NEXT J
170 NT=N*PMT-BAL0
180 APR=1200*I
190 PRINT "YOUR ANNUAL PERCENTAGE RATE ¬
      ¬IS ";APR;"%."
200 PRINT "YOUR INTEREST IS $";NT
210 END

WHAT IS THE BALANCE OF THE LOAN?
?3000
ENTER YOUR MONTHLY PAYMENT.
?143.79
ENTER THE NUMBER OF PAYMENTS.
?23.2123282
YOUR ANNUAL PERCENTAGE RATE IS 10.8000279 %.
YOUR INTEREST IS $ 337.700674
```

```
10 REM PROGRAM V-CALCULATES THE PAYMENT ¬
      ¬SIZE FOR ADD-ON INTEREST
20 PRINT "HOW MUCH WAS YOUR LOAN"
30 INPUT BAL0
40 PRINT "HOW MANY MONTHLY PAYMENTS ¬
      ¬MUST YOU MAKE"
50 INPUT N
60 PRINT "WHAT IS YOUR ADD-ON INTEREST ¬
      ¬RATE IN PERCENT"
70 INPUT I
80 I=I/100
90 M=N/12
100 NT=BAL0*M*I
110 PMT=(BAL0+NT)/N
120 PRINT "YOUR PAYMENTS WILL BE $";PMT
130 PRINT "YOUR INTEREST ON THE ENTIRE ¬
      ¬LOAN IS $";NT
140 PRINT "IF YOU WANT TO KNOW THE ¬
      ¬ANNUAL PERCENTAGE RATE, EXECUTE ¬
      ¬PROGRAM IV."
150 END

HOW MUCH WAS YOUR LOAN
?4400
HOW MANY MONTHLY PAYMENTS MUST YOU MAKE
?36
WHAT IS YOUR ADD-ON INTEREST RATE IN
PERCENT
?7.25
YOUR PAYMENTS WILL BE $148.805556
YOUR INTEREST ON THE ENTIRE LOAN IS $957
IF YOU WANT TO KNOW THE ANNUAL
PERCENTAGE RATE, EXECUTE PROGRAM IV.
```

©

Expand with the best

FOR GRAPHICS...It's MIPLOT,

the intelligent plotter

MIPLOT by Watanabe Corporation

Designed for straight forward interface to any microcomputer that outputs the ASC11 code, MIPLOT can even be used by operators with no plotter experience.

- Incorporates pre-programmed "intelligent" functions required for producing graphs and drawings
- Solid and broken line types can be specified
- Built in character generator for letters, numbers and symbols
- Characters can be enlarged and rotated to four orientations
- Special printer mode outputs character data as-is
- Uses commonly available hard fiber-tip pens
- Maximum plot speed approximately 2 inches per second
- Built in self-test mode

Only $1,200

at Systems Formulate Corporation

FOR CHARACTER GENERATION...It's the new HAL PCG 6500

Programmable Character Generator

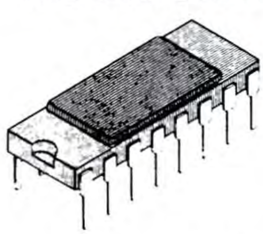
Only $200

at Systems Formulate Corporation

At last, available exclusively at Systems Formulate Corporation is the HAL PCG 6500, a programmable character generator for your PET. With the PCG 6500, you can program 64 of your own characters into the system. Other features:

- Built in CB2 style sound amplifier
- Interface to PET/CBM with 24 pin ROMS
- User program not required for displaying characters (only for programming them)
- Fully independent of user memory
- All software needed to write programs using programmed characters included

FOR MORE MEMORY...



Whether for Apple, TRS-80, PET or Sorcere, you'll like the reliability and price on this 16K Dynamic RAM Ceramic Package by Fujitsu.

Only $80

at Systems Formulate Corporation

*For more information or to order by phone: (415) 326-9100

Whatever your system or system expansion needs, give Systems Formulate a call. We will be happy to send along more information to you or take your phone order. We honor Master Charge, Visa or will ship COD.

Store Hours: Monday-Saturday, 10-7, Sunday 12-6

SYSTEMS FORMULATE CORPORATION

39 Town & Country Village • Palo Alto, California 94301

How to Program in Basic With the Subroutine Power of FORTRAN

Elizabeth Deal
337 W First Ave
Malvern, Pa.
19355

Fortran and PL/I programmers may take subroutines out of the library and append them easily to any main program. Those of us programming in Basic have to be very careful when we append previously written subroutines, since the variable names have to be identical, rather than passed through a COMMON storage area or a parameter list (as in Fortran). This article describes a way to simulate a COMMON storage area when programming in Basic. It is written for a new PET, but can be modified for an old PET or adapted for other microcomputers, such as Apple or Ohio Scientific. It requires some extra code which occupies about 2500 bytes, but it is fast and can handle all PET-supported n-dimensional arrays of strings and arithmetic values.

Subroutines in Basic are internal procedures in that variables active in the main program interact with those in the subroutine. In order to pass parameters to and from a subroutine that has previously been written for another application we must either rewrite the subroutine or assign all names from the calling program to names used by the subroutine. For single variables it is a nuisance, but easy to do. It is more complicated for arrays. Several options are available: (1) rewrite the subroutine or the main program, (2) move one element of the array at a time, (3) move the entire array at one time if the subroutine requires presence of the whole array. We'll avoid these unwieldy options.

To illustrate the problem, imagine that you have written a subroutine to find minimum and maximum values of a 1000 element array S, but the main program uses array M. Perhaps your subroutine uses S so many times that you dread the idea of rewriting it, or the main program calls the subroutine for many different variables. To use the appended subroutine, the main program will assign dimension of 1000 elements to arrays M and S. It will then move all elements of M array into S array and then use the subroutine. The trouble with this procedure is that two thousand elements are used where only one thousand are needed and it takes eight seconds to move the array one way while Basic interprets the instructions into machine code one thousand times. Were we to pass one value at a time, the memory area would be saved, but the program still would have to spend eight seconds moving the array element into a single variable.

My solution to this memory and time wasting lies in moving *only the name of the array* instead of the array elements. The array stays put. It is never moved, but its name is changed back and forth in about half a second.

The procedure to use for the entire project is to write the main program, append whatever subroutines one needs (via Toolkit or any merge or append program), initialize single variables, append the routine presented here (from line 2520 to 3680), and insert the linking information between two dotted lines in the main program.

More than one variable can share a name with the name used in the subroutine (see array MK% and M%). The number of such variables used in one program is limited only by what can fit in 255 bytes of string QS$. String QS$ together with the DIM statement is the key to the program. It is a directory of array names in the order in which they appear in the DIM statement. For instance:

Position in the QS$ string: 1 2 3 4 5 6 7
QS$ string contains: M%, MK%, M, MQ$, SI% S, SQ$
DIM statement contains: M%, MK%, M, MQ$.
* * *

In this example names that begin with "S" refer to arrays used in the subroutine. The asterisk is used to emphasize that they **must not** appear in the DIM statement. They should also not be dimensioned in your subroutine. Actually, it is delightful to get a REDIMENSIONED ARRAY ERROR here, since it is the proof that the name change works.

PET provides us with all the information we need to be able to change names. Pointer to start of arrays is stored in locations 44 and 45 (125 and 127 in old PET). Pointer to the end of arrays is in location 46 and 47 (128 and 129 for old PET). Array names are stored in the first two bytes preceding each array, and the memory size required by that array is in bytes three and four. In the initializing routine several Q-arrays are declared. The end of those arrays becomes the pointer to the beginning of the arrays that we are concerned with - those in the DIM statement between two dotted lines. The program (lines 2720 to 2840) reads a name of an array and stores each character, separately, and the address, in a table. The pointer is then moved to the next array by the number of bytes the first array used. When the names and addresses of arrays in common have been processed, the program then modifies the names in the QS$ string (lines 2960 to 3100) so that each byte will correspond to the internal format of the PET name storage. The two lists are then compared (lines 3100 to 3200) byte by byte and if they match, the program continues. If there is any discrepancy it is indicated in the error message and the job is abandoned. (If during the execution the arrays are moved, the job will also be abandoned and the program will count the byte shift. However, if it happens in a spot undetectable by my routines, the standard PET error messages will result.)

In order to tell the find + set subroutines which names to change we use position numbers in the QS$ string. Thus to replace name M with S we will set variable QM = 3 which corresponds to the third name in the QS$ list, and variable QS = 6 which corresponds to the sixth name in the QS$ list. The find + set routine will put a name referred to by the QS variable into the name referred to by the QM variable, or S over M. Now the work subroutine uses S, and on return, the original name is reset so that the calling program can continue whatever it is to do with array M.

The output of the demonstration program consists of showing what happens to the variables at any given time. It occupies about a screenful. Two trivial subroutines are used. #1 adds one to integers, finds MIN and MAX of a floating point array and adds two characters to the elements of string array. #2 creates an array in the subroutine and passes those values to the main program, showing that the process works in both directions and that it does not matter where the array is created.

To make it easier to understand the listing I have named all variables that are related to or used in the initialize, find + set and reset routines with a letter "Q" in the first position. Non-Q variables are general variables used in the main program and mathematical routines.

There are several important restrictions that must be observed. Most "BAD SUBSCRIPT ERROR", "ILLEGAL QUANTITY ERROR" messages occurring during the find + set routine and most of the erroneous data subsequent to return from reset routine will be due to failure to comply with these rules:

1. All single variables and functions (from the main program and subroutines) **must** be assigned a value prior to the DIM statement in the main program and prior to using undimensioned, small arrays. This is a good rule to follow anyway. Commodore advises that arrays are actually moved seven bytes each time a new single variable is defined in a program. Our program depends on arrays staying in one place throughout the entire program and subroutines. I have not yet found a simple way to have a really independent COMMON area.

2. Provide a string (QS$) of one or two character array names that will be used in common. The array name used in a calling program **must** precede the corresponding names in the subroutine. In the demonstration program M% and MK% precede the listing of SI%, but M% and MK% need not be adjacent to SI% or each other in the QS$ string.

3. Arrays from the main program that will be subject to name change **must** be dimensioned first. They **must** be contiguous. They must be in the same order in which they were listed in the QS$ string. The type of arrays that are common between the calling program and the subroutine **must** agree (i.e., integer with integer), the number of dimensions **must** agree, and their size must agree only to the extent required by the logic of the program. (If array M was dimensioned to twenty elements, but the subroutine tried to use an array S of fifty, you'd end up with a subscript error or garbage; if the array was dimensioned to fifty, and the work subroutine used twenty, there is no problem).

4. Just before going into the find + set subroutine, tell the program by use of QM and QS variables which names will be changed. QM must be smaller than QS.

The initialize, find + set and reset routines take up about 2500 bytes at execution time. That is equivalent to two 250 element arrays of floating point numbers. From the memory point of view it makes sense to use these routines only when the array size exceeds that amount. From the time point of view it makes sense to use them when the array size exceeds fifty elements. It takes half a second to change and reset the name, but it takes two seconds to move the array one way and two seconds to move it back. From the editing point of view it is, for me, easier to use this procedure than fool around with names. The hassle of defining single variables is made easier by PAICS Programmer's Toolkit DUMP command.

Note on terms used in this article:

Fortran programmers should note that this system simulates COMMON in its ability to pass the arrays in Basic, but that it is based on the concept of EQUIVALENCE, and can be used as such. PL/I programmers will see that it is similar to the DEFINED (and not BASED) attribute of variables, and that the allocation is still STATIC.

```
1000 REM==============================
1020 REM SIMULATION OF COMMON IN BASIC
1040 REM              BY
1060 REM        ELIZABETH DEAL
1080 REM 337 W.FIRST AVE, MALVERN, PA
1100 REM     19355; (215)647-4876
1120 REM        APRIL 9, 1980
```

```
1140 REM REF:
1160 REM 1) GENE BEALS OF AB COMPUTERS
1180 REM 2) PET USER NOTES 3) COMPUTE
1200 REM 4) LEN LINDSAY :OLD PET/NEW
1220 REM    PET IN MICROCOMPUTING
1240 REM 5) COMMODORE MANUAL
1260 REM==============================
1280 REM
1300 REM---DEFINE NON-Q SINGLE-V,FN---
1320 IN=20:FL=15:LI$="":FORI=1TO39:
     ¬LI$=LI$+"-":NEXTI:JI=Ø:JF=Ø:Q=Ø:
     ¬KX=Ø
1340 MN=Ø:MX=Ø:I=Ø:J=Ø:K=Ø:S1=2:S2=2:
     ¬S3=2:Z=64:L=Ø:M=Ø:N=Ø:TX=Ø
1360 REM
1380 REM..............................
1400 :
1420 REM LINK MAIN WITH Q SUBROUTINES
1440 REM DEFINE QS$,INITIALIZE IN THE
1460 REM SUB, DIM ARRAYS IN COMMON,
1480 REM BACK TO INIT., THEN MAIN....
1500 :
1520 QS$="M%,MK%,M,MQ$,SI%,S,SQ$"
1540 GOSUB2520
1560 DIM M%(IN),MK%(IN),M(FL)
1580 DIM MQ$(S1,S2,S3)
1600 GOSUB2720
1620 :
1640 REM..............................
1660 REM
1680 REM   BACK TO MAIN PROGRAM
1700 REM
1720 REM---DIM ALL OTHER ARRAYS-------
1740 REM         (NONE HERE)
1760 REM---ASSIGN VALUES--------------
1780 FORJ=1TOIN:MK%(J)=J:NEXT
1800 FORJ=1TOFL:M(J)=-J/100:NEXT
1820 FORI=1TOS1:FORJ=1TOS2:FORK=1TOS3:
     ¬MQ$(I,J,K)=CHR$(I+Z)+CHR$(J+Z)+CHR
     ¬$(K+Z)
1840 NEXTK,J,I
1860 REM---ASSIGN QS,QM FOR NAME
1880 REM   CHANGE; POKE NEW NAME; USE
1900 REM   IN SUBROUTINES; RESET NAME,
1920 REM   CONTINUE.
1940 REM
1960 PRINTLI$: REM DEMO FEW ARRAYS----
1980 QS=7:QM=4:GOSUB3300:QS=6:QM=3:
     ¬GOSUB3300:QS=5:QM=2:GOSUB3300:
     ¬REM FIND+SET
2000 JI=IN:JF=FL:I=S1:J=S2:K=S3:
     ¬GOSUB2260:PRINT"SUB#1, ";
2020 GOSUB3460:REM RESET ALL NAMES----
2040 PRINT:PRINT"(MK%)";:FORJ=1TOIN:
     ¬PRINTMK%(J);:NEXT:PRINT
2060 PRINT:PRINT"(M  )";:FORJ=1TOFL:
     ¬PRINTM(J);:NEXT:PRINT
2080 PRINTTAB(10)"MIN="MN", MAX="MX
2100 PRINT:PRINT"(MQ$)";:FORI=1TOS1:
     ¬FORJ=1TOS2:FORK=1TOS3:PRINTMQ$(I,
     ¬J,K);
2120 NEXTK,J,I:PRINT:PRINTLI$
2140 REM           DEMO ONE ARRAY-----
2160 QS=5:QM=1:GOSUB3300:KX=IN:GOSUB2380
     ¬:PRINT"SUB#2, ";:GOSUB3460
2180 PRINT:PRINT"(M%)";:FORJ=1TOIN:
     ¬PRINTM%(J);:NEXT:PRINT:PRINTLI$
2200 END:  REM END MAIN
2220 REM
2240 REM===WORK SUB#1=================
2260 ::FORQ=1TOJI:SI%(Q)=SI%(Q)+1:
     ¬      NEXTQ
2280 FORL=1TOI:FORM=1TOJ:FORN=1TOK:
     ¬SQ$(L,M,N)="*"+SQ$(L,M,N)+"/":
     ¬NEXTN,M,L
2300 MX=-1.7E38:MN=1.7E38:FORJ=1TOJF:
     ¬IFS(J)<=MNTHENMN=S(J)
2320 IFS(J)>=MXTHENMX=S(J)
2340 NEXTJ:RETURN
2360 REM===WORK SUB#2=================
2380 ::FORK=1TOKX:SI%(K)=K:NEXT:RETURN
2400 REM
2420 REM==*==*==*==*==*==*==*==*==*==*
2440 REM
2460 REM===SUB: INITIALIZE============
2480 REM
2500 REM---DCL Q'S: SINGLE,FN,ARR-----
2520 ::PRINT"INIT1":DEFFNQA(QP)=PEEK(QP)
     ¬+256*PEEK(QP+1)
2540 QB=Ø:QC=Ø:QD=Ø:QE=Ø:QF=Ø:QH=Ø:QI=Ø:
     ¬QJ=Ø:QK=Ø:QL=Ø:QM=Ø:QN=Ø
2560 QP=Ø:QQ=Ø:QR=Ø:QS=Ø:QT=Ø:QU=Ø:QW=Ø:
     ¬QX=Ø:QY=Ø:QZ=Ø:Q=Ø
2580 QK$="":QM$="":QL=LEN(QS$)/2
2600 IFPEEK(50003)=1THENQ5=44:Q6=46:
     ¬Q7=54:Q8=152:GOTO2640
2620 Q5=126:Q6=128:Q7=136:Q8=516
2640 ::DIMQG(QL),Q1(QL),Q2(QL),QD$(QL),
     ¬Q3(QL),Q4(QL),QN$(QL):FORQ=1TOQL
2660 QD$(QL)="":QN$(QL)="":NEXT
2680 QZ=FNQA(Q5):QI=FNQA(Q6):PRINT"MAIN"
     ¬:RETURN
2700 REM---ARRAYS FROM DIM------------
2720 ::PRINT"INIT2":QD=FNQA(Q7):
     ¬GOSUB3640:QH=QI:QE=FNQA(Q6)
2740 ::QK=QK+1:QG(QK)=QH:QC=QG(QK):
     ¬Q1(QK)=PEEK(QC):Q2(QK)=PEEK(QC+1)
2760 QX=Q1(QK):QY=Q2(QK)
2780 QT=Ø:QT=QT+37*(-(QX>=128ANDQY>=128)
     ¬)+36*(-(QX<128ANDQY>=128))
2800 QX=QX-128*(-(QX>128)):QY=QY+128*(-(
     ¬QY=Ø))-128*(-(QY>128))
2820 QD$(QK)=CHR$(QX)+CHR$(QY)+CHR$(QT):
     ¬REM DO NOT COMPARE QD$ TO QN$ !!
2840 QN=FNQA(QC+2):QH=QH+QN:IFQH<QEGOTO2
     ¬740
2860 REM---ARRAYS FROM QS$------------
2880 QU=1:FORQ=1TOLEN(QS$):QK$=MID$(QS$,
     ¬Q,1):IFQK$=","THENQU=QU+1:NEXTQ
2900 QN$(QU)=QN$(QU)+QK$:NEXTQ
2920 IFQU<2THENPRINT"***CORRECT QS$":END
2940 REM---FIX NAMES IN QS$-----------
2960 FORQ=1TOQU:QM$=QN$(Q):QF=Ø:QJ=Ø:
     ¬QT=ASC(RIGHT$(QM$,1))
2980 QT=QT*(-(QT=36ORQT=37)):QF=128*(-(Q
     ¬T=37)):QJ=128*(-(QT=37ORQT=36))
3000 IFLEN(QM$)>2GOTO3060
3020 IFQJ=ØANDQT=ØTHENQM$=QM$+CHR$(Ø):
     ¬GOTO3060
3040 QM$=LEFT$(QM$,1)+CHR$(128)+CHR$(QT)
3060 ::QX=QF+ASC(LEFT$(QM$,1)):QY=QJ+ASC
     ¬(MID$(QM$,2,1)):QY=QY-QJ*(-(QY>255
     ¬))
3080 Q3(Q)=QX:Q4(Q)=QY:REM PRINTQ;TAB(4)
     ¬QM$;TAB(10)QX;QY
3100 NEXTQ:QQ=Ø:FORQ=1TOQK:IFQ1(Q)=Q3(Q)
     ¬ANDQ2(Q)=Q4(Q)GOTO3140
3120 QQ=QQ+1:QN$(Q)=QN$(Q)+"  <-- ??"
```

```
3140 ::NEXTQ:IFQQ=0GOTO3200
3160 PRINT:PRINT"***NO MATCH !":PRINT:
     ¬PRINT"QS/QM  QS$"TAB(20)"DIM":
     ¬PRINT
3180 FORQ=1TOQU:PRINTTAB(1)QTAB(7)QN$(Q)
     ¬TAB(20)QD$(Q):NEXTQ:END
3200 PRINT"MAIN.":PRINT"PRESS 'SHIFT' ¬
     ¬TO CONT MAIN PRGM":WAITQ8,1:
     ¬PRINT" OK"
3220 RETURN
3240 REM
3260 REM===SUB: FIND+SET NAME=========
3280 REM
3300 ::QD=FNQA(Q7):GOSUB3640
3320 IFQS<=QMTHENPRINT"***BAD POSITION ¬
     ¬NUMBERS":PRINT"QS="QS", QM="QM:END
3340 QR=QG(QM):PRINT"SET "QN$(QS)TAB(8)"
     ¬OVER "QN$(QM);:PRINTTAB(18)"AT"QR
3360 QB=QB+1:QG(QM)=-QG(QM):POKEQR,
     ¬Q3(QS):POKEQR+1,Q4(QS)
3380 RETURN
3400 REM
3420 REM===SUB: RESET NAME============
3440 REM
3460 ::IFQB<1THENPRINT"***IMPROPER CALL ¬
     ¬TO RESET":END
3480 QD=FNQA(Q7):GOSUB3640:PRINT"RESET ¬
     ¬NAME AT";
3500 FORQ=1TOQK:IFQG(Q)>0GOTO3540
3520 QG(Q)=-QG(Q):QR=QG(Q):POKEQR,Q1(Q):
     ¬POKEQR+1,Q2(Q):PRINTQR;
3540 ::NEXTQ:PRINT:RETURN
3560 REM
3580 REM---CHECK COMMON ARRAYS POS----
3600 REM   CALLED BY INIT,SET,RESET
3620 REM
3640 ::QW=FNQA(Q5):IFQW=QZTHENRETURN
3660 PRINT: PRINT"***ARRAYS MOVED"QW-QZ"
     ¬BYTES"
3680 PRINT"   PRIOR TO LINE"QD:END
3700 REM==============================
```

©

1979 Periodical Guide For Computerists

The January-December **1979 PERIODICAL GUIDE FOR COMPUTERISTS** indexes over 2000 articles from 20 personal computing and professional electronic publications. Articles, editorials, book reviews and letters from readers which have relevance to the Personal Computing field are indexed by subject under 110 categories. A list of the authors is cross-referenced by subject to aid in locating articles. The 80 page book is available for $5.95 from E. Berg Publications, 622 East Third, Kimball, NE 69145 or from local computer stores. A free brochure which describes the book in more detail is available on request. Editions covering 1976, 1977, and 1978 are available for $5.00 each. ©

Computers and The Handicapped

Susan Semancik

The Delmarva Computer Club
P.O. Box 36
Wallops Island, VA 23337

One of the biggest problems facing programmers who have neither hardware nor electrical expertise is finding "off-the-shelf" items that will interface directly to their computer allowing input alternatives for the handicapped. Programmers can devise specialized coding for different applications of the interface device, but first they need a device with which to work.

The Delmarva Computer Club is trying to develop quick, inexpensive and versatile interfaces for both the blind and the quadriplegics in our area. A device we are exploring that is readily available and plugs into the PET's User Port is INNOVISION's Prestodigitizer board. The original software of the board trains the computer to recognize the user's stroke set for either digits or letters of the alphabet. This board can be as versatile an input device as the programmer needs in many applications.

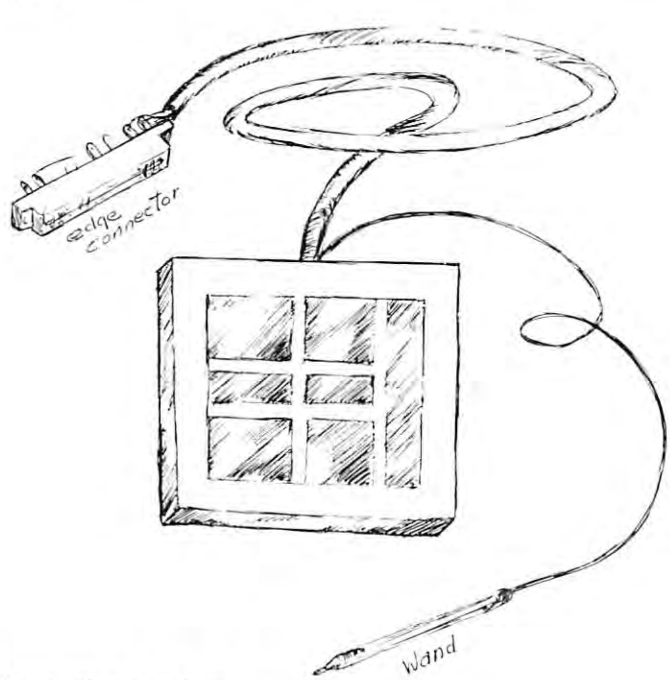
For those not familiar with the board, it is a small tablet partitioned into seven copper sections. As can be seen in Figure 1, the largest section is a vertical strip extending the length of the right edge of the tablet. This is used in the original software to indicate the end of a sequence of input strokes made on the other six horizontal sections. Each section is wired to one of the PA lines of the User Port. A metal-tipped wand is used to touch the copper sections, thereby completing the electrical circuit indicating which line is active. In the original software, the sequence of sections selected in defining a character is packed into a string during the training part of the program and compared against the user's inputs during the recognition part. The software is writted so that each stroke is not influenced by the length of time contact is made by the wand. The original program prevents the choice of the same section consecutively without an intervening choice. This is fine for defining the shape of digits and letters, but for other purposes it can be programmed differently, as Program 2 will illustrate.

Also, originally only five of the six horizontal sections were wired to the PA pins of the User Post. This is acknowledged in the Prestodigitizer's Manual, but no reason is given for its exclusion. Those of you handy enough to pry the glued back cover off the board, drill into the material backing the unattached region, and solder a connecting wire between the copper of the region and PA1, can easily remedy this situation. Only the application used in Program 3 below needs this hardware change.

The Prestodigitzer board can be implemented in a computer system for a quadriplegic by having the wand held in the user's mouth or attached to a spring on a frame within grasp of the user's lips. A sample program to show how the different regions can be used as inputs without any modifications to the board or software is Program 1, which oroginally appeared in the PET USER NOTES, Volume 1, Issue 7, under the name ARROW, and was written by Jim Butterfield. It has been modified for variable snake speed and Prestodigitizer input.

Since the input selections needed in the program are YES, NO, RIGHT, LEFT, UP, and DOWN, each one of the wired regions of the board can be uniquely assigned an input response. Not only can the program explicitly describe the regions corresponding to each word, but the words can be written on pieces of index cards and taped on the board adjacent to the appropriate region. Figure 2 shows the arrangement used in Program 1. Overlays could be designed for different programs requiring no more than six types of responses, adding to the versatility of the board.

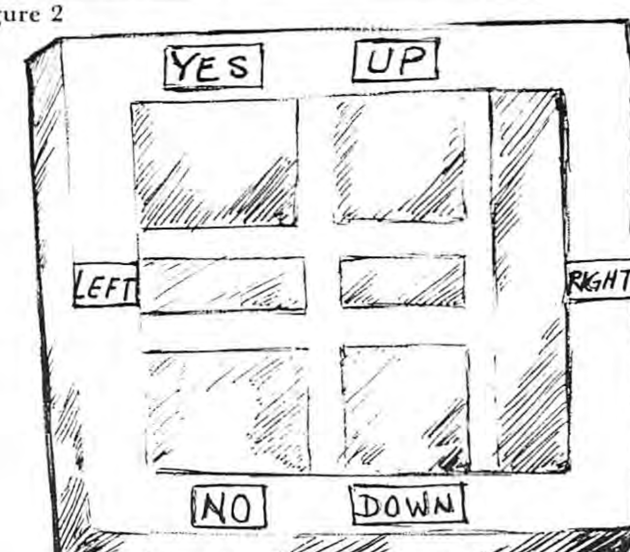
Figure 1



Illustrations courtesy C. Marshall Curtis

```
12 GOSUB13:GOTO20
13 P=59471:POKE59459,0
14 PRINT"ĥṟ ARROW r̂ GAME"
15 PRINT:PRINT"SET THE TIMER FOR THE ¬
      ¬SPEED THE SNAKE   WILL MOVE BY";
16 PRINT" TOUCHING THE REGION MARKED ";
17 PRINT"ṟRIGHTr̂ ON THE TABLET WHEN ¬
      ¬YOUR NUMBER    CHOICE APPEARS.↓"
18 PRINT"THE NUMBERS WILL CONTINUOUSLY ¬
      ¬MOVE FROM  0(FASTEST) TO 100(SLOWE
      ¬ST)."
19 GOSUB2000:RETURN
20 PRINT"ĥINSTRUCTIONS? - TOUCH THE ¬
      ¬REGIONS MARKED ṟYESr̂, OR ṟNOr̂."
21 LS=PEEK(P):IFLS=255THEN21
22 IFLS=247THEN100
23 IFLS<>254THEN21
30 PRINT"ĥGUIDE THE MOVING 'SNAKE' WITH ¬
      ¬THE          REGIONS MARKED:"
35 PRINTTAB(15)"↓ṟUPr̂":PRINT"ṟLEFTr̂"TAB(
      ¬28)"ṟRIGHTr̂"
40 PRINTTAB(15)"ṟDOWNr̂":PRINT:PRINT:
      ¬PRINT
50 PRINT"DON'T HIT THE BOUNDARY (OR THE ¬
      ¬SNAKE);"
60 PRINT" ..TRY TO HIT THE BOXES FOR ¬
      ¬POINTS."
65 PRINT"↓BUT BEWARE! THE SNAKE GROWS ¬
      ¬AS IT HITS  BOXES!!":PRINT:PRINT
70 PRINT"YOU HAVE 60 SECONDS OF PLAY. ¬
      ¬GOOD LUCK!":PRINT:PRINT
80 PRINT:PRINTTAB(2)"TOUCH THE REGION ¬
      ¬MARKED ṟNOr̂ TO START"
90 LS=PEEK(P):IFLS=255THEN90
95 IFLS<>247THEN90
100 DIMP(80),D(3)
110 D(0)=22:D(1)=60:D(2)=62:D(3)=30
120 T9=32768:T6=3599:POKE59468,12
130 M1=59467:M2=59466:M3=59464
140 PRINT"ĥ  SCORE: 0"
150 FORJ=32808TO32847:POKEJ,81:POKEJ+920
      ¬,81:NEXTJ
160 FORJ=32848TO33728STEP40:POKEJ,81:
      ¬POKEJ+39,81:NEXT
170 V=20:H=35:V1=0:H1=-1:P2=10:D1=1
180 TI$="000000"
190 PRINT"h";RIGHT$(TI$,2):IFTI>T6THEN48
      ¬0
200 LS=PEEK(P):IFLS=255ORLS=OLTHEN230
205 OL=LS
215 IFLS=239THENZ=0:GOTO220:REM DOWN
216 IFLS=251THENZ=1:GOTO220:REM LEFT
217 IFLS=223THENZ=2:GOTO220:REM RIGHT
218 IFLS=253THENZ=3:REM UP
220 D1=Z:D=Z-1.5:V1=INT(ABS(D))*SGN(D):
      ¬H1=SGN(D)-V1
230 V=V-V1:H=H+H1
240 SP=32768+V*40+H
250 P9=PEEK(SP):POKEM1,16:POKEM3,
      ¬29*D1+80:POKEM2,15
260 R6=R7:R7=R7+1:IFR7>P2THENR7=0
270 P1=P(R7):P(R7)=SP:POKEM1,0:IFP1<>0TH
      ¬ENPOKEP1,32
280 POKESP,D(D1):P1=P(R6):IFP1<>0THENPOK
      ¬EP1,81
285 FORXY=1TOXTY:NEXTXY
290 IFP9<>32GOTO400
300 IFRND(1)>.05GOTO190
310 FORV3=V2-1TOV2+1:P3=V3*40+T9:
      ¬FORH3=H2-1TOH2+1:IFPEEK(P3+H3)<>10
      ¬2GOTO330
320 POKEP3+H3,32
330 NEXTH3,V3:T=0:POKEP8,32
340 V2=INT(RND(1)*20)+3:H2=INT(RND(1)*36
      ¬)+2
350 FORV3=V2-1TOV2+1:P3=V3*40+T9:
      ¬FORH3=H2-1TOH2+1:IFPEEK(P3+H3)<>32
      ¬GOTO340
360 NEXTH3,V3:FORV3=V2-1TOV2+1:P3=V3*40+
      ¬T9:FORH3=H2-1TOH2+1
370 POKEM1,16:POKEM2,15:POKEM3,30
380 POKEP3+H3,102:POKEM1,0
390 NEXTH3,V3:T=9*RND(1):P8=V2*40+H2+T9:
      ¬POKEP8,49+T:GOTO190
400 IFP9<>102GOTO460
410 T$=TI$
420 T=T-1:S=S+1:POKEP8,T+49:POKEM1,16:
      ¬POKEM2,15
430 PRINT"h→→→→→→→→→";S
440 FORJ=100TO30STEP-1:POKEM3,J:NEXT:
      ¬POKEM1,0:IFT>=0GOTO420
450 P2=P2+1:TI$=T$:GOTO310
460 POKEM1,16:POKEM2,15:POKEM3,200:
      ¬FORJ=1TO1000:NEXT
470 POKEM1,0
480 PRINT"ANOTHER GAME? Y OR Nr̂";
490 LS=PEEK(P):IFLS=255ORLS=OLTHEN490
496 IFLS=247THENZ$="N"
497 IFLS=254THENZ$="Y"
500 IFZ$="Y"THENCLR:GOSUB13:GOTO100
510 IFZ$<>"N"THEN490
520 PRINT"ṟ";
530 POKE59467,0:POKE59466,0:POKE59464,
      ¬221:END
2000 FORI=0TO100STEP5
2010 ZZ=33268:AA=LEN(RIGHT$(STR$(I),
      ¬LEN(STR$(I))-1))
2020 FORJ=0TOAA:VV=VAL(MID$(STR$(I),J+1,
      ¬1)):POKEZZ+J,VV+176:NEXTJ
2030 FORK=1TO50
2031 LS=PEEK(P):IFLS<>255THEN2050
2040 NEXTK:NEXTI:FORJJ=0TO4:POKEZZ+JJ,
      ¬32:NEXTJJ:GOTO2000
2050 IFLS<>223THEN2031
2060 XTY=I:RETURN
```

Figure 2



Another possible use of the board by the handicapped who know Morse Code is to input characters using this code. Only two regions are needed to signal either a dot or dash selection, and either a time delay or the touching of the vertical section can be used to indicate the end of a character.

Program 2 is an example of how to input the letters of the alphabet using Morse Code on the Prestodigitizer board. The disassembled listing for Program 2 is the 6502 assembly language part of the program and allows repeat selection of a section, using the top left horizontal region for a dot, the top right horizontal region for a dash, and the verical region for the end of a character. The dots and dashes are rotated into memory location 981 as either zeroes or ones, with memory location 983 containing the number of characters entered.

Each alphabetic character has two entries in the defining table starting at memory location 5000. The first entry has the number of dots and dashes in the character's code, and the second entry has the bit pattern corresponding to the dots and dashes. Once a character has been entered from the Prestodigitizer, the program searches the table for characters with the same number of entries. When one is found, the next table entry is compared with

