Volume One/Number One March/April 1980

APPLE

ORCHARD

THE APPLE NEWSMAGAZINE OF THE FIFTH ANNUAL WEST COAST COMPUTER FAIRE

Introducing

INTERNATIONAL APPLE CORE™

In this issue:

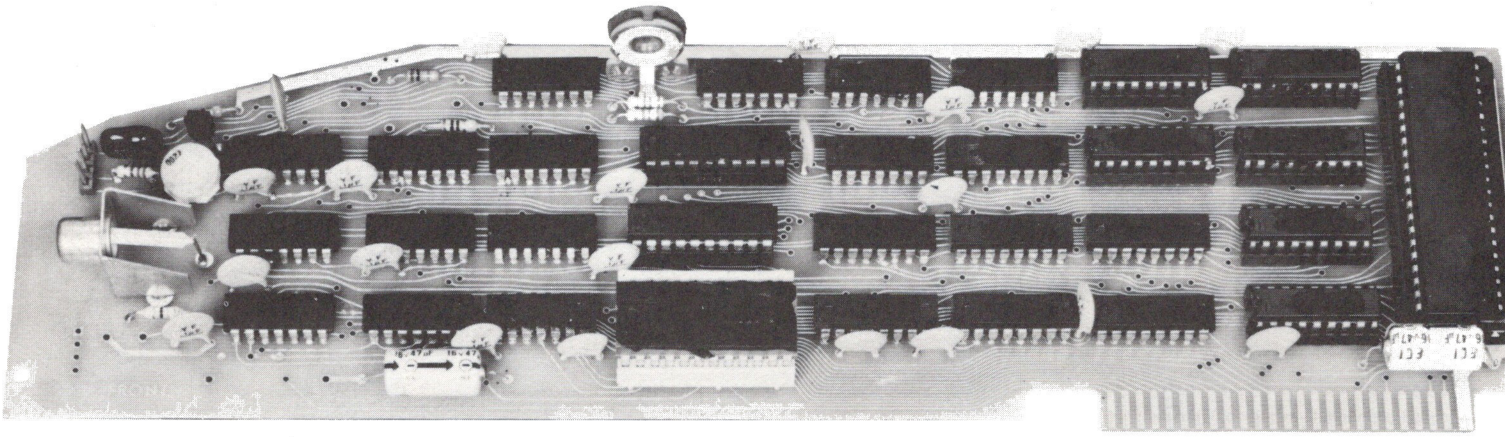
- What is a User Group?
- Programs and Features

Published by the International Apple Core in cooperation with:

• Apple Bay Area Computer Users Society •
• S.F. Apple Core •
• Apple Pugetsound Program Library Exchange •
•Houston Area Apple Users Group •
• Michigan Apple • Original Apple Cores •
• New England Apple Tree •
• Philadelphia Apple User Group • Washington Apple Pi

$5.00

SUP'R'TERMINAL™



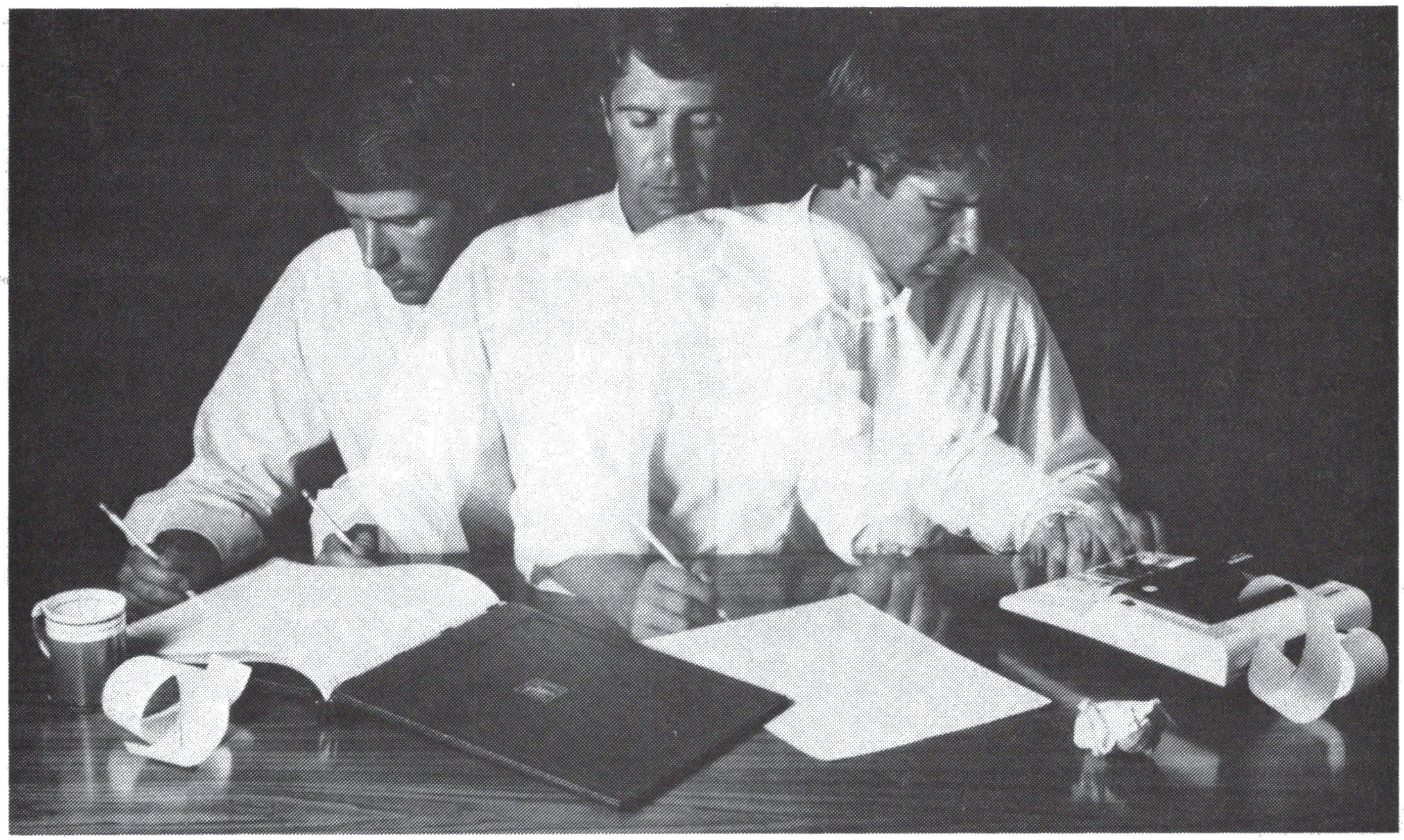
SUP'R'TERMINAL IS AN 80 COLUMN BY 24 LINE PLUG-IN COMPATIBLE BOARD FOR THE APPLE II COMPUTER

SPECIFICATIONS

- 80 Columns by 24 lines, upper and lower case; all 128 ASCII characters.
- Upper and Lower case data entry using the APPLE II keyboard.
- Includes an Upper and Lower case 5x8 dot matrix ASCII character set, and inverse alpha characters.
- Character set can be user definable
- Includes VBC™ (video balance circuit) which enables the use of displaying 80 columns on an inexpensive 8 MHz CRT monitor
- Shift Lock Feature
- Works with APPLE PASCAL and APPLE BASIC
- Incorporates PASCAL and BASIC control characters
- ALL monitor-type escapes are valid
- Follows protocols of PASCAL and BASIC operating systems
- Compatible with ALL APPLE II peripherals.
- Effective baud rate greater than 10,000; fast scrolling and clearing
- Synchronous operation with APPLE II
- Can be used with APPLE II communication interface board to act as self contained terminal for time-sharing or other applications.

PATENT PENDING

M&R ENTERPRISES
P.O. BOX 61011, Sunnyvale, CA 94088



Solve your personal energy crisis. Let VisiCalc™ Software do the work.

With a calculator, pencil and paper you can spend hours planning, projecting, writing, estimating, calculating, revising, erasing and recalculating as you work toward a decision.

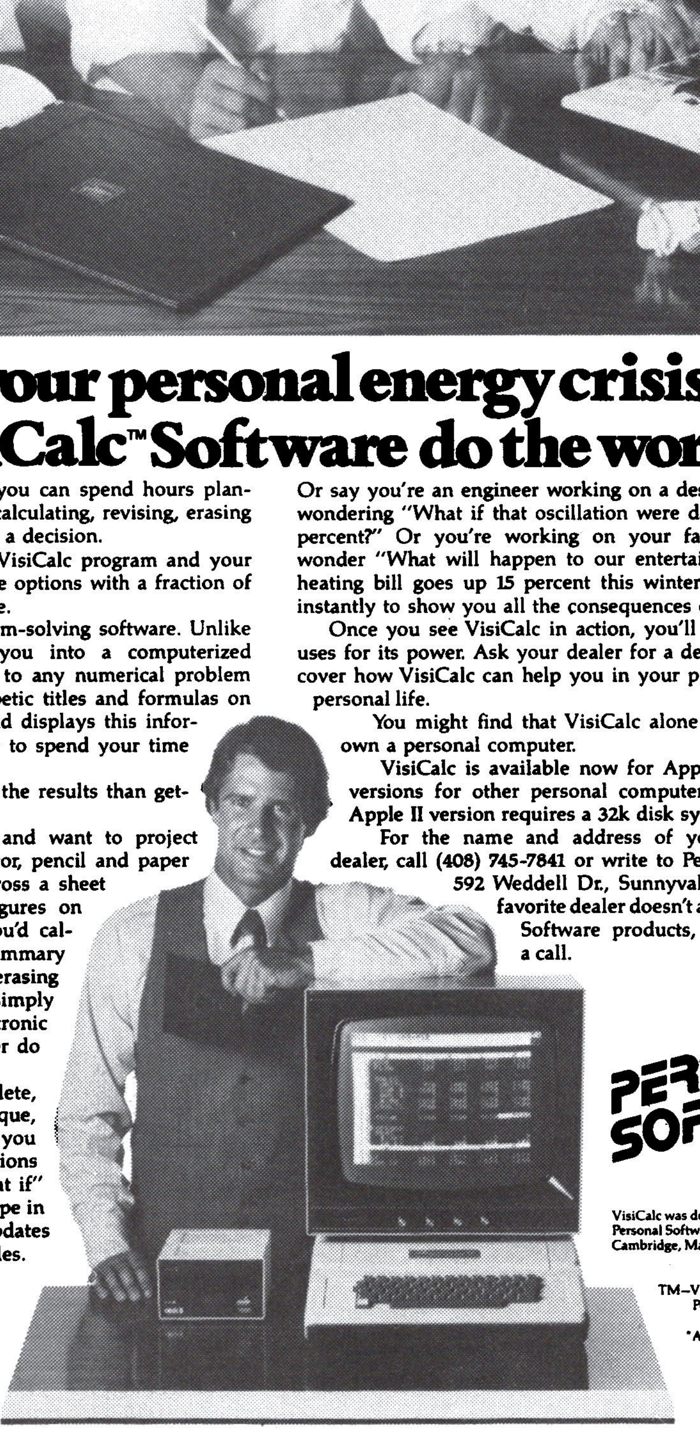
Or with the Personal Software™ VisiCalc program and your Apple* II you can explore many more options with a fraction of the time and effort you've spent before.

VisiCalc is a new breed of problem-solving software. Unlike prepackaged software that forces you into a computerized straight jacket, VisiCalc adapts itself to any numerical problem you have. You enter numbers, alphabetic titles and formulas on your keyboard. VisiCalc organizes and displays this information on the screen. You don't have to spend your time programming.

Your energy is better spent using the results than getting them.

Say you're a business manager and want to project your annual sales. Using the calculator, pencil and paper method, you'd lay out 12 months across a sheet and fill in lines and columns of figures on products, outlets, salespeople, etc. You'd calculate by hand the subtotals and summary figures. Then you'd start revising, erasing and recalculating. With VisiCalc, you simply fill in the same figures on an electronic "sheet of paper" and let the computer do the work.

Once your first projection is complete, you're ready to use VisiCalc's unique, powerful recalculation feature. It lets you ask "What if?," examining new options and planning for contingencies. "What if" sales drop 20 percent in March? Just type in the sales figure. VisiCalc instantly updates all other figures affected by March sales.



Or say you're an engineer working on a design problem and are wondering "What if that oscillation were damped by another 10 percent?" Or you're working on your family's expenses and wonder "What will happen to our entertainment budget if the heating bill goes up 15 percent this winter?" VisiCalc responds instantly to show you all the consequences of any change.

Once you see VisiCalc in action, you'll think of many more uses for its power. Ask your dealer for a demonstration and discover how VisiCalc can help you in your professional work and personal life.

You might find that VisiCalc alone is reason enough to own a personal computer.

VisiCalc is available now for Apple II computers with versions for other personal computers coming soon. The Apple II version requires a 32k disk system.

For the name and address of your nearest VisiCalc dealer, call (408) 745-7841 or write to Personal Software, Inc., 592 Weddell Dr., Sunnyvale, CA 94086. If your favorite dealer doesn't already carry Personal Software products, ask him to give us a call.

PERSONAL SOFTWARE

VisiCalc was developed exclusively for Personal Software by Software Arts, Inc., Cambridge, Mass.

TM—VisiCalc is a trademark of Personal Software, Inc.

*Apple is a registered trademark of Apple Computer, Inc.

MOVING DATA AT A SNAIL'S PACE BECAUSE YOU'RE FLOPPY BOUND?

Let Corvus Systems put you back in the race!

- **For TRS-80†, Apple‡ (including Apple Pascal), S-100 Bus—and now LSI-11.**
- **Fully compatible hardware/software.**
- **10-million byte disk: IMI-7710**
- **Proven Winchester technology.**
- **Z-80 based Corvus disk controller.**
- **Comprehensive disk diagnostics.**
- **Up to 4 disks per system.**
- **System $5350 (LSI-11 $5950), add-on disk $3690.**

Corvus offers a complete systems solution to the mass storage problem of micro computers. In a package smaller than a briefcase, we provide an intelligent controller, disk, and personality module. Call or write today for additional information. Get up to speed with Corvus.

NOW! CORVUS SPEAKS APPLE™ PASCAL™!

†TRS-80 is a registered trademark of Radio Shack, a Tandy Co.
‡Apple is a registered trademark of APPLE Computers, Inc.

CORVUS SYSTEMS, Inc.
900 S. Winchester Boulevard
San Jose, California 95128
408/246-0461

QUESTIONS & ANSWERS FROM SSM

"What equipment can be used with the AIO?"

Since the introduction of the AIO Serial & Parallel Apple Interface in September 1979, thousands of units have been sold to interface the Apple with a variety of printers and terminals. A partial list of devices that have *actually* been tested with the AIO includes:

- IDS 440 Paper Tiger
- Centronics 779
- Qume Sprint 5
- NEC Spinwriter
- Comprint
- Heathkit H14
- IDS 125
- IDS 225
- Hazeltine 1500
- Lear Siegler ADM-3
- DTC 300
- AJ 841

"Will the AIO work with a PAPER TIGER at 1200 baud serial?"

Yes. The AIO has 3 handshaking lines for serial connections. The baud rate can be set with a rotary switch to 110, 300, 600, 1200, 2400 and 4800 baud. (Ask for a data sheet for more details on how to go up to 19,200 baud.)

"Does the AIO work with Pascal?"

Yes. The current AIO serial firmware works great with Pascal. If you want to run the parallel port, or both the serial and parallel ports with Pascal, order our "Pascal Patcher Disk".

"I'm an OEM with a particular need. Can SSM help me?"

Yes. The AIO is just one of several boards for the Apple that SSM will be introducing over the next year. We are also receptive to developing products to meet special OEM requirements. So please contact us if you have a need and there is nothing available to meet it.

SSM will soon be moving to a new and larger facility in San Jose. Look for our new address and telephone number in our ads in Byte magazine, page 11.

We welcome inquiries from new dealers, distributors and OEM's.

Please send in any suggestions, or applications information (AIO uses, printer and terminal hook-up diagrams, etc.) or your ideas for new products. We welcome your comments!

APPLE ORCHARD

THE APPLE ORCHARD

VOL 1. **NO. 1**

March/April 1980

Entire Contents Copyright 1980
by **International Apple Corps**
P.O. Box 976 Daly City, Ca. 94017

Val J. Golding . . . . Editor
Randall Hyde . . . . Assistant Editor
Ken Silverman . . . . Assistant Editor
Patricia Boner . . . . Editorial Assistant
Michael Weinstock . . . . Advertising Manager
COVER ART by Art Department, Apple Computer, Inc.
Postcard furnished by Buck Evans

Typeset and printed by Carlos Printing, Inc.
214 North Washington Ave. Kent Wa. 98031

INTERNATIONAL APPLE CORPS

Interim Officers

Val J. Golding — President
Neil Lipson — Vice President
Dave Gordon — Treasurer
Ken Silverman — Secretary

Interim Regional Directors

WEST

Fred Wilkinson
Joe Alinsky

NORTH

Jon Lawrence

SOUTH

R.V. "Bob" Collins
DeWayne VanHoozer

EAST

Bernard Urban
Dick Sedgewick

INTERNATIONAL

Reger Ossie

Committee Chairmen

APPLE ORCHARD — Val J. Golding
THE SOURCE — Joe Alinsky
FAIRE — Matthew McIntosh
CONSTITUTION — Ken Silverman
CARROT — Randall Hyde
BUDGET — Dave Gordon
SOFTWARE — Neil Lipson
INFORMATION/DOCUMENTATION — Randall Hyde
KIT — Dick Sedgewick

CONTRIBUTING PUBLICATIONS

Abacus
Applesauce
Apple Barrel
Apple Pi
Call – A.P.P.L.E.
Cider Press
Contact
Michigan Apple
Neat Notes
Philadelphia Apple Club

IN THIS ISSUE

Select One . . . . Val J. Golding 5
What is a User Group . . . . Val J. Golding 7
Original Apple Corps . . . . Kip Reiner 7
San Francisco Apple Core . . . . . Ken Silverman 8
A.P.P.L.E. . . . . Val J. Golding 9
Philadelphia Apple Club . . . . . . Neil Lipson 9
Michigan Apple Club 9
A.B.A.C.U.S. . . . . Ed Avelar 10
International Apple Corps. . . . Ken Silverman 10
Space Trip . . . . Mark Cross 10
Applesoft Internals . . . . John Crossley 12
The &Lomem: Utility . . . . Neil Konzen 21
Connecting With the UCSD BIOS . . . . . Randall Hyde 25
Software Development Tools For the Apple II . . Bob Stout 35
Shaping Up With The Apple II . . . . . Mark L. Crosby 37
Apple DOS Booting Process. . . . . Ted Burns 46
Converting Brand X to Brand Y . . . . . . Randall Hyde 47
Game Paddle Driver . . . . Ron & Darrell Aldrich 53
Integral Data Hi-Res Dump . . . . Ron & Darrell Aldrich 54
Modifications to the Apple II to Display Upper and Lower
Case Letters . . . . John MacDougall 57
Page Flip . . . . Glen Hoag 60
Apple Typer. . . . . W.C. Deegan 62
Apple Typer II . . . . W.C. Deegan 64
R & R For Decimal Dumps . . . . Max J. Nareff 65
Converting Integer Basic to Assembly Language Randall Hyde 67
The Language Ssytem – The Apple Grows Up . Bill Wurzel 74
Hex ASCII Memory Dump . . . . W.C. Deegan 79
Dos Tidbits . . . . Jerry Rivers 83
Flash Cards . . . . . . Val J. Golding & Rick Williams 89
Disk Bootstrap Without ROM . . . . . . Richard Suitor 95
Apple Rumor Mill 96
Hex Dec Converter . . . . Val J. Golding 96
Software Reviews . . . . Robert Dennison 98

*** SYNTAX ERR

At press time, it is not known whether there will be a second issue of the Apple Orchard, or when. Inevitably, errors may have crept into program listings, etc. (They were not there to start; gremlins stuck them in). In this event, corrections will be jointly published in subsequent issues of Call – A.P.P.L.E., Applesauce and Cider Press.

FROM THE EDITOR'S

SCRATCHPAD ( )
NOTEBOOK ( )
RAVINGS ( )
HARD DISK ( )
MAXFILES ( )

SELECT ONE

by
Val J. Golding

What the above contrived title boils down to is that from a massive maze of confused clutter, almost as if by magic, comes a cohesive, structured neat appearing magazine. This has always been the case with Call —A.P.P.L.E., the journal of Apple Pugetsound Program Library Exchange, which we also edit, and it is no less true of the instant APPLE ORCHARD.

Our idea here is to give the Orchard reader a glimpse — just a little glimpse — of how this magazine was turned from a stack of loose copy, notes and printer listings into what you see here. Now if the reader can visualize a neat, orderly modern office, fluorescent lights, two or three IBM selectrics, a couple of Apples and disk drives, four or five desks, canned music in the background, ets., then they are thinking obviously about Playboy or the San Francisco Chronicle, but definitely not Call — A.P.P.L.E. or the APPLE ORCHARD.

Both of the above magazines are produced in the editor's basement, where there is not room for a single new piece of copy or a single new program diskette. Programs/articles have been misplaced, lost, recovered and lost again, promised stories have never arrived. These are an editor's woes; they are the price he pays for a labor of love. They are not the basis of complaint; they are a statement of fact.

The process of following a project such as the Orchard from infancy to fruition holds a strange fascination; it is as compulsive and addictive as alcohol or nicotine. The task of putting together a 50 page magazine about every six weeks is not easy, and when a "extracurricular" 100 page magazine is thrown in the middle of things, it only adds to the confusion. It is at this point that we would like to offer a special word of thanks to our editorial assistant, **Ms. Patricia Boner,** without whose help the Orchard would never have survived. Pat spent many hours running to and from the printers with proofs, keeping track of the drastically late arriving ad copy, helped lay out the magazine, entered and tested programs and innumerable other details. Thanks, Pat.

The Orchard was born in September of 1979, amidst talk of some type of super user group that could be organized to assist new groups in getting under way and to perform other services. The Orchard idea came about as a result of telephone conversations between **Val Golding** of Call— A.P.P.L.E., **Randy Hyde** of Applesauce, **Dave Gordon, Ken Silverman** and other individuals who felt the need for a publication that could be issued in commemoration with the 1980 West Coast Computer Faire, and perhaps also be the organ for the proposed "super" user group.

In late October, nineteen individuals gathered in San Francisco as guests of Apple Computer, Inc., and met in a breathtaking brainstorming session which ultimately produced the Internation Apple Corps, committees, and the Apple Orchard. It was decided that each of the invited clubs, who among them represented more than 5000 Apple users, would each contribute original material for a 100+ page magazine, to be known as the Apple Orchard, and to be published in March, 1980, concurrently with the fifth West Coast Computer Faire.

With a number of goals set, each representative left the meeting secure in the knowledge that "history" had been made in San Francisco, and that each individual had pledged his efforts to make the IAC a success.

Just when things start looking gloomy, it seems that everything starts falling into place. The material from the Original Apple Corps arrived first, the only material to do so ahead of the deadline. That gave us a chance to start setting some type right away. Layout cannot commence until you know your format, until you have galleys that can be measured. Material arrived late, or not at all. Uncle Sam lost at least a dozen pages of copy that had to be replaced. On the other hand, Carlos Printing doing just their second typesetting job for us, turned out such relatively error-free copy that proofreading was a snap.

Suddenly, our January printing deadline was almost upon us. We were flooded with Federal Express, Express mail and other speed delivery services, all with new copy or ads. In fact, our residential street at times looked like a delivery service convention. And then a foot of snow in just over 24 hours, and everything stopped. Seattle closed its eyes and went to sleep. Most outlying areas were inaccessible for two or three days, and despite the old saying about rain, snow, sleet, etc., Uncle missed us at least once. Seattle is simply not prepared to cope with snow in those quantities, since it is quite infrequent.

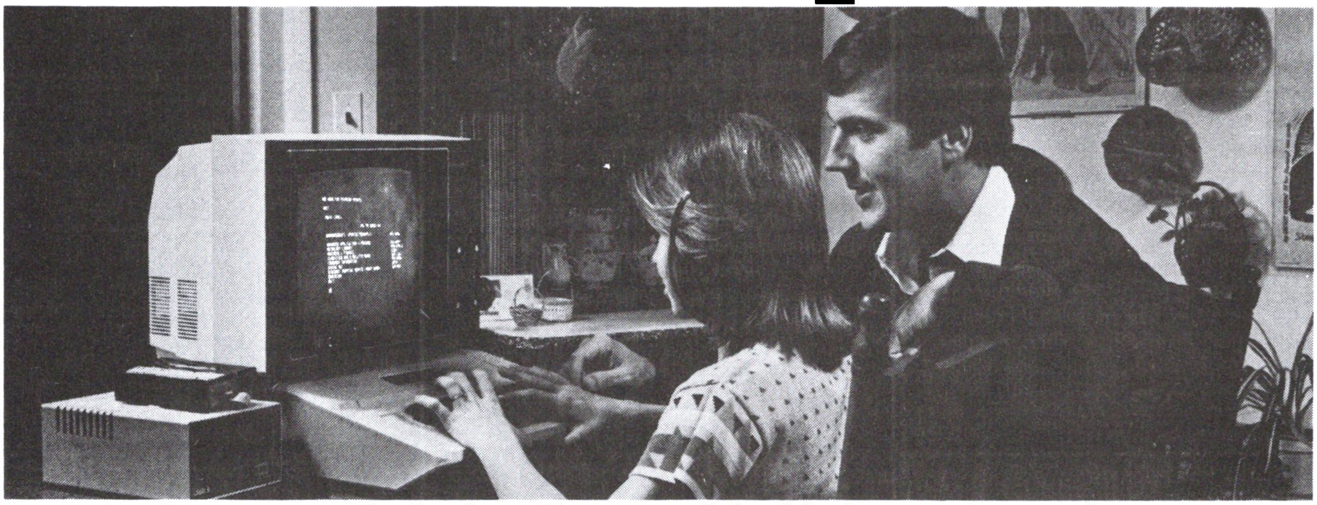
To make a long story short, in five days, the rains came and washed away the snow, express mail arrived with the last copy for the Orchard, Patricia and I together laid it out, and we were off and running.

In San Francisco, a decision was made by Apple Computer, Inc. that "Contact", issue no. 7 would not be published as such, but that the material for it would be furnished to the I.A.C. for inclusion in the Orchard, which brings us to our lead article: "Applesoft Internals" by John Crossley. Here is information about the internal structure and operation of Applesoft that has never before been printed. It is extremely well researched and most comprehensive. In it are furnished all the tools that an Assembly Language programmer needs to be able to access Applesoft routines from 6502 code. Thanks, John.

For those of you that are new to either the Apple or to user groups, you will find a symposium on articles commencing with our "What is a User Group", which continues with brief sketches of some of today's major groups and how to join and how you may benefit by belonging to a user group.

We could go on and describe each of the feature articles in the Orchard, but we believe by the time you read one or two, you will find yourselves "hooked" without any further help from us, so read on . . .

Give your APPLE™ mainframe power



and give your whole family a world of information, education, entertainment and communication with the first information utility.

Enhance your APPLE to give you and your family more use and value of your current hardware than you thought possible through THE SOURCE . . . the world's first information and communication utility for microcomputers.

As Near As Your Telephone

With a simply accoustical connection to your telephone, your APPLE can become a part of a vast information and communication network which now serves thousands of home and business computers . . . through a local phone call in over 300 U.S. cities.

From UPI, New York Times to Local Entertainment Guides

With THE SOURCE's on-line keyword access, you can have important information sources in your own home . . . the United Press International wire service, over 5,000 informational abstracts from the New York Times Consumer Data base, stock market reports, restaurant and entertainment information, dozens of sophisticated computer games, educational programs, travel information and reservations, a discount buying service . . . and hundreds of other information and entertaining services.

Instant Nationwide Communication with Other Personal and Business Computers

With your SOURCE network subscription, you will have the capability of instant nationwide communication with thousands of other microcomputer users on THE SOURCE network. Use it to transmit programs and data, to relay business information, or simply to chat with friends . . . at a fraction of the cost of other message systems.

Expand Your Capacity for Programming and Storage Through Mainframe Power

For your larger or more complicated programs, THE SOURCE offers you the ability to upload and download information to a large mainframe with unlimited storage capacity. You can program in FORTRAN, COBOL, EXTENDED BASIC, and RPG II with your APPLE and you'll never have a capability or a capacity problem again.

The Source . . . Easy to Use for Your Whole Family

You don't have to be a computer whiz to enjoy THE SOURCE's many information, entertainment, and communication advantages. All SOURCE services and information resources can be accessed in plain English with absolutely no prior knowledge required of computer programming so the whole family can enjoy full use and value.

The Source . . . An Affordable Compliment to Your APPLE System

You'd pay thousands of dollars to duplicate the information and entertainment programs THE SOURCE offers. But THE SOURCE costs as little as $2.75 per connect hour (non-prime time), giving your family a value which is impossible to beat.

Find Out How The Source Can Give More Power and Value to Your APPLE

To find out how THE SOURCE's many features can expand and complement the value of your APPLE and how easy it is to connect to the world's first information utility, simply complete the coupon below or call our toll-free information hotline. There's no obligation.

AI 3/80

THE SOURCE
Information Utility

Free Information Request

Please send me your free guide to THE SOURCE Information Utility for personal computers. I understand that I am under no obligation.

☐ I currently have a home computer.
MAKE: ________, *MODEL:* ________

☐ Please send me information on compatible home microcomputers including the special low-cost equipment packages offered by THE SOURCE.

NAME: ________

ADDRESS: ________

CITY: ________

STATE: ________ ZIP: ________

Please mail this coupon to:
THE SOURCE
1616 Anderson Road ■ McLean, Va 22102
Toll-free hotline: 800-336-3330

WHAT IS A USER GROUP

by
Val J. Golding

To define user group, we must first define user. Very simply a user is an individual who has purchased or otherwise acquired a product. Webster defines user as "a person or thing that uses". In the context to which we refer herein, user means an Apple II computer user. Further, in our own definition user group implies in our own definition, user *group* implies a gathering or association of people with a common goal: to share with the others the knowledge in various areas that individuals may have gained.

Currently in the United States, it is estimated that there are between 50 and 100 active Apple computer user groups. User groups are not restricted to Apple users by any means. There are also a number of groups devoted to the Pet, TRS80, etc.

In addition, going back a bit further there were (and still are) computer *clubs* with a wider range of interest, i.e., not devoted to a specific brand or type of computer. These clubs were built around first generation microcomputers such as Imsai, Sol, etc., a breed which for the most part require some background in computer technology.

These micros are not what the consumer expects today of machines like Apple or Exidy, where one can walk into a computer store, plunk down a thousand or so and walk out with a computer that can be taken home, plugged in and immediately be put to use.

In the mid 70's, buying and assembling a microcomputer was akin to buying and assembling a stereo system, taking it home and plodding your way through a maze of wires and manuals, hoping that your pride and joy eventually would "run". Out of this was born the first "user groups", the computer clubs of the period. These clubs were of necessity hardware-oriented.

Today the user group has taken on a new meaning and significance; they are groups where now the primary accent is on software and the exchange of information more closely allied to programming and operation.

To understand the functions of a user group, it is necessary to look at some of the groups that are acknowledged as leaders and that were among the earliest formed.

The first group around, to the best of our knowledge, was the Original Apple Corps in Los Angeles. They existed as early as December, 1977. Other groups that qualify as "pioneer" Apple user groups include the San Francisco Apple Core and Seattle's Apple Pugetsound Program Library Exchange. It was only natural that as the microcomputer industry developed on the west coast, that the first user groups would be from that area.

We have requested that each of the groups that contributed to the formation of the International Apple Corps furnish us with a bit of history and background on their respective organizations. Those vignettes follow here.

ORIGINAL APPLE CORPS

by
Kip Reiner

The Original Apple Corps was founded by AVIDD Electronics in Long Beach, CA. Sandy Tiedeman was the first president of the club. He organized the first meeting of the Original Apple Corps in December 1977. This gave the Original Apple Corps the distinction of being the first Apple computer club in the country. The Los Angeles area was the early hot bed of activity for the Apple II as shown by an increase from six members at the first meeting to over thirty at the second meeting a month later.

The Original Apple Corps has held two meetings at computer shows with very good results. The April 1978 meeting was held at the Percomp Computer Show in Long Beach. Over one-hundred members listened to Phil Roybal, marketing manager for Apple Computer, Inc. as he showed the Disk II prototype with a makeshift operating system which had to be loaded into memory in three different segments. At the third West Coast Computer Faire in Los Angeles, the November 1978 meeting was held. Over three hundred members in a packed meeting room listened to Steve Wozniak, the designer of the Apple computer, as he told about the history of Apple Computer, Inc.

The Original Apple Corps meets on the second Sunday of the month in Lecture Hall 151 at Cal State University, Long Beach, at 12:00 noon. They use an Advent color projection television for video display along with black and white monitors along the sides of the room. Free software is given out at most meetings normally consisting of a full disk. In over two years of monthly meetings, the Original Apple Corps has had representatives of many major hardware and software manufacturers as speakers.

The Original Apple Corps has made available a ready to run library with over 500 programs on disk for a nominal charge. These include programs in business, scientific, education, utilities, and games.

This month marks the one year anniversary of the Original Apple Corps magazine "Applesauce." Our club is very proud of the magazine as a major source of information about the Apple computer and related products. In the last year the magazine has had numerous articles on programming in BASIC, Pascal, and assembly languages. The editor of "Applesauce," Randy Hyde, is very experienced in Pascal. Both software and hardware reviews are included as regular features. In a recent issue Hi-res graphics was presented in depth.

The Original Apple Corps has an electronic bulletin board system up and running as an information exchange for Apple Computer users.

The club welcomes all hardware and manufacturers to speak at our meetings.

Information on membership, subscriptions to "Applesauce", and the program library are available at the following address:

Original Apple Corps
12804 Magnolia
Chino, CA 91710

Kip Reiner
President
Original Apple Corps

DOES "GET" GET YOUR GOAT?

In APPLESOFT when using the "GET" command in conjunction with DOS, follow it with a PRINT, else GET tends to do nasty things and mess up your DOS.

THE SAN FRANCISCO APPLE CORE

by
Ken Silverman

The APPLE CORE OF SAN FRANCISCO is a non-profit organization comprised of and supported by Apple II Computer owners. The Apple Core is run entirely by volunteer officers and committees. The club endeavors to aid other APPLE owners. All members are individuals (and their families), and NO shops, stores or corporations are directly registered. (However, any shop may register an employee c/o that shop).

MONTHLY MEETINGS are held at Homestead Savings, at 5757 Geary Avenue, in San Francisco (at the corner of Geary and 22nd Avenue). Meetings are held on the first Saturday of each month at 10:00 a.m.

THE CIDER PRESS is the official publication provided to the membership February through June, and September through December. Included are program listings, tips, special features, reviews, editorial comment . . . The "Best of the Cider Press' is published every January. "Apple Peelings' takes over in July and August with an abbreviated format consisting of minutes of meetings, Disk of the Month listing, product information and any News Flashes.

The APPLE CORE LIBRARY of contributed programs is arranged by general categories. Members living in the San Francisco Bay area may copy programs from the library at the following locations:

Village Electronics	668-4243
Computerland of San Francisco	546-1592
Computerland of Belmont	595-4232
Computerland of Marin	459-1767
Computerland of the Castro	864-8080
AIDS	221-8500

Courtesy and common sense dictate that a member call in advance to reserve use of required equipment. The stores provide this service without charge. (Their aid helps us survive, so remember to return the favor with your patronage. The local stores have been VERY HELPFUL).

Out of area members can get programs from the library through the mails in the following manner:

1. A member is required to donate at least one original or public domain program (not Copyrighted, please).
2. Donated programs must be sent on a disk or a computer tape placed in a self-addressed, stamped proper mailer, suitable for returning the disk or tape. Please use a Program Submission form. Include a note indicating the desired volume from the library that you would like to have copied. Carefully package the mailer and Note:

CONTAINS LIVE COMPUTER PROGRAMS – DO NOT EXPOSE TO X-RAYS OR ELECTRICAL FIELDS – DO NOT BEND OR FOLD.

Send to:

The Apple Core
Library Exchange
P.O. Box 4816
San Francisco, CA 94101

INFORMATION/APPLECATION

Please follow instructions as we do not want to see your disks or tapes ruined any more than you do. Only one library disk or tape will be processed per month. (The DOM-Disk of the Month is considered separately).

The complete LIBRARY #1 is available to members for $150. Over 340 programs on 20 diskettes are packaged in diskette holders, and bound in a SF Applecore binder. Over 30 pages about the library are included.

The DISK OF THE MONTH is a group of recently donated programs or updated utilities, etc. It was originated to encourage new members to be able to write programs by having examples to study and enjoy.

Members unable to come to the meetings can send in $7.50 (US) for the current DOM which covers the cost of the disk, mailing and handling. Three past months are also available for $7.50 each.

Members who come to the meetings can obtain the same DOM's for $5.00 each. Prices are subject to change.

NOTE: All programs on the DOM's go into the library according to category. The stores do not have the COM's on file.

Current interest groups include NU'ERS (for new users), ASSEMBOLEERS, CAI (Computer Assisted Instruction), Personal Finance, Medical & Health, DOS (Disk Operating Syntax), Income Tax Programmers, etc., etc.

APPLECATION FORM

NAME: . . . .
ADDRESS: . . . .
. . . .
. . . .
. . . .
PHONE NO.. . . . .

Check One:
☐ New Member
☐ Renewal
☐ Address Update
☐ Newsletter Trade (Subject to Approval)

MEMBERSHIP RATES:

$15.00/yr. USA-Newsletter sent 3rd class
$20.00/yr. Foreign-Newsletter by surface mail
$25.00/yr. Foreign-Newsletter by air.
Send check or money order, US only. Send no cash!

NOTE: In April yearly dues will be paid by all members. Amount will be pro-rated on your renewal date.

APPLE PUGETSOUND PROGRAM LIBRARY EXCHANGE

by
Val J. Golding

It is curious, in retrospect, to examine ones past and attempt to determine what went wrong or what went right. As one might surmise, there is no one single factor responsible for the current popularity of Apple Pugetsound.

Certainly there was no hope or expectation on behalf of A.P.P.L.E.'s founders that the group would grow to its present strength of well over 3000, or that it would achieve the measure of acceptance that it has, nearly strangling to death along the way, under the burden of a staggering work load.

If any single factor can be held to account, it would have to have been the news releases printed in the summer, 1978 issues of magazines such as *Byte* and Kilobaud, at a time when Apple users were starving for information of any kind about their new computers, and user groups were few and far between.

Even by June of 1978, Call --A.P.P.L.E., this group's newsletter, had grown to 16 pages in size, and was acquiring some of the ear-marks of a "real" newsmagazine. No doubt the many sample copies mailed out at that time provided the impetus for the growth that was to come.

Also in mid-1978, A.P.P.L.E. implemented the concept of "Library Paks", the popular format in which 20 or more assorted programs were made available to the members on a single low priced cassette. This author, in collaboration with **Darrell Aldrich** and numerous other individuals also conceived and wrote the "Programmer's Workshop", a collection of handy Integer Basic utility routines, one of the earliest of it's type.

On April 10, 1979, Apple Pugetsound Program Library Exchange was incorporated as a Washington state non-profit corporation. That status continues currently, and none of the officers draw compensation for their services, despite the long and hard hours turned in by all. The membership and order office currently processes over one hundred pieces of mail daily, the routine portions of which is handled by a full time contract employee. The other divisions, production and shipping, treasury, and editorial are similarly staffed.

Today, Apple Pugetsound has achieved a vast reputation for its high quality software and documentation. Call –A.P.P.L.E., now a slick cover 56 page newsmagazine is available at computer dealers throughout the country, and is read by A.P.P.L.E. members in more than 20 countries. It too is known for its quality of content, and is considered by many as the authoritative source of Apple computer information.

Last, but not least, through the "Call –A.P.P.L.E." hot line, the club provides its members with a useful inquiry and information service, ranging from answering or referring programming problems to current events and new product information.

It is not inexpensive to join, but the benefits far outweigh the cost. There is currently a one-time Apple-cation fee of $25.00 and annual dues of $15.00 through December, 1980, for a total of $40.00. Members joining at any time during 1980 will receive all 1980 issues of Call –A.P.P.L.E at no additional cost. Checks should be made payable to "A.P.P.L.E." and mailed to our new address, which may be determined by calling (206)271-6939.

New members will receive by return mail an Apple-cation blank, order form and description sheets of the various software and publications available.

THE PHILADELPHIA APPLE CLUB

by
Neil Lipson

The Philadelphia Apple Club was started in Feb., 1978 by Neil D. Lipson. Since then it has expanded to include numerous small clubs, and membership in total exceeds 200.

Dues for the Philadelphia branch are only $10. At present there is no newsletter. The meetings are held the third Saturday of each month, at LaSalle College in Philadelphia, 11:00 a.m. and last about three hours.

There are numerous subgroups, among which include machine language, music, education, scientific, hardware, graphics, utilities, video, and light pen. The group is very eager to help and assist other groups in other areas of interest.

About two diskettes of programs are distributed each month, consisting of public domain software only. Disks are distributed at the meetings only, at present.

The meetings are held in a somewhat formal fashion, with an introduction, description of new events and news, new products, and then new software, with a question and answer session at the end. Periodically, we have speakers to give talks on various topics.

We also coordinate closely with the Philadelphia Area Computer Society, which meets with us at LaSalle College, in projects than can be used on the Apple as well as other computers, and there is a close interaction between PACS and the Apple Club. PACS has in excess of 350 members (excluding Apple owners). PACS was formed in 1977 by Dick Moberg, and the present president is Eric Hafler, also a member of the Apple group.