```
5 REM DIGITIZER MORSE PROGRAM
6 REM
7 REM USE REGION 1 TO INPUT A DOT; ¬
      ¬REGION 2 FOR A DASH; REGION 7 TO ¬
      ¬END INPUT
8 REM
9 REM TO END PROGRAM, TURN COMPUTER OFF
10 DATA169,0,141.67,232,141.215,3,169,0,
      ¬141,213,3,169,255,141.212,3
20 DATA173,79,232,201,255,240,244,201,
      ¬223,240,51,205,212,3,240,240
30 DATA201,254,208,10,141,212,3,24,46,
      ¬213,3,76,117,3
40 DATA201,253,208,222,141.212,3,56,46,
      ¬213,3
50 DATA238,215,3,160,150,140,214,3,160,
      ¬255,136,208,253,206,214,3,208,246
60 DATA76,76,3,169,64,141.216,3,160,0,
      ¬174,215,3,177,0,205,215,3,240,12,
      ¬200,200
70 DATA206,216,3,192,51,16,19,76,145,3,
      ¬200,177,0,205,213,3,208,237
80 DATA152,24,109,216,3,32,210,255,76,
      ¬58,3,-1
90 DATA2,1,4,8,4,10,3,4,1,0,4,2,3,6,4,0,
      ¬2,0,4,7,3,5,4,4,2,3,2,2,3,7
95 DATA4,6,4,13,3,2,3,0,1,1,3,1,4,1,3,3,
      ¬4,9,4,11,4,12,0,0,-1
100 I=826
110 READOP:IFOP=-1THEN125
120 POKEI,OP:I=I+1:GOTO110
125 I=5000
126 READOP:IFOP=-1THEN130
127 POKEI,OP:I=I+1:GOTO126
130 POKE0,136:POKE1,19:PRINT"ĥ";:SYS826:
      ¬PRINTPEEK(983),PEEK(981):STOP
```

location 981's contents. If these are identical, the correct ASCII value is calculated so the alphabetic character can be printed on the PET's screen. If no table value corresponds, then nothing is printed on the screen. Since this is just a sample program, it is designed for the user to be able to input characters forever, and the only way to terminate the program is to turn the computer off.

As an example of its use, to input the Morse Code for the character F, which is ..-., the user would take the wand and touch it to the upper left horizontal region, lift it up, touch the upper left horizontal region, lift it up, touch the upper right horizontal region, lift it up, touch the upper left horizontal region, lift it up, and finally touch the vertical region. You can enter this code remarkably fast, but you must be careful to hold the wand perpendicular to the region when you make contact, and not to slide the wand when you lift it from the surface. Otherwise, since the computer will accept consecutive entries from the same region, more than one entry may be mistakenly entered because of either the uneven surface or dirt on the surface of the region.

We envision that this arrangement could also be used by the blind to input braille characters to the computer, since the braille cell arrangement is also a two column, three row arrangement. This may or may not be an easy way for the blind to input characters to the computer, but at least it gives a blind person who is not familiar with a keyboard a way to communicate with the computer in a familiar code. All too often people devise new codes for the blind, putting an extra burden ont hem in that the code must be learned before it can be used by them. Why not put the burden on the programmer to incorporate the braille code into the input device, thereby making the device immediately accessible to the blind for their use. Program 3 will translate the braille input code for braille letters into corresponding screen characters. The disassembled listing for Program 3 is the 6502 assembly language part of the program. It does not allow repeat selection of a section, since it is not needed in defining a braille character, which is a combination of dots chosen in a six cell arrangement.

For example, the letter 0 has the following dot arrangements:

This can be signaled to the computer in this program by the user taking the wand and touching the upper left horizontal region, lifting it up, touching the middle right horizontal region, lifting it up, touching the lower left horizontal region, lifting it up, and finally touching the vertical region. For the blind to be able to use this system, they must have some

Computer Supplies for Small Computer Systems

Commodore Printer Ribbons

IN STOCK

Our full line of supplies includes:

Dikettes (3M, basf)
Diskette retrieval devices
BPI Business Package for the Commodore
Tractor Printer Paper
Tractor Printer Labels
Printer Ribbons for NEC, Commodore and Xymec
Dust Covers
Adapters
Uncrashers
Computer Desks
NEC Spinwriters
Print Thimbles
Xymec Intelligent Printers
"New Dealer" Kits

ZEIGLER ELECTRONIC PRODUCTS

DEALERS:
We no longer market COMMODORE Equipment. (Jerry Zeigler is with Commodore) but Connie and I have the computer supplies you need to support Commodore System Sales.

Give us a call:
Trudie Zeigler
or
Connie Rittenberry
(404) 289-1596 or 289-2265
3661 CALUMET RD.
DECATUR, GEORGIA, 33034

NOW YOU ARE A PART OF THE GREATEST STORIES EVER CREATED!

INTERACTIVE FICTION ™

What is it?

Interactive Fiction is story-telling using a computer, so that you, the reader, can actually take part in the story instead of merely reading it.

How does it work?

The computer sets the scene with a fictional situation, which you read from the CRT. Then, you become a character in the story: when it's your turn to speak you type in your response.

The dialogue of the other characters and even the plot will depend on what you say.

Is it a game?

No. In a game the situation is rigidly defined and you can select from only a limited number of responses. But in Interactive Fiction you can say anything you like to the other characters. (Of course if your response is too bizarre they may not understand you).

What is available now?

Robert Lafore, writer, columnist, and programmer, has created a series of works in Interactive Fiction. Eash is available on a 5¼" diskette for use on a TRS-80* Level-II with at least 32K memory and one disk drive. Requires user to have a copy of TRSDOS 2.2 or 2.3.

Six Micro Stories offers a good introduction to Interactive Fiction. Six very short stories involve you, the reader, in a variety of situations: you are an American spy in Hitler's Third Reich, the pilot of a doomed 747, and more.

Local Call for Death is a detective story in the style of Lord Peter Wimsey. Considerably more challenging than the above program, this one will put your analytic skills (and social savoir-faire) to the test.

Two Heads of the Coin is a psychological mystery set in the London of Sherlock Holmes. Most challenging of all, this program will tax your observational skills and above all your imagination.

Please send me the following works of Interactive Fiction:

Six Micro-Stories $14.95
Local Call for Death $19.95
Two Heads of the Coin $19.95
(Florida residents add 4% sales tax.)

"Also available worldwide through your local adventure dealers"

Name ____________
Address ____________
City, State, Zip ____________

TO ORDER: Mail the above form with your payment to: **Adventure International, P.O. Box 3435, Longwood, FL 32750.**

☐ MasterCharge ☐ Visa ☐ Check ☐ C.O.D.

M.C. or Visa Number: ____________

Signature ____________

*TRS-80 is a trademark of the Tandy Corporation.

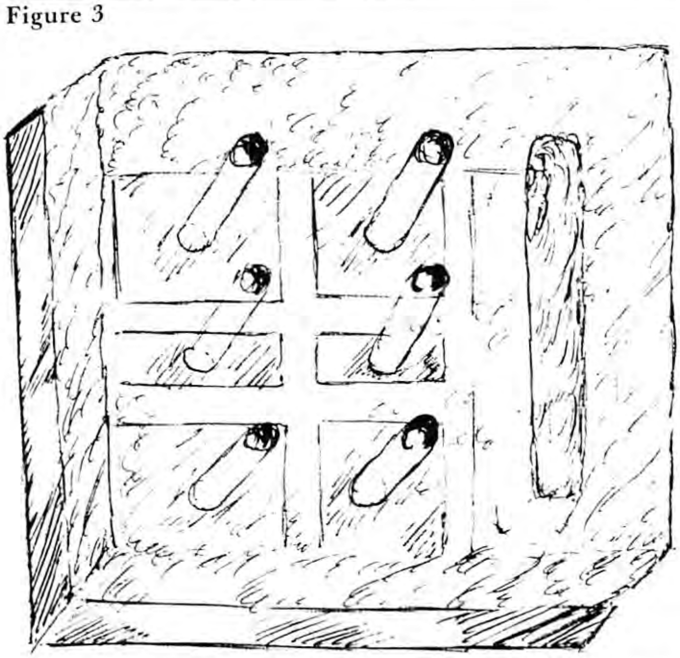
adventure international

means of determining the location of a region without making any contact. For this purpose, a piece of foam can be cut the size of the board and notched at the upper edge for the blind to be able to determine the top of the board. Then cut round holes over each of the six horizontal regions to represent the braille cells, and a vertical region to represent the "end-of-character" region. See figure 3 for a rough sketch of this arrangement. This operation takes no hardware skill as such, and certainly serves the purpose.

As each region is touched, its input value is converted to an appropriate power of two from the table which starts at location 990. These powers of two are added together to uniquely determine a number for the character. This number will point to the ASCII value of the alphabetic character in the table starting at location 929. Note that unused numbers have a zero entry in the table. Then the character is printed on the PET's screen. This program is also designed to run continuously, and can only be terminated by turning the computer off.

Several new codes will have to be developed for the special function keys such as RETURN, STOP, SHIFT, GRAPHICS, and CURSOR keys to be a fully implemented braille programming system. We will be using this device with some blind programmers and will try to develop a braille compatible code. We welcome other people's suggestions in these endeavors and hope that others of you in need of "off-the-shelf" merchandise will start to look at devices designed for other initial purposes, but which may in fact be ideal for computer applications for the handicapped. Please share your discoveries with us. The information you provide may be the breakthrough others need in order to solve their programming challenges.

Figure 3



```
5 REM DIGITIZER BRAILLE PROGRAM
6 REM
7 REM USE REGIONS 1THROUGH 6 FOR THE ¬
      ¬BRAILLE CELL INPUTS
8 REM USE REGION 7 TO END INPUT
9 REM TO END PROGRAM, TURN COMPUTER OFF
10 I=826
20 READOP:IFOP=-1THEN100
30 POKEI,OP:I=I+1:GOTO20
40 DATA169,0,141.67,232,169,255,141,152,
      ¬3,169,0,141.153,3,173,79,232,201,
      ¬255
50 DATA240,249,205,152,3,240,244,201,
      ¬223,240,34,141.152,3,160,6
60 DATA209,6,240,6,136,208,249,76,63,3,
      ¬169,1,136,240,4,10,76,106,3
70 DATA24,109,153.3,141,153,3,76,73,3,
      ¬172,153.3,192,60,48,3,76,63,3
80 DATA192,0,16,3,76,63,3,177,0,32,210,
      ¬255,76,63,3,-1
100 I=929
110 READOP:IFOP=-1THEN200
120 POKEI,OP:I=I+1:GOTO110
130 DATA0,65,0,67,0,66,73,70,0,69,0,68,
      ¬0,72,74,71,0,75,0,77,0,76,83,80,0,
      ¬79
140 DATA0,78,0,82,84,81,0,0,0,0,0,0,0,0,
      ¬0,0,0,0,0,0,87,0,0,85,0,88,0,86
150 DATA0,0,0,90,0,89,-1
200 I=990
210 READOP:IFOP=-1THEN300
220 POKEI,OP:I=I+1:GOTO210
230 DATA254,253,251,191,247,239,-1
300 POKE0,161:POKE1,3:POKE6,221:POKE7,3
400 PRINT"ñ";: SYS(826)
```

Disassembled Listing For Program #2

826:	**LDA# 0**	Set User Port for input.
	STA 59459	
	STA 983	Location 983 will count the number of entries for each character.
	LDA # 0	Location 981 will contain the bit
	STA 981	pattern for the entries.
	LDA# 255	Location 980 will contain the last
	STA 980	selection and is preset to indicate no selection has been made.
844:	**LDA 59471**	Look for an input.
	CMP# 255	If none, then keep looking.
	BEQ -12	
	CMP# 223	If it is the vertical section, then a
	BEQ +51	character has been completely entered, so go to the table lookup.
	CMP 980	If contact has not been broken, then
	BEQ -16	keep looking.
	CMP# 254	Is it the upper left horizontal region?
	BNE +10	If not, then go to the next question.
	STA 980	If yes, then store the last selection in
	CLC	location 980, and rotate a zero
	ROL 981	into location 981.
	JMP 885	Continue at the delay loop.
	CMP# 253	Is it the upper right horizontal region?
	BNE - 34	If not, then keep looking.
	STA 980	If yes, then store the last selection in
	SEC	location 980, and rotate a one into
	ROL 981	location 981.
885:	**INC 983**	Increase the entry count by one.
	LDY# 150	Delay loop.
	STY 982	

```
        LDY# 255
        DEY
        BNE -3
        DEC 982
        BNE -10
        JMP 844     Go back and keep looking.
        LDA# 64     Table Look-up starts here. Location
        STA 984         984 will hold the value needed to
                        calculate the ASCII of the character.
        LDY# 0      If it is the same as the last input, then
                        keep looking.
913:    LDX 983
        LDA(0), Y   Locations 0 and 1 are pointing to the
                        start of the look-up table.
        CMP 983     Find a table value with the same
        BEQ +12         number of entries.
        INY
        INY
        DEC 984     If not the same, update the ASCII
                        information location.
        CPY# 51     If beyond the table limits, then go
        BPL +19         back to looking for another input.
        JMP 913     Otherwise, keep looking in the table.
        INY
        LDA(0), Y   Is the entry pattern in the table the
        CMP 981         same as the input pattern stored
                        in location 981?
        BNE - 19    If not, then keep looking in the table.
        TYA         If so, then use the table location to
        CLC             adjust the ASCII value stored in
                        location 984.
        ADC 984
        JSR 65490   Go to the printing routine.
        JMP 826     Get another input.
```

Disassembled Listing For Program #3

```
826:    LDA# 0      Set User Port for Input.
        STA 59459
831:    LDA# 255    Location 920 will contain the last
        STA 920         selection, and is preset to indicate
                        no selection has been made.
        LDA# 0      Location 921 will hold the combination
        STA 921         inputs for a braille character.
841:    LDA 59471   Look for an input.
        CMP# 255        If none, keep looking.
        BEQ -7
        CMP 920     IF it is the same as the last input, then
        BEQ -12         keep looking.
        CMP# 223    If it is the vertical section, then a
        BEQ +34         character has been completely
                        entered, so go to the table lookup.
        STA 920     Store the input in location 920.
        LDY# 6      Change the input value into a section
        CMP(6), Y       number which is stored in a table
        BEQ +6          to which locations 6 and 7 are
        DEY             pointing.
        BNE - 7
        JMP 831
        LDA# 1      Change the section number into a
874:    DEY             power of two.
        BEQ +4
        ASLA
        JMP 874
        CLC         Add to previous powers entered and
        ADC 921         store in location 921.
        STA 921
        JMP 841
        LDY 921     Check that number is within the table
        CPY# 60         range.
        BMI +3
        JMP 831
        CPY# 0
        BPL +3
        JMP 831
        LDA(0), Y   Find the ASCII value in the table to
                        which locations 0 and 1 are
                        pointing.
        JSR 65490   Go to the printing routine.
        JMP 831     Get another input.
```

©

Al Baker's Programming Hints for Atari/Apple

Programming is the most complex and least organized of human endeavors. Well, maybe after the U.S. Government and raising children. Many people have tried to bring order out of this chaos. In this column, I will join that noble company. With your help, we just might pull it off.

Two of my favorite computers are the Apple and Atari. They are superbly designed machines. (It's not that I don't like the PET. I do. I guess I'm just hopelessly addicted to sound and color, joysticks and paddles.) In this column, I am going to help you use the sound, color, and attachments of these two computers.

In each issue, I will show you one or two short routines fully utilizing some feature of an Atari or Apple. I'll put the routines to work and leave you with a chance to work on a programming exercise, answered in the next issue.

I said I needed your help. Send me any routines you have and would like to share. If I use them, you'll be given credit as the source.

The Atari Joystick

This month, we are going to explore the Atari joystick. The position of a joystick is read with the function STICK. The joysticks are numbered from 0 to 3. Thus, the position of the second joystick is STICK(1). This function returns the number 15 when the joystick is centered. Here are the results of STICK for the other joystick positions.

14 10 6 11 15 7 9 5 13

The button on the joystick is read with the function STRIG. STRIG(0) reads the button of the first joystick. The function is zero if the button is pushed and one if the button is not pushed.

Most programs use the joystick to move objects around on the screen. As soon as the program needs a yes or no response, however, the players must use the keyboard. This is inconvenient, especially when they are several players, none sitting close to the keyboard. Why not use the joystick to make the selection?

Two Entry Menu Selection

Look at the first listing. This is a routine which uses the joystick to get a yes or no response from a player. Line 45 turns off the cursor on the screen. Lines 60 through 140 set the default answer and display the options, YES NO, on the screen. The default answer, in this case YES, is highlighted in reverse video.

Listing 1. Two Entry Menu Select
The words in the boxes are typed using the Atari key to put them in reverse video.

```
10 REM ... TWO ENTRY MENU SELECT ...
20 REM         FROM JOYSTICK
30 REM
40 REM
43 REM TURN OFF CURSOR
45 POKE 752,1
47 REM
50 REM DEFAULT ANSWER:
60 A$="Y"
70 REM
80 REM  DISPLAY MENU
90 REM  XY,YY IS POSITION OF YES
100 REM XN,YN IS POSITION OF NO
105 REM
110 POSITION XY,YY
120 PRINT "YES";
130 POSITION XN,YN
140 PRINT "NO";
150 REM
160 REM SCAN JOYSTICK FOR YES
170 REM
180 IF STICK(PLAYER-1)<>11 THEN 280
190 A$="Y"
200 POSITION XY,YY
210 PRINT "YES";
220 POSITION XN,YN
230 PRINT "NO";
240 GOTO 180
250 REM
260 REM SCAN JOYSTICK FOR NO
270 REM
280 IF STICK(PLAYER-1)<>7 THEN 380
290 A$="N"
300 POSITION XY,YY
310 PRINT "YES";
320 POSITION XN,YN
330 PRINT "NO";
340 GOTO 180
350 REM
360 REM SCAN TRIGGER FOR CHOICE
370 REM
380 IF STRIG(PLAYER-1) THEN 180
390 REM
400 REM WE HAVE ANSWER
410 REM
420 PRINT A$
```

The routine assumes that the word YES is to the left of the word NO. If the joystick is moved to the left then lines 180 through 240 sets the answer to Y and highlights the word YES on the screen. If the joystick is moved to the right, then lines 280 to 340 sets the answer to N and highlights the word NO. Pushing the button ends the routine. This is handled

Listing 2. Do You Love Me?

```
10 REM ... DO YOU LOVE ME ...
20 REM
30 REM
40 REM DECLARE STRINGS AND CONSTANTS
50 DIM A$(1)
60 C1=1
70 REM
80 REM ASK MY OWNER IF HE LOVES ME
90 REM
95 GRAPHICS 0
100 XY=12:YY=18
110 XN=24:YN=18
120 PLAYER=1
130 POSITION 12,10
140 PRINT "DO YOU LOVE ME?"
150 GOSUB 1000
160 REM
170 REM RESPOND TO ANSWER
180 REM
190 POSITION 3,22
200 IF A$="Y" THEN PRINT "     SHUCKS, I
 LOVE YOU TOO."
210 IF A$="N" THEN PRINT "WELL, I LOVE Y
OU ANYWAY. (SNIFFLE)"
220 FOR DELAY=1 TO 1000
230 NEXT DELAY
240 GOTO 95
960 REM
970 REM JOYSTICK ROUTINE
980 REM (DISCUSSED ELSEWHERE)
990 REM
1000 POKE 752,C1:A$="Y"
1010 POSITION XY,YY:? "YES";:POSITION XN
,YN:? "NO";
1020 IF STICK(PLAYER-C1)=11 THEN 1000
1030 IF STICK(PLAYER-C1)=7 THEN A$="N":P
OSITION XY,YY:? "YES";:POSITION XN,YN:?
"NO";:GOTO 1020
1040 IF STRIG(PLAYER-C1) THEN 1020
1050 RETURN
```

in line 380. The IF statement is true if STRIG (PLAYER-1) is 1. Remember that this means the button is not pushed. The program loops back to line 180 as long as the button is not pushed.

Lots of lines and REM statements take up memory and slow the program down. Look at the second listing. Here is a short program which uses the menu selection routine. The routine has been compressed into lines 1000 through 1050. The program needs no explanation. Play it and get some feel for the convenience of using the joystick instead of the keyboard.

At the tone the number is. . .

Listing 3 is another joystick input routine. This time we are using the joystick to input a number. It is similar to the first routine. Lines 60 through 140 set the default input number and print it on the

Listing 3. Number Select