THE MICHIGAN APPLE

The Michigan Apple is the most prominent Apple users group currently active in Michigan. The group is most visible through a newsletter that they publish 10 times a year. They also conduct meetings 10 times a year on the last Tuesday of the month at alternate computer stores in the metropolitan Detroit area. They discuss club business and new products and applications for the Apple.

Additional activities conducted through the Michigan Apple include the following: Club disks of original programs donated by members are made available to the members. Meetings of members with a common special interest are held periodically in member's homes. The club also offers meetings of System Analysis Groups (SAGs) that explore many aspects of the Apple computer in depth. A library of periodical and newsletter material is also maintained and cataloged for the use of members in doing research.

To receive the Michigan Apple-Gram (our newsletter) or to join the club for the various other activities, here is the current address.

THE MICHIGAN APPLE COMPUTER CLUB
P.O. BOX 551
MADISON HEIGHTS, MICHIGAN 48071

CONVERT WITHOUT DISK

If you have an Apple Com Card and a printer with tape punch and read you can convert Integer Basic to Applesoft by punching tape from an Integer program listing and reading back into Applesoft.

APPLE BAY AREA COMPUTER USERS SOCIETY (ABACUS) < AB-A-CUS>

THE ABACUS group is located in the San Francisco East-Bay area, specifically, Castro Valley, Calif. The club meets on the 2nd Monday of each month, at 7:00 p.m. Our group is approximately 200 strong and growing.

The ABACUS-II (NEWSLETTER) is published monthly and is included in the annual membership fee of $12.00.The ABACUS library contains in excess of 500 programs and is available to all members, local or out of state.

Additional information may be obtained by writing Larry Danielson, club treasurer at 5302 Camino Alta Mira, Castro Valley, CA 94546.

ABACUS OFFICERS: President . . .Ed Avelar 2850 Jennifer Dr., Castro Valley, CA 94546 – (415) 538-2431. Vice President . . . Stephen Shank (415) 820-4374. Secretary . . . Dave Wilkerson Treasurer . . .Larry Danielson 5302 Camino Alta Mira, Castro Valley, CA 94546. (415) 581-2748. Librarian . . .Bill Walsh.

The ABACUS is a founding member of the I.A.C. (INTERNATIONAL APPLE CORPS).

MEMBERSHIP APPLICATION FORM:

NAME. . . . .

ADDRESS . . . .

CITY, STATE, ZIP . . . .

Phone #home . . . .

Phone #work . . . .

Occupation . . . .

Mail to: ABACUS, 2850 Jennifer Dr., Castro Valley, CA 94546.

WHAT IS THE INTERNATIONAL APPLE CORPS

by
Ken Silverman, Secretary

On October 27-28, 1979 a meeting was held in San Francisco to discuss the formation of a non-profit organization to pass on all forms of information, both hardware and software, from Apple user groups and users, software and hardware companies, and Apple Computer Incorporated. This is to provide a flow of information in both directions through one organization for the benefit of the end Apple users.

The meeting was attended by representatives of the larger Apple user groups in the United States. At the end of the weekend the "INTERNATIONAL APPLE CORPS" was formed. An interim Board of Directors and Officers were elected, a basis for a constitution and bylaws were formed, and organizational goals and objectives were set.

The International Apple Corps will be made up of Apple User Clubs all over the United States with membership open to clubs in other countries.

The International Apple Corps is set up to distribute public domain software, application notes, general Apple information and product information. There is also a committee just to help persons who wish to start an Apple club.

In order for this organization to work, like your own club, it needs funding. We plan to obtain monies from many areas and user groups is just one of these areas. To keep us going we are requesting each club to send a "one time" $50 initiation fee for the *CLUB* to be a member of the INTERNATIONAL APPLE CORPS. At the first annual meeting in March, to coincide with the West Coast Computer Faire, a dues structure will be set. After your club joins it will be entitled to:

1. Access to an international library of programs
2. An input-output device for questions and problems on any subject dealing with the Apple
3. Access to printed information
4. Reduced subscription rate for publication

Upon receipt of your $50 we will send the club a package that contains most of Apple's reference manuals (including the new reference book), a collection of new application notes, and very soon the distribution of software. If you have any questions you can write us at our post office box, as shown on page 3.

From applesauce

SPACE TRIP

by
Mark Cross

You're zooming through space in your galactic scout. Souped up ion generators have boosted you to six times the speed of light. The stars are moving past your viewport.

But wait! This is impossible! Any old spaceship can go faster than light, but you don't know how to program your Apple to make the stars move past the viewport. This program simulates the view of moving stars seen by a fast spaceship.

There is a background of distant stars that are so far away they seem to stay in the same place in the sky (lines 310-320). There are ten stars close to the moving spaceship which seem to spread out as the ship approaches. The math involved is simple. To make a star move away from the center by a constant such as 4/3 in line 420. The stars should move slower when they are near the center of the screen, straight ahead and far away. Stars at the edge of the screen will seem to move very fast and disappear off the edge of the viewport.

The speed of the moving stars leaves something to be desired, even though this is Integer Basic with only ten points to be moved. You can make them move faster by using fewer stars in line 400 or multiplying by a bigger constant in line 420. Crosshairs or a frame around the viewport would look good. More advanced programmers might try a machine language routine to move the stars, or a shape table with an asteroid that grows in scale as it moves closer to the spaceship.

NEW RELEASES FROM

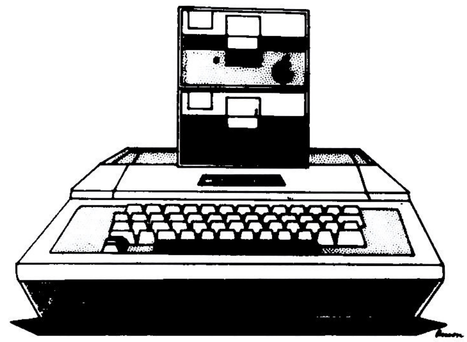
SYNERGISTIC SOFTWARE

THE MODIFIABLE DATABASE
by Chris Anson & Robert Clardy

The Modifiable Database is a general purpose, user oriented database program that can be easily customized for your specific data management application. Create any number of application programs such as mailing lists, bibliography files, inventory controls, personnel files, accounting programs, etc. The only limitations is your own imagination.

The program uses fast and flexible machine language search and sort routines, provides for easy record editing, and can search or print up to 2 disks of records with a single command. All commands are invocable by a few keystrokes. There's never been an easier to use or more flexible data management program.

Modifier Module 1 includes accounting/numeric functions
Modifier Module 2 includes output formatting functions
Requires Applesoft, 48K disk



PROGRAM LINE EDITOR
by Neil Konzon

The Program Line Editor is one of the most powerful program editing tools available. Inserting, deleting, or replacing characters within a line can be accomplished with a few simple keystrokes without re-entering the entire line. Lower case text can be inserted into print statements with a simple command.

The package also includes a keyboard macro capability. You can assign common commands or convenient routines to a keyboard character. Thereafter, pressing ESC and that character will cause the function to be executed. A complete set of macros is provided for your use or modification.

These uniquely powerful capabilities make programming in Integer or Applesoft Basic twice as fast as it used to be. (16K)

HIGHER GRAPHICS
by Robert Clardy

Higher Graphics is a high resolution graphics package that lets you create detailed displays for business or game use and add graphics effects to your software. Package includes 3 utility programs, a text explaining the use of high-res graphics for display and animation effects, and sample shape tables with over 100 of the more commonly desired shapes. The programs allow new shapes and shape tables to be created and existing shape tables to be combined, edited, or rearranged. The screen manipulation program allows the simple placement of shapes, areas of color, and text anywhere on the screen.

(48K Integer & Disk)

HIGH-RES TEXT
by Ron & Darrell Aldrich

Add colorful customized text to your high resolution graphics displays. The High Resolution Text Generator allows you to define your own character sets with normal or double sized characters. The defined characters can be printed in the size defined or in an expanded size. Large characters can be printed in any of 10 colors, the high res colors plus yellow, aqua, pink, navy blue, and grey.

The text generator has all the features of a normal text screen such as scrolling, tracing over text with the cursor, etc. plus full lower case capability with no hardware modifications required. Use the character sets provided (standard, countdown, sideways, etc.) or define your own for special purpose promotional displays, graphing, or games.

(Machine language routines require 16K)

ODYSSEY
by Robert Clardy

Embark on a heroic quest across the dreaded Sargalo Sea stopping to explore dangerous islands, caverns, and old castles. Gather the forces and weapons that you will need to destroy the fortress of the cruel Caliph of Lapour.

Three interlocking programs using several high-res and low-res maps provide an unending variety of hazards, opponents and adventures.

(48K Integer on Disk)

NOW AVAILABLE

GAMES		CASS	DISK
Dungeon Campaign	(I, 16K)	12.50	15.00
Dungeon Campaign	(A, 32K)	12.50	15.00
Wilderness Campaign	(I, 48K)	15.00	17.50
Dungeon & Wilderness	(I, 48K)		30.00
Odyssey	(I, 48K)		25.00
UTILITIES			
Higher Graphics	(I, 48K)		25.00
High Res Text	(I,A, 32K)		32.50
Program Line Editor	(I,A, 16K)		37.50
BUSINESS			
Mailing List Database	(A, 48K)		34.50
Modifiable Database	(A, 48K)		49.50
Modifier Module 1			15.00
Modifier Module 2			15.00

AVAILABLE AT YOUR LOCAL DEALER OR SEND CHECK OR INQUIRY TO SYNERGISTIC SOFTWARE, 5221 120 AVE. S.E., BELLEVUE, WA 98006. WA RESIDENTS ADD 5.3% SALES TAX

APPLESOFT INTERNAL ENTRY POINTS

by
Apple Computer, Inc.
From: Contact
John Crossley

CONTENTS

INTRODUCTION 12
ABBREVIATIONS 12
LABELS 12
TXTPTR INPUT ROUTINES 12
TXTPTR TO INTEGER ROUTINES 13
FLOATING POINT MATH PACKAGE
  INTRODUCTION 13
  REGISTERS 13
  OPERATORS 13
  CONSTANTS 13
  FUNCTIONS 14
  MOVE ROUTINES 14
  UTILITIES 14
  CONVERSIONS
    INTEGER TO FAC 14
    FAC TO INTEGER 14
    TXTPTR TO FAC 15
STRING UTILITIES 15
DEVICE INPUT ROUTINES 15
DEVICE OUTPUT ROUTINES 16
INTERNAL LOCATOR ROUTINES 16
INITIALIZATION ROUTINES 16
STORAGE MANAGEMENT ROUTINES 16
MISCELLANEOUS BASIC COMMANDS 16
HIRES GRAPHICS ROUTINES 17
CASSETTE ROUTINES 17
ERROR PROCESSOR ROUTINES 17
SYNTAX CHECKING ROUTINES 17
INDEX 18

INTRODUCTION

This is a guide for the 6502 machine language programmer who wants to take advantage of the various subroutines in Applesoft. The addresses included assume that the user has an Apple II Plus, an Applesoft firmware card, or a Language Card. This list is believed to be correct, but be warned that it was a spare time project. If you find errors, contact your user group. This data is meant for the experienced programmer, *NOT THE BEGINNER*. Read your Applesoft Reference manual for more information.

Take special note of CHRGET. This subroutine is the heart of Applesoft. When Applesoft wants the next character or an instruction it points TXTPTR at the program or the input buffer and JSRs to CHRGET. When Applesoft READs DATA, TXTPTR is temporarily set to the last used DATA statement.

ABBREVIATIONS

A the 6502 accumulator
X the 6502 X register
Y the 6502 Y register
Z the zero flag of the 6502 status register
C the carry flag of the 6502 status register

A,X is a 16 bit number where A has the most significant byte and X the least significant byte.

(Y,A) is the number or string whose address is in Y and A with the msb in Y and the lsb in A.

FAC	the floating point accumulator
ARG	the ARGument register
msb	most significant bit or byte
lsb	least significant bit or byte
eol	end of line token ($00)

LABELS

LABELS	HEX ADDR	
A1	3C,3D	Apple monitor pointer for cassette routines
A2	3E,3F	Apple monitor pointer for cassette routines
ARYTAB	6B,6C	Start of array storage
BUF	200,2FF	Line input buffer
CHARAC	OD	Used by STRLT2
CURLIN	75,76	The current line number (=FF if in direct mode.
DATLIN	7B,7C	Line number of current DATA statement
DATPTR	7D,7E	The address of the next DATA comes from
DSCTMP	9D,9E, 9F	Temp string descriptor
ENDCHR	OE	Used by SRTLT2
ERRFLG	D8	$80 if ONERR active
ERRLIN	DA,DB	Line number where error occurred
ERRNUM	DE	Which error occurred
ERRPOS	DC,DD	TXTPTR save for HNDLERR
ERRSTK	DF	Stack pointer value before error
FBUFFR	100-110	FOUT buffer
FIRST	F0	Used by PLOTFNS
FORPNT	85,86	General pointer. see COPY
FRESPC	71,72	Temp pointer for string storage routines
FRETOP	6F,70	Bottom of string storage
H2	2C	Used by PLOTFNS
HIGHDS	94,95	Used by BLTU
HIGHTR	96,97	Used by BLTU
HPAG	E6	HIRES page to plot on. ($20 for HGR, $40 for HGR2)
INDEX	5E,5F	Temp pointer for moving strings
INVFLG	32	Mask for inverse output
LASTPT	53	Last used temp string pointer
LINNUM	50,51	General purpose 16 bit number location
LOWTR	9B,9C	General purpose register. GETARYPT' FINDLN, BLTU
MEMSIZ	73,74	HIMEM
OLDLIN	77,78	Last line executed
ORMASK	F3	Mask for flashing output
PRGEND	AF,BO	The end of the program text
REMSTK	F8	Stack pointer saved before each statement
SPDBYT	F1	Speed = delay number
STREND	6D,6E	The top of array storage
STRNG1	AB,AC	Pointer to a string. See MOVINS
STRNG2	AD,AE	Pointer to a string. See STRLT2
SUBFLG	14	$00 subscripts allowed, $80=no subscripts
TEMPPT	52	Last used temporary string descriptor
TXTTAB	67,68	Start of program text

V2	2D	Used by PLOTFNS
VALTYP	11	Flags last FAC operation 0=number, FF= string
VARPNT	83,84	Used by PTRGET
VARTAB	69.6A	Start of variable storage

TXTPTR INPUT ROUTINES

CHRGET 00B1(177) (Increment TXTPTR)
CHRGOT 00B7(183) (No increment)

These routines load A from TXTPTR and set certain 6502 status flags. X and Y are not changed.

On exit:

A=the character
Z is the set if A ':' or eol ($3A or $00)
C is clear if A is an ASCII number ('0' to '9').

TXTPTR TO INTEGER

LINGET DAOC (55820)

Read a line number (integer 0 to 63999) from TXTPTR into LINNUM. LINGET assumes that the 6502 registers and A have been set up by the CHRGET that fetched the first digit. Normally exits through CHARGET which fetches the character after the number. If the number is greater than 63999 then LINGET exits via SYNTAX ERROR. LINNUM is zero if there is no number at TXTPTR.

GTBYTC E6F5 (51925)

JSR to CHRGET to gobble a character and fall into GETBYT.

GETBYT E6F8 (59128)

Evaluates the formula at TXTPTR, leaves the result in FAC, and falls into CONINT. 'n the entry TXTPTR points to the first character of the formula for the first number. PLOTFNS puts the first number in FIRST and the second number in H2 and V2.

PLOTFNS FIEC (61932)

Get 2 LORES plotting coordinates (0-47,0-47) from TXTPTR separated by a comma. On entry TXTPTR points to the first character of the formula for the first number. PLOTFNS puts the first number in FIRST and the second number in H2 and V2.

HFNS F6B9 (63161)

Get HIRES plotting coordinated (0-279,0-191) from TXTPTR. On entry TXTPTR points to the first character of the formula for the first number. Leaves the 6502 registers set up for HPOSN.

On exit:

A= vertical coordinate
X= lsb of horizontal coordinate
Y= msb of horizontal coordinate.

FLOATING POINT MATH PACKAGE INTRODUCTION

This is the number format used throughout Applesoft:

The exponent is a single byte signed number (EXP) in excess $80 form (the signed value has $80 added to it). The mantissa is 4 bytes (HO, MOH, MO,LO). The binary point is assumed to be to the right of the most significant bit. Since in binary floating point notation the msb is always 1, the number's sign is kept there when the number is stored in packed form in memory. While in the math package the sign is kept in a separate byte (SGN) where only bit 7 is significant. If the exponent is zero then the number is zero although the mantissa isn't necessarily zero.

Examples:

	EXP	HO	MOH	MO	LO	SGN
Packed format						
-10	84	A0	00	00	00	
10	84	20	00	00	00	
FAC format						
-10	84	A0	00	00	00	FF
10	84	A0	00	00	00	00

Arithmetic routine calling conventions:
For single argument functions:
The argument is in FAC.
The result is left in FAC.
For two argument functions:
The first argument is in ARG (see CONUPK).
The second argument is in FAC.
The result is left in FAC.

FLOATING POINT REGISTERS

NOTE: many of the following locations are used for other things when not being used by the floating point math package.

	FAC	ARG	TEMP1	TEMP2	TEMP3	RND
EXP	9D	A5	93	98	8A	C9
HOHO	9E	A6	94	99	8B	CA
MOH	9F	A7	95	9A	8C	CB
MO	A0	A8	96	9B	8D	CC
LO	A1	A9	97	9C	8E	CD
SGN	A2	AA		(packed format)		

FLOATING POINT OPERATORS

FMULT E97F (59775)
Move the number in memory pointed to by Y,A into ARG and fall into . . .

FMULTT E982 (59778)
Multiply FAC and ARG. On entry A and Z reflect FACEXP.

FDIV EA66 (90006)
Move the number in memory pointed to by Y,A into ARG and fall into . . .

FIDVT EA69 (60009)
Divide ARG by FAC. On entry A and Z reflect FACEXP.

FADD E7BE (59326)
Move the number in memory pointed to by Y,A into ARG and fall into . . .

FADDT E7C1 (59329)
Add FAC and ARG. On entry A and Z reflect FACEXP.

FSUB E7A7 (59303)
Move the number in memory pointed to by Y,A, into ARG and fall into . . .

FSUBT E7AA (59306)
Subtract FAC from ARG. On entry A and Z reflect FACEXP.

FPWRT EE97 (61079)
Exponentiation (ARG to the FAC power). On entry A and Z should reflect the value of FACEXP.

NOTE: Most FAC move routines set up A and Z to reflect FACEXP but a LDA $9D will insure the proper values.

FLOATING POINT CONSTANTS

NOTE: The following addresses point to numbers in packed form suitable for use by CONUPK and MOVMF.

RND	00C9	(201)
1/4	F070	(61552)
1/2	EE64	(61028)
-1/2	E937	(59703)
1	E913	(59667)
10	EA50	(59984)
SQR(.5)	E92D	(59693)
SQR(2)	E932	(59698)
LN(2)	E93C	(59708)
LOG(e)2	EEDB	(61147)
PI/2	F063	(61539)
PI*2	FO6B	(61547)
-32768	EOFE	(57598)
1000000000	ED14[1E9]	(60692[489])

FLOATING POINT FUNCTIONS

SGN EB90 (60304)

Calls SIGN and floats the result in the FAC.

On exit:

FAC=1 If FAC was greater than 0
FAC=0 If FAC was equal to 0
FAC=-1 If FAC was less than 0

ABS EBAF (60335)

Absolute value of FAC

INT EC23 (60451)

Greatest integer value of FAC. Uses QINT and floats the result.

SQR EE8D (61069)

Take the square root of FAC

LOG E941 (59713)

Log base e of FAC

EXP EF09 (61193)

Raise e to the FAC power

RND EFAE (61358)

Form a 'random' number in FAC

COS EFEA (61418)

COS(FAC)

SIN EFF1 (61425)
SIN(FAC)

TAN F03A (61498)
TAN(FAC)

ATN F09E (61598)
ARCTAN(FAC)

FLOATING POINT NUMBER MOVE ROUTINES

MOVFM EAF9 (60153)

Move memory pointed to by Y,A, into FAC. On exit A and Z reflect FACEXP.

MOV2F EB1E (60190)

Pack FAC and move it into temporary register 2. Uses MOVMF. On exit A and Z reflect FACEXP.

MOV1F EB21 (60193)

Pack FAC and move it into temporary register 1. Uses MOVMF. On exit A and Z reflect FACEXP.

MOVML EB23 (60195)

Pack FAC and move it into zero page area pointed to by X. Uses MOVMF. On exit A and Z reflect FACEXP.

MOVMF EB2B (60203)

Pack FAC and move it into memory pointed to by Y,X. On exit A and Z reflect FACEXP.

MOVFA EB53 (60243)

Move ARG into FAC. On exit A=FACEXP and Z is set.

MOVAF EB63 (60259)
Move FAC into ARG. On exit A=FACEXP and Z is set.

CONUPK E9E3 (59875)

Load ARG from memory pointed to by Y,A. On exit A and Z reflect FACEXP.

SUMMARY OF MOVES:

FAC	=> (Y,A)	EB2B
FAC	=> (O,X)	EB23
FAC	=> TEMP 1	EB21
FAC	=> TEMP 2	EB1E
FAC	=> ARG	EB63
(Y,A)	=> FAC	EAF9
(Y,A)	=> ARG	EB63
ARG	=> FAC	EB53

FLOATING POINT UTILITIES

SIGN EB82 (60290)

Set A according to the value of FAC.

On exit:
A=1 if FAC is positive.
A=0 if FAC=0
A=FF if FAC is negative

FOUT ED34 (60724)

Creates a string in FBUFFR equivalent to the value of FAC. On exit Y,A points to the string. The string ends in a zero. FAC is scrambled. Use STROUT to then print the number.

FCOMP EBB2 (60338)

Compare FAC and a packed number in memory pointed to by Y,A.

On exit:
A=1 if (Y,A) < FAC
A=0 if (Y,A) = FAC
A=FF if (Y,A) > FAC

NEGOP EEDO (61136)

FAC= -FAC

FADDH E7A0 (59296)

Add 1/2 to FAC

DIV10 EA55 (59989)

Divide FAC by 10. Returns positive numbers only.

MUL10 EA39 (59961)

Multiply FAC by 10. Works for both positive and negative numbers.

INTEGER TO FAC

SNGFLT E301 (58113)

Float the unsigned integer in Y.

GIVAYF E2F2 (58098)

Float the signed integer in A,Y.

FLOAT EB93 (60307)

Float the signed integer in A.

FAC TO INTEGER

CONINT E6FB (59131)

Convert FAC into a single byte number in X and FACLO. Normally exits through CHRGET. If FAC is greater than 255 or less than 0 then CONINT exits via ILLEGAL QUANTITY ERROR.

AYINT E10C (57612)

If FAC is less than +32767 and greater than -32767 then perform QINT.

QINT EBF2 (60402)

Quick greatest integer function. Leaves INT(FAC) in FACHO, MO, LO signed. QINT assumes FAC < 2 to the 23rd (8388608 decimal)

GETADR E752 (59218)

Convert the number in FAC (-65535 to 65535) into a 2 byte integer (0-65535) in LINNUM.

GETNUM E746 (59206)

Read a 2 byte number into LINNUM from TXTPTR, check for a comma, and get a single byte number in X. On entry TXTPTR points to the first character of the formula for the first number. Uses FRNUM, GETADR, CHKCOM, GETBYT.

COMBYTE E74C (59212)

Check for a comma and get a byte in X. Uses CHKCOM, BETBYT. On entry TXTPTR points to the comma.

TXTPTR TO FAC

FRMEVL DD7B (56699)

Evaluate the formula at TXTPTR using CHRGET and leave the result in FAC. On entry TXTPTR points to the first character of the formula. This is the main subroutine for the commands that use formulas and works for both strings and numbers. If the formula is a string literal, FRMEVL gobbles the opening quote and executes STRLIT and ST2TXT.

FRMNUM DD67 (56679)

Evaluate the formula at TXTPTR, put it in FAC, and make sure it's a number. On entry TXTPTR points to the first character of the formula. TYPE MISMATCH ERROR results if the formula is a string.

FIN EC4A (60490)

Input a floating point number into FAC from CHRGET. FIN assumes that the 6502 registers and A have been set up by the CHRGET that fetched the first digit.

STRING UTILITIES

In Applesoft strings have three parts: the descriptor, a pointer to the descriptor, and the ASCII string. A string descriptor contains the length of the string and the address of its first character. See page 137 of the Applesoft Reference Manual. Through most of the routines the descriptor is left in memory and a pointer is kept in FAC. The pointer is the address of the descriptor. The actual string could be anywhere in memory. In a program, 1A$= "HI" will leave a descriptor pointing into the program text.

CAT E597 (58775)

Concatenate two strings. FACMO,LO point to the first string's descriptor and TXTPTR points to the '+' sign.

STRINI E3D5 (58325)

Get space for creation of a string and create a descriptor for it in DSCTMP. On entry A=length of the string.

STRSPA E3DD (58333)

JSR to GETSPA and store the pointer and length in DSCTMP.

COPY DAB7 (55991)

Free the string temporary pointed to by Y,A and move it to the memory pointed to by FORPNT.

MOVINS E5D4 (58836)

Move a string whose descriptor is pointed to by STRNG1 to memory pointed to by FRESPA.

MOVSTR E5E2 (58850)

Move the string pointed to by Y,X with a length of A to memory pointed to by FRESPA.

STRTXT DE81 (56961)

Sets Y,A equal to TXTPTR plus C and falls into STRLIT.

STRLIT E3E7 (58343)

Store a quote in ENDCHR and CHARAC so that STRLT2 will stop on it.

STRLT2 E3ED (58349)

Take a string literal whose first character is pointed to by Y,A and build a descriptor for it. The descriptor is built in DSCTMP, but PUTNEW transfers it into a temporary and leaves a pointer to it in FACMO,LO. Characters other than zero that terminate the string should be saved in CHARAC and ENDCHR. Leading quotes should be skipped before STRLT2. On exit the character after the string literal is pointed to by STRNG2. Falls into PUTNEW.

PUTNEW E42A (58410)

Some string function is returning with a result in DSCTMP. Move DSCTMP to a temporary descriptor, put a pointer to the descriptor in FACMO,LO, and flag the result as a string.

GETSPA E452 (58450)

Get space for character string. May force garbage collection. Moves FRESPC and FRETOP down enough to store the string. On entry A= number of characters. Returns with A unaffected and pointer to the space in Y,X, FRESPC, and FRETOP. If there's no space then OUT OF MEMORY error.

FRESTR E5FD (58877)

Make sure that the last FAC result was a string and fall into FREFAC.

FRETMP E604 (58884)

Free up a temporary string. On entry the pointer to the descriptor is in Y,A. A check is made to see if the descriptor is a temporary one allocated by PUTNEW. If so, the temporary is freed up by updating TEMPPT. If a temp is freed up a further check is made to see if the string is the lowest in memory. If so, that area of memory is freed up also by updating FRETOP. On exit the address of the string is in INDEX and Y,X and the string length is in A.

FRETMS E635 (58933)

Free the temporary descriptor without freeing up the string. On entry Y,A point to the descriptor to be freed. On exit Z is set if anything was freed.

DEVICE INPUT ROUTINES

INLIN D52C (54572) (No prompt)
INLIN+2 D52E (54574) (Use character in X for prompt)

Input a line of text from the current input device into the input buffer, BUF, and fall into GDBUFS.

GDBUFS D539 (54585)

Puts a zero at the end of the input buffer, BUF, and masks off the msb on all bytes.

On entry:

X= the end of the input line

On exit:

A=0
X=FF
Y=1

INCHR D553 (54611)

Get one character from the current input device in A and mask off the msb. INCHR uses the main Apple input routines and supports normal handshaking.

DEVICE OUTPUT ROUTINES

STROUT DB3A (56122)

Print string pointed to by Y,A. The string must end with a zero or a quote.

STRPRT DB3D (56125)

Print a string whose descriptor is pointed to by FACMO, FACLO.

OUTDO DB5C (56156)

Print the character in A. INVERSE, FLASH, and NORMAL in effect.

CRDO DAFB (56059)

Print a carriage return.

OUTSPC DB57 (56151)

Print a space.

OUTQST DB5A (56154)

Print a question mark.

INPRT ED19 (60697)

Print "IN" and the current line number from CURLIN. Uses LINPRT.

LINPRT ED24 (60708)

Prints the 2 byte unsigned number in X,A.

PRNTFAC ED2E (60718)

Prints the current value of FAC. FAC is destroyed. Uses FOUT and STROUT.

INTERNAL LOCATOR ROUTINES

PTRGET DFE3 (57315)

Read a variable name from CHRGET and find it in memory. On entry TXTPTR points to the first character of the variable name. On exit the address to the value of the variable is in VARPNT and Y,A. If PTRGET can't find a simple variable it creates one. If it can't find an array it creates one dimensioned to 0 to 10 and set all elements equal to zero.

GETARYPT F7D9 (63449)

Read a variable name from CHRGET and find it in memory. On entry TXTPTR points to the first character of the variable name. This routine leaves LOWTR pointing to the name of the variable array. If the array can't be found the result is an OUT OF DATA ERROR.

FNDLIN D61A (54810)

Searches the program for the line whose number is in LINNUM.

On exit:

1. If C set LOWTR points to the link field of the desired line.
2. If C clear then line not found. LOWTR to the next higher line.

DATA D995 (55701)

Move TXTPTR to the end of the statement. Looks for ':' or eol (0).

DATAN D9A3 (55715)

Calculate the offset in Y from TXTPTR to the next ':' or eol (0).

REMN D9A6 (55718)

Calculate the offset in Y from TXTPTR to the next col (Ø).

ADDON D998 (55704)

Add Y to TXTPTR.

INITIALIZATION ROUTINES

SCRTCH D64B (54859)

The 'NEW' command. Clears the program, variables, and stack.

CLEARC D66C (54892)

The 'CLEAR' command. Clears the variables and stack.

STKINI D683 (54915)

Clears the stack.

RESTOR D849 (55369)

Sets the DATA pointer, DATPTR, to the bebinning of the program.

STXTPT D697 (54935)

Set TXTPTR to the beginning of the program.

STORAGE MANAGEMENT ROUTINES

BLTU D393 (54163)

Block transfer makes room by moving everything forward.

On entry:

Y,A and HIGHDS=destination of high address + 1
LOWTR=lowest address to be moved
HIGHTR=highest address to be moved + 1

On exit:

LOWTR is unchanged
HIGHTR=LOWTR – $100
HIGHDS=lowest address transferred – $100

REASON D3E3 (54243)

Makes sure there's enough room in memory, Checks to be sure that the address Y,A is less than FRETOP. May cause garbage collection. Causes OMERR if there's no room.

GARBAG E484 (58500)

Move all currently used strings up in memory as far as possible. This maximizes the free memory area for more strings or numeric variables.

MISCELLANEOUS BASIC COMMANDS

Note that many commands are not documented because they jump into the new statement fetcher and cannot be used as a subroutine.

CONT D898 (55448)

Moves OLDTXT and OLDLIN into TXTPTR and CURLIN.

NEWSTT D7D2 (55250)

Execute a new statement. On entry TXTPTR points to the ':' preceding the statement or the zero at the end of the previous line. Use NEWSTT to restart the program with CONT. *THIS ROUTINE DOES NOT RETURN.*

RUN D566 (54630)

Run the program in memory. *THIS ROUTINE DOES NOT RETURN.*

GOTO D93E (55614)

Uses LINGET and FNDLIN to update TXTPTR. GOTO assumes that the 6502 registers and A have been set up by the CHRGET that fetched the first digit.

LET DA46 (55878)

Uses CHRGET to get address of the variable, '=', evaluate the formula, and store it. On entry TXTPTR points to the first character of the variable name.

HIRES GRAPHICS ROUTINES

NOTE: Regardless of which screen is being displayed, HPAG (location $E6) determines which screen is drawn on. ($20 for HGR, $40 for HGR2)

HGR2 F3D4 (62420)

Initialize and clear page 2 HIRES.

HGR F3DE (62430)

Initialize and clear page 1 HIRES.

HCLR F3EE (62446)

Clear the HIRES screen to black.

BKGND F3F2 (62450)

Clear the HIRES screen to last plotted color.

HPOSN F40D (62477)

Positions the HIRES cursor without plotting, HPAG determines which page the cursor is pointed at.

On entry:

Horizontal=Y,X
Vertical=A

HPLOT F453 (62547)

Call HPOSN then try to plot a dot at the cursor's position. No dot may be plotted if plotting non-white at a complementary color X coordinate.

HLIN F530 (62768)

Draws a line from the last plotted point or line destination to the coordinate in the 6502 registers.

On entry:

Horizontal =X,A
Vertical=Y

HFIND F5CB (62923)

Convert the HIRES cursor's position to X-Y coordinates. Used after SHAPE to find where you've been left.

On exit:

$E0=horizontal lsb
$E1=horizontal msb
$E2=vertical

DRAW F601 (62977)

Draw the shape pointed to by Y,X by inverting the existing color of the dots the shape draws over. On entry A=rotation factor.

SETHCOL F6EC (63213)

Set the HIRES color to X. X must be less than 8.

SHLOAD F775 (63349)

Loads a shape table into memory from tape above MEMSIZ (HIMEM) and sets up the pointer at $E8.

CASSETTE ROUTINES

SAVE D8B0 (55472)

Save the program in memory to tape.

LOAD D8C9 (55497)

Load a program from tape..

VARTIO D8F0 (55536)

Set up A1 and A2 to save 3 bytes ($50 -$52) for the length.

PROGIO D901 (55553)

Set up A1 and A2 to save the program text.

ERROR PROCESSOR ROUTINES

ERROR D412 (54290)

Checks ERRFLG and jumps to HNDLERR if ONERR is active. Otherwise it prints <or> '?' <error message &X> 'ERROR'. If this is during program execution then it also prints 'IN' and the CURLIN.

HANDLERR F2E9 (62185)

Saves CURLIN in ERRLIN, TXTPTR in ERRPOS, X in ERRNUM, and REMSTK in ERRSTK. REMSTK is equal to the 6502 stack pointer and is set up at the start of each statement. X contains the error code. This may be used to interrupt the execution of a BASIC program. See the Applesoft Reference Manual page 136 for the value of X for a given error.

RESUME F317 (62231)

Restores CURLIN from ERRLIN and TXTPTR from ERRPOS and transfers ERRSTK into the 6502 stack pointer.

SYNTAX CHECKING ROUTINES

ISCNTC D858 (55384)

Checks the Apple keyboard for a control – C ($83). Executes the BREAK routine if there is a control – C.

CHKNUM DD6A (55682)

Make sure FAC is numeric. See CHKVAL.

CHKSTR DD6C (56684)

Make sure FAC is a string. See CHKVAL.

CHKVAL DD6D (56685)

Checks the result of the most recent FAC operation to see if it is a string or numeric variable. A TYPE MISMATCH ERROR results if FAC and C don't agree.

On entry:

C set checks for strings
C clear checks for numerics

ERRDIR E306 (58118)

Causes ILLEGAL DIRECT ERROR if the program isn't running. X is modified.

ISLETC E07D (57469)

Checks A for an ASCII letter ('A' to 'Z'). On exit C set if A is a letter.

PARCHK DEB2 (57010)

Checks for '(', evaluates a formula, and checks for ')'. Uses CHKOPN and FRMEVL then falls into CHKCLS.

CHKCLS DEB8 (57016)

Checks at TXTPTR for ')'. Uses SYNCHR.

CHKOPN DEBB (57019)

Checks at TXTPTR for '(', Uses SYNCHR.

CHKCOM DEBE (50722)

Checks at TXTPTR for ','. Uses SYNCHR.

SYNCHR DECO (57024)

Checks at TXTPTR for the character in A. TXTPTR is not modified. Normally exits through CHRGET. Exits with SYNTAX ERROR if they don't match.

XDRAW F65D (62977)

Draw the shape pointed to by Y, X by inverting the existing color of the dots the shape draws over. On entry, A=rotation factor.

—A—		
A1	3C,3D	12
A2	3E,3F	12
ABS	EBAF	14
ADDON	D998	16
ARYTAB	6B,6C	12
ATN	F09E	14
AYINT	E10C	14
—B—		
BKGND	F3F2	17
BLTU	D393	16
BUF	200-2FF	12
—C—		
CAT	E597	15
CHARAC	OD	12
CHKCLS	DEB8	17
CHKCOM	DEBE	17
CHKNUM	DD6A	17
CHKOPN	DEBB	17
CHKSTR	DD6C	17
CHKVAL	DD6D	17
CHRGET	00B1	13
CHRGOT	00B7	13
CLEARC	D66C	16
COMBYTE	E74C	15
CONINT	E6FB	14
CONT	D898	16
CONUPK	E9E3	14
COPY	DAB7	15
COS	EFEA	14
CRDO	DAFB	16
CURLIN	75,76	12
—D—		
DATA	D995	16
DATAN	D9A3	16
DATLIN	7B,7C	12
DATPTR	7D,7E	12
DIV10	EA55	14
DRAW	F601	17
DSCTMP	9D-9F	12
—E—		
ENDCHR	OE	12
ERRDIR	E306	17
ERRFLG	D8	12
ERRLIN	DA,DB	12
ERRNUM	DE	12
ERROR	D412	17
ERRPOS	DC,DD	12
ERRSTK	DF	12
EXP	ER09	14
—F—		
FADD	E7BE	13
FADDH	E7A0	14
FBUFFR	100-1FF	12
FCOMP	EBB2	14
FDIV	EA66	13
FIN	EC4A	15
FIRST	FO	12
FLOAT	EB93	14
FMULT	E97F	13
FNDLIN	D61A	16
FORPNT	85,86	12
FOUT	ED34	14
FPWRT	EE97	13
FREFAC	E600	15
FRESPC	71,72	12
FRESTR	E5FD	15
FRETMP	E604	15
FRETOP	6F,70	12
REMEVL	DD7B	15
FRMNUM	DD67	15
FSUB	E7A7	13
—G—		
GARBAG	E484	16
GDBUFS	D539	15
GETADR	E752	15
GETARYPT	F7D9	16
GTBYTC	E6F5	13
GETBYT	E6F8	13
GETNUM	E746	15
GETSPA	E452	15
GIVAYF	E2F2	14
GOTO	D93E	16
—H—		
H2	2C	12
HANDLERR	F2E9	17
HCLR	F3EE	17
HFIND	F5CB	17
HFNS	F6B9	13
HGR	F3DE	17
HGR2	F3D4	17
HIGHDS	94,95	12
HIGHTR	96,97	12
HLIN	F530	17
HPAG	E6	12
HPLOT	F453	17
HPOSN	F40D	17
—I—		
INDEX	5E,5F	12
INCHR	D553	15
INLIN	D52C	15
INLIN+2	D52E	15
INPRT	ED19	16
INT	EC23	14
INVFLG	32	12
ISCNTC	D858	17
ISLETC	E07D	17
—L—		
LASTPT	53	12
LET	DA46	16
LINGET	DAOC	13
LINNUM	50,51	12
LINPRT	ED24	16
LOAD	D8C9	17
LOG	E941	14
LOWTR	9B,9C	12
—M—		
MEMSIZ	73,74	12
MOV1F	EB21	14
MOV2F	EB1E	14
MOVAF	EB63	14
MOVFA	EB53	14
MOVFM	EAF9	14
MOVINS	E5D4	15
MOVMF	EB2B	14
MOVML	EB23	14
MOVSTR	E5E2	15
MUL10	EA39	14
—N—		
NEGOP	EEDO	14
NEWSTT	D7D2	16
—O—		
OLDLIN	77,78	12
ORMASK	F3	12
OUTDO	DB5C	16
OUTQST	DB5A	16
OUTSPC	DB57	16
—P—		
PARCHK	DEB2	17
PLOTFNS	F1EC	13
PRGEND	AF,BO	12
PROGIO	D901	17
PRTFAC	ED2E	16
PTRGET	DFE3	16
PUTNEW	E42A	15
—Q—		
QINT	EBF2	14
—R—		
REASON	D3E3	16
REMN	D9A6	16
REMSTK	F8	12
RESTOR	D849	16
RESUME	F317	17
RND	EFAE	14
RUN	D566	16
—S—		
SAVE	D8BO	17
SCRTCH	D64B	16
SETHCOL	F6EC	17
SGN	EB80	14
SHLOAD	F775	17
SIGN	EB82	14
SIN	EFF1	14
SNGFLT	E301	14
SPDBYT	F1	12
SQR	EE8D	14
STKINI	D683	16
STREND	6D,6E	12
STRINI	E3D5	15
STRLIT	E3E7	15
STRLT2	E3ED	15
STRNG1	AB,AC	12
STRNG2	AD,AE	12
STROUT	DB3A	16
STRPRT	DB3D	16
STRSPA	E3DD	15
STRTXT	DE81	15
STXTPT	D697	16
SUBFLG	14	12
SYNCHR	DECO	17
—T—		
TAN	F03A	14
TEMPPT	52	12
TXTTAB	67,68	12
—V—		
V2	2D	13
VALTYP	11	13
VARPNT	83,84	13
VARTAB	69,6A	13
VARTIO	D8FO	17
—X—		
XDRAW	F65D	17

NEW FROM MOUNTAIN HARDWARE. CONTROL FROM YOUR APPLE.

INTROL/X-10.

COMPUTERIZE YOUR HOME.
The Introl/X-10 peripheral system for your Apple* Computer allows you to remotely control lights and electrical appliances in your home.