```
10 REM      ... NUMBER SELECT ...
20 REM          FROM JOYSTICK
30 REM
40 REM
43 REM TURN OFF CURSOR
45 POKE 752,1
47 REM
50 REM DEFAULT ANSWER:
60 A=10
70 REM
80 REM  DISPLAY NUMBER:
90 REM  X,Y IS POSITION OF NUMBER
105 REM
110 POSITION X,Y
120 PRINT A;" ";
130 FOR SND=0 TO 15
135 SOUND 0,100-A,10,15-SND
140 NEXT SND
150 REM
160 REM SCAN JOYSTICK FOR SUBTRACT
165 REM DON'T GO BELOW LOW LIMIT
170 REM
180 IF STICK(PLAYER-1)<>11 THEN 280
185 IF A=LOW THEN 180
190 A=A-1
240 GOTO 110
250 REM
260 REM SCAN JOYSTICK FOR NO
265 REM DON'T GO ABOVE HIGH LIMIT
270 REM
280 IF STICK(PLAYER-1)<>7 THEN 380
285 IF A=HIGH THEN 180
290 A=A+1
340 GOTO 110
350 REM
360 REM SCAN TRIGGER FOR CHOICE
370 REM
380 IF STRIG(PLAYER-1) THEN 180
390 REM
400 REM WE HAVE ANSWER
410 REM
420 PRINT A
```

screen. Notice that line 120 prints a blank after the number. This prevents garbage from appearing on the screen if 'A' goes from 2 digits to 1 digit.

Look at lines 130 through 140. This generates a muted bell sound, very similar to striking a xylophone. The SOUND statement has four parameters. The first is the sound register. This can be any number from 0 to 3. Up to four sounds can be created at one time. The second parameter is the pitch of the sound. The higher the number, the lower the pitch. Using 100-A gives a pitch that goes up as A gets bigger and goes down as A gets smaller.

The third parameter is the sound quality. A 10 gives a clear tone. The fourth parameter is the loudness of the note. The note goes from 15-0 = 15 or loud to

15-15 = 0 or quiet. This creates the bell effect.

Lines 180 through 240 decrease the input number as long as the joystick is pushed to the left.

Lines 280 through 340 increase the input number as long as the joystick is pushed to the right. Line 380 ends the routine if the button is pushed.

Conclusion

Next time we will compress the number input routine and use it in a program. Try it yourself and let's see who does a better job at compressing it! We'll also try our hand at using the Apple paddles to do a menu select.

Al Baker is Programming Director of The Image Producers, Inc., 615 Academy Dr., Northbrook, IL 60062. ©

TINY PASCAL for the PET and APPLE II

ABACUS SOFTWARE has announced the availability of its TINY PASCAL to run on either the PET or APPLE II personal computers.

TINY PASCAL is a complete package allowing the user to create, compile and execute programs written in the TINY PASCAL language. The TINY PASCAL language is a subset of the full standard PASCAL language but includes the structured programming features: IF-THEN-ELSE, WHILE-DO, CASE-OF-ELSE, FOR-TO/DOWNTO-DO and REPEAT-UNTIL. This block structured language allows for functions and procedures using local or global variables.

TINY PASCAL has three components:

The **LINE EDITOR** allows you to create, modify and save source language statements written in the TINY PASCAL language.

The **COMPILER** translates the source statements into an intermediate P-code that is ready for execution.

The **INTERPRETER** executes the P-code by turning the PET or the APPLE II into a pseudo 16-bit stack computer. Trace and single-step modes are built into the INTERPRETER.

The source statements and the P-code files may be saved on either tape or diskette. The APPLE II version supports diskette only. Includes comprehensive user's manual.

REQUIREMENTS:

PET 16K/32K NEW ROMS Cassette	$40
PET 16K/32K NEW ROMS Diskette	$35
APPLE II 32K ROM Applesoft w/DOS	$35
APPLE II 48K RAM Applesoft w/DOS	$35
TINY PASCAL User's Manual (refundable w/software)	$10
6502 Interpreter Listing	$20

For more information contact:

ABACUS SOFTWARE
P.O. BOX 7211
GRAND RAPIDS, MI 49510 ©

Fun With the 6502

Len Lindsay
1929 Northport Drive #6
Madison, WI 53704

A TALKING PET

Yes, it is true. You now can buy a PET program for a mere $15.95 that will LOAD, RUN and TALK. The only attachment you need is a speaker attached for CB2 sound (two wires connected to pins M & N of the user port- see Compute #1 or Best of the PET Gazette for details on how to hook up the sound). The same speaker you may be accustomed to using for sound effects in PET programs now can be used to let your PET talk.

The program is **TALKING CALCULATOR** from Programma (3400 Wilshire Blvd, Los Angeles, CA 90010). It comes professionally packaged (as are all their programs). It LOADed the first time for me. The fact that the program only will let your PET function as a calculator is completely over shadowed by having it talk to you as you calculate.

Although it has a limited vocabulary, it can talk you through all your calculations. It can pronounce the name of all 10 digits 0 thru 9 as well as the words "plus", "minus", "times", "divided by", "equals", "point", and "clear". A picture of a calculator is displayed on the screen. The program will speak each digit as you enter it, and will update the calculator display as well. Hit the "*5*" key and it says the answer for you, as it prints it on your calculator's display. The problem 6 + 6 has an answer of "one", "two", not "twelve". Each digit is treated individually, and the decimal point is pronounced as "point".

Although you can buy a calculator for less than this program, it is definitely worth having the program for its ability to talk. Kids will love it.

ATARI SOFTWARE

Image Computer Products (615 Academy Drive, Northbrook, IL 60062) has a line of ATARI software. I have four of their cassettes. They are very attractively packaged, and the tapes each LOADed the first time for me.

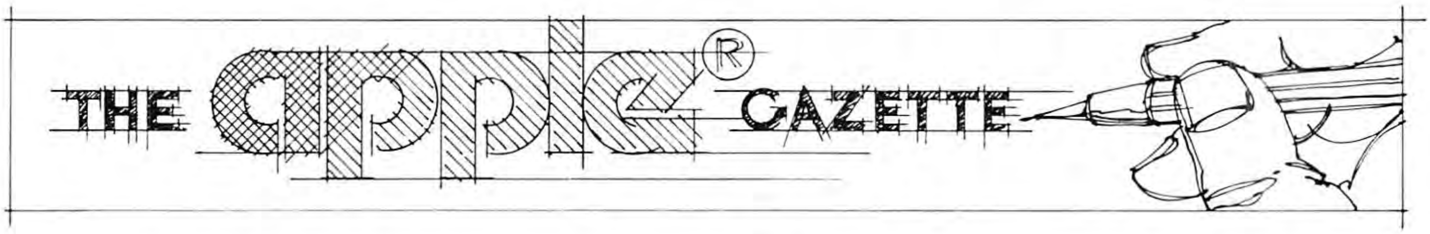
All Star Baseball (#6401) is a very well done, and interesting game. The tape comes with two versions of the game. The first requires only an 8K ATARI and uses the keyboard for user interaction. A 16K version is on the other side of the tape. This version allows the use of joysticks for each player.

The game is for two players. One is the Home team, the other the Visitor. The player at bat, simply hits the SPACE bar to swing at the pitch (or hits the RED button on the joystick). Timing is important. The other player gets to pitch the ball, and can control its speed as well as its course. He can slow the ball down just before it reaches the plate, or add a final "curve" to the pitch.

The game employs beautiful high resolution graphics, and includes nicely drawn fielders and each runner (if any) on base. The game is animated as well. Hit the ball and watch it go into the field (or into the stands for a foul ball). At times the computer will declare the hit an "out" or a hit ("single", "double", or "triple") or it may leave the outcome up to the fielders. The player who pitched the ball, must control the fielder, trying to catch the ball for an out. If he misses, it is an error, and the computer declares the type of hit as a result. A home run goes over the fence (too bad the fielders can't jump that high).

After each inning, a scoreboard is presented for the players review. Hit the space bar or red joystick button to continue. The game is over after nine innings. The game is very realistic, and lets each player in on the action. However, the program has one frustratig aspect. It determines whether a pitch is a "ball" or strike" after the ball is past the plate. Thus it is very easy (and tempting) to pitch the ball far outside (beyond the batters reach) and hook the ball in again after it reaches the plate. This guarantees a strike. It is possible to hit it, but it will always result in a foul ball. The game is alot of fun. If you like ATARI BASKETball, you should enjoy this BASEBALL game. ©

A Column in Transition:
The Consumer Computer column is turning into Fun With the 6502. Len's new column will cover strategic games, contests, and other "leisure-time" software news and activities. Watch Issue 6 for his expanded column. Space didn't permit reviews of other Image software, but Len also looked at Mind Master (#6403), Strategy Pack I (#6404), and Strategy Pack II (#6405). Of these, his only "NOT RECOMMENDED" was Mind Master.
RCL



Dear Apple Readers:

Here's a revised and expanded Apple Gazette. We need your help and input to keep it that way. We're interested in product reviews, short programming hints and application notes, and how-to articles. Help keep the Apple Gazette healthy! Submit your articles to Robert Lock, Publisher/Editor, COMPUTE, P.O. Box 5406, Greensboro, NC, 27403. Please mark the outside of the envelope clearly with "Attn: Apple Gazette'. ©

BOOK REVIEW

Computer Graphics Primer

author: Mitchell Waite publisher: Howard W. Sams book # 21650/$12.95

Review by Eric Rehnke

For a really excellent introduction to computer graphics get this book. In 2 chapters the authore takes you from answering your initial questions on what graphics are and what kinds of things they can do to how they do the things they do and a survey of some of the more popular computers with built-in graphics. Lots of good data here including block diagrams of some graphic displays and information on one particular LSI color video chip, the AMI 68047.

As informative as the first two chapters are, the third chapter alone would justify the cost of the book. Lots of in-depth explanations of the Apple graphics programming interface in particular, and many programming examples. The most important stuff in the whole book was the detailed explanation of how the Apple shape tables are designed and how they can be used to build a graphic character that can be made to grow, shrink and rotate under software control. That alone should keep me busy for awhile.

Other things that were discussed in the book include other types of graphics I/0 devices like digitizer pads, light pens, plotters, joysticks, digital cameras, and image digitizers.

I can't really say enough about Computer Graphics Primer except that it would prove very useful to anyone even remotely interested in computer generated graphics, and INDISPENSABLE to anyone who owns a Apple. ©

NCC '80 And The APPLE III

Joseph H. Budge

By now everyone has heard of the Apple III, but no ones knows much about it. In May I was able to view the Apple III at NCC '80 in Anaheim. Apple computer put on a special demonstration for the International Apple Core which I was able to attend.

According to Apple, the Apple III was designed specifically for high capability use by professional and managerial people. It's features include a new Apple-designed central processor, up to 128K bytes of main memory, a self-contained floppy disk drive, and a new keyboard design with 10-key pad and programmable keys. Video allows for 80-column by 24-row text in black and white, or 40 x 24 in 16 colors as well as high resolution graphics. The III also has four analog to digital (A/D) converters, one digital to analog (D/A) converter, and a programmable character generator. Software available for the III will include Apple Business Basic, Fortran, Pascal, and Pilot. Apple Business Basic will include PRINT USING with string justification and 18-digit mathematics. Pilot is a teaching language with enhanced graphics for the III.

As a professional tool the Apple III will be promoted as part of application packages. At least three application packages are pending. The first, to be available in July, will be the Information Analyst package. It will contain an Apple III with 96K bytes of RAM and a 12-inch black and white monitor. Software will include Apple Business Basic, a mail list manager, and Visicalc III. Visicalc III is Personal Software's Visicalc with new features and functions added. Visicalc III has been additionally enhanced to utilize the Apple III's larger display and memory. Base price for the information analyst package will be $4,400.

The second application package, available in September, will be the Apple III word processor. In addition to the hardware of the Information Analyst, this package will include the buyer's choice of two printers. Either a Silentype thermal printer or a Qume-type letter quality printer will be available. The Silentype plugs directly into the back of the III, while the letter quality printer will include an interface card. Software for the package will feature a word processor and word processor training course. Base price will be $5,400 with the Silentype or $7,800 with the letter quality printer.

The third application package will be a software development package. This will come with a 128K Apple III, a 12-inch black and white monitor, and a Silentype printer. All Apple III languages will probably be included. No price or release date has been announced, but look for it in 6-9 months.

All options from each application package will be available separately. Extra disk drives and memory will probably be in most demand. The memory upgrade, from 96K to 128K, will be priced at $500. The Apple III will be available by itself, of course, although Apple has not announced the price or anticipated availability date for the hardware alone.

The III will, indeed, be a great business computer. But it's also the ultimate hobby computer. Listen to Wendell Sanders, the III's designer: "What do you expect? They gave me a blank check and told me to go build a computer. It's the ultimate homebrew!"

To understand the III, you really need to know something about it's processor. The processor is built around a 6502A running at 2MHz. The fact that there's a 6502 chip in there is a bit misdirecting. A large number of support chips are hooked onto the 6502 which lift it out of the microprocessor class and into the miniprocessor class. Output from the 6502 goes to a PIA and a ROM. Outputs from those two devices feed back to each other and to the 6502. About a dozen other logic chips are thrown in for good measure, just to confuse the issue. Apparently the ROM is the key to the whole system. It doesn't act like memory, as in the Apple II. Rather, it's a giant logic gate with many thousand possible states.

This arrangement allows for a whole host of new processor instructions and actions not possible with the regular 6502. To begin with, page zero can be located virtually anywhere in memory. An interesting possibility which results in putting banks of page zero's holding data throughout large programs. Program speed should be greatly enhanced. Of course if page zero is to be portable, the stack must be portable as well. Can you imagine what kind of Byzantine programs you could write jumping from stack to stack? Interrupts are handled differently in the III's processor as well. An added command is the equivalent of interrupt - stand-by. When this mode is enabled the processor recognizes interrupts but takes no action. Instead it flags the fact that an interrupt must be dealt with later. This allows the III to still do most everything in software but support interrupts. It's now possible, for example, to simultaneously enter text to the keyboard while disk access is going on. The processor also has more direct control over I/0 devices than has previously been possible. Scrolling is possible with machine commands. Other commands involve enabling and disabling the programmable character generator and keyboards. Finally, the most obvious benefit of the processor's enhanced logic is its ability to address 128K of memory. That's the official figure at least. Reliable sources indicate that, with some minor modification, the III is capable of handling 384K of memory in 64K chips. Whew!

Graphics freaks will love the new graphics capabilities in the III. At maximum resolution the screen has a 560 x 180 dot grid. This is exactly twice the horizontal resolution of the Apple II. Unfortunately the high resolution is only available in black and white (same for the 80 - column text). But lo-res will blow your mind! Lo-res on the III is the same resolution as hi-res on the II. But unlike hi-res on the II, the III's graphics allow for a full 16 colors, any of which can be mixed at will. This contrasts to some of the multi-color machines on the market now which only allow four colors to be on the screen at a time. And the 16 colors are available for either graphics or 40 x 24 text. Speaking of text, a real show-stopper was the programmable character generator. As you might expect, this allows the user to set up his own character set. That's lots of fun if you're into Gothic or Chinese (For some reason they didn't have a Pet graphics demo running. . .). The character generator is even more fun if you like animation. By changing the definition of characters you can easily change a character's shape. If you do this to adjacent characters in the right order you can make much larger objects move. Andy Hertzfeld showed a couple of outstanding demos of birds flying and horses running. Just like the movies. Sound from the III is literally in the 6th dimension. The speaker is run by a 6-bit d/a converter which allows true waveform generation. Unfortunately the speaker is the same crummy one found in the II. Apple ran a wire to a jack in the back so you can easily plug in an amplifier and good speaker. Apple expects to market a 12-bit A/D converter as a peripheral card. While we were discussing the sound, one of Apple's software experts noticed that the six-bit d/a converter nicely coincided with the six-bit data storage on the disk. Look for some interesting sound disks to come along soon.

The disk drive itself is pretty much the same as the Disk II we're already used to. To please the FCC's radio emission standards a change has been made that squelches direct compatability with Disk II drives. To keep radio emission down you can't have a million cables running out the back of the computer. So the disk interface was changed such that disks daisy-chain. That way only one cable comes out of the computer, for the second drive. The third plugs into the back of the second, and so on down the line. After hearing this several spectators suggested that the first non-Apple product for the III will be conversion kits for Disk II's. Data on diskettes will be compatible between the language system II and the III.

Apple received some criticism for introducing the III with only 143K byte drives. They apparently feel that this won't be a problem in early applications. Even more tantalizing is their elaboration of this point: "We will also expand our line of mass storage peripherals in the future, since some applications need several megabyte of memory."

No system would be complete without some software to make it work. This is especially important in the Apple III, where all kinds of strange tricks are going on inside the processor. In anticipation of this problem Apple provides the Sophisticated Operating System (SOS) for the III. SOS manages memory, peripherals, the keyboard and screen, graphics drawing, and interrupt handling. Most utility functions are built into it. Being configurable, SOS only pulls in from disk those parts of itself needed for the tasks at hand. With most programs SOS will take up about 14K of memory. One of its main functions is to efficiently manage the memory that's left. SOS automatically finds and allocates free memory for program use. If a memory sensitive function, such as graphics, is switched on, the SOS will reallocate memory around the graphics page. SOS's other main function is to handle I/O. It keeps track of what peripheral card is in what slot and has built in programs to control the cards. Peripherals can be interrupt driven for maximum efficency, SOS also has built in routines for graphics and sound. SOS thus provides a foundation for higher level languages. The user's application software can be truthfully independent of the hardware.

Most Apple II owners will be interested in compatability between their computers and the III. For this reason the III comes with an Apple II emulator mode. The emulator mode copies the II as faithfully as possible. It allows you 48K of RAM with either Applesoft or Integer Basic available in "ROM". Graphics and text appear as on the II; 6 color hi-res, lores, and uppercase only text. The microprocessor clock even slows down to 1MHz. Put in the emulator disk and it effectively performs a frontal lobotomy on the III.

Naturally there are some differences between the emulator and the real thing due to hardware differences. There is no cassette recorder port, for instance. Game paddles will need adapter sockets, and there are no annunciators. There are no slots 0, 5, 6, or 7. The emulator supports two disk drives which appear from the software's viewpoint to be in slot 6,1 and 6,2. The physical dimensions of peripheral cards in the III are different: higher and shorter. Your Mountain Hardware clock and Sup'R'Term won't fit in the III - but you won't need them anyway. Most other peripheral cards for the II should work in the III since most of the buss lines are identical. The III uses the no-connects, DMA, and User1 lines for its own purposes.

Apple has made a full-scale commitment to continued support of the Apple II. Taylor Pohlman, the product manager, told us that Apple intends to sell as many Apple II's in the next 12 months as have already been sold to date. Two of the new languages, Fortran and Pilot, will be available for the II before they are available for the III.

And Apple intends to release sometime this summer a Pascal interpreter that will run in a 48K Apple II with no Language System. Thus all Apple owners will be able to run compiled P-code. Apple sees the II as a beginner's system and as a basic workhorse for small applications.

Apple's introduction of their own new computer was done in the Grand Style. NCC is a very political convention, where pecking orders and pull count heavily for floor space and location. Both are important, since in excess of 80,000 people attend the convention. As you might expect, NCC wanted to put Apple in the personal computing section. Apple, on the other hand, wanted to insist that the III was a small business comupter. The haggling went on long enough that Apple finally wound up as neither; they were put in the "overflow" section in the basement garage of the Disneyland Hotel. The garage was over a mile from the main convention floors. Apple wasn't even listed in the convention's guide to exhibitors. Nevertheless the Apple booth was the second - most popular booth at the show; only IBM drew a bigger crowd. At Apple's booth one had to stand in line 20 minutes just to get near a machine! By the show's third day officials had grown so tired of "Where's Apple?" questions that 10 signs were up on the main convention floor giving instructions on how to get over to Apple. So much for politics.

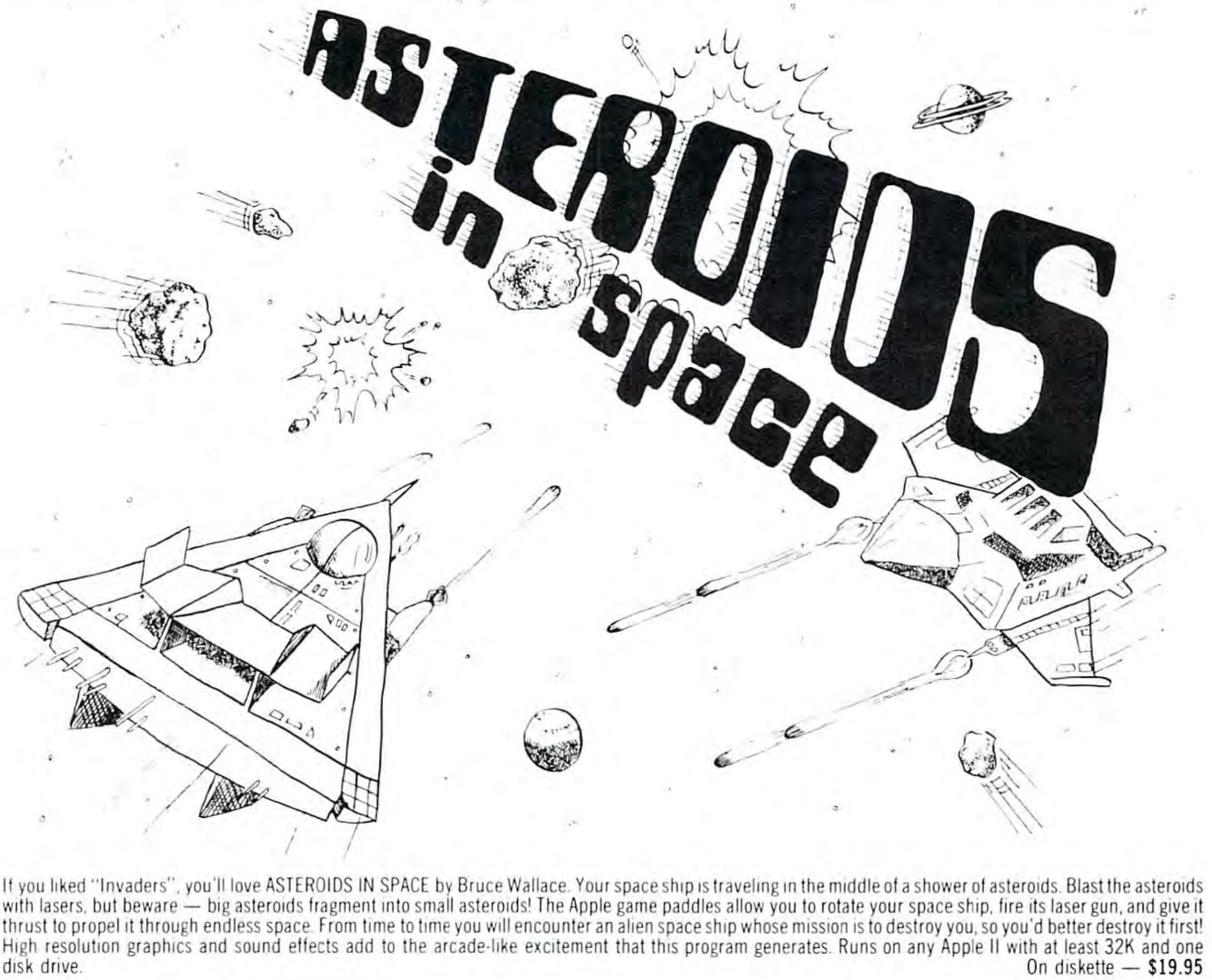
Several interesting products were introduced for the Apple II by other manufacturers at NCC '80. Mountain Hardware announced their new music board. Actually, it's two boards and a light pen. The boards allow 16 voice music synthesis, with each voice having a fully programmable waveform and envelope. Software included allows input of sheet music in standard notation using the Apple's graphics screen and the light pen. Output is in stereo, naturally, and sounds excellent. The complete system will be priced at $545. Mountain Hardware also had a bit of information on their Apple buss expansion module. Contrary to ads you may have read, the expansion box won't be ready for at least 4 months. It will have its own power supply, a switching supply slightly greater than Apple's (my sources indicate that this supply is manufactured in the Far East and is difficult to get in large quantities). The box will have 9 slots, one to replace the slot which the box plugs into and eight more (0'-7'). It employs the User1 line to disable the boards on the Apple itself when appropriate. PR#14!

Programma showed the prototype of their dual sided 8" disk controller. It's a neat board, absolutely crammed with chips. To overcome power supply limitations, the controller card gets all its power from the disk drives. That should be ready in about 6 months. Programma also showed their new text editor, PIE 2.0. It does almost everything you can imagine (I didn't find out if it works with LISA). One of it's nice features is footnoting. Unfortunately, for some reason it's still a line-oriented editor. Oh well, it's probably the best line-oriented editor you'll ever see.

Stoneware Computer Products was demonstrating the prototype of their Data Base Management System. It is indexed so that any data can be found in 6 seconds. Each entry can be up to 1K long, and have up to 100 fields. Data types can be defined, and the user has complete control over report generation. The system has been elegantly human engineered to be useable by the non-programmer. A first version will be released this summer which will support one disk full of data. The second version, which will soon follow, will support up to 25 megabytes and have many other features.

Epson America introduced a new printer for the hobbyist and small business market. Called the TX-80, it's an 80-column dot-matrix impact printer with either tractor or friction feeds. The printer comes with a Centronics - type parallel port and options for RS-232 serial or IEEE 488 interfaces. It prints at 125cps. Epson designed the printer with high reliability and ease of use in mind. The ribbon comes on standard typewriter spools, for instance. Of the thousands of TX-80's sold in Europe, less than 1% needed service of any kind. Of course the added quality costs: the printer is expected to sell in the $750 range when marketing begins in this country. ©

Exciting, entertaining software for the Apple II and Apple II Plus*



If you liked "Invaders", you'll love ASTEROIDS IN SPACE by Bruce Wallace. Your space ship is traveling in the middle of a shower of asteroids. Blast the asteroids with lasers, but beware — big asteroids fragment into small asteroids! The Apple game paddles allow you to rotate your space ship, fire its laser gun, and give it thrust to propel it through endless space. From time to time you will encounter an alien space ship whose mission is to destroy you, so you'd better destroy it first! High resolution graphics and sound effects add to the arcade-like excitement that this program generates. Runs on any Apple II with at least 32K and one disk drive. On 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 like the Ten Foot Spider and the Headless Horseman. 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. 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. Requires at least 32K of RAM. Cassette: **$14.95** Diskette: **$24.95**

FASTGAMMON™ by Bob Christiansen. Sound, hi res color, and cartoons have helped maked this the most popular backgammon-playing game for the Apple II. But don't let these entertaining features fool you — FASTGAMMON plays serious backgammon. Requires at least 24K of RAM. Cassette: **$19.95** Diskette: **$24.95**

QUALITY SOFTWARE

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

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. MasterCharge and Visa cardholders may place orders by telephone and we will deduct $1 from orders over $19 to compensate for phone charges. Or mail your order to the address above. 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 — payable in U.S. currency.

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

APPLE WRITER™

A Review Of The Text Editing System From Apple Computer, Inc.

David D. Thornburg
Innovision
P.O. Box 1317
Los Altos, CA 94022

The use of personal computers in small businesses is clearly bringing tremendous sophistication to tasks which a few short years ago were being carried out manually. To take just one example from the data processing (DP) world, the Apple II is being used along with Personal Software's Visicalc program for numerous data processing applications in thousands of businesses.

In addition to data processing, all businesses (and homes for that matter) generate documents which have to be sent to others in the form of letters, reports, or memoranda. The creation of these documents is aided by "Word Processing" (WP) software. It is reasonable to expect the computer that handles the data processing (DP) tasks to handle the WP tasks as well, since the primary ingredient for doing the job is software. There are some hardware constraints, as we will see, which can make a good DP computer less than perfect for WP applications, but it is still the software which carries 99% of the load.

Since Innovision uses a 48K Apple II for its business applications, I was only too happy to get my hands on the Apple text editing system, "Apple Writer", when it was released several months ago. This text editing system comes with two disks (one for back-up) and a manual which continues the fine tradition of Apple in that the documentation is superb. Since this package was priced inexpensively at $75, I did not expect to have all the features one expects from a $16,000 dedicated word processor. What I did find was a compact text editing system with a lot of the essential features needed for the rapid preparation of letters and small (under 100 page) reports.

HARDWARE CONSTRAINTS PROVE CHALLENGING. . .

If you are an Apple owner, you may have run up against two Apple characteristics which might make you skeptical of this machine's ability to be used for document creation. These characteristics are the lack of lower case character entry from the keyboard, and the inability of an unmodified Apple II to display lower case characters on the screen. These constraints, coupled with the limitations of a forty character by twenty-five line display, might make one shudder to think of doing text editing on such a machine. My experience suggests that things aren't as bad as they might first appear - and I use this system for text editing every day of the week.

Apple uses its own convention for character presentation on the screen. Upper case letters are shown in reverse field (black on white) and lower case letters are shown as normal (white on black) upper case letters..The keyboard is handled in the following manner. All alphabet keys are accepted as lower case letters. Upper case letters are obtained by preceding a letter with the ESC key. Since the ESC key is located in the same column as the SHIFT key, this doesn't cause too much trouble. Normally shifted characters (the punctuation marks above the numerals, for example) are obtained with the SHIFT key. As characters are entered into the computer, they are displayed on the screen just as they are typed in. This means that words will be broken at line boundaries (40th character position). Thus making it hard to do extensive proof reading from the screen. Since the goal is to produce printed documents, this is not a terrible inconvenience, as I am willing to perform proofreading tasks on the printed output.

Please note that while I find the constraints imposed by the Apple hardware tolerable for this application, I am not suggesting that they could not be improved. In fact, I think that all personal computers should have full upper and lower case keyboards and displays. What I am saying is that, given the constraints imposed by the hardware, the Apple Writer word processing system is easy to learn and use.

BRINGING THE SYSTEM UP . . .

If you have Applesoft and an auto-start ROM, the Apple Writer is virtually crash-proof. Accidental pressing of the RESET key, for example, does nothing more serious than moving you to the menu. Note, however, that this is not true unless you have the auto-start ROM and Applesoft BASIC, and that it is a bad practice to return to the menu this way. If the system is brought up with the disk in place, the user is automatically presented with a menu of commands. The commands allow one to create a new document, bring in an existing document from the disk, save a document on the disk, and bring up the printer program (which will be discussed later). The "Edit" command clears the menu window and places the document to be edited on the screen.

Since it is important to be able to move the cursor around on the screen to edit various portions of a document as well as to bring new portions of the document into view, text editing systems need some method for cursor control. Apple Writer lets the user move between text entry and cursor control modes by use of the ESC key. If the ESC key is pressed once, the cursor becomes a carat ( ). This symbol indicates that the next typed letter will be printed in upper case. If the ESC key is pressed twice in succession, the cursor becomes a plus sign (+) which indicates that the system is now in the cursor control mode. The cursor can be moved anywhere on the screen using the I, J, K, and M keys to move the cursor up, left, right, or down respectively. Once the cursor is in the desired place, the text entry mode can be entered by pressing any of the other alphanumeric keys. A little practice makes this seem quite natural.

Large scale cursor motion is available from several control functions. CTRL B, for example, moves the cursor to the beginning of the document.

The deletion of unwanted text can be carried out on a character-by-character basis, as well as by word, or by paragraph. These capabilities are easy to master. As characters are deleted they are stored in a 255 character buffer so that they can be re-entered later if desired. The easy way to move a word, for example, is to delete it from its old position with the back arrow key (<-) and to then move the cursor to the location at which the word is to be entered. Once this has been done, the word may be entered with the forward arrow (->) key.

More sophisticated editing tasks, such as search and replace, file insertion, and others, are also implemented in Apple Writer.

In addition to the documentation provided in the 70 page manual accompanying the software, Apple also provides an on-line tutorial to bring you up to speed with "hands-on" practice. All in all, this is a fine text editor for simple document creation.

Since the keystrokes are only saved on the disk as a result of your formal request to that effect, I find it useful to save my documents every fifteen minutes or so, just in case the power goes out. The user can find the amount of space remaining in free memory by typing CTRL F. With a 48 K Apple, one should be able to create about sixteen pages of text in one file.

GETTING IT PRINTED . . .

Sooner or later, you will want to take the jumble of broken words which appear on the screen and turn them into a nicely formatted document. During the document creation process you can enter non-printing commands (which do appear on the display) to let the printer do some fancy stuff like centering, fill justification, etc. These commands occupy one line each and are preceded by an exclamation point. For example,

!cj

is a command to tell the printer to center all the text which follows. Since each command stays active until superceded by another one, it is easy to enter text in the midst of a long document without worrying about how it will be formatted.

Once the document is to be printed, one enters the print mode from the editor menu. At this point, one gets a print menu which includes the functions Print and Continue. If the Print function is selected, a list of default printer settings is displayed on the screen (left and right margins, lines per page, interline spacing, etc.). The user has the option of changing any of these parameters, in which case the parameter set is stored on the disk and becomes the new default. This feature is quite valuable, since there are likely to be few changes in overall document format once your printer is set up the way you like it.

Once the printer parameters are set, the user is given the option of entering a page heading which appears on each page of the document. Finally, the printer springs to life and the document gets put on paper for all to see.

Large documents which require more than one file to hold them can be printed with no trouble at all by using the Continue command from the Print program menu. A file can end in mid sentence, in which case the remainder of the text can be loaded into the computer and the printing resumed with no problem whatsoever. The only problem with printing large documents is that the process requires lots of operator interaction. This becomes especially noticeable if several copies of a multiple file document are being printed.

We have used Apple Writer with a Qume daisy wheel printer and with the Comprint electrosensitive printer connected to the Apple parallel interface card. In both cases the documents turn out beautifully, although I think there is a small bug in Apple Writer which makes it a little tricky to use the Comprint printer in the single page mode.

As you may know, the Comprint printer uses roll paper. If you want to print documents one page at a time (so that you can easily tear them off at 11 inch lengths), you must use the "single page" mode during printing. When this is selected, there is an extra line feed command sent just prior to printing the first page. Unless the user anticipates this and puts the printer in the "no-print" mode just prior to printing the first page, the resulting document will be improperly spaced.

Given that this product has only recently been placed on the market, I am happy that this is the only "bug" I have uncovered to date.

AND IN CLOSING

I have looked at other text editors for the Apple, some of which were overloaded with features. Given the hardware limitations of the Apple II, I feel that Apple Writer is a very useful document creation tool.

After all, how do you think this manuscript was created? ©

Apple II Rom Card Documentation

Jeff Schmoyer
Andromeda
Computer
Systems

The only schematic that has not been made available by Apple (that I know of) is for their ROM card. The reason for the presentation at this time is not out of looking for something to do, but to obtain the solution to my blowing out the card for the third time. In each case the computer acted the same, coming up with the screen full of junk in the Lores graphics mode with RESET having no effect. Each time removing the ROM card alleviated the problem.

Also in every case the perpetrator was found to be the 74LS09 that provides the inhibiting of the main board ROMs and the output enable for the data buffer on the card. Apparently this chip inhibits the main board ROMs while not enabling its own ROMs. The processor can't find anything to execute so it hangs up. Replacement of this chip restored the system to its normal operation every time.

While we're here I might as well explain how the board works and about the two options present on it.

The 74LS74 flip-flop on the card determines whether the on-board or off-board ROMs will be used. When RESET comes through the system, the flip-flop is either preset or cleared depending on the position of the switch on the back. The other way of activating the flip-flop is through the use of Device Select. The low order address bit, A0, determines whether to set or reset it in conjuction with a C08X access.

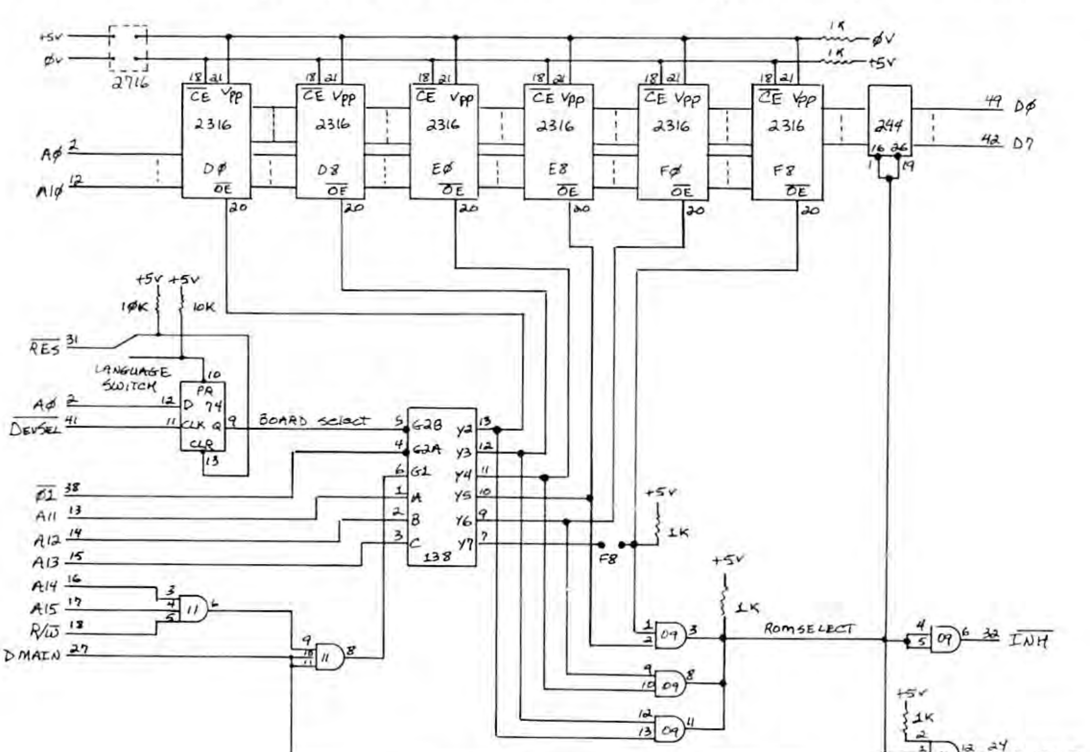
The 74LS138 IC decodes the high order address lines, with enables also coming from 01 and R/W, to provide output enable to the proper ROM. If any of the ROMs are enabled, the 74LS09 activates the INH line to inhibit the main board ROMs, and it enables the 74LS244 output data buffers.

Another service the card performs is the daisy chaining of the DMA line through it.

On to the option jumpers. One of the jumpers on the board permits the use of a ROM in the F8 position. This jumper is labeled F8 and is located near the 74LS138. If a blob of solder is melted across it, this position will be used whenever the card is selected. This could contain an Autostart ROM if a standard monitor is on the main board, or it could contain the old monitor for an Apple II Plus.

The other two jumpers, which are located in the upper right corner of the card, select whether you want to use 2716 (programmable ROMs) or the usual 2316 (Apple ROMs) chips on the card. If both of them are jumpered, 2716's may be used. Both kinds of chips can not be used at the same time. Please notice that on the schematic, 2716 notation is used for the ROMs.

One final note. On at least one of the occasions the 74LS09 blew out, nothing was done to the computer other than picking the whole thing up and moving it as a unit. So don't assume someone pulled out a board with the power on. (I did that the last time!)



Review: DDT (DISC DRIVE TIMER)

Morton Technologies, Inc. $19.95 P.O. Box 11129 Santa Rosa, CA 95406

Reviewed By:
Terry N. Taylor, Apple Pi, Denver, CO

Most of us have seen Disco-Tech's ads on their drive timer for the Apple, and most of us have asked why we should purchase this product when a perfectly good public-domain program called DSPEED is already available. The answer is simple: The program is good and the documentation is excellent-- a 24-page booklet that covers the following topics:

1. Introduction
2. Running the Apple DDT Program
3. Adjusting Disc Drive Motor Speed
4. Care of Magnetic Media

This documentation alone justifies the price of $19.95 for the package. There have been several articles in the Apple user group newsletters on how to adjust your drive's speed with only two screwdrivers and DSPEED, but they are not generally available. In addition, this booklet does a far better job than do the articles.

OK, so the documentation is better; how about the program? It's better too. I have lost count of the number of times that I have made additional copies of DSPEED for those who have forgotten to put a blank diskette in the disc drive and thus ruined the program. You don't need to worry about changing diskettes with DDT. This program offers two major options. First, it allows the user to analyze the motor speed of the disc drive in question. It displays both the motor speed in revolutions per minute and the percentage of error (from 300 rpm, which they say is the correct speed) for the disc drive. DDT averages the disc speed over 40 resolutions, taking about 12 seconds to present the analysis.

The second major option updates the motor speed about once per second and graphically displays the results. Since the program asks you for an rpm range, you are able to adjust the disc motor speed very accurately, from a large-scale spread down to a very small dispersion, which watching your video screen.

To adjust my three disc drives, with either program, took less than 20 minutes using only two screwdrivers (one of which should be a Phillips), so this is not a complicated procedure; and, better yet, does not void your warranty if you follow the very good directions in the DDT booklet. I found only one minor fault with the instructions: if you have an early disc drive (for example, one with a serial number of 1800 as contrasted to one with a serial number of 6000-7000), then the trimpot screw does not face to the side, but faces down. You have to remove four more Phillips screws in order to reach the adjustment screw. Then you must place the disc drive so that the trimpot screw overhangs the edge of the table. Now you can adjust the disc drive's speed while the drive is in a normal position.

Since I was curious not only as to how this program compared to DSPEED but also to the disc speed adjustment program on Apple's diagnostic test disc, I went to a local computer store where we compared all three programs on two different disc drives. The results are shown below. While each one is different, they all result in the same conclusion: Each drive is slightly slow; but, since they are well within tolerances, the two drives did not need adjusting.

Program	Disc Drive 1	Disc Drive 2
Apple's	0 to -3	0 to -2
DSPEED	-1 to -4	0 to -2
DDT	(299.1)	(299.5)

Why should you adjust your drives? If you only have one disc drive, it's not too important to adjust your disc speed unless it's way out of whack. It is important when you are using two disc drives to copy programs from one to the other. If you are using the fast copy method (about 28 seconds) with two disc controller cards, then it is critical that the two drives be closely aligned. If you want all of the data to be properly transferred, the drives should be within one percent of each other, which is less than three revolutions per minute difference. Disco-Tech states that an incorrect motor speed can cause data to be lost when the program is either loaded or saved to disc. Unfortunately, the disc motor is easily jarred out of alignment, so this program should be run every time you transport your disc drives. Otherwise, a monthly check is probably sufficient.

When you order DDT, be sure to specify either Apple or TRS-80 as the price is the same for either ($19.95). The TRS-80 version is also available on cassette for $14.95; the Apple version only comes on diskette. Disco-Tech is a division of:

Morton Technologies, Inc.
P.O. Box 11129
Santa Rosa, CA 95406

If you are a resident of California, you must include 6% sales tax. Happy computing. ©

Assembly Language Programming with UCSD PASCAL

J. M. Moshell

This article is primarily about using the APPLE II version of UCSD PASCAL to write 6502 assembly-language programs, and use them with PASCAL programs. Much of what is described can, however, be used with other UCSD PASCAL microcomputer systems. We include as an example a program to make low-resolution 16-color APPLE graphics available to PASCAL.

General Issues about assembler/PASCAL relations.

The UCSD system uses a very sophisticated " psuedo-machine" (P-machine) or interpreter program, which looks to the user like a sixteen-bit micro-computer with a "pure stack" architecture (no data registers); all operands are on the top of a stack, and the result of an operation is put back there. The UCSD system not only places data on the stack, it actually places executable code on the stack when it is brought in from disk. These chunks of code may either be P-machine code or 6502 code. Their interrelation is quite complex, and the APPLE PASCAL manual (essentially the UCSD PASCAL manual, customized for APPLE) does a reasonable job of explaining the operation of the system, assuming you have a graduate education in computer science. APPLE promises a new book in April, 1980. Here we're only interested in making you a fluent assembly-language user, so we'll skip most of the unnecessary details.

Basically, you may write any number of assembly-language subroutines, and link them together with a PASCAL "host program". Sometimes the host is trivial, doing nothing but calling your assembly-language program. This linkage is done by the LINKER program, and is not complicated to perform, if you have built the proper buttons and hooks into the host and the subroutines.

The examples in the APPLE book are simple and accurate. The .PROC pseudo-operator in your assembly program declares a name that can be used by the LINKER to make a connection with a procedure name that is declared EXTERNAL in the host program. Rather than repeating the APPLE manual we will concentrate on areas they don't stress.

Labels in an assembly program can receive their values in two ways: either by the EQUate pseudo-op, or by being attached to a line of code. In simple-minded absolute assemblers like that of CP/M, you can form complex expressions based on the assignment of value to a label; but the UCSD PASCAL assembler requires care in the construction of expressions. Any label, whose value is determined by being attached to a line of code, is a *relocatable* value. Its actual value will be assigned at run-time. Thus it cannot be used in forming any arithmetic expressions that involve more than adding a constant value to the relocatable value (e.g. LDA LABEL1 + 1 is o.k., and will load accumulator with the contents of the byte just after LABEL1). However, multiplying or dividing a relocatable value is illegal, since the run-time system hasn't got mechanisms for evaluating such expressions. For instance, in absolute assemblers, we would use something like:

```
LDA #TABLE/256
STA POINTER + 1
```

to pick up the high byte of TABLE's address, and

```
LDA #TABLE MOD 256
STA POINTER
```

to get its low byte.

With UCSD PASCAL, we have to use the following slightly more cumbersome technique:

```
LDA TABADDR + 1; high-byte
STA POINTER + 1
LDA TABADDR ; low-byte
STA POINTER
. . . .
TABADDR: .WORD TABLE; store (relocatable) address of table here.
TABLE: .BYTE 0, 2, 4, 15, 40
```

Now, the relocating loader will store the value of label TABLE at location TABADDR. You can then pick it to pieces by bytes, when your program runs.

The second way in which labels receive values is via the EQUate pseudo-op. The expression to the right of the EQU can contain absolute values freely intermixed with arithmetic and logical operators, but can only contain one relocatable expression to be added to the result of the absolute arithmetic. Thus, two legal EQUates are:

```
LABEL1 .EQU 80 ;reference to page zero, location 80
LABEL 2 .EQU L5 + LABEL1/4
. . . . .
L5: .BYTE 44 ;L5 evaluated by its position in the code file.
```

Another common source of confusion is that the UCSD Assembler strictly segregates code-generating operations (.PROC and .FUNC) from non-code-generating directives such as .EQU. Thus you cannot put an .EQU at any convenient place within a code-file; they must all occur before the *first* .PROC in your source file. The "high-level syntax" diagram in the assembler manual is worth some study.

"Local labels" such as $1 in this $1;

```
INC COUNTER
BNE $1
```

are useful but tricky. They are useful because they don't "last"; they are unknown except within a range of code delimited by two "real" labels; thus you can use $1 as a local label for short jumps anywhere within your program as long as another $1 isn't "within sight". The tricky part is that sometimes you will interpose, by accident, a real label during code modification, creating something like

```
$1 INC COUNTER
JUMPIN: DEC RECORD
        BNE $1
        . . .
NEXTLBL: etc
```

The assembler error message will occur at NEXTLBL, even though the error was the insertion of JUMPIN. The problem is that the one-pass assembler keeps on looking (within the region between JUMPIN and NEXTLBL) for a local $1 to jump to; it's been "cut off" from the one above JUMPIN. When NEXTLBL ends the region, an error results, far from its cause. Beware; always check that the jump and the local label can "see each other".

Communication with the PASCAL host program is well-described by the APPLE manual. Communication with other assembly routines is also easy, but there are a few twists. The pseudo-ops. .PROC and .DEF make any label to which they are attached, at the head of a routine, known to everyone else participating in linkage. Thus you can, in a separate source-file, use the .REF pseudo-op to make a label meaningful to *this* procedure, either for data access or for jumping to. For instance:

In one source-file:

```
    .DEF DATATABLE, FIXUP
    .PROC PART1
    . . .
DATATABLE: .BYTE 11, 40, 80, 0B0
    . . .
FIXUP: LDA ETC ;subroutine to be called from
               both within
               ;PART1 and from PART2.
```

In another source file:

```
    .REF DATATABLE, FIXUP
    .PROC PART 2
    . . . . .
    LDA DATATABLE + 1
    JSR FIXUP
    . . . .
```

You can put several .PROCs in one source file; this is convenient since that file becomes a kind of "library", and may be the only thing you have to link to your host. However, you *cannot* use REF and DEF between .PROCS in the same source file! In a strange way, a data-table in another source file is more accessible to some routine than one in its own source file.

Some Semi-Legal Communications Windows between PASCAL and Assembly Language

The "official" ways of communicating between assembly programs and PASCAL hosts are: procedure parameters and function values, and .PUBLIC and .PRIVATE entries in the global symbol table (that is, COMMON, to you FORTRAN types). These are well documented in the APPLE manual.

However, several "unofficial" ways exist for getting from one realm to the other. They rely on a sneaky trick which allows us to violate the type-rules of PASCAL; that is, the rules which separate integers from characters, etc.

We will use **pointer variables** in this discussion. Briefly, a pointer variable is a machine-memory address. In PASCAL they are widely used for linked-list management, and for many other things. If a pointer varible P is to be used as a "pointer to integers", then it is declared by

```
VAR P:^INTEGER;
```

and the expression

```
P^
```

refers to whatever location the pointer variable P is currently pointing at, viewed as a 16-bit integer. Thus, **Y: =^p** would pick up the integer value to which P points and put it into variable Y.

We could thus use pointers to make PEEKS and POKES if we just had a way of assigning an integer value to the pointer itself (the address we want to look at). However, PASCAL doesn't want to let us do that, because **pointer** is a different type than **integer.**

However, there exists a special declaration, which allows *two different* identifiers to refer to the *same* memory location. Thus, we will "trick" PASCAL into giving that location both a pointer-name and an integer name. We can then assign values to the integer name, and get at "what it points to". The code:

```
VAR X : RECORD
        CASE BOOLEAN OF
        TRUE: INT: INTEGER;
        FALSE: PTR:^INTEGER
        END;
```

I won't explain why this particular notation was used; but it works. Thus we can do a PEEK at location 1000, say, by making the assignments:

```
X.INT: = 1000;
Y: X.PTR^
```

Now Y contains the value that is in (decimal) locations 1000 and 1001 (low and high order bytes of a 16-bit word). We illustrate the trick with a PASCAL Function PEEK16:

```
FUNCTION PEEK16 (LOC:INTEGER):
INTEGER;
VAR X: RECORD
        CASE BOOLEAN OF
        TRUE: INT: INTEGER;
        FALSE: PTR: INTEGER
        END;
BEGIN
  X.INT: = LOC;
  PEEK: = X.PTR^
END;
```

A 16-bit POKE, and byte-sized PEEKS and POKES are supplied at the end of the article.

Now, for those of you who know something of pointer variables and dynamic storage allocation, you can use a POKE like the above, to store a pointer-value in some memory location (e.g. in page 0) then access it from assembly language to refer to the structure in dynamic storage to which the pointer points. (For those of you who don't use pointer variables, don't worry about it.)

Some specifically APPLE tricks for coding and debugging assembly language.

All the foregoing has been applicable to any UCSD PASCAL system. Let's look at some specific APPLE stuff.

First, the pseudo-machine interpreter in the APPLE is located at the high end of memory, in the RAM on the language-card. The pseudo-machine uses page zero memory locations 36 to A1 (hex). Locations 0-35 are temporaries; they are used by the P-machine and Turtlegraphics, but you may trash them. Locations A2-EF are "officially" reserved for the P-machine, but unused. I would recommend using them from the top down, since minor mods to the P-machine may come out later that use more space upwards from A2. Thus, data which you want to survive from one call of your assembly program to another can reside in the area A2-EF.

Second, PASCAL's use of memory (specifically, the dynamic heap) starts slightly above location 1000 (hex). To "free up" the memory region 2000-4000 for use as a high-resolution video buffer, one can use the following trick (with our "two-faced pointer", X, above).