YOU'RE ALREADY WIRED.
Introl/X-10 operates by utilizing your computer's intelligence to command the BSR System X-10 to send signals over regular 110 volt household wiring. That means you can control any electrical device in your home without additional wiring.

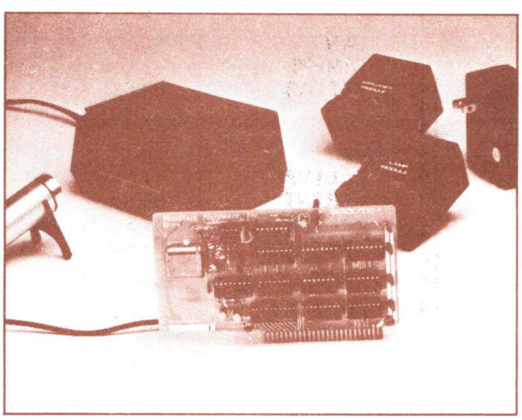
READY TO USE.
Introl/X-10 comes with complete software to control devices on pre-determined schedules, and features:
• Control devices at a specific time. • Select a daily or weekly schedule. • Specify a day of the week, or an exact date for a particular event. • Specify an interval of time for an event. • Rate device wattages for a running account of power consumption during your schedule for energy management. • Used with our Apple Clock™ your schedules may run in "background" while other programs may run at the same time in "foreground."

EVERYTHING YOU NEED.
The Introl Controller board plugs into a peripheral slot of your Apple. With an ultrasonic transducer it transmits control signals to the BSR/X-10 Command Console which may be plugged into any convenient AC outlet near your computer. On command, signals are sent to remote modules located at the devices you wish to control. Up to 16 remote module addresses may be controlled from your Apple. Software requires Applesoft firmware.

AVAILABLE NOW.
The Introl/X-10 System consists of the Introl Controller board with timer and ultrasonic transducer, the X-10 Command Console and three remote modules. $279. Complete and tested. If you already have a BSR System X-10, the Introl Controller board is available separately for $189. Additional remote modules are available at $15. See your computer dealer for a demonstration. Or, return the coupon below for complete information.

Available through computer dealers worldwide

*Apple is a trademark of Apple Computer Inc.
BSR/System X-10 is a trademark of BSR, Ltd.



Mountain Hardware, Inc.
LEADERSHIP IN COMPUTER PERIPHERALS
300 Harvey West Blvd., Santa Cruz, CA 95060
(408) 429-8600

Sounds great.
☐ Home control from my Apple?
That sounds like a great system. Send me all the details.

Name ____________________

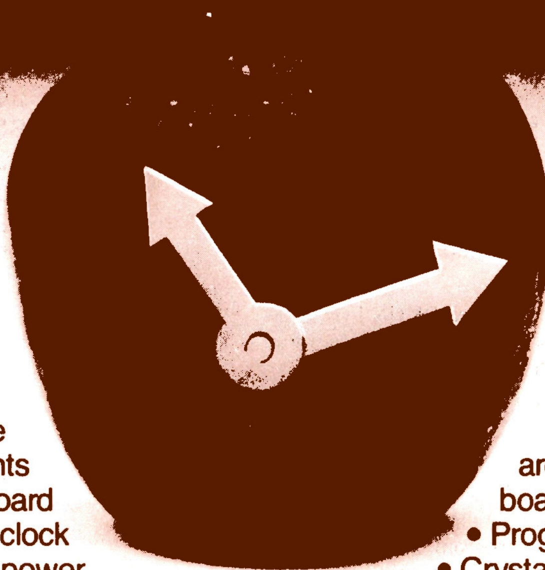
Address ____________________

City ____________ State ________ Zip ______

Phone ____________________

NEW FROM MOUNTAIN HARDWARE.

THE APPLE CLOCK.



NEW UTILITY FOR YOUR COMPUTER.
Until now, there hasn't been a Real-Time Clock for the Apple II*. The Apple Clock from Mountain Hardware keeps time and date in 1mS increments for over one year. On-board battery backup keeps the clock running in the event of power outage. Software controlled interrupts are generated by the clock. That means you can call up schedules, time events, date printouts ...all in real time on a programmed schedule.

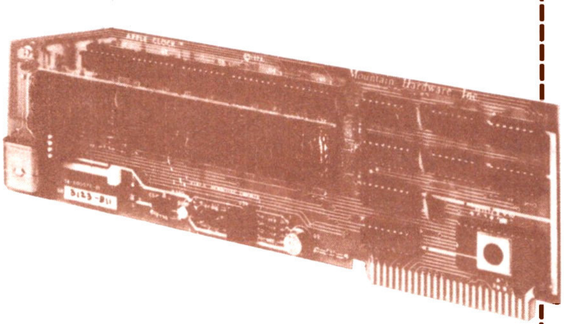
EASY TO USE.
The Apple Clock is easily accessed from BASIC using routines carried in on-board ROM. With it, you can read time and program time-dependent functions for virtually any interval. From milliseconds to days, months or a year.

PLUG IN AND GO.
Plug the Apple Clock into a peripheral slot on your Apple II and you're ready to go.

FEATURES.
- Time and date in 1mS increments for periods as long as one year.
- Software for calendar and clock routines, as well as an event timer are contained on on-board ROM.
- Program interrupts.
- Crystal controlled accuracy of ±.001%.
- On-board battery backup keeps your clock in operation even during power outage.

REAL TIME AT THE RIGHT PRICE.
At $199 assembled and tested, it's the clock your Apple has been waiting for. And, it's available now through your Apple dealer. Drop in for a demonstration. Or return the coupon below.

A COMPLETE LINE.
Mountain Hardware also offers a complete line of peripheral products for many fine computers.



Mountain Hardware, Inc.
300 Harvey West Blvd., Santa Cruz, CA 95060
(408) 429-8600

Sounds great.

____ Send me everything I'll need to know about the Apple Clock.
____ I've got a ____________ computer. Send me information.

Name ____________________

Address ____________________

City __________ State ______ Zip ______

*Apple II is a trademark of Apple Computer, Inc.

THE &LOMEM: UTILITY

by Neil Konzen

From Call-A.P.P.L.E.

This program solves the problem of finding a place in memory for machine language programs in Applesoft. Until now, programs were usually written to reside above HIMEM:, which has its drawbacks: a program written for a 48K machine would not run on 32K. Some programs, like RENUM/MERGE on the DOS 3.2 master disk, relocate themselves to accommodate different size systems, but this is often more work and hassle than the program is worth.

&LOMEM: solves this problem in a unique way: it actually moves the Applesoft program upward in memory, freeing up the memory left behind. Do not confuse &LOMEM: (pronounced Ampersand LOMEM) with the Applesoft LOMEM: statement – Applesoft LOMEM: affects only the location of Applesoft variables, whereas &LOMEM: affects the location of the variables *AND* program.

To use the &LOMEM: utility, you must first BRUN LOMEM: to initialize the Ampersand jump vector and load it into memory. The syntax for the &LOMEM: statement is as follows:

&LOMEM: [addr] , where [addr] is the new start of program address in decimal.

&LOMEM: can be used within programs or in immediate mode. The program does an implicit "CLEAR", which wipes out all variables. The memory freed up by &LOMEM: will always start at $800 hex (the normal start of program address). So, a program line to reserve 2K of memory for a machine language program assembled to run at $800 hex, would look something like this:

10 PRINT CHR$(4) "BRUN LOMEM:":&LOMEM: 4096

Since the freed up memory space is the same as that freed up with the LOMEM: statement in Integer Basic, many machine language programs originally written for Integer Basic can now be easily adapted for use with Applesoft.

There are a few things that should be kept in mind when using this program. First of all, this program will *NOT* work with RAM Applesoft, since it calls routines within Applesoft itself. Ampersand LOMEM: can only move programs *UPWARD* in memory; attempting to move the program back to its original location or to a lower address will *DESTROY* your program. The action of Ampersand LOMEM: statement is permanent; "NEW", "CLEAR", and "LOMEM" have no effect on the start of program address. The start address of another program loaded after the execution of a &LOMEM: statement will also be the value set by &LOMEM:. Use the DOS FP command to reset the program address to its normal $800.

This program was originally written for Ron and Darrell Aldrich's Hi-Res Text Generator to allow its use in ROM Applesoft systems of any RAM size. The program as presented here is identical to the program included on the Hi-Res Text Generator disk, from *Apple Pugetsound Program Library Exchange.* The Ampersand LOMEM: resides at $330 hex, and uses approximately 160 bytes. (This is because the Hi-Res Text Generator uses other locations in the $300 page.)

&LOMEM:'s potential uses are unlimited. Dig in and let us know the unique applications you turn up!

```
*330.3CF

0330- A9 4C 8D F5 03 A9 40 8D
0338- F6 03 A9 03 8D F7 03 60
0340- A9 A4 20 C0 DE 20 67 DD
0348- 20 52 E7 A5 67 85 96 A5
0350- B0 85 97 C6 96 38 A5 50
0358- 85 94 E5 67 85 50 A5 51
0360- 85 95 E5 68 85 51 A5 AF
0368- E5 67 A8 A5 B0 E5 68 AA
0370- 18 65 95 85 95 C8 D0 01
0378- E8 E8 C8 20 C3 D3 A2 69
0380- 20 C1 03 A2 AF 20 C1 03
0388- A2 67 20 C1 03 A2 79 20
0390- C1 03 A5 B9 C9 02 F0 05
0398- A2 B8 20 C1 03 A5 67 A4
03A0- 68 85 5E 84 5F A0 00 38
03A8- B1 5E 65 50 91 5E AA C8
03B0- B1 5E F0 0A 65 51 91 5E
03B8- 86 5E 85 5F 90 E7 4C 6C
03C0- D6 38 B5 00 65 50 95 00
03C8- B5 01 65 51 95 01 60 FF
```

```
*
* & LOMEM: X
*
* BY NEIL KONZEN
*
             ORG $330
             OBJ $330
*
* EQUATES
*
LOMEM        EQU $50
RNMPTR       EQU $5E
PRGBEG       EQU $67
VARTAB       EQU $69
FRETOP       EQU $73
OLDTXT       EQU $79
DATXT        EQU $7D
HIGHDS       EQU $94
HIGHTR       EQU $96
PRGEND       EQU $AF
TXTPTR       EQU $B8
AMPRSND      EQU $3F5
*
*
```

SUBROUTINES

```
                  *
                  CHRGET    EQU $00B1
                  BLT2      EQU $D3C3     ;PART OF FP BLOCK TRANSFER
                  CLEAR     EQU $D66C     ;'CLEAR' STMT ENTRY
                  FRMEVL    EQU $DD67     ;EVALUATE A FORMULA
                  SYNCHK    EQU $DEC0     ;COMPARE A W/(TXTPTR)
                  SNERR     EQU $DEC9
                  FRMNUM    EQU $E752
                  *
0330: A9 4C                 LDA #$4C      ;SET UP AMPERSAND VECTOR
0332: 8D F5 03              STA AMPRSND
0335: A9 40                 LDA #<LOMEM:
0337: 8D F6 03              STA AMPRSND+1
033A: A9 03                 LDA #>LOMEM:
033C: 8D F7 03              STA AMPRSND+2
033F: 60                    RTS
0340: A9 A4       LOMEM:    LDA #$A4      ;'LOMEM:' TOKEN?
0342: 20 C0 DE              JSR SYNCHK    ;SYNTAX CHECKING HERE
0345: 20 67 DD              JSR FRMEVL    ;OK, SO GET VALUE
0348: 20 52 E7              JSR FRMNUM
                  *
                  *
034B: A5 67                 LDA PRGBEG
034D: 85 96                 STA HIGHTR
034F: A5 B0                 LDA PRGEND+1  ;FP MOVE ROUTINE USED
0351: 85 97                 STA HIGHTR+1  ;IN A VERY BIZZARE WAY!
0353: C6 96                 DEC HIGHTR
                  *
0355: 38                    SEC
0356: A5 50                 LDA LOMEM
0358: 85 94                 STA HIGHDS
035A: E5 67                 SBC PRGBEG
035C: 85 50                 STA LOMEM
035E: A5 51                 LDA LOMEM+1
0360: 85 95                 STA HIGHDS+1
0362: E5 68                 SBC PRGBEG+1
0364: 85 51                 STA LOMEM+1
                  *
0366: A5 AF                 LDA PRGEND
0368: E5 67                 SBC PRGBEG
036A: A8                    TAY
036B: A5 B0                 LDA PRGEND+1
036D: E5 68                 SBC PRGBEG+1
036F: AA                    TAX
0370: 18                    CLC
0371: 65 95                 ADC HIGHDS+1
0373: 85 95                 STA HIGHDS+1
0375: C8                    INY           ;STRANGE MATH HERE BECAUSE
0376: D0 01                 BNE *+3       ;WE HAVE TO MOVE (PRGBEG)-1
0378: E8                    INX           ;AND 'BLT2' IS CALLED FUNNY
0379: E8                    INX
037A: C8                    INY
037B: 20 C3 D3              JSR BLT2
                  *
037E: A2 69       SKP5      LDX #VARTAB   ;NOW GO UPDATE THESE PTRS
0380: 20 C1 03              JSR ADD
                  *
0383: A2 AF                 LDX #PRGEND
0385: 20 C1 03              JSR ADD
                  *
```

```
0388: A2 67              LDX #PRGBEG
038A: 20 C1 03           JSR ADD
                    *
038D: A2 79              LDX #OLDTXT
038F: 20 C1 03           JSR ADD
                    *
0392: A5 B9              LDA TXTPTR+1
0394: C9 02              CMP #$02
0396: F0 05              BEQ DONT
0398: A2 B8              LDX #TXTPTR
039A: 20 C1 03           JSR ADD
                    *
039D: A5 67         DONT     LDA PRGBEG
039F: A4 68              LDY PRGBEG+1
03A1: 85 5E              STA RNMPTR
03A3: 84 5F              STY RNMPTR+1
                    *
03A5: A0 00         RENUM    LDY #0        ;FIX 'NEXT LINE' ADDRESSES
03A7: 38                 SEC           ;(ASSUMING THEY'RE ALREADY OKAY)
03A8: B1 5E              LDA (RNMPTR),Y
03AA: 65 50              ADC LOMEM
03AC: 91 5E              STA (RNMPTR),Y
03AE: AA                 TAX
03AF: C8                 INY
03B0: B1 5E              LDA (RNMPTR),Y
03B2: F0 0A              BEQ DONE
03B4: 65 51              ADC LOMEM+1
03B6: 91 5E              STA (RNMPTR),Y
03B8: 86 5E              STX RNMPTR
03BA: 85 5F              STA RNMPTR+1
03BC: 90 E7              BCC RENUM
03BE: 4C 6C D6      DONE     JMP CLEAR     ;NOW LET APPLESOFT DO THE REST
                    *
03C1: 38            ADD      SEC           ;ADD LOMEM+1 TO ZERO
03C2: B5 00              LDA $00,X     ;ZERO PAGE LOC IN X
03C4: 65 50              ADC LOMEM
03C6: 95 00              STA $00,X
03C8: B5 01              LDA $1,X
03CA: 65 51              ADC LOMEM+1
03CC: 95 01              STA $1,X
03CE: 60                 RTS
```

```
--- END ASSEMBLY ---

TOTAL ERRORS: 00

159 BYTES GENERATED THIS ASSEMBLY
```

```
          --- SYMBOL TABLE ---

LOMEM    $50          RNMPTR   $5E
PRGBEG   $67          VARTAB   $69
FRETOP   $73          OLDTXT   $79
DATXT    $7D          HIGHDS   $94
HIGHTR   $96          PRGEND   $AF
TXTPTR   $B8          AMPRSND  $03F5
CHRGET   $B1          BLT2     $D3C3
CLEAR    $D66C        FRMEVL   $DD67
SYNCHK   $DEC0        SNERR    $DEC9
FRMNUM   $E752        LOMEM:   $0340
SKP5     $037E        DONT     $039D
RENUM    $03A5        DONE     $03BE
ADD      $03C1
```

From applesauce

CONNECTING WITH THE USCD BIOS

by Randall Hyde

After purchasing my language system, my first goal was getting it to work with my ComputerWorld printer interface. For those of you who are unfamiliar with the Pascal system, let me mention that it only supports Apple peripheral cards. For those of you unfamiliar with the ComputerWorld printer interface, it only costs $80 (which is $100 less than the Apple printer interface). Since I have a ComputerWorld interface I certainly did not feel like spending an additional $180 just so I could use my printer with Pascal. So the obvious solution was to "patch" the existing system so that it would recognize, and utilize, the ComputerWorld printer interface.

After two weeks of pulling my hair out trying to dissassemble the UCSD BIOS (BIOS stands for Basic Input Output System) Dave Smith (of ComputerWorld) turned me on to an application note on the BIOS which was given to him by Apple Computer Inc. This Ap note, among other things, gave a source listing of the BIOS as well as directions on patching your own drivers to the BIOS. This Ap note should have been distributed to each of the local clubs as part of the "Introductory Package" sent out by the International Apple Corp. Ask your local club director for a copy if you're interested. Anyway, with this application note in hand I proceeded to add my printer driver to the UCSD BIOS. The only problem I encountered was the fact that the free memory mentioned in the application note is not really free. So I encountered quite a few problems when trying to figure out where to put my driver routines. Since I have a Mountain Hardware ROM Plus board, I decided to store my drivers in ROM and utilize the ROM Plus' capabilities. While burning the first version of my driver routine it occurred to me that in addition to the printer driver I should also take advantage of the ROM Plus lowercase entry capabilities as well as the lower case display capabilities of the Dan Paymar lower case mod. So, back to the BIOS source listings to incorporate the required patches. Since I wanted this for use by the Pascal System, I needed the capability to input such characters as "[", "]", "(", ")", etc. To allow this I not only hooked up the shift key (to TTL Input #1) but I also wired up the control key to TTL input #2 (pin #4 on the ROM Plus connector). Now, by pressing the shift key and the control key simultaneously I can input the full 96 upper/lowercase/special character ASCII character set!

Some other things I added include a "CAPSLOCK" feature (toggled by pressing control – R) and of course I allow the use of the shift key when using the Pascal System for upper and lower case entry. This "filter'. routine can be used anywhere a "LDA $C000" instruction is used. Simply replace the LDA $C000 with a JSR $C800 (assuming of course that the ROM Plus board is selected) and upon return, if the accumulator is negative, then a key has been pressed, otherwise no key has been pressed. The listing for this routine appears in listing #l (the assembler used is LISA), The routine is called "CONSOLE".

Also included in listing one is the initialization procedure for the ComputerWorld interface card as well as the printer write routine for the ComputerWorld interface card. Note that these routines are assembled at different pages in the expansion ROM memory space to allow customization for the user's particular needs.

Once these routines are burned into ROM I have made the assumption that this ROM will be stuck in the number two ROM socket. Once the ROM is placed in this socket these routines are available for use by the Pascal system. There is one problem however. The UCSD BIOS does not know that these drivers even exist. This means that we have to actually go in and change certain locations in the BIOS to tell it when to use these routines. This problem is handled by a special program called a "SYSGEN" program. This SYSGEN program goes in and modifies the Apple Pascal BIOS so that the printer card is recognized and the lower case modifications can take place.

Typically there are two ways to write a SYSGEN program. You can write it as a "one-shot" which you execute once and it modifies the BIOS on the Pascal disk. Or you can write it as a "dynamic SYSGEN" which must be executed each time the disk is booted. The one-shot method has the obvious advantage in that you don't have to worry about running the SYSGEN program everytime you boot your Pascal disks. It has the not-so-obvious disadvantage in that it can possibly make your Pascal disks incompatible with other systems. Since compatibility is a major concern I chose to use the dynamic method for my SYSGEN program.

Listing #2 is the listing for the SYSGEN program. This program is written in 6502 assembly language using the Pascal Adaptable Assembler. This routine simply turns off the language card write protect and proceeds to overwrite portions of the BIOS with the patches for use with the drivers I had stored previously in the ROM Plus. Other patches are made to allow lower case display capability, as well as to allow the Pascal system to recognize the ComputerWorld ROM.

Since the Pascal system cannot execute an assembly language program directly (at least to my knowledge), a short Pascal program appears in listing #3. After its compilation you must link this Pascal program to the previously assembled SYSGEN routine.

Once the above steps have been taken, you can simply "eXecute" the SYSGEN program at the Pascal command level, and presto! Lowercase capability and printer capability are yours.