```
X.INT : = 16384 ; (* = 4000 hex *)
RELEASE(X.PTR); (*Moves Heap to 4000*)
```

This should be done "first thing" in the main body of your PASCAL host program, if you're going to do high-resolution graphics. Of course, you can also do that by calling INITTURTLE; this is exactly what they do.

One of the most difficult things about programming in any computer's assembly language is debugging. With DDT in CP/M, or equivalent systems, you can "trace" a program, operation by operation, and see where it goes wrong (assuming you understand your own code.) In fact the APPLE ROM MONITOR used to contain trace and breakpoint facilities; but they were removed when the auto-start feature was added; so your APPLE with the Pascal Language card won't help much for debugging.

However, it *is* possible to "breakout" of an assembly program, use the Monitor to examine memory locations, and then resume operation of the program, even returning to the PASCAL host program. The main trick is to know how to turn off the Language Card's RAM memory, which is overlaying the ROM MONITOR. The following pieces of code are a small "macro library", which I include in the head of each of my assembly routines.

```
; DEBUGGING AIDS FOR APPLE ASSEMBLER
;
.MACRO MONWHERE ;store address of $1
JSR $1 ;at location (15, 06)
$1 PLA
STA 16 ;assembler assumes numbers are
hexidecimal
PLA
STA 15
.ENDM
.MACRO MONCALL ;store registers at
STA 12 ;locations (12, 13, 14);
STX 13
STY 14
STA 0C082 ;select the monitor ROM
JMP 0FF59 ;jump to the monitor
.ENDM
.MACRO RETPAS
STA 0C08B ;RESTORE RAM TO PRE-BREAK
;CONDITION.
RTS ;AND RETURN TO PASCAL.
.ENDM
```

MONWHERE is used to determine where in memory the code-file is loaded at present. Since JSR "pushes" the return address on the stack, we merely pull it off again and store it for examination by the monitor.

MONCALL stores the register contents, then enables the monitor-ROM and jumps to the monitor. Several monitor-calls can be put into a given assembly program and used almost like break-points. It is also possible, using the monitor, to put JMP instructions to the head of the MONCALL macro (JMP is easier to install than the whole thing, when you're working in hex-code).

RETPAS is used to return to a PASCAL host program. Since MONCALL turns off the Language Card RAM to get at the Monitor ROM, we must switch back to RAM to go back to PASCAL. To use RETPAS it is necessary to have saved the return address from the stack, before entering the ROM monitor. In the example program PUT40 (at the end of this article) we use an "interface sequence" to connect PASCAL to the assembly routine PLOT:

```
POP RETURN ; POP is a macro, stores return
addr. at RETURN.
POP YLOC ; Pull 16-bit param. arguments off
stack.
POP XLOC
POP COL
LDA COL ; Move low-order half to another place
STA COLOR ; in Page 0
JSR PLOT ; Go to actual routine
PUSH RETURN ; Put PASCAL return address
back on stack.
RTS ; and return to host program.
RETURN: .WORD 0
XLOC: .WORD 0
YLOC: .WORD 0
COL: .WORD 0
```

To modify and reassemble the routine for monitor breaks, one would remove RTS and insert the macro RETPAS. MONWHERE and MONCALL could be inserted at various points in PLOT for debugging purposes.

RETPAS will work whether or not a monitor call actually occurs; so it could be left in place if desired, for future debugging.

A useful plan for debugging an assembly/PASCAL program package is to put "debugging statements" in the PASCAL host, which print out all the variables. A READLN command is also inserted inside any loop which contains calls to the assembly routines. Thus, whenever you are in PASCAL carriage-return moves you through the loop, allowing time to note the values printed by the debugging statements. When you hit a MONCALL in the assembly routines, you can note the state of assembly-level variables by examining location (15, 16 -hex) to find the code's absolute address, then dumping memory (or deassembling it with the L command) to locate the memory variables. This combination of PASCAL-level and assembly-level debugging output is usually sufficient to debug anything you may write, if you are careful in the first place.

We will conclude with a "guided tour" of a simple program called QUILT.

QUILT: an example of PASCAL with assembler.

The QUILT program is a simple "pattern-repeating" program. A cursor color is selected with game-paddle 0. Typing keys around 'K' moves the cursor (e.g. typing 'I' moves it upward.) When you type 'R' the pattern is repeated, wrapping around the screen as it runs off an edge. This program is used as a first lesson in the concept of loops, in the high school computer science curriculum which we are developing.

Let's "walk through" QUILT; first, we'll examine the PASCAL code, then the assembly code.

PASCAL has a very rigid structure; you must declare constants, then variables, then procedures, then have the main program. Procedure EXPLAIN is used so as to place the explanation-text close to the beginning of the source program, to avoid having to repeat it in a block comment. Procedures C0L0R40 and PUT40 are declared as EXTERNAL, meaning that they will be attached later by the LINKER. They will be in assembly language.

Procedures PUTBUMP and FIXIT are convenient, since their task is needed several places in QUILT.

The main program consists of two parts: First, the input keystroke sequence is stored until 'R' is hit. Then, the sequence is replayed until any key is hit. We see that the main program begins with calls to procedures EXPLAIN and READLN. The call to READLN just "stalls" while you read the explanation, and hit a carriage return. COLOR40 turns on the APPLE in low-resolution mode. CLEAR40 clears the screen. We then go through a "case" statement to make entries in two arrays, XLIST and YLIST, which store changes in the X and Y position of the cursor.

When an 'R' is typed we fall out of the REPEAT.. UNTIL loop, and into another loop REPEAT... UNTIL KEYPRESS. This loop repeatedly goes through the XLIST and YLIST arrays, using them to move and display the colored cursor.

Finally, we have a call to TEXTMODE which is hit after the program detects 'Q' (quit). If this is not done you wind up back at the UCSD PASCAL monitor, but still in graphics mode; this is confusing.

Now let's look at the assembly routines:

Note first that we use .INCLUDE MACROS TEXT. We don't see the macros expanded here, but they are present when the file is assembled just as if typed in.

We see PROC COLOR40, which just hits the correct memory addresses to set the APPLE up in low-resolution mode with a 4-line text buffer still visible (for user instructions).

PROC PUT40 uses three parameters; these are "POP" ped off the stack and stored in memory words (since integers are pushed as 16-bit words by PASCAL). POP is another macro in MACROS. TEXT; see the end of this article for macros POP, PUSH and COPY, useful accessories.

The actual 40 x 40 plot routine is just copied from the ROM monitor; we could have actually jumped to it rather than including it here. It performs an address calculation to deal with the APPLE's strange way of mapping memory, puts the color in, and returns to the "interface" routine at the top. That routine pushes RETURN on the hardware stack and returns to PASCAL.

Good luck in writing assembly programs with PASCAL!

(Much thanks to Charlie Hughes and Al Hoffman for assistance and ideas.)

More commonly used macros:

```
.MACRO PDP
PLA
STA %1
PLA
STA %1+1
.ENDM
.MACRO PUSH
LDA %1+1
PHA
LDA %1
PHA
.ENDM
.MACRO COPY
LDY #%3 ; COPIES UP TO 256 BYTES
$1; LDA %1,Y ;USED AS FOLLOWS
STA %2,Y ;COPY SOURCE, DEST, LENGTH
DEY ;SOURCE, DEST ARE LABELS
BNE $1 ;LENGTH IS INTEGER,0-255
.ENDM ;0 MEANS COPY 256 BYTES.)
```

POKE and PEEK require the following declarations:

```
TYPE BYTE = 0..255; ARRAY8 = PACKED ARRAY [0..1]
OF BYTE;
VAR X: RECORD
        CASE BOOLEAN OF
        TRUE: (INT: INTEGER);
        FALSE: (PTR:^ INTEGER)
        END;
    Y: RECORD
        CASE BOOLEAN OF
        TRUE: (INT: INTEGER);
        FALSE: (PTR:^ARRAY8)
        END;
PROCEDURE PIKE16(WHAT,WHERE:INTEGER);
BEGIN
X.INT: = WHERE;
X.PTR^: = WHAT
END;
PROCEDURE POKE8(WHAT,WHERE:INTEGER();
BEGIN
Y.INT: = WHERE;
Y.PTR^[0]: = WHAT
END;
FUNCTION PEEK8(WHERE:INTEGER):INTEGER;
BEGIN
Y.INT: = WHERE;
PEEK8: = Y.PTR^[0]
END;
```

```
PROGRAM QUILT;
        USES APPLESTUFF,TURTLEGR;
        CONST SEEDSIZE=50;
        VAR  LISTPTR,X,Y,I,COL:INTEGER;
             XLIST,YLIST,CLIST:
             ARRAY[0..SEEDSIZE]OF INTEGER;
             INCHAR: CHAR;
PROCEDURE EXPLAIN;
  BEGIN
  WRITELN;
  WRITELN('THIS PROGRAM LETS YOU DRAW');
  WRITELN('A PICTURE AND THEN MAKE A');
  WRITELN(' ''QUILT'' OF IT BY REPEAT-');
  WRITELN('ING THE PATTERN. TO DRAW:');
  WRITELN;
  WRITELN('SET PADDLE 0 FOR A COLOR');
  WRITELN('TYPE KEYS AROUND K TO MOVE');
  WRITELN('THE DOT. FOR INSTANCE ''I'' ');
  WRITELN('MOVES THE DOT UPWARD.');
  WRITELN;
  WRITELN('TYPE ''R'' TO SEE THE PATTERN');
  WRITELN('REPEATED.');
  WRITELN;
  WRITELN('TYPE ''H'' TO HALT THE REPEATING.');
  WRITELN('TYPE ''C'' TO CLEAR SCREEN.');
  WRITELN('TYPE ''QUIT'' TO END PROGRAM.');
  WRITELN;
  WRITELN('NOW TYPE ''RETURN'' TO BEGIN.');
  WRITELN;
  WRITLEN('REMINDERS:');
  WRITELN('KEYS AROUND K:MOVE;I=UP,J=LEFT,ETC.');
  WRITELN('C)LEAR  R)EPEAT  H)ALT REPEATING  Q)UIT');
  END;
```

```
PROCEDURE COLOR40; EXTERNAL;
PROCEDURE PUT40(COLOR:INTEGER;
                XLOC:INTEGER;
                YLOC:INTEGER);EXTERNAL;
PROCEDURE CLEAR40;
      BEGIN
      FOR X:=0 to 39 DO
       FOR Y:=0 to 39 DO
        PUT40(0,X,Y);
      X:=20;Y:=20;
      END;
```

```
PROCEDURE PUTBUMP;
      BEGIN
      LISTPTR:=LISTPTR+1;
      IF LISTPTR=SEEDSIZE THEN LISTPTR:=0
      PUT40(COL,X,Y)
      END;
```

```
FUNCTION FIXIT(ARG:INTEGER):INTEGER;
      (*THIS IS NECESSARY BECAUSE
      UCSD'S 'MOD' FUNCTION DOESN'T
      WORK FOR NEGATIVE NUMBERS.DON'T
      REALLY KNOW WHY THAT IS.*)
      BEGIN
      FIXIT:=ARG;
      IF ARG>39 THEN FIXIT:=ARG-40;
      IF ARG<0 THEN FIXIT:=ARG+40;
      END;
```

```
(* MAIN PROGRAM ---------------------
   ---------------------------------
   THIS CONSISTS OF TWO PARTS, THAT
   ARE REPEATED UNTIL SOMEONE TYPES 'Q':
   THE FIRST PART BUILDS UP AN IMAGE
   IN THE ARRAYS XLIST.YLIST.CLIST,
   WITH XLIST AND YLIST STORING THE
   'MOVES' -THAT IS, 0,+1 OR -1,THAT
   RESULT FROM EACH KEYSTROKE. CLIST
   IS A LIST OF COLORS THAT WERE IN
   EFFECT AT THE TIME OF EACH KEYSTROKE.
   THE SECOND PART RUNS THROUGH THE
   THREE LISTS AGAIN AND AGAIN, MOVING
   THE 'CURSOR' by ADDING XLIST[I]
   TO X, AND YLIST[I] TO Y,
   AND CHANGING THE COLOR 'COL' to
   CLIST[I].
   WE LET I RUN FROM 0 TO 'LISTPTR-1'
   BECAUSE LISTPTR WAS 'BUMPED' EVERY
   TIME A PICTURE ELEMENT (PIXEL) WAS
   ADDED; SO IT IS ONE TOO BIG.
   *)
      BEGIN
       (*SETUP STUFF*)
       EXPLAIN;
       READLN;
       COLOR40;
       CLEAR40;
       INCHAR:=' ';
       WHILE INCHAR<>'Q' DO
        BEGIN
       (* PART 1 : GET A PATTERN *)
       X:=20; Y:=20;
       LISTPTR:=0;
       REPEAT (* UNTIL 'R' *)
           REPEAT
           COL:=PADDLE(0) DIV 16;
           PUT40(COL,X,Y)
          UNTIL KEYPRESS;
          CLIST[LISTPTR]:=COL;
          READ(KEYBOARD,INCHAR);
          CASE INCHAR OF
          'U': BEGIN
                   XLIST[LISTPTR]:=-1;
                   X:=FIXIT(X-1);
```

```
            YLIST[LISTPTR]:=-1;
            Y:=FIXIT(Y-1);
            PUTBUMP
         END;
   'I': BEGIN
            XLIST[LISTPTR]:=0;
            YLIST[LISTPTR]:=-1;
            Y:=FIXIT(Y-1);
            PUTBUMP
         END;

   'O': BEGIN
            XLIST[LISTPTR]:=1;
            X:=FIXIT(X+1);
            YLIST[LISTPTR]:=-1;
            Y:=FIXIT(Y-1);
            PUTBUMP
         END;
   'J': BEGIN
            XLIST[LISTPTR]:=-1;
            X:=FIXIT(X-1);
            YLIST[LISTPTR]:=0;
            PUTBUMP
         END;
   'L': BEGIN
            XLIST[LISTPTR]:=1;
            X:=FIXIT(X+1);
            YLIST[LISTPTR]:=0;
            PUTBUMP
         END;

   'M': BEGIN
            XLIST[LISTPTR]:=-1;
            X:=FIXIT(X-1);
            YLIST[LISTPTR]:=1;
            Y:=FIXIT(Y+1);
            PUTBUMP
         END;

   ',': BEGIN
            YLIST[LISTPTR]:=1;
            Y:=FIXIT(Y+1);
            XLIST[LISTPTR]:=0;
            PUTBUMP
         END;

   '.': BEGIN
            XLIST[LISTPTR]:=1;
            X:=FIXIT(X+1);
            YLIST[LISTPTR]:=1;
            Y:=FIXIT(Y+1);
            PUTBUMP
         END;

      'C': CLEAR40;

      'Q': BEGIN
              TEXTMODE;
              EXIT(QUILT)
           END;
     END;(* CASE STATEMENT *)

UNTIL INCHAR='R';

  (* PART 2:

  NOW WE FALL OUT  OF THAT 'DRAWING'
  CODE (REPEAT BLOCK) AND FILL
  THE SCREEN WITH COPIES OF THE
  DRAWN IMAGE.*)
   REPEAT
      FOR I:=0 TO LISTPTR-1 DO
      BEGIN

        X:=FIXIT(X+XLIST[I]);
        Y:=FIXIT(Y+YLIST[I]);
        COL:=CLIST[I];
        PUT40(COL,X,Y)

      END
   UNTIL KEYPRESS;

   READ(KEYBOARD,INCHAR);

  END;(* BIG 'DO WHILE INCHAR<>'Q''
       BLOCK. WHEN Q IS HIT,
       PROGRAM ENDS. *)

  TEXTMODE;

 END.
```

```
        ; MOSHELL'S IMPROVED 40X40
        ; PIXEL-PUTTER... HAS THREE
        ; ARGUMENTS (IN PASCAL ORDER:
        ;   COLOR, X-COORD,Y-COORD).
        ;   ALL 3 ARE OF TYPE INTEGER.
        ;
        ; THE ORIGIN IS IN UPPER LEFT
        ; CORNER; X IS HORIZONTAL, Y IS
        ; VERTICAL.
        ;
        ; THESE ROUTINES COME FROM THE
        ; APPLE ROM MONITOR. (IF I'D HAVE
        ; DONE IT I WOULD HAVE PUT THE
        ; ORIGIN IN LOWER LEFT...)
        ;
        ; THE FOLLOWING PAGE 0 LOCATIONS
        ; ARE JUST LIKE THOSE USED BY THE
        ; MONITOR. THEY ARE TEMPS, ONLY.
GBASL   .EQU      26
GBASH   .EQU      27
MASK    .EQU      2E
COLOR   .EQU      30
        ;
        ; THESE MACROS PROVIDE THE PUSH
        ; AND POP WE USE BELOW.
        .INCLUDE MACROS.TEXT
        ;
        ; THIS FIRST ROUTINE TURNS ON THE
        ; DISPLAY IN 40 X 40 COLOR MODE,
        ; WITH 4 LINE TEXT AT BOTTOM. TO
        ; GET 48 (VERT) X 40 (HORIZ), OMIT
        ; LAST LINE.
        .PROC   COLOR40
        STA     0C050   ;SELECT COLOR
        STA     0C054   ;PAGE 1
        STA     0C056   ;LO-RESOLUTION
        STA     0C053   ;TEXT @BOTTOM.
        RTS
        ;
        ;PUT40 TAKES 3 ARGUMENTS (SEE MAIN
        ;HEADER. THIS PRELIMINARY ROUTINE
        ;IS THE 'INTERFACE' TO PASCAL.
        ;
        .PROC   PUT40,3
        POP     RETURN
        POP     YLOC    ;WE POP STUFF OFF THE
        POP     XLOC    ;STACK in 16-BIT
        POP     COL     ;WORDS, THEN DISSECT
        LDA     COL     ;BY BYTES.
        STA     COLOR
        JSR     PLOT
        PUSH    RETURN
        RTS
RETURN  .WORD   0
XLOC    .WORD   0
YLOC    .WORD   0
COL     .WORD   0
        ;
        ;HERE IS THE PLOTTER.
        ;
PLOT:   LDA     YLOC
        LSR     A
        PHP
        JSR     GBASCALC; DO THE BIT PERMUTATIONS ...
        LDY     XLOC
        LDA     COLOR
        AND     #0F
        PLP             ;LO-BIT OF Y-ADDR
        BCC     LONYB   ;INTO CARRY.
      ;
      ; HIGH-NYB CASE (TOP HALF OF PIXEL)
      ;
        CLC
        ROL     A
        ROL     A
        ROL     A
        ROL     A       ;COLOR TO HI-NYB
        STA     COLOR
        LDA     #0F
WINDUP: AND     @GBASL,Y
        ORA     COLOR
        STA     @GBASL,Y
        RTS
LONYB:  STA     COLOR
        LDA     #0F0
        BCC     WINDUP  ;ALWAYS TAKEN.
        ;
        ;THE BASE-CALC:ROUTINE
        ;THIS DOES SOME BIT-PERMUTING...
        ;SEE THE NEW APPLE MANUAL FOR EXPLANATIONS.
        ;
        ;
```

```
GBASCALC PHA
         LSR      A
         AND      #03
         ORA      #04
         STA      GBASH
         PLA
         AND      #18
         BCC      GBCALC
         ADC      #7F
GBCALC   STA      GBASL
         ASL      A
         ASL      A
         ORA      GBASL
         STA      GBASL
         RTS
         .END
```

©

THE ATARI® GAZETTE

All right you Atari owners. You've had those machines long enough to start cranking out some useful articles. Let's see some material on getting those nacgubes ti "fly"! Address your articles, reviews, and comments to me at COMPUTE, P.O. Box 5406, Greensboro, NC, 27403. Please mark the outside of the envelope "Attn: Atari Gazette". If your program listing is more than 20 or 30 lines, please include a short cassette tape.

Several months ago I commented that I expected to see hardare vendors entering the Atari market with competitive add-ons. Well, the first non-Atari **memory expansion board** for the Atari 800 is now on the market. We have one here and have been using it for over a month with no problems. Manufactured and marketed by JACC, Inc., (Just Another Computer Company, 543 West Golf Road, Arlington Heights, IL 60005, 312-364-6268), the 16K board sells for $179.95. While it does not have the protective plastic case of the Atari memory, the board plugs right in to the expansion memory slot just as the Atari produced boards do. My only warning would be that you should be especially careful to establish which side is the front of the board before attempting to plug it in. If you've been looking for a professionally produced source of alternate memory for your Atari, this group seems to know what they're doing. Watch them for more new hardware products for the Atari 800. One last note: At the time of this writing, we were waiting to hear from Atari regarding their warranty and JACC's board. I'll update you next time around. If you can't wait, I suggest you check with the company. ©

Robert Lock, Editor

Review:

An Invitation to Programming (cassette tape) Atari, Inc. Sunnyvale, CA Written by Program Design, Inc.

Joretta Klepfer, COMPUTE Staff

Program Design Inc. has produced six tutorial programs (lessons) which invite you to learn about your new Atari 400/800 computer and the BASIC language. The Atari computers have the capability of recording a sound track with a program on cassette. PDI has taken advantage of this feature and created tutorials which not only use the computer to demonstrate itself visually but also provide corresponding verbal instructions. The effect is much more dramatic and lasting when you can *hear* instructions while reading them on the screen. The potential use in education of this technique is fantastic. The tape is used in conjunction with the BASIC cartridge (CXL4002) and the programs are loaded by means of the conventional CLOAD command.

Beginning with the keyboard, the user is led step by step through the elementary concepts of using the various features of the Atari and of beginning BASIC. The first two lessons each have two parts and include introductory items about the course, keyboard, and computer language in general, along with error messages, line numbers, order of program execution, and the keywords PRINT, RUN, LIST, and NEW. In Lessons Three, Four, and Five more BASIC statements and concepts are introduced, such as variables, LET (assignment statement), INPUT, string variables, DIMension, IF THEN, and GOTO. Lesson Six is a demonstration of two programs which display sound, color, and graphics capabilities of the Atari. You can use these programs as building blocks for your own experiments with these concepts.

The first five lessons are written in a tutorial style with a set, you supply the answers before continuing with the lesson. You are given three chances to provide the correct answer and are properly rewarded with a nice musical chord if you do. You receive no "reward" for incorrect answers.

The programs have been developed with the goal of providing a self-teaching tool to the first-time computer user. Obviously you will not find an in-depth discussion of the BASIC language in just six lessons, but you will discover this tape to be a good introduction. Since the six programs are loaded separately you can easily set a comfortable pace for learning, with sufficient practice time between lessons for developing your programming skills. (Very important!)

The programs can be easily understood by a wide range of ages. Verbal instructions make it easier for younger children to use the lessons, but, some reading is necessary, and they may need some assistance in coping with the computer during "start-up". ©

Adding A Voice Track To Atari Programs

John Victor, President
Program Design, Inc.
11 Idar Court
Greenwich, CT 06830

We recently had a chance to see the latest in audio-visual technology-- a video tape machine being controlled by an Apple computer. The student was shown selected film sequences on the video tape. Then the video tape would stop and the student would be asked questions by the computer.

This demonstration had some impressive features, but the most important was the integration of voice with the computer question and answer technique. The same effect can be generated on an Atari 400 by combining text, graphics, animation and color with a sound track recorded on an audio cassette. And the Atari 400 is significantly cheaper and easier to program than the combination video tape player-computer.

There are several ways that a software designer-programmer can synch a cassette voice track to visuals on the computer screen. The cassette player that plugs into the Atari computer records and plays in a stereo format. The right track on the tape records and plays digital information (such as programs or data files), while the left track plays audio recordings. The "Talk And Teach" ROM and tapes supplied with Atari computers use both tracks simultaneously. As the voice explains material, ASCII characters are read off the digital track and shown on the screen. The two are coordinated in the manufacturing process so that they are always synchronized.

The problem with the "Talk And Teach" system for the average Atari owner is that the development of the Talk And Teach cassettes requires different hardware than is supplied with the Atari system. In fact, the system may be developed and run on non-Atari equipment-- we have seen the cassettes run on a modified TRS-80 computer.

The simplest and most practical method for Atari users to synch voice with their own educational programs is to use a "timed-BASIC" method. The visuals are programmed into a BASIC program and run simultaneously with an audio tape cassette. The program would then start and stop the Atari cassette player and change the visuals on the TV screen based on timing routines built into the program. The key to making this system work is that the audio tape must start at the same point each time it is used.

The computer course designer-programmer first writes a script as though he or she were producing a sound-film strip presentation. The spoken words, music, etc. would be specified along with a detailed description of what is to appear on the TV screen. The designer-programmer then writes a BASIC program that will produce the desired visual effects.

The next step is to coordinate the voice with the visuals. The best way to do this is to have a preliminary routine within the computer "freeze" each screen display until the programmer hits the 'RETURN' key. This can be done by sending the program to an INPUT subroutine, but this has the undesired side effect of printing an extraneous question mark on the screen. We prefer using the subroutine shown below since it prints nothing at all on the screen:

```
5000 IF PEEK(764)<>255 THEN POKE
764,255 :RETURN
5010 GOTO 5000
```

Memory location 764 indicates whether a key has been pressed. If no key has been pressed, the number 255 will be stored there. When a key has been pressed, the routine sets the value back to 255 (to keep the computer from printing the key press) and the program returns from the subroutine.

The designer-programmer should read through the script and manually check the screen changes to see that the BASIC program and the script match up and produce the desired results.

The third step is to place timing routines into the program so that the visuals will be in synch with the recorded voice. We do **NOT** recommend using FOR...NEXT timing loops for these routines. FOR...NEXT loop timing is not linear on the Atari. This means that the Atari might take one second to count from 1 to 300 in one loop, and less time to do the same count in another loop of different length. In addition, the length of the program and position of the subroutine also affects the count.

Fortunately, the programmer can utilize a built-in clock used by the Atari computer to count the scan lines in the TV display, which are stored in memory locations 18, 19 and 20. Location 20 counts in "jiffies" or 1/60 second. Each 1/60 increases the value stored in location 20 by 1. When the count reaches 256, the value is cleared to 0 and location 19 is incremented by 1. It takes the computer about 4.27 seconds to count from 1 to 256 in location 20, and about 18.2 minutes to count from 1 to 256 in location 19. You can watch this process with the following program:

```
10 PRINT PEEK(20), PEEK(19), PEEK(18):
GOTO 10
```

The results from these PEEKs could be converted to seconds, but we prefer to work in jiffies, which requires less math on the computer's part.