I will attempt to answer any questions you may have about the BIOS, simply write:

RANDY HYDE
c/o APPLESAUCE
12804 MAGNOLIA
CHINO, CA 91710

```
0800              ;
0800              ;
0800              ;
0800              ; CONSTANTS ETC.
0800              ;
0800              CNTRL.R  EQU $92
0800              CNTRL.K  EQU $8B
0800              LINEFEED EQU $8A
0800              RETURN   EQU $8D
0800              SHFTMASK EQU $20
0800              CNTLMASK EQU $10
0800              TRUE     EQU $1
0800              FALSE    EQU $0
0800              ;
0800              ;
0800              ; SPECIAL LOCATIONS
0800              ;
0800              ROMPLUS  EQU $C0A0          ;ASSUMED IN SLOT #2
0800              SHIFTFLG EQU ROMPLUS        ;LOCATION OF SHIFT FLAG
0800              CNTRLFLG EQU ROMPLUS        ;LOCATION OF CTRL FLAG
0800              KEYBOARD EQU $C000          ;KEYBOARD PORT
0800              KEYSTRB  EQU $C010          ;KEYBOARD CLEAR PORT
0800              ;
0800              ; VARIABLES:
0800              ;
0800              KEYSAVE  EQU $CF00          ;CHAR SAVE AREA (IN ROM+)
0800              CAPSLOCK EQU $CF01          ;SHIFTLOCK BOOLEAN
0800              ;
0800              ;
0800              ;
0800              ;
0800              ;
0800              ;
0800                       PAG
C800                       ORG $C800
C800                       OBJ $1000
C800              ;
C800              ;
C800              ; UCSD PASCAL LOWER CASE CONSOLE
C800              ; DRIVER.  FOR PLACEMENT IN
C800              ; MOUNTAIN HARDWARE'S ROM PLUS.
C800              ;
C800              ;
C800              CONSOLE:
C800 AD00C0                LDA KEYBOARD
C803 3001                  BMI CNSL1
C805              ;
C805              ; NO KEY, SO RETURN
C805              ;
C805 60                    RTS
C806              ;
C806              ;
C806              ; KEY WAS PRESSED, SAVE STATUS
C806              ;
C806 08           CNSL1    PHP
C807              ;
```

```
C807            ;
C807            ; CLEAR KEYBOARD PORT
C807            ;
C807 2C10C0             BIT KEYSTRB
C80A            ;
C80A            ;
C80A            ; TEST FOR CONTROL-R (SHIFT/CAPS LOCK)
C80A            ;
C80A 8D00CF             STA KEYSAVE        ;SAVE INPUT CHAR
C80D C992               CMP #CNTRL.R
C80F D00E               BNE CNSL2
C811            ;
C811            ; NOW TOGGLE CAPS LOCK MODE AND
C811            ; RETURN
C811            ;
C811 AD01CF             LDA CAPSLOCK
C814 4901               XOR #TRUE
C816 2901               AND #TRUE
C818 8D01CF             STA CAPSLOCK
C81B 28                 PLP                ;FIX STACK
C81C A900               LDA #$0            ;FILL ACC WITH NON-NEG #
C81E 60                 RTS
C81F            ;
C81F            ; TEST THE CURRENT CHARACTER
C81F            ;
C81F            CNSL2:
C81F            ;
C81F            ; SEE IF SHIFT IS PRESSED
C81F            ;
C81F A920               LDA #SHFTMASK
C821 2CA0C0             BIT SHIFTFLG
C824 D06B               BNE NORMAL
C826            ;
C826            ; NOW TEST FOR SIMULTANEOUS CTRL KEY
C826            ;
C826 A910               LDA #CNTLMASK
C828 2CA0C0             BIT CNTRLFLG
C82B D031               BNE TSTSHFT
C82D            ;
C82D            ; CONTROL-SHIFT CHAR HERE
C82D            ;
C82D A20C               LDX #12            ;INIT FOR 12 CHARS
C82F AD00CF             LDA KEYSAVE
C832 DD44C8     CSLOOP  CMP SPEC,X
C835 F008               BEQ FND1
C837 CA                 DEX
C838 10F8               BPL CSLOOP
C83A            ;
C83A            ; IGNOR C-S COMBINATION AND PASS
C83A            ; CHARACTER UNCHANGED
C83A            ;
C83A AD00CF             LDA KEYSAVE
C83D 28                 PLP
C83E 60                 RTS
C83F            ;
C83F            ;
C83F            ; SPECIAL CHARACTER WAS DETECTED,
C83F            ; CONVERT AND LEAVE
```

```
C83F               ;
C83F BD51C8        FN01      LDA SPECVAL,X
C842 28                      PLP
C843 60                      RTS
C844               ;
C844 A1A2A3        SPEC      ASC "!""#$%&'()=?><"
C847 A4A5A6
C84A A7A8A9
C84D BDBFBE
C850 BC
C851 FCDE          SPECVAL   ASC "|^"
C853 FF                      BYT $FF
C854 C0A5A6                  ASC "@%&`{}_\][" 
C857 E0FBFD
C85A DFDCDD
C85D DB
C85E               ;
C85E               ;
C85E               ;
C85E               ;
C85E               ; SHIFT KEY IS PRESSED HERE
C85E               ;
C85E               TSTSHFT:
C85E AD01CF                  LDA CAPSLOCK
C861 D01D                    BTR NOCHNG
C863               ;
C863 A202                    LDX #2               ;INIT FOR "@", "]", & "^"
C865 AD00CF                  LDA KEYSAVE
C868 DD7AC8        SFTLOOP   CMP SHFTC,X
C86B F008                    BEQ FND2
C86D CA                      DEX
C86E 10F8                    BPL SFTLOOP
C870               ;
C870               ; AT THIS POINT, LEAVE THE CHARACTER
C870               ; ALONE
C870               ;
C870 AD00CF                  LDA KEYSAVE
C873 28                      PLP
C874 60                      RTS
C875               ;
C875               ; NOW A "@", "]", OR "^" HAS BEEN
C875               ; ENCOUNTERED. CONVERT TO "P", "M", OR "N"
C875               ;
C875 BD7DC8        FND2      LDA SHFTCC,X
C878 28                      PLP
C879 60                      RTS
C87A               ;
C87A               ;
C87A C0DDDE        SHFTC     ASC "@]^"
C87D D0CDCE        SHFTCC    ASC "PMN"
C880               ;
C880               ;
C880               ; IF YOU GET TO THIS POINT THEN THE
C880               ; CAPSLOCK MODE IS IN EFFECT, SO TREAT
C880               ; THE CHARACTER EXACTLY AS IT COMES
C880               ; FROM THE APPLE KEYBOARD.
C880               ;
C880               NOCHNG:
```

```
C880 AD00CF                LDA KEYSAVE
C883 C98B                  CMP #CNTRL.K          ;CONTROL-K?
C885 D005                  BNE NOCHNG1
C887 A9DB                  LDA #"["              ;CHANGE TO "["
C889 8D00CF                STA KEYSAVE
C88C             ;
C88C AD00CF      NOCHNG1   LDA KEYSAVE
C88F 28                    PLP
C890 60                    RTS
C891             ;
C891             ;
C891             ; NORMAL CHARACTERS HERE (NO SHIFT OR CTRL)
C891             ;
C891             NORMAL:
C891 AD01CF                LDA CAPSLOCK
C894 D0EA                  BTR NOCHNG
C896             ;
C896             ; IF NOT IN CAPSLOCK MODE, AND
C896             ; CHARACTER IS ALPHABETIC- "UNSHIFT"
C896             ; IT.
C896             ;
C896 AD00CF                LDA KEYSAVE
C899 C9C1                  CMP #"A"
C89B 90E3                  BLT NOCHNG
C89D C9DB                  CMP #"Z"+$1
C89F B0DF                  BGE NOCHNG
C8A1 0920                  ORA #$20
C8A3 8D00CF                STA KEYSAVE
C8A6 4C80C8                JMP NOCHNG
C8A9             ;
C8A9             ;
C8A9             ;
C8A9                       PAG
C900                       ORG $C900
C900                       OBJ $1100
C900             ;
C900             ;
C900             ;
C900             ;*************************************************
C900             ;
C900             ;
C900             ;
C900             ; COMPUTERWORLD (ALIAS CCI OF OC)
C900             ; PRINTER INTERFACE DRIVER
C900             ;
C900             PINIT:
C900 A900                  LDA #0
C902 8D90C0                STA $C090
C905 8D93C0                STA $C093
C908 A904                  LDA #4
C90A 8D91C0                STA $C091
C90D A9FF                  LDA #$FF
C90F 8D92C0                STA $C092
C912 A93C                  LDA #$3C
C914 8D93C0                STA $C093
C917 A200                  LDX #$0
C919 60                    RTS
```

```
C91A                 ;
C91A                 ;
C91A                 ;
C91A                          PAG
CA00                          ORG $CA00
CA00                          OBJ $1200
CA00                 ;
CA00                 ; CCI PI CHAROUT ROUTINE
CA00                 ;
CA00                 PWRIT:
CA00 297F                     AND #$7F          ;STRIP H.O. BIT
CA02 48                       PHA               ;AND SAVE
CA03                 ;
CA03 2081D6          HANDSHK  JSR $D681         ;TEST KEYBOARD
CA06 AD90C0                   LDA $C090
CA09 30F8                     BMI HANDSHK
CA0B                 ;
CA0B                 ; RETRIEVE CHARACTER AND OUTPUT
CA0B                 ; TO PRINTER
CA0B                 ;
CA0B 68                       PLA
CA0C 8D92C0                   STA $C092
CA0F A931                     LDA #$31
CA11 8D93C0                   STA $C093         ;TOGGLE CHAR AVAILABLE
CA14 A93C                     LDA #$3C
CA16 8D93C0                   STA $C093
CA19 60                       RTS
                              END
***** END OF ASSEMBLY
```

```
PAGE      1  LWRCASE    FILE:L.ASM


0000!                              .PROC LWRCASE
Current memory available:        9527
0000!
0000!
0000! AD 83C0                            LDA 0C083      ;TURN ON RAM WRITE
0003! AD 83C0                            LDA 0C083      ;ON LANGUAGE CARD.
0006!
0006!
0006!                             ; ROM PLUS PATCH TO ALLOW LOWER CASE
0006!                             ; INPUT TO PASCAL SYSTEM.
0006!
0006! A0 00                              LDY #0
0008! B9 ****                     LOOP   LDA DATA1,Y
000B! 99 9AD6                            STA 0D69A,Y
000E! C8                                 INY
000F! C0 10                              CPY #10
0011! 90F5                               BCC LOOP
0013!
0013!
0013!                             ; LOWER CASE DISPLAY PATCH.  CONVERT THE
0013!                             ; SBC #$20 TO A BCC *+$4 INSTRUCTION WHICH
0013!                             ; ALLOWS LOWER CASE TO BE DISPLAYED IF
0013!                             ; THE USER HAS A DAN PAYMAR BOARD
0013!
0013! A9 B0                              LDA #0B0
0015! 8D E8D8                            STA 0D8E8
0018! A9 02                              LDA #2
001A! 8D E9D8                            STA 0D8E9
001D!
001D!
001D!                             ; COMPUTERWORLD PRINTER INITIALIZATION
001D!                             ; PATCH TO BIOS.
001D!
001D!
001D! A0 00                              LDY #0
001F! B9 ****                     LOOP2  LDA DATA2,Y
0022! 99 E0DA                            STA 0DAE0,Y
0025! C8                                 INY
0026! C0 0E                              CPY #0E
0028! 90F5                               BCC LOOP2
002A!
002A!
002A!                             ; PATCH UP A JUMP TO THE ABOVE PRINTER
002A!                             ; INIT ROUTINE.
002A!
002A! A9 4C                              LDA #4C
002C! 8D 8AD7                            STA 0D78A
002F! A9 E0                              LDA #0E0
0031! 8D 8BD7                            STA 0D78B
0034! A9 DA                              LDA #0DA
0036! 8D 8CD7                            STA 0D78C
0039!
0039!                             ; MOVE THE PRINTCHAR ROUTINE INTO BIOS
0039!
0039! A0 00                              LDY #0
```

```
003B| B9 ****          LOOP3  LDA DATA3,Y
003E| 99 F0DA                 STA 0DAF0,Y
0041| C8                      INY
0042| C0 0F                   CPY #0F
0044| 90F5                    BCC LOOP3
0046|
0046|
0046|                  ; PATCH IN A JUMP TO THE PRINT CHAR
0046|                  ; ROUTINE.
0046|
0046| A9 4C                   LDA #4C
0048| 8D 11D8                 STA 0D811
004B| A9 F0                   LDA #0F0
004D| 8D 12D8                 STA 0D812
0050| A9 DA                   LDA #0DA
0052| 8D 13D8                 STA 0D813
0055|
0055|
0055|                  ; FIX UP CARD RECOGNITION BYTES TO
0055|                  ; CORRESPOND TO THE COMPUTERWORLD
0055|                  ; INTERFACE.
0055|
0055| A9 48                   LDA #48      ;SHOULD BE 4 FOR NEWER CARDS
0057| 8D 10D6                 STA 0D610
005A| A9 04                   LDA #4       ;SHOULD BE 93 FOR NEWER CARDS
005C| 8D 14D6                 STA 0D614
005F|
005F|                  ; PLACE A 5 IN LOCATION BFF9.  THIS TELLS
005F|                  ; APPLE PASCAL THAT A PRINTER INTERFACE
005F|                  ; CARD IS IN SLOT #1
005F|
005F| A9 05                   LDA #5
0061| 8D F9BF                 STA 0BFF9
0064|
0064|                  ; SET LINE FEED IGNOR MODE
0064|
0064| AD 0FBF                 LDA 0BF0F
0067| 09 80                   ORA #80
0069| 8D 0FBF                 STA 0BF0F
006C|
006C|
006C|                  ; WRITE PROTECT RAM CARD
006C|
006C| AD 88C0                 LDA 0C088
006F| 60                      RTS
0070|
0070|                  ;LOWERCASE INPUT DRIVER ROUTINE
0070|
0009* 7000
0070| AD FF CF         DATA1 .BYTE 0AD,0FF,0CF    ; LDA $CFFF
0073| A9 8A                  .BYTE 0A9,8A       ; LDA #$8A
0075| 8D A0 C0               .BYTE 8D,0A0,0C0   ; STA $C0A0
0078| 20 00 C8               .BYTE 20,0,0C8     ; JSR $C800
007B| 10 3C                  .BYTE 10,3C        ; BPL JDONCK
007D| 29 7F                  .BYTE 29,7F        ; AND #$7F
007F| EA                     .BYTE 0EA          ; NOP
0080|
0080|
```

```
0080!                                   COMPUTERWORLD PRINTER INIT ROUTINE
0080!
0020* 8000
0080! AD FF CF                   DATA2 .BYTE 0AD,0FF,0CF      ; LDA $CFFF
0083! A9 8A                            .BYTE 0A9,8A           ; LDA #$8A
0085! 8D A0 C0                         .BYTE 8D,0A0,0C0       ; STA $C0A0
0088! 20 00 C9                         .BYTE 20,0,0C9         ; JSR $C900
008B! A2 00                            .BYTE 0A2,0            ; LDX #$0
008D! 60                               .BYTE 60               ; RTS
008E!
003C* 8E00
008E! AD FF CF                   DATA3 .BYTE 0AD,0FF,0CF      ; LDA $CFFF
0091! A9 8A                            .BYTE 0A9,8A           ; LDA #$8A
0093! 8D A0 C0                         .BYTE 8D,0A0,0C0       ; STA $C0A0
0096! 8A                               .BYTE 8A               ; TXA
0097! 20 00 CA                         .BYTE 20,00,0CA        ; JSR $CA00
009A! A2 00                            .BYTE 0A2,00           ; LDX #$0
009C! 60                               .BYTE 60               ; RTS
009D!
009D!                                  .END

                         FILE:L.ASM  SYMBOLTABLE DUMP

AB - Absolute      LB - Label      UD - Undefined      MC - Macro
RF - Ref           DF - Def        PR - Proc           FC - Func
PB - Public        PV - Private    CS - Consts


DATA1     LB 0070!  DATA2     LB 0080!  DATA3     LB 008E!  LOOP      LB 0008!  LOOP
2     LB 001F!  LOOP3     LB 0038!  LWRCASE  PR ----
```

```
2    1    1:D    1 {$L PRINTER: }

3    1    1:0    1 PROGRAM LOWERCASE;

4    1    2:D    1 PROCEDURE LWRCASE;
                   EXTERNAL;
5    1    1:0    0 BEGIN

6    1    1:0    0

7    1    1:1    0      LWRCASE;

8    1    1:1    4

9    1    1:0    4 END.
```

DOS ERROR MESSAGE OVERRIDES

To avoid the NOMON CIO default:
POKE - 25129,234.
POKE - 25128,234.
POKE - 25127,234
To defeat the NOT DIRECT COMMAND error:
POKE - 24543,24!
POKE - 24542,234.
POKE - 24541,234
Thes locations are for 48K. For 32K, add 41952

From the Apple Barrel

SOFTWARE DEVELOPMENT TOOLS FOR THE APPLE II

by
Bob Stout

INTRODUCTION

The introduction of the Apple II+ was hailed by many as the answer for many small systems (read business and similar systems) applications. An Apple with Applesoft as its primary language is obviously better suited to this market than the original Apple II. However the adoption of the Autostart ROM as the new standard monitor chip will unfortunately only serve to further isolate the user from the computer's internal operation and structure. While this is unfortunate for the casual user who may no longer be tempted by the possibilities that the asterisk cursor challenges one to explore, it is critical to the serious user who no longer has the capability to debug his machine language software. The trace and single-step facilities have been sacrificed to make room for new functions.

The Apple II has long had the potential to be developed into a serious and competitive tool to develop applications software for the popular 6500-family of microprocessors. Essential for this use, however, are software tools capable of analyzing and generating object code (machine language software) at a thoroughly professional level. The tools required are:

1. Text editor,
2. Assembler,
3. Debugger,
4. Enhanced monitor software,
5. EPROM programming facilities,
6. Hardware debug capability,
7. High-level language compilers, and
8. Provision for other microprocessors (e.g. Z-80, 6809, etc.)

Since there are many good editors and assemblers on the market (most notably ASM/TED from Carl W. Moser, Winston-Salem, NC), I will discuss in this article the steps that have been taken to fill the next essential requirements of a good debugger. Subsequent articles will deal with the other subjects listed in sequence.

TECHNIQUES

Simply stated, a good debugger must be capable of executing object code in a controlled manner while providing a continuous flow of information back to the operator who must always know what's going on as well as what's going to happen and what has already happened. The key is in the *CONTROLLED* execution of the object code. The operator must be able to conditionally decide when the program will run and when it must stop. Obviously, the normal operating environment of the computer does not provide this capability. However, there are two standard techniques to allow it. This may be demonstrated in fig. 1. The assembly listing for this short section is shown in fig. 1A. Fig. 1B illustrates an implementation of the most popular debug technique. As may be seen, the debug software sequentially picks out and saves each opcode, in turn replacing it with a 'BRK' command, then restores it. As the program executes, each 'BRK' command causes the debugger to be invoked every step of the way. The alternative technique in fig. 1C shows a dummy memory area and program counter (PC). The program is relocated into the dummy area one instruction at a time. In this case the program never really "runs" in any conventional sense, but rather the debugger runs, examining each instruction and then either modifying the dummy program counter or jumping to the dummy instruction as required.

Due to its simplicity, the technique shown in fig. 1B is the more popular but it does have two shortcomings. First the code to be debugged must be in RAM, and secondly we could never use the debugger on any Apple without the Autostart ROM since a programmable 'BRK' vector is required. Although more complex, the technique of fig. 1C is obviously preferable for use with the Apple II. In addition to its complexity, the only other significant shortcoming of the technique we will discuss is its slow free-running execution speed, since the debugger and not the application program is always the one running.

REGISTERS

A key requirement of any good debugger is the ability to step through a program and to feed back a running stream of data. Most important, obviously, are the contents of all internal CPU registers (as displayed by the step and trace functions of the old Apple II monitor ROM). This may not be enough, however, for a comprehensive analysis. In a register-oriented processor such as an 8080 (9 CPU registers) or Z-80 (21 CPU registers) this might be sufficient, but the 6502 is memory-oriented, that is, most working data are in external RAM rather than the 5 CPU registers. It is therefore desirable to be able to define external RAM locations which may be traced during debugging as if they were internal CPU registers. Additionally, since many of these registers may be used to hold indirect addresses, it would be well to treat them as 16-bit registers (i.e. trace 2 bytes beginning with each specified memory address) and to provide a means of tracing the indirect location referenced.

The next consideration must be the stack pointer (SP). Since the debugger software uses subroutines, it will obviously alter the SP. Although we can save the SP after each step and restore it prior to each new instruction, we must also assure the integrity of the stack data. Most of the new-generation micro processors such as the Z-8000, MC6800, etc. neatly solve this sort of problem by incorporating 2 SP's, one for systems use and the other for applications software. The fact that the 6502 restricts the stack to page 1 ($0100-$01FF) of memory allows us to easily implement a similar scheme in software, using the 8-bit 6502 SP. To do this we will adopt the convention that locations $0100-$017F are the user stack while locations $0180-$01FF are the system stack. This allows us complete freedom to manipulate and save the user stack while assuring the integrity of its data. A less obvious benefit is that, having adopted this convention, any SP value less than $7F signifies the presence of data on the user stack which may be displayed along with the CPU registers and external trace addresses. This allows us to quickly see programming errors such as trying to execute an 'RTS' after data have been pushed onto the stack (sound familiar?).

BREAKPOINT and HISTORY

While single-stepping or printed traces may be fine for detailed analysis of a section of code, it is also desirable to be able to just let the program run until a previously specified breakpoint is reached and then examine the CPU status and program history. To do this we need merely to compare the dummy PC with our pre-defined breakpoints and exit if there is a match. To save a history of program execution, we merely implement a software-driven stack and push each new PC address on it in turn. A convenient 256-byte buffer used in this manner will record the past 128 steps of the program's history.

PRACTICAL APPLICATIONS

A simplified flowchart of a complete debugger is shown in fig 2. Although this follows the techniques previously discussed, a 'BRK' instruction debugger could be implemented with the Autostart ROM using many of the same techniques. A debugger with all of these features and more is available as part of The Micro Power System available on diskette from Micro Power Designs, Inc., Alief, TX 77411 for $150. Although this is a copyrighted software system designed for industrial, consultant, and advanced experimenter uses, the reader is encouraged to experiment with the principle presented to implement his or her own debugger software. Figs. 3, 4, 5 and 6 demonstrate the use of the debugger as implemented in the Micro Power System. A similar debugger available from Microproducts, Redondo Beach, CA at an as yet undetermined price. The Micro Power Debugger supports 4 breakpoints, 4 16-bit trace addresses, indirect trace addresses, stack display, 128-step trace history, run trace, and single-step modes all tied to an improved Monitor software package.

With this sort of software available from after-market suppliers, perhaps the Autostart ROM will really be the sort of blessing that Apple intended it to be.

Fig 1A

```
1000-  A5 1E              LDA  $1E
1002-  D0 05              BNE  SKIP
1004-  20 00 11           JSR  ZERO
1007-  A9 00              LDA  #$00
1009-  4C 00 12    SKIP   JMP  NEXT
                          ---
1100   8D 00 03    ZERO   STA  $0300
1200-  38                 SEC
                          ---
```

Fig. 1A

```
A5 1E D0 05 20 00 11 A9 00 4C 00 12 : Original object
                                      code.
00 1E D0 05 20 00 11 A9 00 4C 00 12 : First step.
A5 1E 00 05 20 00 11 A9 00 4C 00 12 : Second step.
A5 1E D0 05 00 00 11 A9 00 4C 00 12 : Third step or. .
A5 1E D0 05 20 00 11 00 00 4C 0012  : Alternate third
                                      step.
```

Fig. 1B

```
A5 1E D0 05 20 00 11 A9 00 4C 00 12 : Original object
                                      code remains
                                      static.
```

Dummy Area

```
A5 1E EA 4C xx xx 4C yy yy          : (PC)= $1000
D0 04 EA 4C xx xx 4C yy yy          : (PC)= $1002
      Alternative #1
8D 00 03 4C xx xx 4C yy yy          : (PC)= $1100
      Alternative #2
A9 00 EA 4C xx xx 4C yy yy          : (PC)= $1007
```

xxxx= Debug 'continue' routine.
yyyy= Debug PC-modifying branch handling routine.
Note the change in relative offset of 'BNE' instruction.

Fig. 1C

From WASHINGTON APPLE PI

SHAPING UP WITH THE APPLE II

by
Mark L. Crosby

INTRODUCTION

The Apple II not only displays colorful low-resolution graphics but has the capability of creating beautiful hi-resolution displays which can be used for graphics design work, architectural design, business graphics, game playing and much more. Using Applesoft II, the Apple II can draw shapes as easily as it can plot a point or draw a line.

Suppose you are an interior designer and want to design a living room layout complete with furniture to see how the space will be filled. You might use the SHAPE DESIGNER to draw chairs, overhead lamps, windows, tables, couches and other items to be located in the room. These would be saved onto a disk under their respective names, e.g., "LAMP", "TABLE", "COUCH", "CHAIR", "WINDOW", etc., and then assembled into a SHAPE TABLE.

For example: DRAW 3 at 139,79

This command translates into "DRAW SHAPE NUMBER 3 – a chair for example – AT THE CENTER OF THE SCREEN". VOILA! A chair appears in the center.

The same concepts are followed for designing games with shapes, adding special shapes to business graphs, etc. Once you have that SHAPE TABLE constructed, your imagination is the limit!

To use the shape concept to the fullest, it is best to first outline a program idea and start writing it. Decide how many different shapes you will need and refer to them by number. Then use the SHAPE DESIGNER to actually draw the shapes you need, save them to disk, and assemble them into a SHAPE TABLE. Once you have done this and added a few statements to your program, you will be ready to draw! *(See "How to Use a Shape Table in Your Program").*

WHAT IS A SHAPE?

As mentioned, a shape usually takes the form of a common object – a flower – but it can also consist of irregularly placed lines. A shape is a series of dots or lines drawn consecutively in The outline or "shape" of an object. It can be drawn on the screen with a single command. A shape can be made up of any number of dots or lines (lines are more commonly known as "vectors"). These vectors can go either up, down, left or right. The APPLE II requires that a "vector table" be constructed in memory that describes for the computer which way each vector goes and whether or not to plot that vector as a line as it moves. The SHAPE DESIGNER does precisely that.

A short explanation of vectors is in order. While you are drawing the shape the first time, each press of a move key draws a single dot on the screen. You should remember that you are presently at SCALE=1. At any scale higher than 1, each press of the key is, effectively, drawing a line in that direction. A single dot at SCALE=1 translates into a 2-dot line at SCALE-2, a 3-dot line at SCALE=3, etc.

In the example below, a programmer has designed a simple shape (a square) which has been enlarged to more clearly show the "vectors" that make up the body of the shape. The table to the right of the shape is a simplified vector table that must be constructed by the SHAPE DESIGNER program.

Vector #	Move
1	
2	
3	↗
4	↗
5	→
6	→
7	↘
8	↘
9	
10	Ø (no move, end of shape

When given the "DRAW" or "XDRAW" command for this shape, the computer starts at point "A" and moves down to point "B" without drawing anything on the screen. Then it draws a line to the left (vector #2). Vectors 3-9 are all drawn as indicated and the shape stops at point"B". The origin of this shape is at its center or point "A".

When the shape is drawn at a standard scale of "1", the shape appears to be a tiny, solid square. When the scale is increased to, say "5", the dots turn into lines (vectors) and the box is clearly distinct with an empty center. By turning the point on or off as you draw, you will either draw a line or not draw a line as you move. In this manner, you may draw a shape that has disconnected lines. In figure 1, vector #1 is a move without drawing. The remainder of the vectors all draw lines as they move.

For most purposes, however, you will not need to know the details of generating a vector table because the SHAPE DESIGNER takes care of all the details.

GETTING STARTED

There are three programs which should all be entered and saved on one diskette. The MENU controls the other two programs. The SHAPE DESIGNER actually permits you to draw a shape of your choice then saves it, by name, on the diskette. The SHAPE ASSEMBLER builds a shape table from several shapes and saves that on any disk of your choosing. These shape tables can be used in any other program dealing with shape drawing by simply loading them under program control, identifying their starting location to Applesoft and drawing. You will need Applesoft ROM.

When you have entered all of the programs and saved them on one diskette, run the MENU program.

To draw a shape, hit "I". The program will then ask you where in main memory you wish to store the shape's vector table. Be sure to select a vacant location that will not interfere with either your Hi-Resolution display, your Disk Operating System or Applesoft II. Consult your manuals. Usually, when using Applesoft II (ROM), the location $1000 (4096 decimal) is available. If you have enough memory, you may use a higher location such as $4000 (16384 decimal). You can always look through memory to see what space is free. Not much space is required for a single shape. An average of 20-50 bytes is about normal.

Next, decide what your first move will be, but before making the move, either turn the point on or off by hitting "P". Then hit the "U", "D", "L" or "R" keys to move the dot in the direction you choose. Remember: each move represents a "line" drawn in that direction. You may turn the point on or off at any time depending on your requirements. If you make a mistake hit (CTRL)W to "wipe" the screen clear and start over.

WARNING — Do not move "up" more than once WHILE THE POINT IS OFF or your shape will not function properly. You may separate "up" moves with moves in another, unrelated direction. If you have the point "on" you may make any number of "up" moves one after the other.

When you have finished drawing the shape, hit (CTRL)F which tells the program you are "finished" drawing. The program will display how many bytes of memory space have been used, the decimal and hexadecimal starting and ending points of the vector table that has been generated and will erase and redraw the shape to verify what you have drawn. Hit any key and the program will ask you is you want to save the shape. If you hit "Y" you will be asked to name the shape. Type in the name and hit return. The shape will be saved on the Disk for future use. The menu will appear again after that.

Do not confuse a VECTOR table with a SHAPE table. The former is a representation of a single shape, the latter is a table with several shapes contained inside such that each shape can be drawn separately when needed. (see page 95 Applesoft Programming Manual). The top of figure 2 shows the construction of a shape table, beginning with the total number of shapes, followed by relative addresses or locations of each shape, followed by the actual shape vectors for each shape. The bottom portion shows a completed shape table using one shape. The ASSEMBLER creates this table automatically, so you need not be too concerned with the details at this point.

For practice, draw several shapes and save each on disk. Then, when the menu appears, hit "2" and read the instructions. Hit any key and a CATALOG of the disk will appear (hit the space bar as necessary to get a complete listing) and then you will be asked to enter the names of each shape you wish to assemble into the table. After the last name, hit return twice. The program will automatically load each shape from the disk, create a SHAPE TABLE, and then draw all of the shapes on the screen to verify the SHAPE TABLE's integrity. You may assemble a maximum of 255 different shapes into one table, but because of space limitations, the upper limit has been arbitrarily set to 128 shapes. You may change "NS" in line #120 as necessary.

Hit any key and the program will then ask you if you want to save the SHAPE TABLE you have just created. If you do, hit "Y" and the program will ask you to name the table. Type the name and hit return. This SHAPE TABLE will then be saved on your disk. The menu will appear after that.

NOTE: All disk commands specify volume Ø so you may insert any disk to either save shapes or retrieve them to create shape tables.

HOW TO USE A SHAPE TABLE IN YOUR PROGRAMS

Remember, you can draw any of the shapes in a SHAPE TABLE by using the command "DRAW I AT X,Y" or "XDRAW I AT X,Y" where "I" is the number of the shape you want to draw and "X" and "Y" are the coordinates of the starting point of the shape on the screen. "X"=Ø-279, "Y"=Ø -191. You should execute an "HGR" or "HGR2" before attmepting to do any Hi-Resolution plotting. ROT, SCALE, and HCOLOR should all be set before attempting to draw a shape. DRAW will draw a shape in the color you have chosen. XDRAW will draw in the complement only. Two XDRAW commands one after the other will draw a shape then erase it *while leaving any other background intact.* (See previously mentioned manual).

You must include the following statements, or equivalents, in your programs before using a SHAPE TABLE:

```
Ø DEF FN MD(B) = B - INT (B/256) * 256
1 D$=CHR$(4)
2 POKE 232, FN MD(B) : POKE 233,B/256
3 PRINT D$; "BLOAD (NAME OF YOUR SHAPE
  TABLE), A";B
```

Line 0 defines the "MOD" function available in Integer Basic but not found in Applesoft II.

Line 1 sets D$ equal to a control-D which is required to get the attention of the Disk Operating System.

Line 2 POKES the shape table starting address into a special pointer used by Applesoft. "B" is the decimal starting address in main memory where you wish to load your shape table. "B" must be set to some number before executing this statement.

Line 3 loads your shape table from disk into memory starting at "B".

Some comments on drawing shapes in your programs:

Before you can draw a shape, you must first load the shape table and set the pointers as indicated above. Then execute an HGR or HGR2 command (the latter for page 2 of hi-resolution graphics). With HGR2, you cannot have text at the bottom of the screen.

Then set SCALE to a reasonable starting point – either 1 for the original size shape or 2,3,4, etc., for larger sizes. Then set ROTation to any number from zero to 63. Actually, you may use any number from zero to 255 depending on the SCALE. For example, any ROT from zero to 3 at SCALE 1 will draw the shape at zero degrees rotation. You can investigate this at your leisure.

HCOLOR must be set for any DRAW command but is not used for any XDRAW command. DRAW will draw the shape at the specified location in the specified color. Keep in mind the Apple's every-other-dot color limitations in the X coordinate direction. Certain colors can only be drawn on even X coordinates and others on odd coordinates. The XDRAW command will take the color/s on the screen and draw their compliments. If the screen were totally black, then, it would draw a white shape. Immediately drawn again, the shape is drawn in the complement of white or black, thereby disappearing. If the screen were filled with various colors, each color would be complemented as the shape is drawn. White would become black, blue-orange, violet-green, etc.

PRECAUTIONS:

Remember not to move more than one space upwards while the point is off when drawing. Each shape table consists of a certain number of shapes. To find out how many shapes you have in any particular shape table, first execute the statements in the previous section. They type X=PEEK(B). "X" will then be equal to the total number of shapes you have available. Trying to draw a number higher than "X" will give you a syntax error and halt execution.

There is an "ONERR GOTO" statement within the SHAPE DESIGNER that will cause a restart if any type of error occurs. This includes errors of spelling when trying to load shape names from a disk. The error handling routines are in lines 3000-3350. You may disable these by deleting line 420.

Certain moves to the right or left or up and down when the point is off or on can cause some weird effects and may cause your shape to be partially drawn. This can usually be corrected by drawing it again being careful not to cross the same point twice. Some experimentation will be helpful here.

When drawing in color, remember that GREEN, VIOLET, ORANGE and BLUE can only plot on every other dot along the "X" axis. Drawing your shape twice at X,Y, and X+1,Y will fill in the missing lines and make your lines "thicker". WHITE and BLACK do not present this problem.

If you have any questions or suggestions for changes in this program or any problems with your particular system configuration you may write to the author:

Mark L. Crosby
1373 "E" Street S.E.

```
1  REM HOW TO MODIFY A VAL
       GOLDING PROGRAM FROM 40
       LINES TO 4

2  REM BY DARRELL ALDRICH

10  GET A$
15  IF  ASC (A$) < 64 THEN  PRINT
     A$;: GOTO 10
20  PRINT  CHR$ ( ASC (A$) + 32);
30  GOTO 10
```

```
]LIST

10  REM  " - SHAPE DESIGNER - "
20  REM  " - COPYRIGHT 1979 - "
30  REM  "RESEARCH  ASSOCIATES"
40  REM  " - MARK L. CROSBY - "
50  REM
60  TEXT : NORMAL : HOME
70 D$ =  CHR$ (4)
80  POKE  - 16303,0
90  VTAB 10: HTAB 11: PRINT " - S
    HAPE DESIGNER -"
100  PRINT : PRINT "     1 - DRAW
     A SHAPE AND SAVE IT": PRINT
    "      2 - ASSEMBLE SHAPES IN
    TO A TABLE": PRINT "     3 -
     END THE PROGRAM"
110  PRINT : PRINT "    HIT KEY O
    F YOUR CHOICE ";
120  GET A$: NORMAL
130 A =  VAL (A$): ON A GOTO 140,
    150,160: GOTO 120
140  GOSUB 170: PRINT : PRINT D$;
    "RUN SHAPER": END
150  GOSUB 170: PRINT : PRINT D$;
    "RUN ASSEMBLER": END
160  NORMAL : HOME : END
170  PRINT : PRINT : INVERSE : HTAB
    4: PRINT "ONE MOMENT PLEASE"
    : NORMAL : RETURN
```

```
]LIST

10  TEXT : HOME :
20  INVERSE : FOR I = 1 TO 24: PRINT
    "
                        ";: NEXT I

30  VTAB 2: HTAB 9
40  PRINT "WELCOME TO   " CHR$ (3
     4 )"SHAPING UP" CHR$ (34)
50  PRINT : PRINT " PURPOSE:";
60  POKE 32,11: POKE 34,1
70  HTAB 12
80  PRINT "TO FACILITATE DRAWING
     SHAPES,";
90  PRINT "AND THE CREATION OF " CHR$
     (34)"SHAPE":
100  PRINT "TABLES" CHR$ (34)" EA
     CH CAPABLE OF CON-": PRINT "
     TAINING MANY SHAPES.  THESE"

110  PRINT "FINISHED TABLES CAN E
     ASILY"
120  PRINT "BE INSERTED INTO OTHE
     R PRO-"
130  PRINT "GRAMS."
140  PRINT
150  PRINT "ALL SHAPES AND TABLES
      CAN BE"
```