```
SECONDS = (PEEK(19)*256 + PEEK(20))/60
JIFFIES = PEEK(19)*256 + PEEK(20)
```

We recommend at this point that the designer-programmer make the final audio cassette that is to go with the computer program. The program can be timed to this cassette, and if all copies of the cassette can be made to use the same starting point, then the program will work with all copies as well.

The task now is to figure out the timing for each change so that the changes will be made in synch with the audio cassette. Figure I shows a program that we developed to automatically make these measurements for an audio tape. The user puts the audio tape in the Atari cassette player and rewinds it to the very beginning. With the play button depressed, the user runs the program. Line 20 starts the cassette player, and the program begins timing. At each point where the user wishes the computer to change the visual (in conjuction with the voice), the user hits the 'RETURN' key. At the end of the program the cassette is shut off, and the user is given the times between each point on the voice track where the computer is to change the visual.

The user should note that memory locations 19 and 20 are set back to 0 after each timing, and that line 55 looks specifically for an input from the 'RETURN' key. This program counts up to 15 changes, but this number can be increased by increasing COUNT in lines 40 and 100.

The last step is to insert the time values into the computer program and to check to see that the voice cassette works in synch with the program. Figure 2 shows a program that we wrote to illustrate how timing values can be coordinated with a teaching program and audio tape. Line 50 of the program defines the subroutines, of which there are three: one to print questions on the screen, one to time the visual so that the voice on the tape can read the question, and one to shut off the tape so that the student can answer the question just asked.

The QUESTIONASK subroutine in lines 4000-4030 gets its information for each question from a DATA line, which includes question number, screen color, answer to the question, number of lines to be read, and the lines of text making up the question. After printing the question, the program sets the time value for the voice, and goes to the clock subroutine at 5000-5020. When the correct time elapses, the program goes to the QUESTION-ANSWER routine. Here the tape is shut off, and the user is required to answer the question. Upon answering, the tape is turned back on.

The time values in this program are based on our own personal reading of the questions.

While it is possible to record both programs and audio on the same cassette and still utilize the method we have described here, the best way to record programs and audio separately. Ideally, the programs would be stored on disk and the voice on cassette.

It is possible that with very long audio cassettes the computer and tape will get out of synch due to small variations of the cassette player. The designer-programmer can correct for this by occasionally having the student press 'RETURN' when he or she hears a beep on the audio track. This gives a frame of reference for the program timing to match up to the tape. The least obvious way of doing this is to have the student press 'RETURN' before answering a question.

FIGURE 1

```
5 REM TIMING PROGRAM BY JOHN VICTOR
6 REM FOR ATARI COMPUTER VOICE TRACK
10 DIM TIME(15),A$(1)
20 POKE 54018,52:REM TURN ON CASSETTE
30 GRAPHICS 0:POSITION 2,6
35 PRINT "START COUNTING..."
40 FOR COUNT=1 TO 15:SETCOLOR 2,INT(RND(
1)*15),4
50 POKE 19,0:POKE 20,0
55 IF PEEK(764)<>12 THEN 55
60 JIFFY=256*PEEK(19)+PEEK(20):TIME(COUN
T)=JIFFY:PRINT "CHANGE #";COUNT
73 POKE 764,255
75 NEXT COUNT
78 POKE 54018,60:REM SHUT OFF CASSETTE
80 PRINT :PRINT "PRESS RETURN TO SEE TIM
E VALUES IN JIFFIES"
90 INPUT A$
100 FOR COUNT=1 TO 15:PRINT "CHANGE #";C
OUNT;"=";TIME(COUNT):NEXT COUNT
200 END
```

FIGURE 2

```
10 REM DEMONSTRATION OF ATARI TIMING
20 REM FOR TUTORIALS USING VOICE AND
30 REM TIMING LOOPS
40 REM PROGRAM DESIGN, INC.
50 CLOCK=5000:QUESTIONASK=4000:QUESTIONA
NSWER=3000:REM SUBROUTINE LABELS AND LOC
ATIONS
60 DIM ANSWER$(10),RESPONSE$(10),LINE$(4
0)
100 GRAPHICS 2+16:POSITION 0,2:PRINT #6;
"   BASIC TUTORIAL":PRINT #6;"   DEMONST
RATION":PRINT #6
105 PRINT #6;"    with voice"
110 TIME=300:GOSUB CLOCK
200 GRAPHICS 0:PRINT :PRINT
205 PRINT "This is a demonstration of th
e ATARI":PRINT "computer's ability to ut
ilize a"
206 PRINT "sound-voice track.  I will as
k four":PRINT "sample questions about AT
ARI BASIC.":PRINT
207 PRINT "Place audio cassette in playe
r and":PRINT "rewind to beginning.":PRIN
T
```

```
210 PRINT "Before starting this demonstr
ation,"
215 PRINT "make sure that the PLAY botto
n is"
220 PRINT "pressed down on your cassette
 player."
230 PRINT :PRINT :PRINT "PRESS RETURN TO
 START.":INPUT RESPONSE$
250 POKE 54018,52:REM STARTS TAPE
300 GOSUB QUESTIONASK:TIME=1274:GOSUB CL
OCK:GOSUB QUESTIONANSWER
310 GOSUB QUESTIONASK:TIME=681:GOSUB CLO
CK:GOSUB QUESTIONANSWER
320 GOSUB QUESTIONASK:TIME=683:GOSUB CLO
CK:GOSUB QUESTIONANSWER
330 GOSUB QUESTIONASK:TIME=800:GOSUB CLO
CK:GOSUB QUESTIONANSWER
340 GOSUB QUESTIONASK:TIME=653:GOSUB CLO
CK:GOSUB QUESTIONANSWER
400 GRAPHICS 1:SETCOLOR 2,0,14:SETCOLOR
4,0,14:POSITION 0,8:PRINT #6;"    END OF
 DEMO ":TIME=392:GOSUB CLOCK
410 POKE 54018,60:REM SHUT OFF CASSETTE
500 GRAPHICS 0:END

2999 REM ANSWERING SUBROUTINE
3000 POKE 54018,60:REM SHUTS OFF CASSETT
E
3010 PRINT :PRINT "YOUR ANSWER IS ";:INP
UT RESPONSE$
3020 IF RESPONSE$=ANSWER$ THEN PRINT CHR
$(253):PRINT :PRINT "CORRECT!":GOTO 3100

3040 PRINT :PRINT "NO, THE ANSWER IS ";A
NSWER$
3100 PRINT :PRINT "PRESS RETURN TO CONTI
NUE...":INPUT RESPONSE$
3110 POKE 54018,52:REM TURN CASSETTE BAC
K ON
3120 RETURN
3999 REM QUESTION SUBROUTINE
4000 GRAPHICS 0:READ NUMBER,COLOR,LINES,
ANSWER$
4010 SETCOLOR 2,COLOR,4:PRINT :PRINT :PR
INT "QUESTION #";NUMBER:PRINT :PRINT
4020 FOR COUNT=1 TO LINES:READ LINE$:PRI
NT LINE$:NEXT COUNT
4030 RETURN
4999 REM TIMING LOOP
5000 POKE 19,0:POKE 20,0:REM SETS CLOCK
TO 0
5010 IF PEEK(19)*256+PEEK(20)<TIME THEN
5010
5020 RETURN
6000 DATA 1,5,3,CLOAD,What is the usual
BASIC command to,tell the computer to lo
ad a program,from cassette tape?
6010 DATA 2,10,2,LIST,What command will
show you the pro-,gram stored in the com
puter memory?
6020 DATA 3,1,2,RUN,What command execute
s a program in,the computer's memory?
6030 DATA 4,3,3,CSAVE,What is the most c
ommonly-used ATARI,BASIC command used to
 record programs,to cassette tape?
6040 DATA 5,14,2,NEW,What command wipes
out the program,in memory?
```

©

COMPUTE needs you! Let us know what interesting applications you're coming up with for your Atari.

The Basics Of Using "Poke" In Atari Graphics

Charles G. Fortner

In order to use the poke statement in Atari graphics, we must first know two things:

1) Where to poke
2) What to poke

To determine where to poke, we must look at the **display list for each graphics mode.** This display list is found by **peek** (560) + **peek** (561) *256. The display list determines how the memory is displayed on the screen. The 5th and 6th byte of the display list hold the address of the first byte to be displayed. Table 1-1 gives the starting address for each graphics mode plus other information.

Determining what to poke involved trial and error with the following results:

1)Graphics Modes 3, 5, 7, 19, 21, 23
These modes are four color modes which display only four pixels for each eight bit byte of memory displayed. Bits 7 and 6, numbered as 7-6-5-4-3-2-1-0, determine the color of the first (left most) pixel; bits 5 and 4 the second; 3 and 2 the third; and 1 and 0 the fourth. The two control bits act as a **"COLOR"** statement for each pixel. If the hex value of the two control bits equals 0 it corresponds to a **"COLOR 0"** statement; if they equal 1, they correspond to a **"COLOR 1"** statement, etc.

2) Graphics Modes 4, 6, 20, 22
These modes are two color modes which display eight pixels for each eight bit byte of memory. Each bit acts as a **"COLOR"** statement for an individual pixel. A 1 in a location corresponds to a **"COLOR 1"** statement and a 0 corresponds to a **"COLOR 0"** statement.

3) Graphics Mode 8, 2, 4
These modes are high resolution modes with only one color. They display eight pixels per memory byte with a "1" bit displaying a pixel of the same color as the background but with a higher luminance. A "0" bit displays a pixel of the same color and luminance as the background.

The "COLOR" statements mentioned in the above explanations indirectly control the color of each pixel by determining which color register is active for an individual pixel. The exact manner a "COLOR" statement chooses this register is explained in Table 9.5 of the **Atari-Basic Reference Manual.**

Here's an interesting program to get started in graphics:

References: "Atari 400/800 Basic Reference Manual", Copyright 1980, Atari, Inc.

```
10 GRAPHICS 5
20 ADDR=PEEK(560)+PEEK(561)*256
30 ADDR=PEEK(ADDR+4)+PEEK(ADDR+5)*256
40 B=INT(RND(0)*800):REM -
 PICK A RANDOM BYTE IN DISPLAY
50 A=INT(RND(0)*255):REM -
 PICK RANDOM VALUE BETWEEN 0 AND 255
60 POKE ADDR+B,A:REM -
 POKE RANDOM VALUE INTO RANDOM BYTE
70 GO TO 40
```

TABLE 1-1

GRAPHICS MODE	DISPLAY DATA ADDR.	# OF ROWS	# OF COLUMNS	BYTES PER ROW	BITS DISPLAYED PER BYTE	# OF COLORS AVAILABLE
3	24176	20	40	10	4	4
4	23936	40	80	10	8	2
5	23456	40	80	20	4	4
6	22496	80	160	20	8	2
7	20576	80	160	40	4	4
8	NOTE 1	160	320	40	8	1
19	24176	24	40	10	4	4
20	23936	48	80	10	8	2
21	23456	48	80	20	4	4
22	22496	96	160	20	8	2
23	20576	96	160	40	4	4
24	16720	192	160	40	8	1

NOTE #1: Graphics Mode 8 has two addresses - 16720 is the starting address for the first 80 lines and 20480 is the starting address for the second 80 lines.

©

A Note on "The Basics of Using "POKE"..."

Robert Lock

Larry rewrote the original program that Charles sent in so it will adjust itself to your machines memory. After you try the program in the article, take a look at these. I expanded them to randomly alter the SETCOLOR parameters... you'll discover some of the versatility of your machine after you let the program run for five minutes or so.

```
10 GRAPHICS 23
20 ADDR=PEEK(560)+PEEK(561)*256
30 ADDR=PEEK(ADDR+4)+PEEK(ADDR+5)*256
35 I=INT(RND(0)*16)
36 J=INT(RND(0)*16)
37 K=INT(RND(0)*5)
38 SETCOLOR K,J,I
40 B=INT(RND(0)*3840):REM -
 PICK A RANDOM BYTE IN DISPLAY
50 A=INT(RND(0)*255):REM -
 PICK RANDOM VALUE BETWEEN 0 AND 255
60 POKE ADDR+B,A:REM -
 POKE RANDOM VALUE INTO RANDOM BYTE
70 GOTO 35
```

```
10 GRAPHICS 7
20 ADDR=PEEK(560)+PEEK(561)*256
30 ADDR=PEEK(ADDR+4)+PEEK(ADDR+5)*256
35 I=INT(RND(0)*16)
36 J=INT(RND(0)*16)
37 K=INT(RND(0)*5)
38 SETCOLOR K,J,I
40 B=INT(RND(0)*3200):REM -
 PICK A RANDOM BYTE IN DISPLAY
50 A=INT(RND(0)*255):REM -
 PICK RANDOM VALUE BETWEEN 0 AND 255
60 POKE ADDR+B,A:REM -
 POKE RANDOM VALUE INTO RANDOM BYTE
70 GOTO 35
```

©

Color Wheel for the Atari

Neil Harris

The Color Wheel program was written to experiment with some of the Atari's color graphic capabilities. The screen clears, and a series of lines radiate from the center of the screen in random colors, forming a shape with the outline of an ellipse. As the color bands sweep the screen, the colors shift in intensity and hue, forming a constantly changing set of contrasts and shapes.

The program itself is quite simple, thanks to the easy Atari BASIC graphics commands. Graphics mode 7 features 160 by 80 points of resolution in four colors, which are set up in registers. One of the things that made this program possible was that you can change a color register value, which causes all points on the screen associated with that register to change color instantly.

Line 100 selects degree mode for trigonometric functions, which in this case leads to less messy numbers in the FOR-NEXT loop in line 140. Lines 120 and 130 select values for DX and DY, which determine the shape of the ellipse for that cycle. The STEP in line 140 was added because the smaller ellipses otherwise took the same time to draw as bigger ones. Line 145 randomly selects the color register for the current line (an interesting variation is to move this line to line 135, making each ellipse a solid color). Line 150 plots a point at the center of the screen. The formula in the DRAWTO in line 160 was arrived at by using simple trigonometry to determine the point on the ellipse at any given angle around the center. The SETCOLOR statement in line 180 changes a random color register on the screen to a random hue and intensity, and is selected 30% of the time by line 170. Line 190 completes the loop, and 200 allows the program to select a new ellipse shape and keep drawing. I usually put some PRINT statements between lines 110 and 120 for a message in the text window.

This program provides a nice way to have the Atari show off its nice range of colors, and the plotting routine has been reduced to its bare essentials.

```
100 DEG
110 GRAPHICS 7
120 DX = INT(RND(1)*80)
130 DY = INT(RND(1)*40)
140 FOR L = 0 TO 360 STEP (140-DX-DY)/20
145 COLOR INT(RND(1)*5)
150 PLOT 80,40
160 DRAWTO 80 + DX*SIN(L),40 + DY*COS(L)
170 IF RND(1)> .3 THEN 190
180 SETCOLOR INT(RND(1)*4), INT(RND(1)*16),
INT(RND(1)*8)*2
190 NEXT L
200 GOTO 120
```

Because the Atari starts playing with the colors if the system editor doesn't do anything for a few minutes, it is a good idea to PRINT something at line 195. An up arrow should work just fine, and not tamper with the display in any way. The high resolution graphics is controlled by the screen, while the editor is running the text window. At least, that's what I *think* is happening. ©

Choose Your Joystick

Len Lindsay

A joystick is just about standard equipment for all ATARI computer users. Most games will probably use it for input of moves. Every ATARI program I have seen thus far TELLS you where to plug your joystick (the program looks only at that one spot for your move). That means that you must continually change which plug your joystick is in, or trade joysticks with your friend in order to switch players. This is unfortunate, for it takes only a very few lines in BASIC to allow you to use whatever joystick plug you wish. Here is the routine:

```
20000 ZJ = 99
20010 FOR Z1 = 0 TO 3
20020 IF STRIG(Z1) = 0 THEN ZJ = Z1
20030 NEXT Z1
20040 IF ZJ>3 THEN 20010
20099 RETURN
```

That is all there is to it (lines 20010 & 20020 can be combined, as can 20030 & 20040). It is a very easy to use routine. Simply use a line like this:

```
500 PRINT "HIT YOUR FIRE BUTTON TO
BEGIN":GOSUB 20000
```

Now, all input via joysticks is taken like this:

```
900 MOVE = STICK(ZJ)
```

If all game programs used this type of routine, you would never have to unplug your joystick. The ATARI is a smart computer, why not use it to its fullest? ©

Star Fleet to All Cruisers...

Who's the current champ of Star Raiders? Send in your best score, comments and playing strategies to COMPUTE, P.O. Box 5406, Greensboro, NC 27403.
Attn: "Atari Gazette"

Input/Output On The Atari

Larry Isaacs, Associate Editor

In this article I will try to explain how to use the various BASIC commands at your disposal to communicate with the peripheral devices in your system. These peripheral devices include the Screen Editor (E:), keyboard (K:), and TV Monitor (S:), all of which are part of your machine. External devices which are currently available include disk drives (D1: through D4:), printer (P:), and cassette (C:). The I/O (Input/Output) commands we will be discussing are the PUT, GET, PRINT, INPUT, XIO5, XIO7, XIO9, and XIO11 commands. Also, the discussion will be limited to the use of these commands as it relates to logical files.

Before we get into details, there are two important facts to remember. The first one is that these I/O commands result in the transfer of one or more bytes of data, and that often, these bytes will be ATASCII characters. The second fact is that the byte or bytes which get transferred will be the same regardless of the device with which you are communicating.

OPEN AND CLOSE

Before you can communicate with a peripheral device, it must first be "opened", and in the case of the disk, a file name provided. The syntax of the open command is as follows:

OPEN #iocb,mode,0,"device:name.ext"

iocb - I/O Control Block. This should be an arithmetic expression which evaluates to a number from 1 to 7. It specifies the I/O Control Block through which BASIC will send its requests to the I/O software.

mode - This should be an arithmetic expression which evaluates to 4, 6, 8, 9, or 12. For now we will just be using 4, 8, and 12. Their meaning is as follows:

4 = open for reading
8 = open for writing
12 = open for reading and writing

device - This should be a letter which identifies which device to associate with the I/O Control Block specified previously.

name - This should be a name of up to six alphanumeric characters, the first of which must be a letter.

ext - This is an extension to the name which is usually used to indicate the type of file, BASIC program, data, etc. It may include up to three alphanumeric characters. The name plus extension form the file name which is needed when communicating with the disk.

Once you have opened a device, you communicate with that device using the "iocb" number. To close a device or file, you use the CLOSE command. The syntax for this command is as follows:

CLOSE #iocb

Only one device can be associated with an IOCB at a time. If you wish to associate a new device with an IOCB that is currently in use, you must close the old device first. In the case of the disk, cassette, and the printer, a CLOSE command may be required for proper operation. For example, the disk can only write groups of 128 bytes, called sectors, which are written once enough data has been received to fill the sector. The CLOSE command is required to cause the last sector of a file, which is only partially filled with data, to be written to the disk. The cassette also needs a CLOSE command to write the last group of bytes. And since the printer doesn't print a line until an EOL (End of Line) character is received, a CLOSE may be needed to print out the last line.

If a program terminates without error, or via an END statement, all open devices and files will be closed automatically. If the program terminates because of an error, due to a STOP statement, or due to the BREAK key being struck, the devices and files will be left open. If you aren't able to continue the program, you may close the devices and files by entering the necessary CLOSE commands directly, i.e. without line numbers. Also, executing the RUN command will close any open devices or files.

Now we will begin our discussion of the I/O commands. Many of the examples make use of the disk. If you wish to use cassette instead, simply, change the file specification in the OPEN commands to the cassette device. Just place a blank cassette in the cassette player. Then whenever you hear two beeps, rewind the tape, press PLAY and RECORD on the player, then hit RETURN on the ATARI. Whenever you near one beep, rewind the tape, press PLAY on the player, then hit RETURN on the ATARI.

PUT AND GET

Lets look first at the PUT and GET commands, which are the most basic of the I/O commands. These two commands result in the transfer of a single byte, with the PUT command sending a byte, and the GET command receiving a byte. Here is the syntax for the commands:

GET #fn,variable where "fn" is a file number, and "variable" is a simple variable, not an array or string variable

PUT #fn, expression where "expression" is an arithmetic expression

Listing 1 provides an example for using the GET and PUT commands. In this program the Screen device is opened for reading and writing. This open command will also cause the screen to be cleared. The letters from "A" to "Z" are sent to the Screen using the PUT command, and after the cursor is repositioned, the letters are fetched back from the Screen using the GET command.

Listing 1

```
10 DIM T$(30)
20 OPEN #1,12,0,"S:":REM OPEN FOR R/W
30 FOR I=ASC("A") TO ASC("Z")
40 PUT #1,I:NEXT I
50 POSITION PEEK(82),0
60 FOR I=1 TO 26:GET #1,CHARACTER
70 T$(I,I)=CHR$(CHARACTER)
80 NEXT I
90 GET #1,I:REM MOVE CURSOR PAST THE Z
100 CLOSE #1
110 PRINT :PRINT T$
```

Listing 2 provides a similar example which communicates with a disk. Note, if you run this program a second time, opening the file for writing will cause the old file to be deleted. Also, if you try to get more bytes than were written to the file, an ERROR 136 (End of File encountered) will be given. Changing the 26 to 27 in line 60 will illustrate this.

Listing 2

```
10 OPEN #1,8,0,"D:TEST.DAT"
20 FOR I=ASC("A") TO ASC("Z")
30 PUT #1,I:NEXT I
40 CLOSE #1
50 OPEN #1,4,0,"D:TEST.DAT"
60 FOR I=1 TO 26
70 GET #1,CHARACTER
80 PRINT ;CHR$(CHARACTER);
90 NEXT I
100 CLOSE #1
```

The advantage of using GET and PUT is that you are controlling the transfer of individual bytes. If this isn't necessary, you will likely find it simpler and faster to use one of the following I/O commands. Each of these commands involves the transfer of a string of bytes.

PRINT AND INPUT

The PRINT and INPUT commands are used to transfer a string of characters. The syntax of these commands is as follows:

PRINT #iocb;list where the "list" is a list of expressions separated by commas or semicolons. The expressions may be numbers, strings, simple variables or string variables. If a semicolon is used prior to an expression, the characters for this expression will be sent immediately following any previous characters. If a comma is used instead of a semicolon, including the one shown in the syntax, tabbing will occur before characters from the expression are sent. If the list doesn't end with a comma or semicolon, an EOL character will be sent at the end of the list. If you wish, the list need not contain any expressions.

INPUT #iocb, list where the "list" is a list of expressions separated by commas. The expressions may be simple variables or string variables.

When printing strings, naturally the characters in the string are sent. However, when you print a number, the number is converted to a string of digits and sent as ATASCII characters. When you input a string, characters will be fetched until an EOL character is received. These characters will be stored in the string's reserved memory until the reserved memory is filled, or the EOL character is received. When you input a number, characters will be fetched until an EOL character or a comma is received. At this point, assuming all the characters were valid digits, the string is converted back to a number.

Listing 3 provides an example of using PRINT and INPUT with the Editor device. Like the Screen device, the Editor will print and fetch characters from the screen memory. However, when printing to the Editor, control characters will perform the associated function instead of printing a character. When you input from the Editor, RETURN must be hit before the Editor will begin sending characters. Also, the Editor remembers the line and column of the cursor when the input request is made. As long as you don't hit a cursor-up or cursor-down, the fetching of characters from the screen memory will begin at this point once RETURN has been hit. If a cursor-up or cursor-down has been hit, the fetching of characters will begin with the first character of the new line which the cursor occupies. The fetching of characters will continue until the last nonblank character of the line occupied by the cursor when RETURN was hit. You can explore the operation of the Editor further by making changes to Listing 3, and finding out what happens.

Listing 3

```
10 DIM T$(80)
20 OPEN #1,12,0,"E:"
30 PRINT #1;"123,CHARACTERS"
40 PRINT #1;CHR$(28);:REM AN UP-CURSOR
50 INPUT #1,T$
60 PRINT #1;CHR$(28);
70 INPUT #1,NUMBER
80 CLOSE #1
90 PRINT "T$=";T$
100 PRINT "NUMBER=";NUMBER
110 REM JUST HIT RETURN WHEN EACH
120 REM OF THE INPUT STATEMENTS
130 REM EXECUTES
```

Listing 4 gives an example of using PRINT with the disk. The program reads back the characters using the GET command so you can see what was sent to the disk by the PRINT command. Again, you can experiment with changes to this program to improve your understanding of how these commands operate.

Listing 4

```
10 DIM T$(10)
20 T$="ABCDEFGHIJ"
30 T$(5,5)=CHR$(155)
40 OPEN #1,8,0,"E:"
45 ? "PRINT DOES THIS"
50 PRINT #1;T$
55 ? :? "XIO DOES THIS"
60 XIO 9,#1,8,0,T$
70 CLOSE #1
```

XIO9 AND XIO5

The XIO9 and XIO5 commands, like the PRINT and INPUT commands, send and receive a string of characters. The syntax for these commands is as follows:

XIO cmdn,#icob,mode,0,exp

"cmdn" is the XIO command number.
9 = PUT RECORD
5 = GET RECORD
"iocb" and "mode" have the same function as in the OPEN statement.
"exp" may be a string or string variable when writing, or a string variable when reading.

The XIO9, or PUT RECORD command will write characters from the specified string until an EOF character is written. If the string contains an EOF character, the XIO9 terminates at this point, and the rest of the string isn't written. If the string does not contain an EOF character, one is appended. This differs from the PRINT command where the entire string is written regardless of content. The program in Listing 5 illustrates this difference.

Listing 5

```
10 DIM D$(1),T$(4),T1$(4),T2$(10)
20 OPEN #1,8,0,"D:TEST.DAT"
30 XIO 9,#1,8,0," ABCDEFGHIJK"
40 CLOSE #1
50 T1$="XXXXXXXXXX":REM RESET T1$
60 T1$="YYYYY":REM MAKE LENGTH 5
70 OPEN #1,4,0,"D:TEST.DAT"
80 INPUT #1,T2$
90 CLOSE #1
100 ? "INPUT DOES THIS"
110 ? T2$,LEN(T2$)
120 T2$="XXXXXXXXXX":REM RESET T2$
130 T2$="YYYYY":REM MAKE LENGTH 5
140 OPEN #1,4,0,"D:TEST.DAT"
150 XIO 5,#1,4,0,T2$
160 CLOSE #1
170 ? :? "XIO5 DOES THIS"
180 ? T2$,,LEN(T2$)
190 T2$(10,10)="Z"
200 ? T2$,LEN(T2$)
210 ? "NOTICE THE X ISN'T WRITTEN OVER"
220 T$="XXXX":T1$="XXXX":T2$="XXX"
230 OPEN #1,4,0,"D:TEST.DAT"
240 XIO 5,#1,4,0,D$
250 CLOSE #1
260 ? :? "OR XIO5 CAN DO THIS"
270 ? T$:? T1$:? T2$
```

The XIO5 command, like the PRINT command, will fetch one string and store it in memory. But where the PRINT command stops (when the memory reserved for the string variable is filled), the XIO5 command keeps going. This means that the XIO5 command can load more than one string variable. A second difference is that the PRINT command doesn't store the EOL character, where the XIO5 command does. And one last difference, the PRINT command will change the length of the string variable to the number of characters stored, where the XIO5 command doesn't change the length of any string variable. Before you can make productive use of the XIO5 command, there is one more necessary fact. Once the XIO5 command fills the first string variable to its current length, the next character fetched is apparently discarded, and the next memory location is left unchanged. This applies only to the string variable specified in the command statement. The program in Listing 6 illustrates the preceding discussion.

Listing 6

```
10 DIM D$(1):REM DUMMY STRING
15 REM FIND ADDRESS OF DISPLACEMENT
20 ADDR=(PEEK(134)+PEEK(135)*256)+2
25 REM FIND BEGINNING OF ARRAY STORAGE
30 BOA=ADR(D$)
40 DIM MARK1$(1),ARRAY1(80),X1$(23)
45 REM WITH 6 BYTES/ARRAY NUMBER,
46 REM THIS DIMENSIONS 510 BYTES
47 REM OR 2*255
48 REM NOW FILL THE ARRAY
50 FOR I=0 TO 80:ARRAY1(I)=I:NEXT I
60 OPEN #1,8,0,"D:TEST.DAT"
65 REM NOW WRITE THE ARRAY IN 2 BLOCKS
70 FOR N=0 TO 1
80 TMP=ADR(MARK1$)
90 GOSUB 1000
110 XIO 11,#1,8,0,D$
120 NEXT N
130 CLOSE #1
140 DIM MARK2$(1),ARRAY2(80),X2$(23)
```

```
150 OPEN #1,4,0,"D:TEST.DAT"
155 REM NOW READ THE ARRAY
160 FOR N=0 TO 1
170 TMP=ADR(MARK2$)
180 GOSUB 1000
190 XIO 7,#1,4,0,D$
200 NEXT N
210 CLOSE #1
220 FOR I=0 TO 80 STEP 10
230 ? ARRAY2(I)
240 NEXT I
250 END
900 REM SUBROUTINE TO FIX THE
905 REM DISPLACEMENT- N=BLOCK NUMBER
1000 TMP=TMP-BOA+(N*255)
1010 POKE ADDR,TMP-INT(TMP/256)*256
1020 POKE ADDR+1,INT(TMP/256)
1030 RETURN
```

XIO11 and XIO7

The XIO11 and XIO7 commands are used to write and read blocks of 255 bytes, respectively. The syntax for these commands is the same as for XIO9 and XIO5 except for the command number. The commands transfer bytes beginning with the reserved memory of the string variable specified in the command. Since you are transfering bytes, their content has no effect on the operation of the command. As with the XIO5 command, once the XIO7 command fills the current length of the first string variable, the next byte fetched isn't stored in memory.

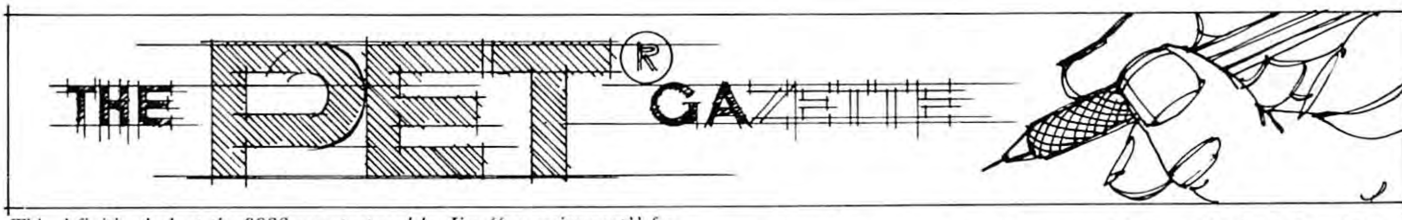
Naturally the XIO11 and XIO7 commands could be used for handling strings of characters. However, if we knew where the address of a string's reserved memory was kept, we could make changes to it, and use these commands to save and restore any portion of memory we want. Fortunately this isn't too difficult. Each string variable will have an entry in the variable storage area, which contains 8 bytes of parameters for the variable. The third and fourth bytes of the parameters contain the displacement from the beginning of the array storage area to the reserved memory for that string. If we dimension a string variable in the first statement of a program, then this displacement can be found by PEEK(134) + (PEEK(135)*256) + 2. Also, the address of the reserved memory for this string will be at the beginning of the array storage area. For more detail about this, see INSIDE ATARI BASIC and ATARI TAPE DATA FILES in COMPUTE #4.

Listing 7 shows how to save an array to disk and then read the data from disk into a different array. In this program we direct the XIO11 command to save the desired portion of memory by POKEing the required displacement into the parameters of D$. We then read the data into a different array, which could have been in a different program, by again POKEing the necessary displacement into the parameters of D$. Note the use of the MARK strings and the ADR function to find where the arrays are in memory. Another application might be to add some machine language routines to a program by reading them from disk or cassette and storing them in the required location in memory.

Listing 7

```
10 DIM T$(10)
20 T$="ABCDEFGH"
30 OPEN #1,8,0,"D:TEST.DAT"
40 PRINT #1;T$
50 CLOSE #1
60 OPEN #1,4,0,"D:TEST.DAT"
70 FOR I=0 TO 8
80 GET #1,A
90 ? A,CHR$(A)
100 NEXT I
110 CLOSE #1
```

This concludes the explanations of the various I/O commands. Hopefully I have explained them well enough for you to put them to productive use. Some of the explanations are fairly brief, so to find out more, or to better understand their operation, I highly recommend that you do some experimenting of your own. This is the best way to find out what the commands will do in specific situations. ©



This definitive look at the 8032 was prepared by Jim "on assignment" for the June issue of PRINTOUT, a PET publication in England. Thanks Jim, for sharing it with us. One p.s.: It's my understanding that production models do have a small "bell" built-in. RCL

Butterfield Reports: The 8032

Jim Butterfield
Toronto

Commodore's newest 8000 series computers—the CBM models 8016 and 8032 — are noticeably different from earlier models. The most obvious change is the width of the screen, which now allows 80-character lines. Yet it's a gentle upgrade in many ways; users will be relieved to find that there's a great deal of compatibility with previous PET/CBM devices.

Some users will dislike any change, of course. Others would like to see Commodore boldly scrap previous architecture and introduce radical changes. But for most of us, the new design looks like healthy evolution: improvements are being introduced without losing continuity.

Physical Appearance

The screen is bigger, of course, increased to 12 inches across the diagonal as compared to 9 inches on previous models. The computer itself hasn't gained appreciably in size, however. The keyboard housing has been lowered, so the overall height of the 8000 is only about an inch higher than the original PET. The width hasn't changed at all, although the new machine appears less 'tapered' because of the larger screen.

Commodore seem to view the 8000 series as strictly business machines, and all models I have seen are equipped with the business keyboard. This is a standard ASCII keyboard plus a numeric pad. Users with experience on other machines will find it quite standard. Programmers who have been spoiled on PET's graphic keyboard will need some adjustment time: it seems hard, somehow, to have to shift for such popular characters as quotation marks, brackets, asterisk, or the question mark. I find it harder in an instructional environment: you can't reach over a user's shoulder and tap in a correction quite as easily. But the business keyboard is an industry standard, and we might as well get used to it. There are interesting new keys: REPEAT, TAB and ESC (escape), which I'll talk about later.

Edge connectors are unchanged from the previous 16K and 32K models: they are all there.

General apperance and quality is comparable to earlier models. Characters are quite readable on the green screen; resolution is good. Text mode (upper/lower case) is the standard power-on condition, but you can get graphics if you want them. Lower-case characters are identical to earlier models, and descenders on characters like y, g, or p still look a little uncomfortable.

Keyboard Features

Cursor movement keys, Insert/Delete, and SPACE automatically repeat if you hold them down. There's an initial pause of a half a second, and then the cursor takes off at a rate of about ten steps per second. The other keys don't repeat unless you hold down the REPEAT key; then they take off right away, with no initial pause. All very handy.

Screen tabulation is built in. Pressing Shift/TAB sets or clears the tab stop wherever the cursor happens to be. After that, pressing TAB takes you to the next tabulation stop in that line; if there isn't one, you'll go to the end of the line. Since Shift/TAB doesn't tell you whether you're setting or clearing the tab, a little care is needed in programming this. Tabulation is very useful for screen tables, but doesn't help for the printer.

The ESC key does a very simple job. It takes you out of the programmed-cursor mode (sometimes called the 'quotes' mode). A minor help for existing activities; but it turns out you'll also need it to invoke some of the clever new screen features that are built into the system.

The RVS key no longer slows down the display when things are printing at a fast clip. This job is now down by the left-arrow key. As usual, the slow-down applies only during screen scrolling.

A new and very handy feature allows screen output to be stopped and then resumed. The colon key freezes the display immediately after a screen scroll. Pressing the left-arrow allows output to continue. This is useful for both freezing program listings and stopping lengthy output runs; it allows the data to be examined before it disappears.

The Bell

The system contains an electronic bell which chimes on power-up, on request, and whenever you near the end of a line. On the models I have seen, there is no speaker or sounding device, but this may be added in final production versions. At any rate, the bell signal can be heard on the CB2 line of the Parallel User Port; a simple amplifier/speaker system will do the trick.

The chiming of the bell interrupts other activities momentarily. If you're repeating characters at high speed on the screen, you'll see a brief pause when you reach the point near end-of-line where the bell sounds. If this is a problem, the chime can easily be shortened or eliminated entirely.

Screen Graphics

The screen still holds 25 lines of text, but each line is 80 characters wide. This would make the characters look tall and skinny if the original screen arrangement were kept. Commodore has restored the proportion of the characters by inserting a strip of empty space between each line.

This looks good, but it means that characters cannot touch above and below; and this in turn would cause odd-looking chopped up graphics. A special arrangement is needed to make graphics look good.

When you go to graphics mode, the screen closes up the empty space so that characters can touch. The effect is rather like Cinemascope: the overall height of the display shrinks to less than five inches, while the width remains unchanged at about eight inches. In this mode, graphics from earlier PET/CBM machines look quite good.

This shrinking and expanding of the screen is accomplished with a new type of chip called a 'CRT Controller'. The 8000 delivers a lengthy list of desired screen characteristics to this controller chip, which takes over the job of arranging things on the screen as desired. Although Commodore have only two types of setup for the screen — text mode and graphics mode — it looks as if the controller chip could be used in many other ways for special effects.

New Screen Controls

A number of useful screen features are included in the 8000 machines. They promise to add greater convenience and versatility to screen usage.

SCROLL DOWN and SCROLL UP move the entire screen up or down one line. A blank line is left at the top or bottom as appropriate.

INSERT LINE AND DELETE LINE are almost self-explanatory. They move the screen from the cursor line, adding a blank line at the appropriate place.

ERASE BEGIN and ERASE END each clear part of a screen line. ERASE BEGIN clears from the beginning of the cursor line up to but not including the cursor position. ERASE END clears from the cursor to the end of line, inclusive; it's a good way to clear a line just before printing on it.

Definable Windows

A dramatic new feature of the 8000 screen is the capability of restricting screen activity to a 'window' area. Once a window is defined, all normal activities are confined within it. Text will go into the window area only; cursor movements will stay within the area; and scrolling, clearing, inserts and deletes will all take effect only within the bounds of the window area.

A window is defined by moving the cursor to the desired upper left location and giving a SET TOP command; then moving the cursor to the desired lower right location and giving a SET BOTTOM command. Only one window may exist at a time; but since they are easy to set up, windows can be switched in and out as desired and the effect of simultaneous windows can be easily achieved.

The window is cleared — or, more precisely, set to the full screen — by giving two HOME commands in succession. This means that a user can always get out of a window with a couple of keystrokes — even if the programmer would rather he couldn't.

The variable window system is nicely done, and careful attention has been given to seeing that all systems interact properly. For example, a user responding to a Basic INPUT statement with a preset window won't have his entered value confused with any adjacent out-of-window data that may be nearby. The bell is even adjusted so that it rings five characters from the end of the window you happen to be in.

Special Programmed Characters

Most users are familiar with existing programmed cursor characters; for example, the reverse-q for cursor down. The new 8000 features have been implemented as additional programmed characters. They appear as reverse field characters within a print string. The new programmed characters can be summarized as follows:

Character	Lower Case	Upper Case
g	Ring Bell	
i	Tabulate	Set/Clear Tab
n	Set Text Mode	Set Graphics
o	Set Top	Set Bottom
u	Delete Line	Insert Line
v	Erase End	Erase Begin
y	Scroll up	Scroll down

There's a small trick to creating these new reversed characters. Unlike the more familiar programmed cursor characters, these functions don't appear as keys on the keyboard. To create them within a quote-

enclosed string, you need to press ESC (which takes you out of programmed cursor mode); then RVS followed by one of the above characters. It works well, but takes a little getting used to.

Software

The current 8000 units are fitted with Basic 4.0. This is the familiar Upgrade ROM with disk operators added to Basic, and with an improved garbage collection routine.

Garbage collection is much faster: a hesitation is visible, but all collections take place within a second or so. The penalty paid by the user for this is a relatively slight one: two more bytes are used for each dynamic string.

The 8000's ROM set is larger. Basic now starts at hexadecimal B000 rather than C000, meaning that users now have an 18K ROM system rather than 14K.

The disk commands are English language equivalents to the commands already available with the DOS or 'wedge' system. At first glance, they don't seem to do anything new. However, they are much easier to use for newcomers; and they strengthen the PET's excellent human interface. It's a great convenience to be able to press Shift and Run/Stop to get the first program from a disk. Another advantage of these commands is that they can be embedded in a program. We can now write 100 SCRATCH "OLDFILE" and gain ease and simplicity of coding.

The disk commands appear to be written with the new Disk system in mind. This is a new set of ROMs for disk which will enhance its ease of use greatly. No hardware changes are needed: fit the new ROMs, and you have a new disk system. This system doesn't need an explicit Initialize command — it's automatic when a new disk is inserted. There are several attractive features of the new disk system, including an APPEND command and a new style of relative or direct access file; the 8000 commands are geared to support all this.

A brand new disk unit, the 8050, is being introduced by Commodore. Its characteristics are identical to those of the enhanced 2040 unit, except for its huge capacity. With 77 tracks of data and more sectors on each track, a mini-floppy can now hold in excess of two thousand blocks of data. This gives a capacity of over one megabyte of data for the two drives. Many commercial and data storage applications will now fall within the scope of the PET/CBM.

The 8050 dual floppy unit has a similar size and appearance to the well-known 2040. It has a power-on indicator: the centre LED now shines green for power applied, and red for errors. Diskette insertion is slightly different. The high-precision drive mechanism requires precise centering; the simple 'tab' door has been replaced with a more elaborate assembly.

Communications between the PET/CBM and the 8050 unit are the same as for the 2040. Most disk programs will be fully compatible unless they use some of the more esoteric advanced commands. You can't read an 8050 disk on a 2040 unit, or vice versa, but Commodore has developed a utility program which will copy files from one to the other. It's interesting to note that the Commodore program seems to be able to change the 8050 device number on the IEEE bus via software.

Compatibility between the two styles of 2040 system — the original and the upgrade — exists in part. Each type of unit can read the other's data, but cannot write to a disk that was new-ed on the other machine. A quick COPY will fix up any problems in this area, but the user should keep in mind that there are six less blocks available on the new style of 2040.

Machine Language Considerations

The various subroutines in Basic ROM have moved around, of course: that's unavoidable. Machine language programmers will be happy to know that the vital zero page locations are virtually unchanged. Adapting existing programs shouldn't be too hard.

The second cassette buffer is now used in two ways. The lower addresses are used for the disk commands; and higher addresses (hex 03FF and up) are used for tabulating TAB positions. Greater care will need to be exercised in using the buffer for staging small machine language jobs.

The Machine Language Monitor is unchanged. It can still be extended with extra commands; and 'uncrashing' is accomplished in the same manner as for previous models.

Some of page zero has been released. Since screen lines no longer need to be linked together into 'double' lines, the fairly lengthy screen-wrap table isn't needed.

Serious programmers will be delighted to find that special vectors are provided for input and output. Now they will be able to link their own I/O routines directly into the system.

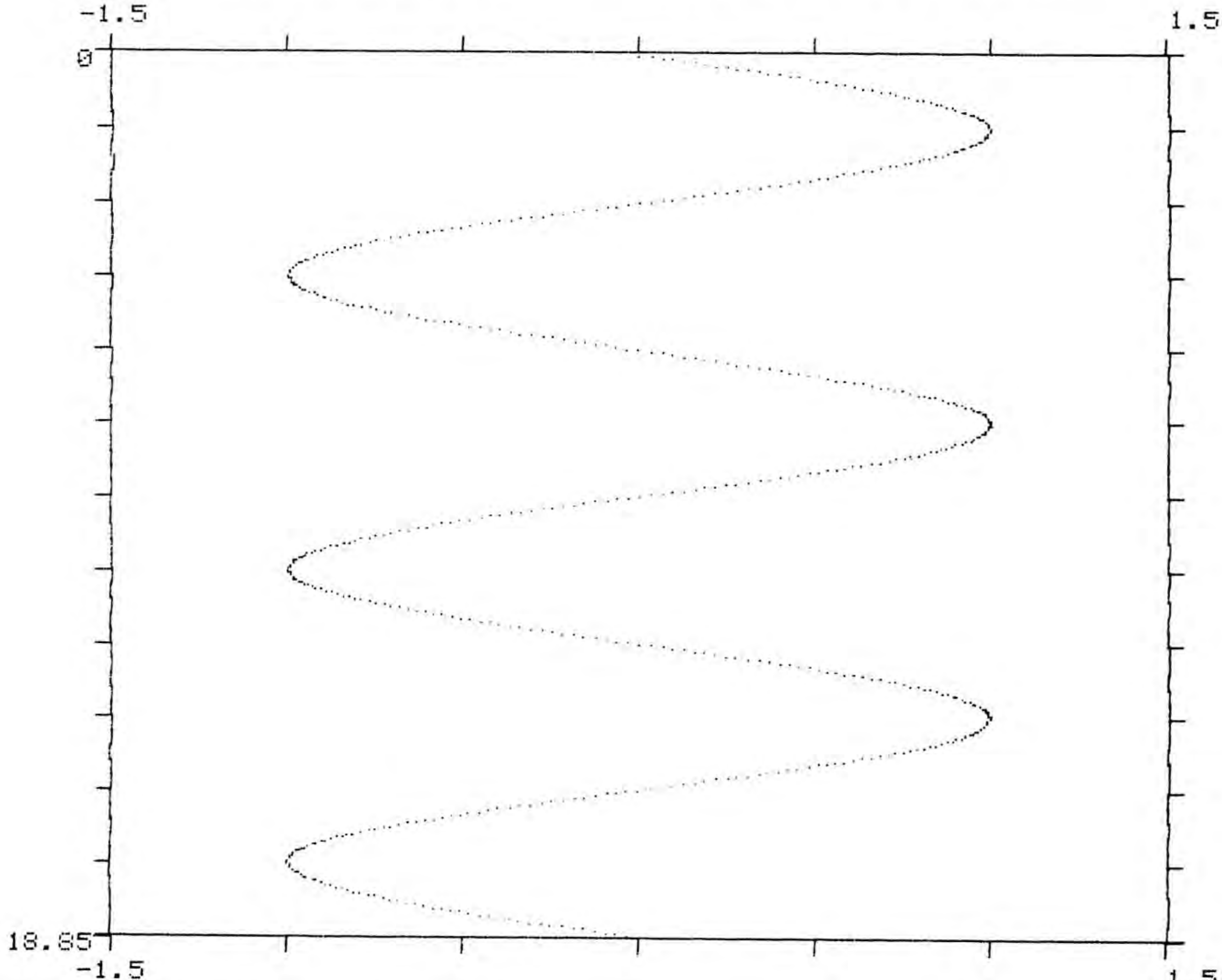
Summary

It's still distinctively a PET/CBM style machine. It retains its friendly personality, and the new features look good.

The 80 character line screen will make many new applications possible — especially those involving columns of figures. Basic hasn't changed at all, and most changes are gentle, so as not to obsolete old machines or programs — or old programmers, for that matter. ©

Plotting With The 2022 Printer

John Winn Department of Chemistry University of California



PLOT OF Y=SIN(X) FOR X= 0 *π TO 6 *π STEP .01 *π
PARAMETERS FOR THIS GRAPH...
NY= 60 NX= 60 YT= 10 XT= 5
SAMPLE PLOT

In the January/February issue of **COMPUTE**, Len Lindsay mentioned using the Commodore Model 2022 Tractor Feed Printer for plotting applications. He pointed out that the ability to vary the line spacing in this printer allows a high degree of vertical resolution. In a somewhat unrelated article in the same issue, W.M. Bunker showed how the user-defined character, CHR$(254), could generate attractive lower case letters with descending tails. He also mentioned the plotting possibilities of this printer. This article puts both ideas together -- the variable line spacing and the user-defined character -- to produce a plotting subroutine of surprisingly high resolution.

The logic behind this subroutine (which is actually a set of subroutines) is to increase resolution from that given by the height of a line and the width of a space to that of a single printer matrix dot. In the way I will describe the subroutine, I will have in mind a graph of computed values of some function, such a Y = SIN (X). In the second part of the article, I will show how data values can be plotted point by point, how bar graphs could be generated, and how more generalized graphics could be done.

Think about how you would graph the function Y = SIN (X) by hand. You would decide first which direction on your graph paper would be in the X direction, and which the Y direction. For us, the X direction will increase down the page as the printer paper advances, and the Y direction will increase across the page, in the direction of the print head motion. Normally, you would hold the graph paper so that X increases from left to right and Y increases from bottom to top. On the printer plot, this orientation is achieved by tilting your head to the right - the plot will be rotated from the usual orientation as it is produced by the printer.

Your next choice in plotting by hand is to pick the minimum and maximum X and Y values to be spanned by the graph. Let's call these values XN, XX, YN, and YX, respectively, in the program. You would then decide how frequently to mark off the two axes with tick marks, and how big to make the graph (how much of the graph paper page to use). For the printer, we can choose the size of the graph by specifying the number of columns wide for the Y axis (variable NY) and the number of rows long for the X axis (variable NX). The size of the page limits NY to a maximum of about 60 for a standard 8 1/2'' x 11'' page. The X axis can be as long as you want (until you run out of paper!), but at the line spacing used in the program, one has about 120 lines (rows) per page. Tick marks are specified by stating the number of columns per tick (variable YT) and the number of rows per tick (variable XT).

Your final step in making the graph is to choose a starting value for X, a final value for X, and an increment for X. The smaller the increment, the more points you will have to plot. You then generate values for Y = SIN (X) at each X value, and put a point on your graph paper at each (X,Y) coordinate. The heart of the plotting subroutine does exactly the same thing. Given X and Y pairs of values, the subroutine figures out which row and column they lie in, and turns on the dot in that space which corresponds to that (X,Y) pair. On the printer, the spacing between lines is set so that each row is six dots tall. The column to column spacing is also six dots, giving a resolution of about 1/60''. (This is comparable to early digital plotters with 0.01'' resolution!)

Now look at the subroutines. The first one (lines 60000-60400) is called via GOSUB 60000 once your main program has established the values for scaling and size mentioned above (NY, NX, YT, XT, XN, XX, YN, and YX). First, it opens files for its own communication with the printer. Secondary addresses 0, 5, and 6 are needed to print, set the special

```
100 REM INITIALIZE PLOT SIZE VARIABLES
110 INPUT "HOW MANY COLUMNS (1 TO ¬
    ¬60)";NY
120 INPUT "HOW MANY COLUMNS PER TICK";YT
130 INPUT "HOW MANY ROWS (120 PER ¬
    ¬PAGE)";NX
140 INPUT "HOW MANY ROWS PER TICK";XT
150 INPUT "X MIN AND X MAX";XN,XX
160 INPUT "Y MIN AND Y MAX";YN,YX
170 INPUT "X START AND END (UNITS OF    ¬
    ¬)";X1,X2
180 INPUT "X INCREMENT (UNITS OF   )";XS
200 REM CALL INITIALIZATION SUBROUTINE
210 GOSUB 60000
300 REM DEFINE THE FUNCTION TO BE ¬
    ¬PLOTTED POINT BY POINT
310 FOR X=X1*   TO X2*   STEP XS*
320 Y=SIN(X)
330 REM CALL MAIN SUBROUTINE FOR EACH ¬
    ¬POINT
340 GOSUB 61000
350 NEXT
400 REM CALL THE PLOT TERMINATION ¬
    ¬SUBROUTINE
410 GOSUB 60500
500 REM ADD A TITLE TO THE BOTTOM OF ¬
    ¬THE GRAPH
510 OPEN1,4,0
520 PRINT#1,"PLOT OF Y=SIN(X) FOR ¬
    ¬X="X1"*   TO "X2"*   STEP "XS"*   "
600 REM PRINT OUT REMAINING GRAPH ¬
    ¬PARAMETERS
610 PRINT #1,"PARAMETERS FOR THIS GRAPH ¬
    ¬..."
620 PRINT#1,"NY="NY" NX="NX" YT="YT" ¬
    ¬XT="XT
999 END
```

SAMPLE DRIVING PROGRAM

```
61000 IF (Y<YN)OR(Y>YX)OR(X<XN)OR(X>XX) ¬
      ¬THEN RETURN
61100 NC% = 1+(Y-YN)/(YX-YN)*NY%
61110 NB% = (NC%-1)/6 : NC% = NC%-6*NB%
61120 NR% = 1+(X-XN)/(XX-XN)*NX%
61130 NL% = (NR%-1)/6 : NR% = NR%-6*NL%
61140 IF NL% <> LL% OR NB% <> LB% THEN ¬
      ¬GOSUB 61200
61150 LL%=NL% :LB%=NB% :C%(NC%)=C%(NC%) ¬
      ¬OR 2^(6-NR%) : RETURN
61200 Q$ = "" : FOR Q=1 TO 6 :Q$=Q$+CHR$
      ¬(C%(Q)) : C%(Q)=0 : NEXT :
      ¬ PRINT#11,Q$
61210 PRINT#10,TAB(LB%+11)SC$CR$
61220 IF NL%=LL% THEN RETURN
61230 IF (LL%+1)/XT% = INT((LL%+1)/XT%) ¬
      ¬THEN 61250
61240 PRINT#10,TAB(10)"Y" TAB(NY)"T" :
      ¬ GOTO 61260
61250 PRINT#10,TAB(10) " " TAB(NY) "L"
61260 IF NL% <> LL%+1 AND NL% <> LL% ¬
      ¬THEN GOSUB 61300
61270 RETURN
61300 FOR Q=1 TO NL%-LL%-1
61310 IF (LL%+Q+1)/XT%=INT((LL%+Q+1)/   ¬
      ¬XT%) THEN 61330
61320 PRINT#10, TAB(10) "Y" TAB(NY)  ¬
      ¬"T" : GOTO 61340
61330 PRINT#10, TAB(10) " " TAB(NY) ¬
      ¬"L"LIST
61340 NEXT : RETURN
```

MAIN PLOTTING SUBROUTINE

Note: These listings were done with a CBM 32B computer which has a slightly different graphics character set and no key for PI. In the Sample Driving Program, Lines 170, 180, 310, and 520 should have the symbol for PI inserted in the spaces left blank. In the Main Plotting Subroutine, lines 61250 and 61330 have a blank in quotes which should be ., i.e. the "opposite" graphics character to the L.

```
60000 OPEN10,4,0:OPEN11,4,5:OPEN12,4,6:IFNY>60THENNY=60
60050 XT%=XT:YT%=YT:LL%=0:LB%=0:NX%=6*NX-1:NY%=6*NY-1
60100 CR$=CHR$(141):SC$=CHR$(254):Q=(NY%+1)/6-LEN(STR$(YN)):IFQ<0THENQ=0
60150 PRINT#10:PRINT#10,TAB(9)YN;TAB(Q)YX:PRINT#10,"                    ⅃";:FORQ=1TONY
60200 IFQ/YT%=INT(Q/YT%)THEN60300
60250 PRINT#10,"_";:GOTO60350
60300 PRINT#10,"⅃";
60350 NEXT:PRINT#10,"_":PRINT#12,CHR$(14):Q$=STR$(XN):Q$=RIGHT$(Q$,LEN(Q$)-1)
60400 PRINT#10,TAB(10-LEN(Q$))Q$;CR$;:RETURN
```

INITIALIZATION SUBROUTINE

character, and set the line spacing, respectively. Variables needed by the subroutine are computed or initialized, the YN and YX values are printed outside the graph, the Y axis is printed, along with tick marks, using graphic symbols, the line spacing is set via PRINT # 12, CHR$(14), and the XN value is printed, setting up the first row for plotting. The first subroutine returns at this point.

Your main program then computes (X,Y) pairs, one at a time, *in increasing order of X*. After each pair is computed, your program goes to the main subroutine via GOSUB 61000. This subroutine first checks to see if the computed point really lies within the graph limits. If not, it simply returns. Lines 61100 to 61130 find out which row (NR%) of matrix dots contain the point, (X,Y). These lines also find out which printer line (NL%) we are dealing with, and how many blank spaces (NB%) preceed the one containing (X,Y). The special character is built up in the array C%. Whenever a dot appears in a new row or column, the current special character is printed (lines 61200 and 61210) without a carriage return/line feed. If we are suddenly on a new line, that condition is sensed (via the variable LL%, standing for "last line") and the X axis borders (at both ends of the graph and with tick marks if required) are printed to finish the line and move on to the next. If it is recognized that several blank lines are required between the last point and the current one, these are generated as well (lines 61220 through 61340 do all this). The subroutine can RETURN to get the next (X,Y) values at several points in this scheme.

Finally, when all the (X,Y) pairs have been computed, your main program calls the termination subroutine via GOSUB 60500. This routine forces a point to be plotted at (X = XX, Y = YX) to guarantee that any remaining blank lines appear at the "end" of the plot. The final X axis label, XX, is printed out, along with the "bottom" Y axis border (lines 60550-60750). The line spacing is graciously reset to the default value with the PRINT # 12, CHR$ (24) statement. Finally, YN and YX values are printed once more, the files are closed, and the subroutine returns.

The graph is now complete, as far as these subroutines are concerned. Any graph titles or other notations you may want can be printed before plotting (before the GOSUB 60000) or after plotting (on return from the termination subroutine).

A sample driving program is shown to illustrate these instructions. The REM statements point out what is happening step by step. Of course, the values set by INPUT statements could be computed directly in many cases, as long as they are established before plot initialization. The final plot produced by this program is shown as well. Note that the limiting factor in the plot quality is the mechanical error in print head motion. The print head does not return to exactly the same point with each carriage return. That's why the X axis is a bit wiggly, at least on my printer.

The speed at which a plot is produced depends on two factors. The first is the computation time needed to generate (X,Y) points themselves. The second is quite variable and depends on the shape of the graph and the X increment chosen. Each print line containing at least one plotted point must be printed twice - once for the point, without a line feed, and once for the X axis borders, with a carriage return/line feed. If several plotted points occur on the same print line in a variety of print columns, the line will be overprinted several times, since the special character can be used in only one way in any single PRINT statement. The example graph took about 5 minutes to plot all 600 points.

A few general comments are in order before turning to ways to extend the subroutine. First, integer variables are used in many places to force truncated integer arithmetic without specifying the INT function. An exception occurs in lines 60200, 60600, 61230 and 61310 where INT is used to sense the need to print a tick mark. Also notice the use of the logical OR in line 61150 to turn on the bits that set the special character. (The 2022 instructions describe how to set this character dot-by-dot.) The OR is needed rather than a simple addition in case the same matrix dot is turned on by more than one (X,Y) pair. The OR will not change a bit already set, while addition will, in general.

Since this subroutine package is meant to be used by a variety of programs, it has a self-contained file structure, opening and closing what it needs

```
60500 X=XX:Y=YX:GOSUB61000:GOSUB61200:GOSUB61250
60550 Q$=STR$(XX):Q$=RIGHT$(Q$,LEN(Q$)-1):PRINT#10,TAB(10-LEN(Q$))Q$"┐";
60600 FORQ=1TONY:IFQ/YT%=INT(Q/YT%)THEN60700
60650 PRINT#10,"‾";:GOTO60750
60700 PRINT#10,"┐";
60750 NEXT:PRINT#10,"‾":PRINT#12,CHR$(24)
60800 Q=(NY%+1)/6-LEN(STR$(YN)):IFQ<0THENQ=0
60850 PRINT#10,TAB(9)YN;TAB(Q)YX
60900 CLOSE10:CLOSE11:CLOSE12:RETURN
```

PLOT TERMINATION SUBROUTINE

independently of the main program. For any particular program, this feature could be removed and all files could be controlled by the main program.

Suppose you want to plot rather widely spaced points, rather than closely spaced points along a curve. Widely spaced single dots are difficult to see. Or suppose two or more different sets of data have to be plotted on the same graph. What is needed in both cases is some way to plot larger and different types of symbols instead of points. One way of plotting symbols is discussed below, but it is not foolproof.

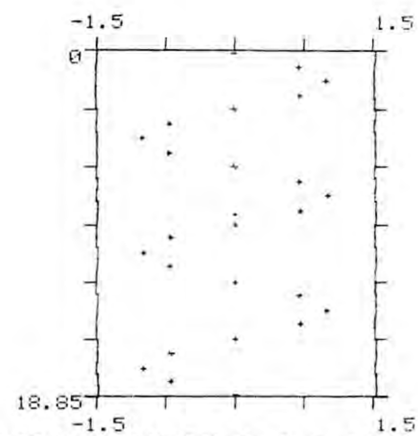
The simplest way to make an arbitrary symbol (a plus, x, square, or whatever) is to begin by defining the dot-to-dot distances, DX and DY, with the statements

DX = (XX-XN) / (6*NX)
DY = (YX-YN) / (6*NY).

Then, after computing each (X,Y) pair, call a new subroutine (let's say it is called by GOSUB 62000) in line 340 of the example program instead of the main (61000) subroutine. The new subroutine constructs the symbol you choose, centered about the correct (X,Y) point. In the sample symbol subroutine, the way to plot a small +, three dots wide, is shown. Other symbols can be created in many shapes and sizes, but all are subject to problems if the X values are too closely spaced. If the symbols overlap from one print line to the next, and the next symbol wants to extend back to the previous print line, you have troubles. The printer won't back up. The only safe way around this problem is to go through your data point by point, and create a new data set representing all the dots in all the symbols for all the original data. This new data set has to be arranged in order of increasing X values, but it can then be plotted point-by-point using the original scheme without problems.

More complex graphics - high resolution pictures of arbitrary shapes - will always have to face the problem of one-directional paper motion. A line-by-line "raster scan" is possible, but 60 print columns contain 60x6x6 = 2160 matrix dots per print line. Turning on the correct dots and scanning all the possibilities could take an intolerable amount of time.

For many applications, a bar graph (histogram) is more useful than a point-by-point or line graph. Bar graphs are easily generated using the built-in PET graphics characters, as long as the bars go "across the page" rather than "up and down." The line spacing must be set to the same close spacing used here, or to a closer spacing, if half-size graphic symbols (and lower resolution) are acceptable. The resolution is lower because half-size symbols don't come in as many widths as full size symbols. Even with the built-in graphics symbols, the bar graph won't look perfect, since small gaps in the bar graph silhouette will exist with these symbols. The best approach would use a mixture of built in graphics plus a special character at the end of the bar to give a continuous silhouette.



PLOT OF Y=SIN(X) FOR X= 0 *π TO 6 *π STEP .25 *π
PARAMETERS FOR THIS GRAPH...
NY= 20 NX= 30 YT= 5 XT= 5

SAMPLE SYMBOL PLOT

```
62000 REM ALTERNATE SUBROUTINE FOR PLOTTING DATA AS A SMALL +
62010 REM DX AND DY MUST HAVE BEEN DEFINED BY THE MAIN PROGRAM
62020 X=X-DX:GOSUB 61000
62030 X=X+DX:Y=Y-DY:GOSUB 61000
62040 Y=Y+DY:GOSUB 61000
62050 Y=Y+DY:GOSUB 61000
62060 Y=Y-DY:X=X+DX:GOSUB 61000
62070 REM RESTORE X TO ORIGINAL VALUE BEFORE RETURNING
62080 X=X-DX
62090 RETURN
READY.
```

SAMPLE SYMBOL SUBROUTINE

©

UN-NEW

Brett Butler

Have you ever had that 'sinking feeling' after you typed in 'NEW' and suddenly realized that you just wiped out your latest program and had neglected to save it?

We all know that a 'NEW' *erases* a program from memory right. . .?

Well not exactly, as far as PET BASIC is concerned it has, but the old program is still actually in memory, only the variable pointers and the first 'line link' of the program chain have been destroyed. Now, as long as you have not input a new Basic line or a direct statement using a variable, 'UN-NEW' will recover that lost program, by rebuilding the chain and restoring the variable pointers.

'UN-NEW' is a machine language program which resides in the '2nd cassette' buffer occupying only 62 bytes ($033A-$0377).

The assembly listing below shows that we start searching from the beginning of Basic Text ($0405) for a single zero byte indicating the end of the first Basic line; when found we increment one location more and store this address in locations $0401 + $0402, our first 'line link'; this restores the chain of the program and now only requires that we find the End of Basic and set the variable pointers. Since all Pet Basic programs terminate in three (3) zero bytes, our search begins at the Start of Basic ($0401) and keeps counting until the third consecutive zero byte is found; when this is done we increment one location past and store this address in all the variable pointers.

To use: load "UN-NEW/SYS826" and type 'SYS826' (RETURN) PET will respond with READY, and that's it. Your program has been recovered.

To save: For UPGRADE ROM's, call the MLM and enter the HEX coding at the locations shown and then type

.S "UN-NEW/SYS826", 01,033A,0378

For original ROM's, load the MLM from tape, enter the HEX coding as per above, ensuring that values $2A thru $2F are replaced with $7A thru $7F respectively then type

.S 01, "UN-NEW/SYS826", 033A,0378

'UN-NEW' is short, simple and easy to save. Hopefully, you will never have to use it but if you need it just once, its well worth the effort to save before hand.

```
                    ;FOR OLD ROMS
                    ;REPLACE LOCATIONS $2A THRU $2F WITH
                    ;$7A THRU $7F RESPECTIVELY
033A  A9 04         START          LDA  #$04      ;START
033C  85 2E                        STA   $2E      ;SEARCH
033E  85 2F                        STA   $2F
0340  A0 00                        LDY  #$00
0342  20 6F  03     MORE           JSR   SCAN
0345  D0 FB                        BNE MORE
0347  20 6F  03                    JSR   SCAN
034A  A5 2E                        LDA   $2E      ;INSERT
034C  8D 01  04                    STA   $0401    ;LINK
034F  A5 2F                        LDA   $2F
0351  8D 02  04                    STA   $0402
0354  A9 01                        LDA  #$01
0356  85 2A                        STA   $2A
                    ;NOW LOOK FOR END OF BASIC
0358  A2 03         NOTEND         LDX  #$03
035A  20 6F  03     LOOK           JSR   SCAN
035D  D0 F9                        BNE   NOTEND
035F  CA                           DEX
0360  D0 F8                        BNE   LOOK
0362  20 6F  03                    JSR   SCAN
                    ;SET ALL VARIABLE POINTERS
0365  A2 03                        LDX  #$03
0367  B5 2C         FILL           LDA   $2C,X
0369  95 2A                        STA   $2A,X
036B  CA                           DEX
036C  10 F9                        BPL   FILL
036E  60                           RTS
                    ;
036F  E6 2E         SCAN           INC   $2E
0371  D0 02                        BNE   OVER
0373  E6 2F                        INC   $2F
0375  B1 2E         OVER           LDA  ($2E),Y
0377  60                           RTS
```

©

Review

PH-001 2114 RAM ADAPTER

Harvey B. Herman

Optimized Data Systems
P. O. Box 595
Placentia, CA 92670

$24.95 Assembled with one 2114 and two sockets
$13.95 Kit with parts (no 2114s)
$ 8.95 Printed circuit board only

Many "old" 8K PETs used 22 pin 6550 RAM memory chips on their circuit boards. These chips, at one time, may have been difficult to obtain and even today cost more than competitive versions (e.g. 2114s). Substitution of 2114s (at least half the cost of 6550s) has heretofore not been possible as the chips are not pin compatible. The circuit board adapter reviewed here is intended to allow one to substitute 2114s for 6550s on the "old" PET circuit board.

I found the adapter easy to install and in my tests performed as expected. It allows substitution of up to 8-6500s (of a total of 16) with just one circuit board. Two adapters may be used to replace all 6550 chips desired. The adapter can be installed immediately with the original 6550s before any failure occurs - a nice feature.

I can see a real value for this product if you are having trouble obtaining 6550 memory chips. Almost anything to keep our PETs healthy is worth it. However, if you have a reliable source for the 6550 chip then the adapter will only be cost effective after replacing 2 or 3 chips. If one had several PETs this might not take an excessive amount of time. I would consider this product carefully if you only own one PET and have no trouble ordering the original memory chips. With these reservations, I recommend the adapter to all "old" 8K PET owners.

©

Disk ID Changer

Rene W. Poirier

When I initially started with the 2040, and would "NEW" a diskette, the name and ID number were grabbed out of the sky. No system of naming or numbering came to mind at that time. Later, I found that this system of no system proved to be a nuisance in building a library that made sense.

This program allows one to change either the name or the ID number on an in-use diskette without disturbing the files on the diskette. Thus, a "working disk" can be easily renamed to a "library disk", etc.

It also allows the ID of a duplicated or "backup" diskette to be changed to a different ID number than the original. Commodore mentioned, in a recent bulletin to dealers, the possibility of errors in creating the BAM when initializing a diskette with the same ID number as the diskette just removed from that drive. Changing the ID number on backup diskettes with this program, or by parts of it being incorporated into a duplication routine, will prevent this problem from occuring.

```
1 REM ***  DISK ID CHANGER
2 REM ***  BY RENE W. POIRIER
3 REM ***  BERLIN, N.H.
4 :
5 :
6 :
7 :
10 OPEN9,0,0:PK=PEEK(59468):POKE59468,12
20 PRINT"ĥ↓↓↓THIS PROGRAM IS TO CHANGE THE ID         CHARACTERS ON A ¬
     ¬DISK";
30 PRINT"ETTE, OR CHANGE THE DISK NAME
40 PRINT"↓USE WITH CAUTION!"
50 PRINT"↓↓ON WHICH DRIVE SHALL WE PERFORM THE     CHANGE? (0/1) ɾ";:
60 INPUT#9,DV$:PRINT:DV=VAL(DV$):IFDV<0ORDV>1THENPRINT"ɾ↑↑↑↑";:GOTO50
65 IFDV=OANDDV$<>"0"THENPRINT"ɾ↑↑↑↑";:GOTO50
70 PRINT"↓DO YOU WISH TO CHANGE"TAB(44)"ɾ1r̂) ID CHARACTERS
75 PRINTTAB(4)"ɾ2r̂) DISK NAME
80 PRINT"  ENTER (1/2)      ɾ";:INPUT#9,Q$:PRINT:Q=VAL(Q$):IFQ<1ORQ>2TH
     ¬EN70
90 OPEN15,8,15:PRINT#15,"I"+DV$
100 OPEN1,8,3,"#":PRINT"↓↓↓↓"
110 PRINT#15,"U1:3";DV;",18,0"
120 ONQGOTO140,500
140 PRINT#15,"B-P:3,162
150 GET#1,A$,A1$:PRINT"THE PRESENT DISK ID IS: ɾ"A$A1$
160 IFA$<>"B"THENA$="B":
170 A1$=CHR$(ASC(A1$)+1)
180 PRINT"THE NEW DISK ID I HAVE CHOSEN IS: ɾ"A$A1$:NID$=A$+A1$
190 PRINT"IS THIS ACCEPTABLE? (Y/N) ɾ";:INPUT#9,Q$:PRINT
200 IFQ$="Y"THEN230
210 IFQ$<>"N"THEN190
220 PRINT"↓WHAT DISK ID DO YOU DESIRE? ɾ";:INPUT#9,NID$:PRINT
225 IFLEN(NID$)<>2THENPRINT"ɾNOT ACCEPTABLE AS A DISK ID!":GOTO220
230 PRINT"SHALL I SEND THAT TO THE DISK? (Y/N) ɾ";
235 INPUT#9,Q$:PRINT:IFQ$="Y"THEN400
240 IFQ$<>"N"THEN220
250 CLOSE1:CLOSE15:POKE59468,PK:END
400 PRINT#15,"B-P:3,162"
410 PRINT#1,NID$;
420 PRINT#15,"U2:3";DV;",18,0"
430 I$="I"+MID$(STR$(DV),2):PRINT#15,I$
440 PRINT"ĥ↓↓DISPLAY THE DIRECTORY TO SEE IF THE DISKID HAS BEEN ¬
     ¬CHANGED";
450 PRINT" CORRECTLY.
460 PRINT"↓I'M ASSUMING THAT THE WEDGE HAS BEEN INITIALIZED
470 PRINT"↓↓↓↓>$"MID$(STR$(DV),2):PRINT"↑↑↑↑"
480 POKE158,1:POKE623,13
490 END
500 PRINT#15,"B-P:3,144
510 DNA$="":FORI=1TO16:GET#1,A$:DNA$=DNA$+A$:NEXT
520 PRINT"↓THE PRESENT DISK NAME IS: ɾ"TAB(44);DNA$
530 PRINT"↓DO YOU WISH TO CHANGE IT? (Y/N) ɾ";:INPUT#9,Q$:PRINT:
     ¬IFQ$="Y"THEN600
540 IFQ$<>"N"THEN520
550 CLOSE1:CLOSE15:PRINT"↓↓↓END":POKE 59468,PK:END
600 PRINT"↓ENTER NEW DISK NAME"TAB(40)"LIMIT TO 16 CHARACTERS ɾ";
605 INPUT#9,NDN$:PRINT:
610 IFLEN(NDN$)>16THENPRINT"↓ɾNOT ACCEPTABLE -- TOO LONG":GOTO600
620 NDN$=LEFT$(NDN$+"                ",16)
630 PRINT"↓↓SHALL I SEND ɾ"NDN$
640 PRINT"TO THE DISK ON DRIVE "DV"? (Y/N) ɾ";:INPUT#9,Q$:PRINT:
650 IFQ$="Y"THEN700
660 IFQ$<>"N"THEN630
670 GOTO550
700 PRINT#15,"B-P:3,144"
710 PRINT#1,NDN$;
720 PRINT#15,"U2:3";DV;",18,0"
730 GOTO430
```

©

SHIFT WORK

Jim Butterfield, Toronto

The SHIFT keys on the PET are pretty straightforward, right? Hold either one down while you hit another key, and you get the key's shifted equivalent: upper or lower case or a graphic. Not much to be said there.

Well, maybe one or two things. . .

Shifted Return

RETURN does two jobs: it takes you to the start of the next line, and it executes the line you're leaving.

Sometimes you don't want to execute the line. You're just drawing a picture on the screen. When you hit RETURN, the computer will take that part of the Klingon attack vessel you've just drawn and try to execute it as a Basic Command; you get ?SYNTAX ERROR, which doesn't help your picture much.

Other times, you have a Basic line, but you don't want to execute it yet. Maybe you've got a little muddled up with the programmed cursor, and every time you try to back up the cursor to fix things, you get another unwanted graphic. You don't want to press RETURN and enter this botched line into your program before you have a chance to fix things up.

Just hold down SHIFT as you press RETURN and you'll go to the next line without trying to execute what you've just done.

Shifted Space

When you press SPACE, the PET prints a space. When you hold down the SHIFT key and press SPACE, the PET prints a space. Same thing. The shifted SPACE, however, is a different character on the screen. Looks the same, but it's not a true space. How can you use this? Here's one very handy application. Suppose you want to do an INPUT and don't want the user to accidentally stop the program by typing RETURN without input. Shifted-space will do the trick. Try this tiny program:

```
10 INPUT " (see note below) ";X$
20 PRINT "THANK YOU.":GOTO 10
```

Here's what to put between the quotes on line 10. After you type the first quote mark, hold down the shift; type three spaces; type three cursor left characters. Now release the shift and complete the line, starting at the second quote mark.

I call this program ABUSE. After a hard day at the computer, you can put this one in, and proceed to call it every name under the sun. It will thank you and ask for the next insult.

The interesting thing is that the program won't stop if you press RETURN without input. That invisible shifted-space that you have printed to the right of the question mark is a genuine input character. If you don't write over it with your own information, it will be accepted as input, and the program won't stop. Instead, it will humbly thank you .. for nothing.

Pseudo-shifted characters

There's a group of characters that you can't input via keyboard/screen but which are useful in certain types of file handling. The shifted RETURN is also useful in this application.

Here's the problem. When you use the INPUT# statement for receiving data from tape or disk, the input procedure stops on three characters: comma, colon, and RETURN.

This is annoying when you're trying to input names from an address list like DR. ALOYSIUS CHIP, PHD or HORACE SCHMEDLAMP, JR. or have address lines like ATTENTION: MURPHY. The input routine neatly drops the PHD, JR. and MURPHY; and you're left to scratch your head over why the data disappeared.

Relief is in sight. If you can catch the comma or colon before you write it to the file, just change it to its shifted equivalent by adding 64 decimal to the ASCII value. It takes a little more work when you write it' but it saves work and puzzlement when you read it back later. The general technique for a single character is:

```
A = ASC(A$) : IF A = 44 OR A = 58 THEN
A$ = CHR$(A + 64)
```

You can write the re-formed A$ to the file and feel secure that it will come back without trouble and print correctly.

Exactly the same thing can be done with quotation marks. The input routine assumes that ordinary quote marks are there so that they can be removed before you see the string. There's a good reason for this, but it doesn't help you when you really want them to be there and part of the input data. Once again, shift the quote by adding 64 to it. Since quotes are often directly program-generated (rather than input), you can just use CHR$ (98) instead of CHR$ (22).

Occasionally, you may want to input two lines at a time from a file. The shifted-return will do the trick. Oddly enough, you must add 128 to the RETURN to make a shifted return: it's CHR$(141). This always works great if your output goes to the screen. If you're using a printer, however, check it out to make sure it recognizes the shifted-return and does the right thing.

Afterthought

Before I leave you to shift for yourself, try this last little keyboard curiosity. Hold down both shift keys. Now, with your third hand, or nose, or whatever, try pressing a few keys on the left-hand side of the keyboard: Q, A, S, Z.

I don't know why you get the odd characters. There's probably a moral here: two shifts are not better than one? Too many shifts spoil the keyboard? ©

Machine Language Code For Appending Disk Files

Robert H. Wollenberg
Department of Chemistry
Stanford University

The attached machine language routine provides a very useful tool for those who have been frustrated (as I have) at the inability of the current disk operating system (DOS) of the Commodore 2040 Dual Drive Disk to append programs. Although firmware recently introduced by Palo Alto IC's (The Programmer's Toolkit) provides some relief to those who require a convenient appending procedure, it suffers a serious drawback. The system operates only by appending a tape file to a program already in memory. Thus, it becomes necessary to first save program pieces to tape and then reload these in proper sequence using the append command. Since I was reluctant to use this slow tape file procedure, I searched for alternatives involving disk files.

A little investigation of the DOS command, Copy, reveals that this instruction goes a long way toward solving the problem. Basic is stored in memory starting at location $0401. The first two bytes are forward pointers to the next line of code (stored by the usual 6502 convention of low byte/high byte). Locations $0403 and $0404 store the line number for this first line of basic. The ASCII Code for this line is stored beginning at location $0405 and ends with the delimiter zero. The next byte begins the second line of basic and is stored at the location pointed to by the forward pointers at locations $0401 - $0402 described above. By following the forward pointers from line to line, one eventually reaches the end of the basic code. This event is signaled when the forward pointers are zero.

When the DOS Copy command concatenates two disk files, the zero page values indicate that both programs were combined; however, listing the concatenated program reveals only the first program. The second program for all purposes remains invisible to the basic interpreter. This is because the forward pointers of the first program eventually point to the two delimiters, zero, and at this point the end of basic is signaled. The next two bytes are pointers to the start of basic text for the second program as originally saved to disk. If the first line of the second program is moved forward in memory by four bytes, then the last line of the first program will point to the first line of the second program and the linking process will be nearly complete. In order to finish the linking procedure, the forward pointers of the second program must be

```
0000                        * = $033A

033A  A9 FF     START   LDA #$FF       ;SETUP TO FIND PROGRAM DELIMITER
033C  85 11             STA $11
033E  85 12             STA $12
0340  20 2C C5          JSR $C52C      ;ROM FIND LINE # ROUTINE
0343  20 6C 03          JSR ENDBAS     ;CHECK FOR END OF BASIC
0346  F0 21             BEQ DONE       ;YES, EXIT AND PRINT READY
0348  A5 5C             LDA $5C        ;CURRENT LINE
034A  D0 02             BNE SKIP
034C  C6 5D             DEC $5D        ;MOVE BACK ONE BYTE
034E  C6 5C     SKIP    DEC $5C
0350  A9 00             LDA #$00
0352  A0 04             LDY #$04       ;SETUP FOR 4 LOOPS
0354  91 5C             STA ($5C),Y    ;STORE NEW PROGRAM DELIMITER
0356  A9 20             LDA #$20       ;LOAD ASCII BLANK
0358  88                DEY
0359  91 5C     AGAIN   STA ($5C),Y    ;STORE BLANKS
035B  88                DEY
035C  10 FB             BPL AGAIN
035E  20 72 C5          JSR $C572      ;ROM "CLR" ROUTINE
0361  20 42 C4          JSR $C442      ;ROM FIX CHAINING ROUTINE
0364  20 6C 03          JSR ENDBAS     ;CHECK FOR END OF BASIC
0367  D0 D1             BNE START      ;DO NEXT PROGRAM IF MORE
0369  4C 8B C3  DONE    JMP $C38B      ;; EXIT TO BASIC
036C  38        ENDBAS  SEC            ;CHECK FOR END OF BASIC ROUTINE
036D  A5 2A             LDA $2A        ;TOP OF VARIABLES
036F  E5 5C             SBC $5C        ;CURRENT LOCATION
0371  C9 02             CMP #$02       ;DIFFERENCE = 2 IF END
0373  D0 04             BNE NOTEND     ;RETURN IF NOT DONE
0375  A5 2B             LDA $2B        ;CHECK HIGH ADDRESS
0377  C5 5D             CMP $5D
0379  60        NOTEND  RTS
```

```
 0 REM ROBERT H WOLLENBERG
 10 REM PROGRAM TO APPEND USING DISK CONCATENATE COMMAND
 15 DEF FNA(X)=PEEK(X)+256*PEEK(X+1):DEF FNB(X)=FNA(X)-X
 20 I=FNA(40):E=FNA(42):X=FNB(I)
 25 IF I+X>=E-2 THEN PRINT"END OF MEMORY-NO LINK":END
 30 I=I+X:IF FNA(I)<>0 THEN X=FNB(I):GOTO 25
 35 FOR J=I TO I+3:POKEJ,PEEK(J+4):NEXT
 40 FORJ=I+4 TO I+7:POKEJ,32:NEXT: Y=I-1025+4 : REM OFFSET
 45 X=FNB(I)+Y:POKEI+1,INT((I+X)/256):POKEI,I+X-256*INT((I+X)/256)
 50 I=I+X:IF FNA(I)<>0 AND I<E-2 THEN 45
 55 IF FNA(I)=0 AND I<E-2 THEN 35
 60 PRINT"LINK-COMPLETE":END
READY.
```

recalculated to compensate for the relocation of code. This is done conveniently using the ROM chaining routine at $C442.

To use the attached machine code, first use the disk Copy command to concatenate up to four program pieces. For example, to append parts A, B and C to form program D, the following command is executed (after loading the DOS support):

►C0:D = 1:A,1:B,1:C

Next load the machine code and the concatenated program D into memory and type:

SYS826

The programs are linked and the message: "READY." appears.

For comparison, I have written a basic program to link concatenated programs. In this case the linking program must be the first program concatenated. Once concatenated, the new program is loaded into memory and then linked by typing:

RUN

The linking program is then deleted, leaving only the desired appended program. Comparison of these two procedures revealed that the machine language code ran nearly a thousand times faster than the basic code. ©

Mixing Basic and Machine Language

Jim Butterfield, Toronto

It's not too hard to put Basic and Machine Language together. Care is needed, of course, but there's no great mystery.

One of the easiest tricks is to put the machine language program behind the Basic program in memory. Once you've created and saved the package, it may be LOADed and SAVEed without special instructions. There's one thing you need to watch, however: when the packge is complete, you must not change the Basic program. If you do, the machine language part will be moved away from its original location. Your SYS command will take you to the wrong place, and the Machine Language program probably won't work anyway.

The following sample programs use this kind of packaging. Here's how to set them up on your machine:

1. Type in the Basic program completely. Check it carefully, since you won't have the option to change it later.
2. Enter the machine language monitor. If you happen to have an early PET with original ROM (no built-in monitor), you should have previously loaded one of the "high-monitors" that are available. (See Roy Busdiecker's article: *"Relocate PET Monitor Almost Anywhere")*.
3. Double check to ensure that your Basic program hasn't somehow crept up above the machine language area that you plan to use. There are several ways to do this. One is to inspect the Basic memory area and spot the three 00 values that signal the end of the Basic program. Another way is to take a look at the start-of-variables pointer (hex 7C and 7D on original ROMs; hex 2A and 2B on newer ROMs) and make sure it's below the area you are about to work in.
4. Now type in the machine language as shown. Check it closely; a single mistake will cause improper operation.
5. Finally - still in the Machine Language Monitor - save the whole thing from start-of-Basic (hex 0400) to end-of-machine-language -plus one. On the Universal Rom Test program, for example, you'd save from 0400 to 084A.

Now your program is ready. It can be loaded, saved or copied without any special knowhow. Just remember - don't change the Basic part of the program.

Universal ROM test

This program loads into any machine and tells you what kind of ROM you have. It will test ROM repeatedly until you stop it... this makes it good for spotting intermittent errors.

All of the standard ROM sets I know about are there. There are also a couple of experimental Commodore ROM sets included - I had a chance to take a look at them during a recent trade show. These may change by the time Commodore releases them, so don't take them too literally.

```
100 PRINT"ñ↓↓ UNIVERSAL ROM TEST     JIM ¬
     ¬BUTTERFIELD"
110 DATA "Ø11 ORIGINAL",59487,12796,
     ¬51858,6198Ø,58622,7753,6792,-1,-1
120 DATA "Ø19 ORIGINAL",59339,12796,
     ¬51858,6198Ø,58622,7753,6792,-1,-1
130 DATA "UPGRADE PERSONAL",41799,42993,
     ¬64959,88Ø3,38129,43129,23Ø93,-1,-1
140 DATA "UPGRADE BUSINESS",41799,42993,
     ¬64959,88Ø3,38129,43129,23Ø93,-1,-1
160 DATA "DOS PERSONAL I",4Ø596,452Ø1,
     ¬349ØØ,Ø82Ø7,4782Ø,ØØ39Ø,44555,
     ¬4Ø847,44239
170 DATA "DOS BUSINESS I",4Ø596,452Ø1,
     ¬2725Ø,Ø82Ø7,4782Ø,ØØ39Ø,44555,
     ¬4Ø847,44239
180 DATA "DOS 8Ø-CHARS I",4Ø596,452Ø1,
     ¬2113Ø,Ø82Ø7,4782Ø,ØØ39Ø,44555,
     ¬4Ø847,44239
190 DATA "*"
200 DATA 192,2Ø8,224,24Ø,2ØØ,216,248,
     ¬176,184
210 DIMA$(8),V(8,9),A(9),R(9),M(9)
220 READX$:IFX$="*"GOTO29Ø
230 R=R+1:A$(R)=X$
240 FORJ=1TO9:READV(R,J):NEXTJ:GOTO22Ø
290 FORJ=1TO9:READA(J):NEXTJ
300 FORJ=1TO9:POKE1Ø23,A(J):SYS2112
310 R(J)=PEEK(1Ø21)+PEEK(1Ø22)*256
320 NEXTJ:P=7
330 FORJ=1TOR:M(J)=Ø:FORK=1TOP:IFR(K)=V(
     ¬J,K)THENM(J)=M(J)+1
340 NEXTK,J
350 L=-1:FORJ=1TOP:IFM(J)>LTHENL=M(J):
     ¬K=J
360 NEXTJ:IFP=7ANDV(K,8)>=ØTHENP=9:
     ¬GOTO33Ø
370 N=N+1:PRINT"hTEST";N:PRINT"↓↓↓↓";
     ¬A$(K):PRINT
380 FORJ=1TOP
390 IFR(J)=V(K,J)THENPRINT:GOTO41Ø
400 A=A(J):B%=A/16:C=A-B%*16:PRINTCHR$(B
     ¬%+55);CHR$(C+48)
```

```
410 NEXTJ:PRINT
420 PRINT"BAD ROMS:";P-L;"<  ":GOTO300
.
.:  0840 AD FF 03 85 B2 A9 08 85
.:  0848 B5 A9 00 85 B1 85 B3 A0
.:  0850 00 18 71 B1 90 02 E6 B3
.:  0858 C8 D0 F6 E6 B2 C6 B5 D0
.:  0860 F0 8D FD 03 A5 B3 8D FE
.:  0868 03 60 00 00 FF 00 00 FF
```

RAM test

This is a very fast memory test, yet it's quite thorough. It's adapted from the memory test in The First Book of KIM - you can dig out more details there if you're curious.

The coding is a little crowded; I wanted to fit the whole thing into 256 bytes so that the rest of memory would be available for testing.

The program tests memory repeatedly until stopped. Users with a full 32K of memory can input a value of 33 and test screen memory too. That way, they can see the actual test on the screen as it happens.

```
10 INPUT"ĥ↓↓HOW MANY K";K:K=K*4-1
20 POKE185,K:POKE184,5
30 N=N+1:SYS1156:PRINT"ħTEST ";N;
40 J=PEEK(187):IFJ>KGOTO30
50 PRINT"FAILED AT";J*256+PEEK(186)
.
.:  0484 A9 00 A8 85 BA 85 BC A2
.:  048C 02 86 BD A5 B8 85 BB A6
.:  0494 B9 A5 BC 49 FF 85 BE 91
.:  049C BA C8 D0 FB E6 BB E4 BB
.:  04A4 B0 F5 A6 BD A5 B8 85 BB
.:  04AC A5 BC CA 10 04 A2 02 91
.:  04B4 BA C8 D0 F6 E6 BB A5 B9
.:  04BC C5 BB B0 EC A5 B8 85 BB
.:  04C4 A6 BD A5 BE CA 10 04 A2
.:  04CC 02 A5 BC D1 BA D0 15 C8
.:  04D4 D0 F0 E6 BB A5 B9 C5 BB
.:  04DC B0 E8 C6 BD 10 AD A5 BC
.:  04E4 49 FF 30 A1 84 BA 60 32
```

Tape Test

The version given is for Upgrade ROM only. This lets you watch any PET tape and see the kind of signals that are coming in from it.

I had hoped that this program would solve head alignment problems once and for all. It doesn't quite make the grade, since in my opinion it's not sufficiently sensitive to slight alignment changes. Even so, you will find the program instructive.

```
100 PRINT"ĥTAPE TEST # JIM BUTTERFIELD"
110 POKE59468,12
120 PRINT:X$="LEADER":GOSUB500
130 X$="DATA":GOSUB500
140 X$="ERROR":GOSUB500
150 INPUT"TAPE UNIT";T
160 IFT>2ORT<1GOTO150
170 POKE212,T
180 SYS(1280):END
500 PRINT" UCCI"
510 PRINT" B  H ~  ";X$
520 PRINT" JFFK"
530 RETURN
READY.
.
.:  0500 20 12 F8 78 A6 D4 CA F0
.:  0508 15 CE 13 E8 A9 90 8D 4E
.:  0510 E8 AD 40 E8 8E FA 00 29
.:  0518 EF 8D 40 E8 10 0B EE 11
.:  0520 E8 A9 34 8D 13 E8 8D F9
.:  0528 00 A9 6E 8D 90 00 A9 05
.:  0530 8D 91 00 58 20 F0 F8 2C
.:  0538 13 E8 10 F8 A2 02 A0 00
.:  0540 A9 20 95 B8 B5 B1 F0 06
.:  0548 94 B1 A9 A0 95 B8 CA 10
.:  0550 EF A5 B8 8D 7A 80 A5 B9
.:  0558 8D F2 80 AD 6A 81 06 BA
.:  0560 69 00 29 1F 8D 6A 81 20
.:  0568 29 F7 10 C7 30 C5 20 7A
.:  0570 05 2C 40 E8 2C 10 E8 4C
.:  0578 E4 E6 AE 49 E8 AD 48 E8
.:  0580 EC 49 E8 D0 F5 A0 FF 8C
.:  0588 48 E8 8C 49 E8 E0 FC 90
.:  0590 08 E0 FF D0 07 C9 50 90
.:  0598 0B E6 B3 60 E0 FE D0 10
.:  05A0 C9 60 90 0C A5 CC 29 FC
.:  05A8 F0 03 E6 B1 60 E6 CC 60
.:  05B0 A9 00 85 CC E6 B2 60 00
```

Leader Write

This is for Upgrade ROM only. It writes continuous "leader" (sometimes called "shorts") to tape. It's useful, in conjunction with Tape Test, in checking out various brands of tape for data quality.

```
100 PRINT"ĥ↓  r WRITE LEADER TAPE "
110 PRINT"↓ #   JIM BUTTERFIELD"
120 PRINT"↓ THIS PROGRAM WRITES A ¬
    ¬CASSETTE TAPE"
130 PRINT"WITH 'LEADER' SIGNAL."
140 PRINT"↓ THE CASSETTE TAPE SO PRODUCED ¬
    ¬MAYBE"
150 PRINT"USED WITH 'TAPE TEST' TO EITHER: ¬
    ¬"
160 PRINT"  --CERTIFY THE TAPE AS OK;"
170 PRINT"  --ALIGN TAPE HEADS OF THE ¬
    ¬OTHER CASSETTE"
180 PRINT"     UNITS. IN THIS CASE, ¬
    ¬ BE SURE"
190 PRINT"     THAT YOU ARE WRITING ON A"
200 PRINT"     PRECISELY ALIGNED TAPE ¬
    ¬UNIT.↓"
210 SYS1472:END
READY.
.
.:  05C0 A9 01 85 D4 20 47 F8 A9
.:  05C8 70 8D C3 00 78 A9 A0 8D
.:  05D0 4E E8 A2 08 20 9B FC A9
.:  05D8 02 85 DE A9 34 8D 13 E8
.:  05E0 8D F9 00 8D 49 E8 58 A9
.:  05E8 70 8D C3 00 20 35 F8 F0
.:  05F0 F6 20 7B FC 4C 84 F2 AA
.
```

©

After the Monitor's Moved

Roy Busdiecker

In a previous article ("Relocate PET Monitor Almost Anywhere", May/June 1980) we described a method for placing copies of Commodore's tape version of the PET Monitor in different locations, and how to make the changes that would enable them to work in the new locations. This article is a continuation which provides more information, some of which is essential.

If you have a copy of the PET operating manual which includes a listing of the Monitor, it might be helpful if you have it handy while reading this.

Although each of the newly relocated versions of the Monitor worked precisely as expected, we got some unexpected results when we loaded both the Monitor and a BASIC program into memory at the same time and used the commands (SYS in BASIC, X in Monitor) to switch control back and forth. What seemed to happen was that every time we tried to RUN a BASIC program *after* we had used the Monitor, we would get an OUT OF MEMORY ERROR, even though we knew there was plenty of memory available.

The PET keeps track of memory by storing "pointer" values in an area where your BASIC programs won't mess them up. Their locations and use are summarized in the following table.

LOCATION DECIMAL	HEX	POINTS TO
122,123	7A,7B	Location in memory where BASIC program begins.
124,125	7C,7D	Beginning of area where simple (single-value) variables are stored.
126,127	7E,7F	Beginning of ARRAY storage area.
128,129	80,81	Beginning of unused (FREE) memory.
130,131	82,83	End of string (character) storage.
132,133	84,85	Beginning of string storage (note: strings are "backwards", beginning at the highest numbered locations and moving down toward the lowest!).
134,135	86,87	End of available memory, plus one (i.e., points to first location number for which memory is not attached).

Each pointer may be decoded by adding the value in its low-numbered byte to 256 times the value in its high-numbered byte. For example, an 8k PET usually contains a zero in location 134 and a 32 in location 135, so the end-of-memory pointer points to location 0 + (256 x 32) = 8192. You can prove this by typing

PRINT PEEK(134) + 256*PEEK(135)

then pressing 'RETURN'.

The problem, we discovered, was that every time we did a SYS 23567 (we were using a 23k Monitor ... the 7k version would have been SYS 7183), the "single-variable pointer" (Locations 7C,7D) would be set to a value greater than the end-of-memory pointer. Following the directions in the previous article had required changing the end-of-memory pointer, but they had also told us to do a NEW, which set the "single-variable pointer" down to the same value as "start of BASIC".

It took a little detective work, but we found the answer. In locations (hex) 1C19 through 1C20 of the 7k version (shown in Figure 1), there was a section of machine-language code which reset the "single-variable pointer"!

Figure 1

```
7193    1C19 A91F    LDA #1F
7195    1C1B 857D    STA 7D
7197    1C1D A96B    LDA #6B
7199    1C1F 857C    STA 7C
```

After those instructions were run, locations 7C and 7D (or 124 and 125 in decimal) would hold the values 6B and 1F (decimal 107 and 31). Our pointer-evaluation rule tells us it would then point to location 8043 (107 + 31 * 256).

Why? Well, since the original Monitor (before relocation) went into the area where BASIC programs were usually stored, a command like FOR I = 1 to 3:PRINT "*":NEXT I would clobber the Monitor unless the "simple-variable pointer" (to the area where the value of "I" would be stored) pointed past the end of the Monitor.

That made sense when the Monitor was in the original location. For a relocated Monitor, it no longer makes sense.

There are several ways to deal with the problem. First (and quickest), we could simply use the Monitor to set the values of 7C and 7D back where they belong, each time we use the Monitor. That gets old fast, and we're bound to forget to do it once in a while (don't ask why I'm so sure!).

Second (and better), we could use the Monitor to change the offending code, locations XX19 to XX20 (in the 7k version, substitute 1C for XX . . . other versions correspondingly), to EA (decimal 234) which is a "no operation" or "micro-delay" command. Then save a copy of the modified version, and you're home free. No more OUT OF MEMORY ERROR's if you follow the procedures in the previous article.

Third (best?). we can use that section of the Monitor to change the end-of-memory pointer instead of the "simple-variable pointer". Then we can omit the POKE 135,YY step described in the previous article to protect the relocated Monitor. Figure 2 shows the set of instructions to do that, while Figure 3 show the memory locations to be changed, and the values to insert, using the Monitor. The figures use values for the 7k Monitor, but other versions only require changing the value in location 1C20 from 1C to 2C for 11k, 3C for 15k, and so on.

```
7193    1C19 A91C    LDA #1C
7195    1C1B 8587    STA 87
7197    1C1D A900    LDA #00
7199    1C1F 8586    STA 86
```

```
        0  1  2  3  4  5  6  7
1C18   02 A9 1C 85 87 A9 00 85
1C20   86 A9 43 85 21 D0 12 A9
```

That's the end of the essential information.

The rest is fun.

If you've read this far, then you've probably done some machine-language programming on the PET. Like most of the rest of us, you used a SYS command to transfer control to your routine (did you lose control?), and probably ended your machine language code with an RTS (return from subroutine) command. So far, so goo! The RTS would return you to control of BASIC, since the SYS command caused the PET to receive a JSR (jump to subroutine) machine instruction when it gave control to your routine.

With the Monitor in place, you might assume that the "G" command would operate in the same fashion as the SYS did in BASIC, since the two descriptions are so similar. Unfortunately, from my point of view, they are not the same. That RTS at the end of your machine-language routine will not bring you back to the Monitor, or to BASIC, or to anywhere else that is particularly useful. Not a very satisfactory arrangement!

After more searching than I would have liked, I did find a way to change the Monitor so that 'G' acted more like SYS (most of the time).

In the 7k version, the G command sends control to location 1DEB in the Monitor (shown at 05EB in your Commodore manual). After a series of LOAD and PUSH instructions, you suddenly encounter an RTI (return from interrupt)!

That seemed like a far cry from a JSR, but it became apparent that we could make the routine *act* like a JSR if we located a sensible return address in the Monitor, then pushed that address on the stack *before* the PHA's (push accumulator onto stack) that set up the RTI to execute the 'G'. To add instructions meant making more room so we "slid" the existing code down, moved the excess to the end of the Monitor, and used a JMP (jump, or transfer control unconditionally) to tie the two pieces together.

The modified machine and assembly-language instructions are shown in Figure 4, as they should be for the 7k version. The only changes for versions at other locations are the contents of 1DF0, 1DF2, and 1DFC. They should be obvious.

Figure 4.

```
7662    1DEE 20F21C    JSR 1CF2
7665    1DF1 A91C      LDA #1C
7667    1DF3 48        PHA
7668    1DF4 A942      LDA #42
7670    1DF6 48        PHA
7671    1DF7 A51A      LDA 1A
7673    1DF9 48        PHA
7674    1DFA 4C6B1F    JMP 1F6B

8043    1F6B A519      LDA 19
8045    1F6D 48        PHA
8046    1F6E A51B      LDA 1B
8048    1F70 48        PHA
8049    1F71 A51C      LDA 1C
8051    1F73 A61D      LDX 1D
8053    1F75 A41E      LDY 1E
8055    1F77 40        RTI
```

If you use the Monitor to modify itself, Figure 5 shows the way the pertinent memory locations should look after the changes are made.

Figure 5

```
        0  1  2  3  4  5  6  7
1DE8   20 B2 1C A6 1F 9A 20 F2
1DF0   1C A9 1C 48 A9 42 48 A5
1DF8   1A 48 4C 6B 1F EA A6 1F
```

```
        0  1  2  3  4  5  6  7
1F68   20 20 B7 A5 19 48 A5 1B
1F70   48 A5 1C A6 1D A4 1E 40
```

I've been using the 23k version of the Monitor for some time now, and am quite pleased with it. The space between the end of the Monitor and the end of memory is enough to do something useful with, too. Mine has a relocated version of an "automatic repeating keys" routine originally published elsewhere for use in the second cassette buffer! ©

Fitting Machine Language into the PET

Jim Butterfield

A PET machine language program must co-exist with Basic. You need at least one Basic instruction, even if it's only a SYS command to start the machine language running. You could give the SYS as a direct command from the keyboard; but it's usually much better to put it in as a program line and let the user type RUN.

The Basic program is usually in a predictable place. It will start at address 1025 (hexadecimal 0401), and will occupy memory space upward from there. The end of the Basic program can be spotted by the fact that memory will contain three consecutive zeros.

Your machine language program can go almost anywhere that's free. There are three favorite places for such programs:

--in the cassette buffer(s);

--immediately above the end of Basic;

--at the top of memory.

Each has its advantages and drawbacks. Let's deal with them one at a time.

Cassette Buffer(s)

If you use only cassette number 1, the second cassette buffer is free and available for your use. Its address is hex 033A to 03F9, which gives you 192 locations to play with. If you don't use cassette tape at all, i.e., you use disk, you may use both buffers; this gives you addresses from 027A to 03F9 hex. or 384 locations.

The newest models of PET/CBMs use a small portion of the second cassette buffer. If you have 4.0 ROMs - that is, your machine accepts English-language commands like SCRATCH or CATALOG - you'll need to leave the bottom twenty locations or so free. And if your machine has a TAB key, you must leave a few locations at the top - or your tab stops will change mysteriously. On these newer models, work in the range of 0350 to 03ED and you'll be reasonably safe.

The cassette buffer area is ideal for machine language programs. Except as noted above, it is completely unused for any basic activity. Loading new Basic programs won't affect it. You may easily save machine language and Basic together by using the Machine Language Monitor's .S (Save) command and specifying the address range, from start-of-machine-language to end-of-Basic. The whole thing will be saved; and later, a LOAD from Basic will load everything back, both machine language and Basic.

There are two problems. First, the amount of space is limited. You'll find plenty of room for your first small programs; but as you get more experienced and more ambitious your programs will become too big to fit in this space.

The second problem may not be a problem at all, depending on your objectives. Programs which have been written and saved using the above techniques can't be copied easily. The naive user who performs LOAD and then SAVE will load the whole program - but will save only the Basic part. To copy the program, you need to go to the Machine Language Monitor - and then you need to know the addresses to give for the .S (Save) command. If you prefer to keep your programs private, this can be a useful technique. But if you'd rather see them passed around, this can be a problem - you might get tired of being the only person that can make copies for other people.

Above the end of Basic

This isn't hard to do, once you get the hang of adjusting the start-of-variables pointer (located at 2A and 2B hex in upgrade ROM, of at 7C and 7D in original ROM). After your Basic program is written, this pointer will direct you to the available memory immediately after that program. Put your machine language program there, and then change the pointer so that it indicates a location above both programs - both Basic and machine language. Give the Basic CLR command after you do this, and all the other pointers will line up correctly.

Now you can use a Basic SAVE to record your program. Both Basic and Machine Language will be saved, and they will both load together at a later time.

Using this system, you'll have lots of space for your machine language program if you need it. The composite program can be copied easily: just do a conventional LOAD and SAVE any time you want an extra copy.

There's one drawback to using this system. Once you have set it up, it is difficult to make a change to the Basic program. If you add or delete anything - even a single character - your machine language program will move to a new location. You'll need to change your SYS command or USR vector. Worse, most machine language programs can't be moved without needing changes. You will have to rewrite the program so that it will work properly in its new location.

This is one of the most convenient ways to position machine language. Keep in mind, however, that it reduces your freedom to change the Basic program.

At the top of memory

This is a very convenient place to put relatively permanent machine language programs. It goes in the high end of memory. Since string variables are written in high memory, you must protect this type of program by moving the limit-of-Basic-memory pointer down so that it points below your program. (This pointer is located at hexadecimal 34 and 35 in upgrade ROM, or hex 86 and 87 in original ROM). Once you have done so, the program will take up permanent residence and will usually remain in your machine until you turn the power off.

In this case, the Basic and machine language programs are no longer adjacent in memory. You can't SAVE and LOAD them together. Be sure to LOAD the machine language program first, followed by the Basic program; otherwise some of the Basic pointers will be mixed up, and you'll probably get an ?OUT OF MEMORY error. Remember, too, that the machine language program will take up permanent residence. It won't go away when you load a new Basic program.

Another problem that you'll have to face with this technique is that different machines are fitted with different amounts of memory. A machine language program that sits neatly at the top of an 8K machine will lie smack in the middle of memory on a 16K unit. You would need different versions for different sizes of machine. Keep in mind, once again, that you usually can't move a machine language program to a new location without making changes to it.

It is possible to write a program which finds the top of memory, parks the machine language program up there (wherever it happens to be), makes all necessary corrections, and then moves the limit-of Basic-memory pointer to the proper place. Many programs, such as Supermon and the DOS "wedge" system, do exactly that. It's an advanced technique, however - don't try your hand at it until you feel you're ready. ©

Using Disk Overlays On The PET

Marty Franz

If you've written very large BASIC programs on the PET, there've probably been times when you've wanted to break them into smaller pieces to save storage or make maintenance easier. As an example of this, consider disk utilities: you don't need to have the code necessary to copy files and check for errors loaded into storage along with the code to list the disk directory, since you will only be performing one of those functions at a given time. Instead, it'd be nice to write small "copy files" and "list directory" programs and just load the one needed. To make accessing these programs more convenient, a master program could then decide which one to load based on your input. When the little utility programs were done, they'd reload the master program again to allow another selection.

This approach to program design is called "overlay segmenting". It has several advantages over putting every desired function into a single large program. One is storage efficiency: only the code needed to perform a specific function is loaded in storage at any given time. Another is modularity: when a bug is found you only have to modify a single, small program. Also, if you want to add another function you only have to write the program that implements it and make the master program "aware" of how to load it.

Fortunately, the PET allows one program to load another by letting you put LOAD statements in BASIC programs, It will even suppress the normal SEARCHING FOR and LOADING messages, so the user won't notice that another program is being loaded.

Unfortunately, there's a serious restriction on this: the overlay program must be smaller than the program loading it! This is because the pointers that tell BASIC how large the program is and where the variables are kept aren't reset when a LOAD is done from another program. The Microsoft people have done this so that the new program can access the previous one's variables. But it means that the master program has to be the largest program in our utility package. What if we want to add an "edit files" function? It'd be nice to use all the memory available to us for keeping text while we were editing it. With the master program hogging storage to insure that its over lays will always be smaller, this will severly limit what's available to our editor.

To have a really useful disk overlay scheme, then, we have to get around this restriction on program size. For that, we have to know how BASIC programs are stored. Addresses 42 and 43 on new PETs are a pointer (low and high bytes) to where variable storage starts in a BASIC program. It also tells where the program text ends. If we poke these addresses with larger pointer value, we've effectively increased the size of our program.

So, a way to insure that our program is always bigger than the one we're going to load next is to set the "program end" pointer right before the LOAD statement. What's the largest possible value we can give to is pointer? It's the highest RAM memory address, kept for us at addresses 52 and 53. We want to make it one page (256 bytes) less than that, actually, because BASIC might need interim storage for variables.

When the new program is loaded and begins running, the first thing it will have to do is tell BASIC how big it really is, so that the rest of storage can be freed up for use by its variables. Luckily, the actual length of a program is kept by the LOAD routines at addresses 201 and 202. So, we reset the pointer at address 42 to this value before we do any processing in our overlay.

The only flaw in this scheme is the previous program's variables. BASIC keeps track of where variable storage begins and ends, and it's based on the end-of-text pointer. Each time we mess with this pointer we have to get BASIC to clean up variable storage for us. The CLR statement will do that.

Our overlay scheme is now complete. Whenever we load an overlay, we make our program as large as possible beforehand to avoid the length restriction:

```
996 POKE 42,PEEK(52)
997 POKE 43,PEEK(53)-1
998 CLR
999 LOAD "next",8
```

And, when the overlay begins execution, we first reset the program-end pointer to the actual program size:

```
1 POKE 42,PEEK(201)
2 POKE 43,PEEK(202)
3 CLR
```

The CLR statements clean up the variables for us, and we allow one page less than the largest RAM address for interim storage of BASIC variables.

There are two "gotchas" involved in this overlay scheme, however. The first is that if we want to pass variables between overlays we're out of luck, because the CLRs will clobber whatever was kept in storage. In a future article, I'll discuss a secure way to pass parameters between programs. For now, we can POKE variables into a protected storage area like the second cassette buffer and have them remain intact during the LOAD process. We'll have to do this for the name of the program you're loading if it's going to be kept in a string. That's why we kept a page of storage free for variables when we increased the program's size: After the pointer is reset, we need to fish the name of the program out of protected storage as a string so we can do the LOAD. The sample master program shows how this is done.

The second "gotcha" relates to writing and testing overlays. It's probably a good idea to omit the entry and exit linkage from them while they're being tested. Why? Each time the program is run, it'll reset its end-of-program pointer to the value it had when it was last loaded. This is very bad when you make a lot of changes and the program size is altered dramatically. You could lose all your changes (at best) or mess up your program to the point where you can't do anything with it (at worst). When debugging an overlay, type NEW before loading it, and be sure to save a copy before running it again.

There are many applications for overlays. For example, the master program could use light pen input to select the overlays, simplifying the user's interface to the PET. If the overlays were carefully written to take all their parameters from memory, the master program could read input from a file, and complex functions could be built up from lists of simpler ones: a file could tell the master program to first assemble, then run, a 6502 machine-language program by calling assembler and loader overlays. This is called "batch" or "command language" processing and is done on much larger computer systems. Suffice it to say that with the ability to do overlay segmenting, the power of the PET for serious programming is greatly enhanced.

```
100 REM  SAMPLE MASTER PROGRAM
110 REM  USING PET DISK OVERLAYS
120 REM
130 POKE 42,PEEK(201):POKE 43,PEEK(202)
140 CLR
150 REM  INITIALIZE PROGRAM
160 DF$="-":SI$="  "+DF$+"████"
170 OPEN 15,8,15
180 REM  ASK FOR PROGRAM NAME
190 PRINT:PRINT:PRINT"PROGRAM";SI$;
200 INPUT P$
210 :IF P$=DF$ THEN 190
220 :IF P$="BYE" THEN NEW:END
230 REM  CHECK DISK FOR PROGRAM
240 OPEN 1,8,2,P$+",P,R"
250 GOSUB 500
260 CLOSE 1
270 IF E=0 THEN 300
280 :PRINT:PRINTEM$
290 :GOTO 180
300 GOSUB 360
310 POKE 42,PEEK(52):POKE 43,PEEK(53)-1
320 CLR
330 GOSUB 430
340 LOAD P$,8
350 END
360 REM  SAVE P$ IN TAPE BUFFER
370 POKE 827,LEN(P$):A=828
380 FOR I=1 TO LEN(P$)
390 :POKE A,ASC(MID$(P$,I,1))
400 :A=A+1
410 NEXT I
420 RETURN
430 REM  GET P$ FROM TAPE BUFFER
440 P$="":A=828
450 FOR I=1 TO PEEK(827)
460 :P$=P$+CHR$(PEEK(A))
470 :A=A+1
480 NEXT I
490 RETURN
500 REM  CHECK DISK ERRORS
510 E=0
520 INPUT#15,EN,EM$,ET,ES
530 IF EN=0 THEN RETURN
540 E=1
550 RETURN
```

©

Joystick Revised

Harvey B. Herman
Chemistry Department
University of North Carolina at Greensboro
Greensboro, N.C. 27412

My previous article, "PETing" with a Joystick", (Compute #4) game some general hints on how to interface a joystick to the PET. In particular I showed how to modify BASIC programs so that they worked with this device. Nevertheless, it is still a pain to change the large number of old programs that may already be in one's library. I looked around for an easier way to interact with the joysticks that would entail minimal modification of pre-existing programs. This article discusses a machine language program that allows one to use joysticks, without any changes in programs which use the conventional keyboard pattern to indicate the direction of player movement.

Last year Kilobaud Microcomputing (March 1979) had an article on the PET User Port. The author, Gregory Yob, illustrated some software which would service an additional keyboard connected to the User Port. The PET interrupt routine uses a pointer in RAM which can be changed to point to special User code. In the PET every 60th of a second an interrupt is generated to service, among other things, peripherals like the screen and keyboard. It is possible to modify the pointer to service a foreign device (in the above article the new keyboard) before proceeding with the normal interrupt processing. I used this idea to write a machine language program which would service a joystick connected to the User Port.

The joystick controller program (shown in the figure) has three entry points. The first (XON) is used to manually initialize the interrupt vectors, program variables and User Port. The second (XOFF) is used to manually disable the operation of the program and to restore the old software vectors. The third (PCODE) is automatically entered after the next interrupt once the program has been initialized.

The code from line 500 on is the main section of the joystick handler. It puts a number in the keyboard buffer which indicates the direction to move next. This number, of course, depends on the position of the joystick. The program attempts to minimize contact bounce by determining if the character is stable at the User Port for two successive (2 x 1/60 sec) jiffies. This value can be experimented with to suit one's taste. The program follows the Yob example and does not allow the keyboard buffer to overflow. A simplified flow chart of the main section is shown in the figure. Modifications necessary for new ROMs are shown in the table (not tested).

The number returned from the User Port falls in the range 63 to 255 (decimal). It is converted by a shift and AND operation to a unique number in the simpler scale of 3 to 15 (decimal) (INT(T/16) and T). This number in turn, is converted by table look up to an indicator of direction movement and stored in the keyboard buffer. A simple example should help to clarify matters. Use the table in my previous article for reference.

If joystick 1 is positioned left, the hex value read at the User Port is DF (223 decimal). The binary equivalent of this number is:

1101 1111

Next shift right four times(INT(T/16)) and the number is:

0000 1101

Finally, AND with the original number (AND T).

The result of these operations:

0000 1101

is 0D hex or 13 decimal

This number is *not* in the normal keypad movement pattern. However, we can convert it to 'move left 'by reading a table whose 14th value is ASCII '4'. The later number is stuffed into the keyboard buffer. By positioning the joystick left, we end up with the same result as if we had typed '4' on the keyboard. BASIC programs which use '4' to move left can be used without *any* modifications. A similar analysis applies to both joysticks and to all the other numbers in the normal keypad movement pattern. In addition, I chose to make the button press mean the number five and straight up mean no keypress.

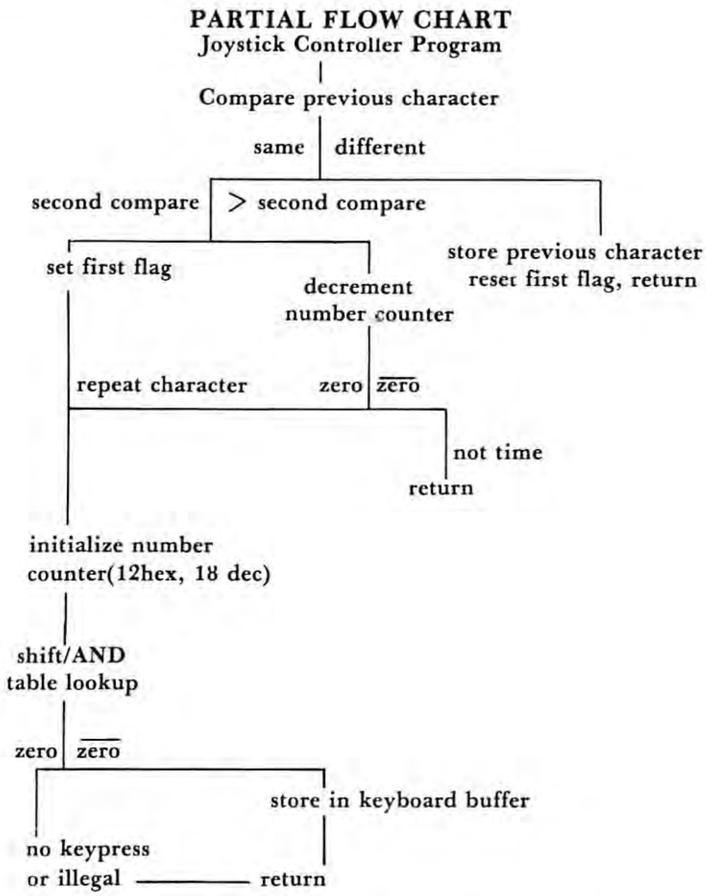
The hex values given in the listing, can be converted to data statements using a program which appeared in the PET Gazette (H. Sherman, Spring '79, p.14). Alternatively, one could use a monitor program to load this program. In either case a monitor program is probably the best way to enter the hex values initially.

The joystick controller program, described here has been helpful to me when converting programs that use one joystick. A slightly more complicated version would be necessary if one needs to use both joysticks in a two player game. I will be happy to answer questions about the interface or program if you include a SASE.

TABLE

Old PET/New PET Equivalent Locations

Description	Old	New
Hardware interrupt vectors	$219/$21A	$90/$91
Hardware interrupt routine	$E685	$E62E
Keyboard buffer index	$20D	$9E
Start of keyboard buffer	$20F	$26F
Interrupt return	$E67E	$E6E4




```
                0110 ; JOYSTICK CONTROLLER PROGRAM
                0120              .BA $33A
                0130 ; DISABLE INTERRUPTS
033A- 78        0140 XON          SEI
                0150 ; SET UP NEW INTERRUPT VECTOR
033B- A9 6A     0160              LDA #$6A
033D- 8D 19 02  0170              STA $219
0340- A9 03     0180              LDA #$3
0342- 8D 1A 02  0190              STA $21A
                0200 ; INITIALIZE USER PORT
0345- A9 00     0210              LDA #$0
0347- 8D 43 E8  0220              STA $E843
                0230 ; INITIALIZE PROGRAM VARIABLES
034A- A9 FF     0240              LDA #$FF
034C- 8D CB 03  0250              STA PREV
034F- 8D CC 03  0260              STA FIRST
                0270 ;ENABLE INTERRUPTS AND RETURN
0352- 58        0280              CLI
0353- 60        0290              RTS
                0300 ; DISABLE INTERRUPTS
0354- 78        0310 XOFF         SEI
                0320 ; RESTORE INTERRUPT VECTOR
0355- A9 85     0330              LDA #$85
0357- 8D 19 02  0340              STA $219
035A- A9 E6     0350              LDA #$E6
035C- 8D 1A 02  0360              STA $21A
                0370 ; ENABLE INTERRUPTS AND RETURN
035F- 58        0380              CLI
0360- 60        0390              RTS
                0400 ; ADJUST STACK
0361- A9 00     0410 STAX         LDA #$0
0363- 48        0420              PHA
0364- 48        0430              PHA
0365- 48        0440              PHA
0366- 48        0450              PHA
                0460 ; CONTINUE INTERRUPT PROCESSING
0367- 4C 85 E6  0470              JMP $E685
                0480 ; MAIN JOYSTICK HANDLER
                0490 ; LOOK AT USER PORT
036A- AD 4F E8  0500 PCODE        LDA $E84F
                0510 ; SAME?-YES.
                     SECOND?-ADD TO KEYBOARD BUFFER.
                0520 ;    THIRD OR GREATER?-KEEP COUNT.
                0530 ;             ADD TO BUFFER IF > 12 TIMES
                0540 ;             RETURN IF NOT
                0550 ; NO.  RESTORE VARIABLES AND RETURN.
036D- CD CB 03  0560              CMP PREV
0370- F0 0B     0570              BEQ EQ
                0580 ; NOT SAME
0372- 8D CB 03  0590              STA PREV
0375- A9 00     0600              LDA #$0
0377- 8D CC 03  0610              STA FIRST
037A- 4C B4 03  0620              JMP FINISH
                0630 ; SAME
037D- 2C CC 03  0640 EQ           BIT FIRST
0380- 10 08     0650              BPL SKIP
                0660 ; THIRD OR GREATER
0382- CE CD 03  0670              DEC NUMB
0385- F0 08     0680              BEQ RESET
                0690 ; < 12 REPEATS
0387- 4C B4 03  0700              JMP FINISH
                0710 ; SECOND TIME
038A- A9 FF     0720 SKIP         LDA #$FF
038C- 8D CC 03  0730              STA FIRST
                0740 ; PICKUP > 12 TIMES
038F- A9 0C     0750 RESET        LDA #12
0391- 8D CD 03  0760              STA NUMB
                0770 ; ACC/16 AND ACC
0394- AD CB 03  0780              LDA PREV
0397- 4A        0790              LSR A
0398- 4A        0800              LSR A
0399- 4A        0810              LSR A
039A- 4A        0820              LSR A
039B- 2D CB 03  0830              AND PREV
                0840 ; CONVERT TO ASCII
039E- AA        0850              TAX
039F- BD BB 03  0860              LDA TABL,X
                0870 ; ZERO? RETURN
03A2- F0 10     0880              BEQ FINISH
                0890 ; OTHERWISE STORE IN KEYBOARD BUFFER
03A4- AE 0D 02  0900              LDX $20D
03A7- 9D 0F 02  0910              STA $20F,X
03AA- E8        0920              INX
                0930 ; WATCH FOR BUFFER OVERFLOW
03AB- E0 0A     0940              CPX #10
03AD- D0 02     0950              BNE COUNTER
03AF- A2 00     0960              LDX #$0
03B1- 8E 0D 02  0970 COUNTER      STX $20D
                0980 ; BACK TO NORMAL CODE
03B4- 20 61 03  0990 FINISH       JSR STAX
03B7- EA        1000              NOP
03B8- 4C 7E E6  1010              JMP $E67E
                1020 ;
03BB- 00 00 00  1030 TABL         .BY 0 0 0 '5' 0 '798'
03BE- 35 00 37
03C1- 39 38
03C3- 00 31 33  1040 TABL1        .BY 0 '132' 0 '46' 0
03C6- 32 00 34
03C9- 36 00
03CB- 00        1050 PREV         .BY 0
03CC- 00        1060 FIRST        .BY 0
03CD- 00        1070 NUMB         .BY 0
                1080              .EN
```

©

PLEXI — VUE™ SOLAR SCREEN

DRAMATIC IMPROVEMENT?

YOU BE THE JUDGE!

SOLAR SCREEN

We urge you to read this and consider our PLEXI-VUE High Contrast SOLAR SCREEN. You will see it is a small price to pay for a big improvement! Some consider it as an Rx for tired eyes! As you will see from our offers, you can even get one FREE!

CONTRAST ENHANCEMENT IS IT FOR YOU?

We have all experienced the eyestrain acquired from sitting in front of the computer for too many hours playing games or working. That is now a thing of the past due to recent developments in the area of CONTRAST ENHANCEMENT. First, the construction of most home computers is such that generally the CRT screens are of a phosphor light gray in color, which makes it difficult to distinguish between the white letters and gray background. The former method was to turn the CRT brightness up which increased contrast but also increased GLARE. This is where the eyestrain comes in. What you need is something that will INCREASE the CONTRAST while DECREASING the GLARE. Several products on the market will accomplish this but to different degrees of success and a drastic difference in cost, as you will see.

OUR PRODUCT

We produce the PLEXI-VUE High Contrast SOLAR Screen from General Electric LEXAN which is unbreakable. It is neutral in color and will work with all phosphor screens including green. This SOLAR LEXAN is a sixteenth of an inch thick and mounts to your computer within a minute after you receive it. You do not need tools of any kind. you just strip off the masking from the foam adhesive and apply the unit to the front of the computer! Then you are ready to enjoy the DRAMATIC difference in viewing WHITE letters on a BLACK screen. Or if you have a GREEN phosphor screen you will see GREEN letters contrasted against BLACK as you do on the expensive WANG and IBM computers! You will also notice that long periods of activity at the computer will not bring on the resultant eyestrain that normally accompanies GLARE. You've been driving into the sun and noticed the difference when you put on a pair of polarizing sunglasses; this is the kind of change you will see by installing PLEXI-VUE!

SATISFIED USERS!

We have been producing and selling the PLEXI-VUE for about a year and a half and have quite a following among users. We have sold to a large number of Colleges, Universities, Schools, Lawyers, C.P.A.'s, Doctors, Hospitals and Laboratories. We know they are pleased because they immediately re-order more PLEXI-VUEs for other computers. Testimonials from users range from, "AMAZING!" to ". . . really like it, enter my order for another!" Don't take our word for it, ask somebody who owns one!

THE COMPETITION

The SUN-FLEX Optical Filter at $20.00 for the smallest version, consists of a fine screening material with a plastic border. It increases contrast but creates a moire effect on the screen if your program has animation. It will also bother you if you have any degree of astigmatism. You can blow through the fine screening material, and have to be very careful not to damage it.

The POLAROID CP-70 Polarization Filter also increases contrast and decreases glare. It costs from $26.00 for a formed plastic filter or from $68.00 for laminated glass versions. Brackets are provided for mounting.

The GLARE-GUARD by Optical Coating Laboratory sells for $95 for certain computers only. It is a very high quality circular polarization material laminated between two pieces of reflection cancelling coated glass. It's thick and heavy, and requires special mounting. Used in some expensive terminals. Material can be purchased for custom fabrication for small computers. As with the SUN-FLEX AND POLAROID versions the screens look "added on" due to the way they are mounted.

SOME OTHER COMPANIES PRODUCE A MYLAR FILM THAT YOU WET AND APPLY TO THE CRT TO DARKEN THE SCREEN. WE WILL NOT MENTION THEM ANY FURTHER THAN TO SAY THEY ARE A RIP-OFF!

DRAWBACKS

We, like other manufacturers of contrast enhancement devices do not recommend their use if you intend to use a light pen. Since the screens cut GLARE, they also reduce light intensity which is needed by the photocell in the pen. Our screen works with some pens, but we would rather caution you than disappoint you after you purchased it.

FREE TRIAL OFFER

We urge you to test the PLEXI-VUE Screen now. Order one for our 30-day no obligation trial. See the dramatic difference it makes on your computer. See how much easier it is to read text with the higher contrast, and how much more you enjoy your computer. Your friends will notice the new appearance as the above photos show. We can make this offer because we have a QUALITY PRODUCT, at a REASONABLE PRICE that we feel will meet with your approval, if you will GIVE IT A TRY!

After you have used it, decide if you want to keep it. If you do you'll own the most affordable contrast enhancement device on the market. If for any reason you're not completely satisfied, simply return your screen with-in 30 days for a prompt and courteous refund. You can't lose!

To order your PLEXI-VUE for our free trial, simply send your personal check or money order for $14.95 + $1.00 Shipping. We accept MASTERCHARGE or VISA! Give ACCOUNT NUMBER, INTERBANK NUMBER, EXPIRATION DATE, AND SIGN your order. Give Model Number needed from CHART:

PX1 = PETs/CBMs with METAL CRT Cases.
PX2 = PETs/CBMs with PLASTIC CRT Cases.
PX3 = NEW 80 Character CRT CBMs.

★ ★ FREE PLEXI-VUE! ★ ★

Order a SOFTPAC-1 at the Regular price of $34.95 + $1.00 Shipping and we'll GIVE you the PLEXI-VUE FREE! The 30 day TRIAL OFFER applies to both! (You must return BOTH for a refund.)

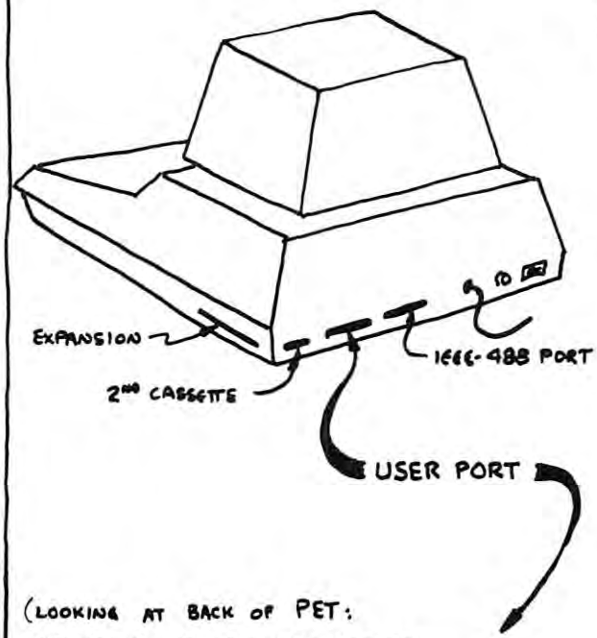
SOFTPAC-1 contains 17 programs on DISC or TAPE (SPECIFY) in a Notebook with Back-up copies, printed instructions & program listings! GAMES W/SOUND too!

ORDER FROM:

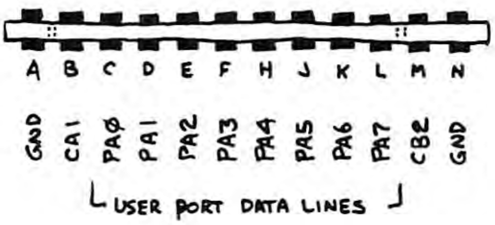
COMPETITIVE SOFTWARE
21650 Maple Glen Drive
Edwardsburg, MI 49112

RAMBLIN'

About the User Port....



(LOOKING AT BACK OF PET: TOP CONNECTORS ARE ASSIGNED TO DIAGNOSTIC & SPECIAL FUNCTIONS)



IT'S UNDERSTANDABLE THAT A NEWCOMER TO THE PET MAY FIND THE OPERATION OF THE USER PORT A LITTLE FORMIDABLE AT FIRST, BUT IT'S REALLY NOT AS HARD TO UNDERSTAND AS IT LOOKS.

THE KEY IS IN THE FUNCTIONING OF THE VIA CHIP (NO. 6522), WHICH IS THE COMPUTER'S LINK TO THE OUTSIDE WORLD. LINES FROM THIS CHIP CONNECT TO EXTERNAL DEVICES AT THE USER PORT, WHILE THE MICROPROCESSOR COMMUNICATES WITH THE CHIP OVER THE DATA BUS, AS IF IT WERE A REGULAR MEMORY REGISTER.

ONE MEMORY LOCATION, ADDRESS 59471, IS DIRECTLY WIRED TO EIGHT OF THE USER PORT CONNECTIONS, PAØ TO PA7. THESE EIGHT LINES CAN BE INDIVIDUALLY PROGRAMMED TO BE EITHER INPUT OR OUTPUT LINES, DEPENDING ON WHAT IS IN THE DATA DIRECTION REGISTER, ADDRESS 59459. IF YOU POKE 59459,Ø, (BINARY ØØØØ ØØØØ), THEN ALL THE PORT A (PA) LINES WILL BE INPUTS. ON THE OTHER HAND, POKE 59459,255 (BINARY 1111 1111), WILL MAKE THEM ALL OUTPUTS. POKE 59459, 240 (BINARY 1111 ØØØØ) WILL CAUSE PAØ TO 3 TO BE INPUTS, AT THE SAME TIME PA4 TO 7 WILL BE OUTPUTS. (PAØ IS CONNECTED TO THE <u>LEAST</u> SIGNIFICANT BIT IN THE REGISTER). IN THIS LAST CONDITION, IF POKE 59471,128 (BINARY 1ØØØ ØØØØ) IS EXECUTED, AN OUTPUT 'HIGH' WILL APPEAR ON PA7, WHICH MIGHT, FOR EXAMPLE, ACTUATE A RELAY ATTACHED TO IT.

THIS MIGHT BE CLEARER AS A DIAGRAM:

	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PAØ	
DATA DIRECTION REGISTER: POKE 59459, 240 →	1	1	1	1	Ø	Ø	Ø	Ø	1 CAUSES LINE TO BE OUTPUT; Ø CAUSES LINE TO BE INPUT
	OUTPUTS				INPUTS				
PORT A DATA REGISTER: POKE 59471, 128 →	1	Ø	Ø	Ø	x	x	x	x	(x = DON'T CARE)
	↑ OPERATES RELAY OR ...								

IN THIS CONFIGURATION, DEVICES CONNECTED TO TERMINALS PA4-7 WILL RESPOND TO THE MOST SIGNIFICANT 4 BITS OF THE BINARY NUMBER IN 59471, AND CAN BE DRIVEN BY POKEING THAT ADDRESS WITH A SUITABLE NUMBER. AT THE SAME TIME, DEVICES CONNECTED TO PAØ TO 3, SUPPLYING A DIGITAL HIGH OR LOW SIGNAL, CAN BE "READ" BY THE COMPUTER WITH A PEEK(59471). IF YOU GET INTO EXPLORING THE PORT, WITH RELAYS OR LEDs, BE CAREFUL NOT TO OVERLOAD THE OUTPUT LINES ~ FROM THE 6522 DATA SHEET, IT LOOKS LIKE 3mA IS THE MOST YOU CAN DRAIN WITHOUT BUFFERING.

-ROY O'BRIEN BOX 426 BEAUMONT CA 92223

Capute

Wherein we acknowledge recent goofs. . .

This page brought to you by Robert Lock, Editor/Publisher and our (sometimes hostile) but always active readers.

Larry's article, "Enhancing Word Pro III" in Issue 4, contained one major "oops!" He did the patches in Basic 4.0 and forgot to change them back. Here are the fixes:

Line 05F0 was printed as:
01 43 55 E4 AD C2 05 48

It should be:
01 43 2E E6 AD C2 05 48

Zapped!

In my review of 6502 Software Design by Leo J. Scanlon, I said that Example 5-14 on page 139 should add a value of two to location 22 rather than the value of one. In fact, the carry bit is set, so that adding one, plus the carry, effectively adds two.

In other words: the book was right, and I was wrong. Oops!

Jim Butterfield

©

Advertisers Index

COMPUTE.
The Journal For Progressive Computing.™

compute II.
The Single-Board COMPUTE.™

☐ Please enter my 1 year (6 issue) subscription to COMPUTE.
☐ Please enter my 1 year (6 issue) subscription to compute II.

My computer is: PET _____ APPLE _____ ATARI _____ OSI _____ SYM _____ KIM _____
AIM _____ Other _____ (Specify) _______________

Check One:
_____ $9.00 (U.S. Mail Order Subscription)
_____ $12.00 U.S. (Canadian Subscription)

☐ I want Both!
_____ $15.00 (U.S. Mail Order Subscription)
_____ $18.00 U.S. (Canadian Subscription)

My Name, Address: _______________

Charge my Visa _____ MC _____
Number _______________ Expires ____/____/____

COMPUTE and compute II are publications of Small System Services, Inc., 900-902 Spring Garden St., Greensboro, N.C. 27403 919-272-4867

The Editor's Feedback:

My computer is _____ PET/CBM _____ ATARI _____ APPLE _____ OTHER
My application is (check all that apply):
_____ Home/Personal _____ Business _____ Educational
_____ Industrial _____ Other
Comments:

Content:
Best Article This Issue (page #, title)

I'd like to see more articles in COMPUTE. _______________
I'd like to see fewer articles in COMPUTE. _______________
Other suggestions:

COMPUTE'S BOOK CORNER

Best of The PET Gazette
$10.00

Collected PET User Notes
Volume 1, Issues 2 - 7
$9.00
Volume 2, Issue 1
$1.50
All 7 issues $10.00

My Name, Address:

Charge my Visa _____ MC _____
Number _______________
Expires ____/____/____

Back issues, when available, $2.00 each.
Issue 1, not presently available.

Add $2.00 shipping & handling

Place
Stamp
Here

COMPUTE./compute II
Post Office Box 5406
Greensboro, NC 27403

Place
Stamp
Here

COMPUTE./compute II
Post Office Box 5406
Greensboro, NC 27403

Place
Stamp
Here

COMPUTE./compute II
Post Office Box 5406
Greensboro, NC 27403

Dear Micro Technology Unlimited:

Please rush me the Fall 1979 catalog of 6502 expansion and application products.

I have the following computer(s):

Old PET____, New PET____, AIM-65____, KIM-1____, SYM-1____

Thank you for expediting.

Sincerely,

____

NAME: ______________________________

STREET: ______________________________

CITY/STATE/ZIP: ______________________________

COMPUTE 80

COMPUTE.
The Journal For Progressive Computing.™

compute II.
The Single-Board COMPUTE.™

☐ Please enter my 1 year (6 issue) subscription to COMPUTE.
☐ Please enter my 1 year (6 issue) subscription to compute II.

My computer is: PET _____ APPLE _____ ATARI _____ OSI _____ SYM _____ KIM _____
AIM _____ Other _____ (Specify) ______________________________

Check One:
______ $9.00 (U.S. Mail Order Subscription)
______ $12.00 U.S. (Canadian Subscription)

☐ I want Both!
______ $15.00 (U.S. Mail Order Subscription)
______ $18.00 U.S. (Canadian Subscription)

My Name, Address: ______________________________

____

____

____

Charge my Visa __________ MC __________
Number ____________________ Expires _____/ _____/ _____

COMPUTE and compute II are publications of Small System Services, Inc., 900-902 Spring Garden St. Greensboro, N.C. 27403 919-272-4867

Skyles Electric Works

YES, I WANT MORE INFORMATION ABOUT:
PET PERIPHERALS

☐ 8, 16, 24 K bytes of PET Memory Expansion Systems

☐ Full sized PET Keyboard with numeric pad

☐ PAL-40 (40 column), PAL-80 (80 column) PET Printers

PET PROGRAMMING AIDS

☐ BASIC PROGRAMMERS TOOLKIT

☐ MacroTeA 6502 Software Development System

Name ______________________________

Address ______________________________

City/State/Zip ______________________________

☐ *Please send name of local Skyles Electric Works dealer.*

Place Postage
Stamp Here

Micro Technology Unlimited
P.O. Box 12106
2806 Hillsborough St.
Raleigh, NC 27605

PLACE
STAMP
HERE

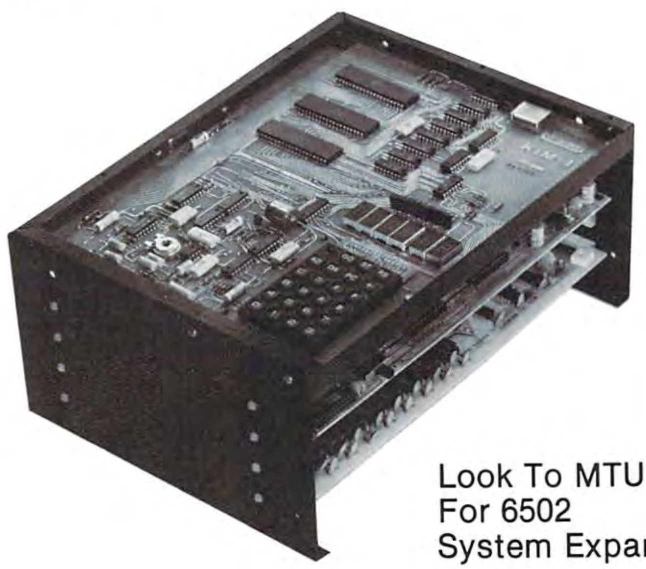
COMPUTE./compute II
Post Office Box 5406
Greensboro, NC 27403

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
First Class Permit No. 503 Cupertino, CA

Postage will be paid by addressee:

Skyles Electric Works
10301 Stonydale Drive
P. O. Box 574
Cupertino, CA 95015



The Great American Solution Machine.

Meet Commodore. The business computer that's providing solutions for more than 100,000 people all over the world. Built by one of the pioneers in office machines. With a reputation for quality that can only come from vertical integration of the total manufacturing process. Commodore builds, not assembles.

Compare Commodore's word and data processing capabilities with computers costing twice or even three times as much. You'll see why so many small businesses are turning to Commodore for solutions to problems as varied as these:

☐ A car leasing company's customers were terminating too early for account profitability. Solution: A 16K Commodore. It analyzes cash flow on over 1200 accounts, identifies those for early penalties, and even writes up lease contracts. Commodore paid for itself within weeks.

☐ A law firm needed a high quality, easy-to-use, affordable word processing system. Solution: Commodore plus its WordPro software package. At a $6,000 savings.

☐ A gasoline retailer needed to inventory, order and set prices; determine Federal and state income taxes; and comply with Federal pricing and allocation regulations. All done daily, weekly, monthly and yearly. Solution: Commodore. It keeps his business on track—and Uncle Sam off his back.

☐ A paint and wallpaper store had to inventory over 600 expensive wallpaper lines for profitability, monitor distributor sales, set and track salesmen's goals, and help the customer select the right size, pattern and quantity. Solution: Two 32K Commodore computers, floppy disk and printer. Commodore does it all—and accounting, too.

In applications like these, and many more, Commodore solves the problems that stand in the way of increased profitability. Commodore can provide the solution in your Great American business, too. Find out more by calling or writing any of Commodore's District Sales Offices.

COSTA MESA, CA 2955 N. Airway Avenue 92626. (714) 979-6307.
SANTA CLARA, CA 3330 Scott Boulevard 95051. (408) 727-1130.
DECATUR, GA 5360 Snapfinger Woods Drive 30035. (404) 987-3311.
BENSENVILLE, IL 790 Maple Lane 60106. (312) 595-5990.
NORRISTOWN, PA 950 Rittenhouse Road 19401. (215) 666-7950.
DALLAS, TX 4350 Beltwood Parkway South 75234. (214) 387-0006.

Commodore Business Machines, Inc., Computer Sales Division, Valley Forge Corporate Center. 950 Rittenhouse Road, Norristown, PA 19401.

commodore

COMMODORE