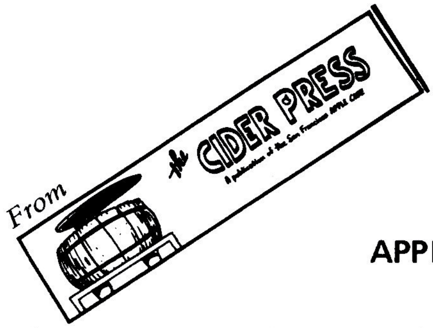
```
160  PRINT "STORED AND RETRIEVED
     BY NAME."
170  POKE 32,0: POKE 33,40: POKE
     34,0
180  PRINT : PRINT "------------
     ----------------------------"
     ;
190  POKE 32,1
200  PRINT : PRINT "ONE USE IS IN
      GAMES THAT USE MOVING"
210  PRINT "OBJECTS ACROSS THE SC
     REEN. ANOTHER"
220  PRINT "MIGHT BE ARCHITECTURA
     L MODELING."
230  PRINT : PRINT : HTAB 5: PRINT
     "HIT ANY KEY TO CONTINUE..."
     ;
240  GET A$
250  VTAB 2: HTAB 1: POKE 32,0
260  PRINT "DURING THE NEXT SECTI
     ON, A MENU WILL"
270  PRINT " GIVE YOU THE OPPORTU
     NITY TO DRAW A "
280  PRINT " SHAPE AND THEN SAVE
     IT ON A DISK.        ";
290  PRINT "
                          ";
300  PRINT " AFTER SAVING THE VAR
     IOUS SHAPES YOU     ";
310  PRINT " REQUIRE, YOU CAN THE
     N ASSEMBLE THOSE ";
320  PRINT " OF YOUR CHOICE INTO
     A SHAPE TABLE THAT"
330  PRINT " CAN BE USED IN ANY O
     THER PROGRAM JUST"
340  PRINT " BY LOADING IT AND PE
     RFORMING 2 POKES."
350  FOR I = 1 TO 10
360  PRINT "
                          ";
370  NEXT I
380  PRINT : HTAB 32
390  GET A$
400  NORMAL : POKE 32,0: POKE 33,
     40
410  FOR I = 1 TO 24: CALL  - 922
     : NEXT
420  ONERR  GOTO 3000
430 D$ =  CHR$ (4): PRINT D$;"MON
      I,O,C": PRINT D$;"NOMON C":
      HOME : DEF  FN MD(B) = B -
      INT (B / 256) * 256
440  DIM D(3),LLL(4),H(4):HEX$ =
     "0123456789ABCDEF"
450  HOME : VTAB 3: HTAB 2: PRINT
     "ENTER THE STARTING ADDRESS"
     : HTAB 2: PRINT "OF SHAPE IN
      HEX: ";
460  INPUT LOC$
470  HOME : VTAB 2
480  PRINT "          POSSIBLE COMM
     ANDS..."
490  VTAB 8: HTAB 3: PRINT "CTRL-
     F    WHEN FINISHED": HTAB 3:
      PRINT "CTRL-W    TO WIPE CL
     EAN AND START OVER": HTAB 8:
      PRINT "U - TO MOVE UP": HTAB
     8: PRINT "D - TO MOVE DOWN":
      HTAB 8: PRINT "L - TO MOVE
     LEFT": HTAB 8: PRINT "R - TO
      MOVE RIGHT"
500  HTAB 8: PRINT "P - POINT ON/
     OFF": VTAB 18: PRINT "     H
     IT ANY KEY TO CONTINUE ";: GET
     A$: HGR : HOME : NORMAL
510 PFLAG = 0
520 YCO = 79:XCO = 139
530 OLDY = 79:ODX = 139
540  HCOLOR= 3: GOSUB 910
550  FOR I = 1 TO 4:H(I) =  ASC (
      MID$ (LOC$,I,1)) - 48: IF H
     (I) > 9 THEN H(I) = H(I) - 7

560  NEXT I
570 LOC = 0: FOR I = 1 TO 3:LOC =
     LOC + H(I):LOC = LOC * 16: NEXT
     I:LOC = LOC + H(4):LC1 = LOC
```

```
580  VTAB 22: PRINT "BYTES USED:"
     ;
590   HTAB 33: PRINT "ON ";: INVERSE
     : PRINT "OFF": NORMAL
600  VTAB 24: PRINT "COMMANDS: CT
     RL-F, CTRL-W, U,D,L,R,P";: VTAB
     1
610   GET A$
620  IF A$ = "U" THEN 700
630  IF A$ = "D" THEN 740
640  IF A$ = "R" THEN 780
650  IF A$ = "L" THEN 820
660  IF A$ = "P" THEN 860
670  IF  ASC (A$) = 6 THEN 900
680  IF  ASC (A$) = 23 THEN 560
690  GOTO 610
700  REM  UP
710 YCO = OLDY - 1:A = 4
720  IF PFLAG = 0 THEN A = 0
730  GOSUB 1010: GOSUB 910: GOTO
     610
740  REM  DOWN
750 YCO = OLDY + 1:A = 6
760  IF PFLAG = 0 THEN A = 2
770  GOSUB 1010: GOSUB 910: GOTO
     610
780  REM  RIGHT
790 XCO = ODX + 1:A = 5
800  IF PFLAG = 0 THEN A = 1
810  GOSUB 1010: GOSUB 910: GOTO
     610
820  REM  LEFT
830 XCO = ODX - 1:A = 7
840  IF PFLAG = 0 THEN A = 3
850  GOSUB 1010: GOSUB 910: GOTO
     610
860 PFLAG =  NOT PFLAG
870  IF PFLAG = 0 THEN 890
880  VTAB 22: HTAB 33: INVERSE : PRINT
     "ON";: NORMAL : PRINT " OFF"
     : VTAB 1: GOTO 610
890  VTAB 22: HTAB 33: NORMAL : PRINT
     "ON ";: INVERSE : PRINT "OFF
      ": NORMAL : VTAB 1: GOTO 61
     0
900  GOTO 1150
910  IF YCO < 0 THEN YCO = 0
920  IF YCO > 159 THEN YCO = 159
930  IF XCO < 0 THEN XCO = 0
940  IF XCO > 279 THEN XCO = 279
950  IF PGLAG = 0 THEN 980
960  HPLOT XCO,YCO
970  GOTO 1000
980  HCOLOR= 0: HPLOT ODX,OLDY
990  HCOLOR= 3: HPLOT XCO,YCO
1000 ODX = XCO:OLDY = YCO: RETURN
1010 F = 0:D = D + 1:D(D) = A: IF
     D < 3 THEN  RETURN
1020  IF D(3) <  > 0 THEN 1040
1030 F = 1: GOTO 1060
1040  IF D(3) < 4 THEN 1060
1050 Q = D(3):D(3) = 0
1060 Z = D(1) + D(2) * 8 + D(3) *
     64
1070 BYTE = BYTE + 1: VTAB 22: HTAB
     13: PRINT BYTE: VTAB 1
1080  POKE LOC,Z:LOC = LOC + 1
1090  IF F = 1 THEN 1110
1100  IF Q = 0 THEN 1140
1110  IF D(2) <  > 0 THEN 1130
1120 D(1) = 0:D(2) = Q:D(3) = 0:Q
      = 0:D = 2: RETURN
1130 D(1) = Q:D(2) = 0:D(3) = 0:Q
      = 0:D = 1: RETURN
1140  FOR I = 1 TO 3:D(I) = 0: NEXT
     I:D = 0: RETURN
1150  REM  CTRL F GOES TO HERE
1160 Z = D(1) + D(2) * 8 + D(3) *
     64: POKE LOC,Z
1170  IF Z = 0 THEN 1220
1180 LOC = LOC + 1: POKE LOC,0
1190 LOC = LOC + 1: POKE LOC,0
1200  VTAB 22: HTAB 30: CALL  - 9
     58
1210  VTAB 23: HTAB 1
1220  PRINT "VECTOR TABLE: FROM "
     ;LC1;" TO ";LOC:C = LOC
1230  PRINT "HEX: FROM ";LOC$;" T
     O ";
1240  FOR I = 0 TO 4:LLL(I) = 0: NEXT
     I
1250  FOR I = 3 TO 0 STEP  - 1
1260  IF LOC < 16 ↑ I THEN 1280
1270 LLL(I) = LLL(I) + 1:LOC = LO
     C - 16 ↑ I: GOTO 1260
1280  NEXT I:LOC$ = ""
1290  FOR I = 3 TO 0 STEP  - 1: GOSUB
     1300: NEXT I: PRINT : GOTO 1
     340
1300  FOR J = 0 TO 15
1310  IF LLL(I) = J THEN  PRINT  MID$
     (HEX$,J + 1,1);
1320  NEXT J
1330  RETURN
1340  REM  DRAW SECTION
1350  HGR : ROT= 0
1360 B = LC1 - 4
1370  POKE 232, FN MD(B): POKE 23
     3,B / 256: POKE B,1: POKE B +
     1,0: POKE B + 2,4: POKE B +
     3,0
1380 X = 30:Y = 79: FOR SC = 1 TO
     3: SCALE= SC: DRAW 1 AT X,Y:
     X = X + 50: NEXT SC
```

```
1390  INVERSE : VTAB 24: PRINT "
         HIT ANY KEY TO CONTIN
     UE...       ";: HTAB 34
1400  NORMAL : GET A$
1410  HOME : VTAB 23
1420  PRINT "DO YOU WANT TO SAVE
     THIS": PRINT "SHAPE? (Y OR N
     ) ";: GET A$: IF A$ = "Y" THEN
     1440
1430  GOTO 1490
1440  HOME : VTAB 23
1450  PRINT "NAME OF SHAPE TABLE:
       ";: INPUT SHAPE$
1460  IF SHAPE$ = "" THEN 1410
1470  PRINT D$;"BSAVE ";SHAPE$;",
     A";LC1;",L";C - LC1;",V0"
1480  PRINT D$;"LOCK ";SHAPE$;",V
     0"
1490 : PRINT : HOME
1500  PRINT D$;"RUN MENU"
1510  END
3000 A =  PEEK (222):B =  PEEK (2
     18) +  PEEK (219) * 256: REM
       A IS ERROR CODE AND B IS L
     INE # WHERE ERROR OCCURRED.
3010  IF A < 4 OR A > 12 THEN 333
     0: REM  CONTROLLED SHUTDOWN
3020 C = A - 3: REM  C=1 THROUGH
     9,REPRESENTING ERROR CODES 4
      THROUGH 12
3030  ON C GOTO 3060,3090,3110,31
     70,3190,3220,3250,3280,3310
3040  VTAB 21: INVERSE : CALL  -
     958: RETURN
3050  FOR J = 1 TO 3: PRINT  CHR$
     (7);: NEXT J: FOR J = 1 TO 3
     500: NEXT : RETURN
3060  REM  ERROR $4: WRITE PROTEC
     T
3070  GOSUB 3040: PRINT "DISK IS
     WRITE PROTECTED !!": PRINT "
     CHANGE DISKS...             "
     : NORMAL : GOSUB 3050
3080  GOTO 640
3090  REM  ERROR #5: END OF DATA,
      SHOULDN'T HAPPEN
3100  GOTO 3330: REM  CONTROLLED
     SHUTDOWN
3110  REM  ERROR #6: FILE NOT FOU
     ND
3120  IF B = 440 THEN 3150: REM
     ELSE "MENU" IS BEING RUN.
3130  GOSUB 3040: PRINT " MENU IS
      NOT ON THIS DISK ! ": PRINT
     " PROGRAM WILL TERMINATE...
      ": NORMAL : GOSUB 3050
3140  GOTO 3330: REM  CONTROLLED
     SHUTDOWN
3150  GOSUB 3040: PRINT " ";SHAPE
     $(I);" NOT ON THIS DISK ! ":
      NORMAL : GOSUB 3050
3160  GOTO 270
3170  REM  ERROR #7: VOLUME MISMA
     TCH, SHOULDN'T HAPPEN
3180  GOTO 3330: REM  CONTROLLED
     SHUTDOWN
3190  REM  ERROR #8: DISK I/O
3200  GOSUB 3040: PRINT " DISK ER
     ROR - USE NEW DISK !! ": NORMAL
     : GOSUB 3050
3210  GOTO 3330: REM  CONTROLLED
     SHUTDOWN
3220  REM  ERROR #9: DISK FULL
3230  GOSUB 3040: PRINT " DISK IS
      FULL, USE ANOTHER ! ": NORMAL
     : GOSUB 3050
3240  GOTO 640
3250  REM  ERROR #10: FILE LOCKED
     , WHICH MEANS TABLE NAME IS
     BEING SAVED THAT ALREADY EXI
     STS ON THE DISK.
3260  GOSUB 3040: PRINT " YOU ALR
     EADY HAVE A ";SHAPE$: PRINT
     " TABLE ON THIS DISK "; NORMAL
     : GOSUB 3050
3270  GOTO 640
```

```
3280  REM  ERROR #11: COMMAND SYN
     TAX, WHICH MEANS THAT FILE I
     S BEING SAVED THAT BEGINS WI
     TH A NUMBER, ETC.
3290  GOSUB 3040: PRINT " YOU MUS
     T START NAME WITH A LETTER A
     -Z ";: NORMAL : GOSUB 3050
3300  GOTO 640
3310  REM  ERROR #12: NO FILE BUF
     FERS, SHOULDN'T HAPPEN
3330  REM  CONTROLLED SHUTDOWN
3340  TEXT : HOME : NORMAL : PRINT
     " PROGRAM TERMINATED DUE TO"
     : PRINT " ERROR "A" LINE #"B
     "
3350  END

]LIST

10  REM     - SHAPE DESIGNER -
20  REM       COPYRIGHT 1979
30  REM    RESEARCH ASSOCIATES
40  REM    ALL RIGHTS RESERVED
50  REM
60  REM      ASSEMBLER MODULE
70  REM
80  POKE 74, PEEK (76): POKE 75, PEEK
    (77) - 8: CLEAR
90 M = 150 -  PEEK (77):H = 169:S
    L = 181:LL = 163:H = H - M:S
     = 1:K = 256:I =  - 384:N =
     - 300:G =  - 198: IF H < 12
    8 THEN 110
100 S =  - 1:H = K - H:PTCH = 0: IF
     PEEK (977) = 191 THEN PTCH =
    189
110 SA = S * K * H + SL + PTCH:LA
     = S * K * H + LL + PTCH
120  TEXT : HOME :NS = 128: DIM S
    HAPE$(NS)
130  DEF  FN MD(B) = B -  INT (B /
    256) * 256
140  ONERR  GOTO 1000
150 D$ =  CHR$ (4)
160  PRINT D$;"MONI,O,C": PRINT D
    $;"NOMON C": HOME
170  REM  " ASSEMBLER "
180  HOME : VTAB 4: HTAB 1
190  PRINT "THIS PROGRAM WILL ASS
    EMBLE PREVIOUSLY"
200  PRINT "SAVED SHAPES INTO A M
    ULTI-SHAPE SHAPE"
210  PRINT "TABLE THAT CAN BE ACC
    ESSED THROUGH"
220  PRINT "APPLESOFT II OR PROGR
    AMMERS AID"
230  PRINT  CHR$ (34)"DRAW" CHR$
    (34)" COMMANDS."
240  PRINT : PRINT : PRINT "A CAT
    ALOG WILL FOLLOW.  ENTER NAM
    E OF    SHAPE OR HIT RETURN
    TO FINISH."
250  PRINT : PRINT "YOU MAY ASSEM
    BLE UP TO ";NS;" SHAPES."
260  PRINT : PRINT "HIT ANY KEY T
    O CONTINUE ";: GET A$
270  HOME
290  FOR I = 1 TO NS
300  PRINT "NAME OF SHAPE #";I;":
     ";: INPUT SHAPE$(I)
310  IF SHAPE$(I) = "" THEN 340
320  NEXT I
330  GOTO 350
340 NUM = I - 1
350  REM  ASSEMBLER SECTION
360  HOME : PRINT "STARTING LOCAT
    ION ASSUMED TO BE": PRINT "2
    4576 DECIMAL.":START = 24576
370  IF START > 38000 THEN  PRINT
    "TOO HIGH !": FOR J = 1 TO 3
    000: NEXT : GOTO 360
380  POKE START,NUM: POKE START +
    1,0:B = START: POKE 232, FN
    MD(B): POKE 233,B / 256:B =
    0
390 ADR = START + 2: REM   START
    OF SHAPE ADDRESSES
400 SH = ADR + (2 * NUM): REM  ST
    ART OF FIRST SHAPE
410  PRINT
420  FOR I = 1 TO NUM
430  PRINT "LOADING ";SHAPE$(I)
440  PRINT D$;"BLOAD ";SHAPE$(I);
    ",A";SH: REM  LOAD SHAPE BE
    GINNING AT  "SH"
450 LN =  PEEK (LA) + 256 *  PEEK
    (LA + 1): REM   SHAPE LENGTH
     FROM DOS
460 B = SH - START: REM   ABSOLUT
    E  ADDRESS FOR SHAPE "I" MEA
    SURED FROM "START" (0)
470  POKE ADR, FN MD(B): REM  POK
    E ADR,B MOD 256 IN INTEGER B
    ASIC
480  POKE ADR + 1,B / 256
490 B = 0
500 ADR = ADR + 2: REM   INCREMEN
    T  ADDRESS TABLE FOR NEXT SH
    APE
510 SH = SH + LN: REM   NEXT SHAP
    E  START LOCATION
520  NEXT I
530  REM   DRAW SHAPES ON SCREEN
    TO VERIFY
540  ROT= 0: HCOLOR= 3: HGR
550  VTAB 22: HTAB 6: PRINT "SCAL
    E=1";: HTAB 26: PRINT "SCALE
    =4";
```

```
560  INVERSE
570 Y = 79:X1 = 55:X2 = 190
580  FOR I = 1 TO NUM: VTAB 24: HTAB
     1: CALL  - 868: HTAB 20 - ( LEN
     (SHAPE$(I)) / 2): PRINT SHAP
     E$(I);
590  SCALE= 1: XDRAW I AT X1,Y: SCALE=
     4: XDRAW I AT X2,Y
600  FOR J = 1 TO 2000: NEXT J
610  SCALE= 1: XDRAW I AT X1,Y: SCALE=
     4: XDRAW I AT X2,Y
620  NEXT I
630  NORMAL
640  HOME : PRINT "START: ";START
     ;"    LENGTH: ";(SH - 1) - S
     TART
650  PRINT
660  FOR I = 1 TO NUM: PRINT I;"-
     ";SHAPE$(I),: NEXT I: PRINT
     : PRINT : PRINT "DO YOU WANT
      TO SAVE THIS": PRINT "SHAPE
      TABLE ? (Y OR N) ";: POKE 1
     6303,0: GET A$: IF A$ = "Y" THEN
     680
665  IF A$ <  > "N" THEN 640
670  GOTO 730
680 : PRINT : PRINT : PRINT "TYPE
      THE TABLE'S NAME: ";
690  INPUT SHAPE$
700  IF SHAPE$ = "" THEN 660
710  PRINT D$;"BSAVE ";SHAPE$;",A
     ";START;",L";(SH - 1) - STAR
     T;",V0"
720  PRINT D$;"LOCK ";SHAPE$;",V0
     "
730  PRINT : HOME
740  PRINT D$;"RUN MENU"
1000 A =  PEEK (222):B =  PEEK (2
     18) +  PEEK (219) * 256: REM
      A IS ERROR CODE AND B IS LI
     NE # WHERE ERROR OCCURRED.
1010  IF A < 4 OR A > 12 THEN 200
     0: REM  CONTROLLED SHUTDOWN
1020 C = A - 3: REM  C=1 THROUGH
     9,REPRESENTING ERROR CODES 4
      THROUGH 12
1030  ON C GOTO 1100,1200,1300,14
     00,1500,1600,1700,1800,1900
1050  VTAB 21: INVERSE : CALL  -
     958: RETURN
1070  FOR J = 1 TO 3: PRINT  CHR$
     (7);: NEXT J: FOR J = 1 TO 3
     500: NEXT : RETURN
1100  REM  ERROR #4: WRITE PROTEC
     T
1110  GOSUB 1050: PRINT "DISK IS
     WRITE PROTECTED !!": PRINT "
     CHANGE DISKS...              "
      : NORMAL : GOSUB 1070
1120  GOTO 640
1200  REM  ERROR #5:END OF DATA,
     SHOULDN'T HAPPEN
1210  GOTO 2000: REM  CONTROLLED
     SHUTDOWN
1300  REM  ERROR #6:FILE NOT FOUN
     D
1310  IF B = 440 THEN 1350: REM
     ELSE "MENU" IS BEING RUN.
1320  GOSUB 1050: PRINT " MENU IS
      NOT ON THIS DISK ! ": PRINT
     " PROGRAM WILL TERMINATE...
      ": NORMAL : GOSUB 1070
1330  GOTO 2000: REM  CONTROLLED
     SHUTDOWN
1350  GOSUB 1050: PRINT " ";SHAPE
     $(I);" NOT ON THIS DISK ! ":
      NORMAL : GOSUB 1070
1360  GOTO 270
1400  REM  ERROR #7:VOLUME MISMAT
     CH, SHOULDN'T HAPPEN
1410  GOTO 2000: REM  CONTROLLED
     SHUTDOWN
1500  REM  ERROR #8:DISK I/O
1510  GOSUB 1050: PRINT " DISK ER
     ROR - USE NEW DISK !! ": NORMAL
     : GOSUB 1070
1520  GOTO 2000: REM  CONTROLLED
     SHUTDOWN
1600  REM  ERROR #9:DISK FULL
1610  GOSUB 1050: PRINT "DISK IS
     FULL, USE ANOTHER ! ": NORMAL
     : GOSUB 1070
1620  GOTO 640
1700  REM  ERROR #10:FILE LOCKED,
      WHICH MEANS TABLE NAME IS B
     EING SAVED THAT ALREADY EXIS
     TS ON THE DISK.
1710  GOSUB 1050: PRINT " YOU ALR
     EADY HAVE A ";SHAPE$: PRINT
     " TABLE ON THIS DISK ": NORMAL
     : GOSUB 1070
1720  GOTO 640
1800  REM  ERROR #11:COMMAND SYNT
     A, WHICH MEANS THAT FILE IS
     BEING SAVED THAT BEGINS WITH
      A NUMBER, ETC.
1810  GOSUB 1050: PRINT " YOU MUS
     T START NAME WITH A LETTER A
     -Z ";: NORMAL : GOSUB 1070
1820  GOTO 640
1900  REM  ERROR #12:NO FILE BUFF
     ERS, SHOULDN'T HAPPEN
1910  GOTO 2000: REM  CONTROLLED
     SHUTDOWN
2000  REM  CONTROLLED SHUTDOWN
2010  TEXT : HOME : NORMAL : PRINT
     " PROGRAM TERMINATED DUE TO"
     : PRINT " ERROR "A" LINE #"B
2020  END
```



APPLE DOS BOOTING PROCESS

by
Ted Burns

The disk booting process is done in three stages. Stage 1 is done by the code on the disk controller card located at Cn00 (where n is the slot of your disk controller). This reads in track sector 0 (0, 0) from the diskette. This information is scrambled due to code space limitations on the boot PROM.

The very last 2 bytes of sector (0, 0) are in normal format. They are used as parameters to the second stage boot routine. After this CODE is loaded into page 3 (300-3FF Hex), the disk controller software then jumps to location $301. The code at location $301 performs the second stage boot.

The *SECOND STAGE BOOT* reads in from disk, sectors (0, 0) to (0, n) where n is specified by the last byte of sector (0, 0). The last byte of sector (0, 0) is equal to N*8. So to find out how many sectors to load, we divide this byte by 8. Normally, a value of $48 is assigned to this byte, which is equal to 9 sectors. Tracks (0, 2) through (0, 9) on the diskette contain the code for the RWTS (Read Write Track Sector) routines which get read by the second level boot routine.

The second to the last byte in (0, 0) contains the page number minus one of where the code is located that boots in the rest of the DOS. Normally, this byte is a $B6 for a 48K Apple system. This means that the second stage boot routine will jump to memory location $B700. This code, which gets loaded into $B700, is located on (0, 1) on the diskette. The second stage boot procedure can be modified by the user to put substitute code for the RWTS, although this is risky unless you know what you are doing.

The *THIRD STAGE BOOT* is DOS defined and we can change it whenever we want. Normal APPLE DOS builds a special table called the I.O.B. This table is read by the RWTS routines which specify desired TRACK/SECTOR, DRIVE, READ/WRITE MODE, SLOT and other important parameters needed to operate the disk. Some description of this I.O.B. is described in the APPLE II D.O.S. 3.2 manual. Finally it jumps to 9D84 at which point DOS takes over (for 48K only).

The disk controller card collects 5 bits at a time from the diskette and passes it to the computer. This 5 bit chunk of data is a nybble. A set of routines called DOS core routines are located from B800 to BC77 (in the RWTS). These routines convert normal data to nybblized format & vice versa.

Since a nybble has five bits of data, there are 32 different data nybbles. Two specified nybbles ($D5 & $AA) are NOT data nybbles. These nybbles are used to format the diskette.

A sector consists of 2 parts: An address mark and a data mark. The address mark contains addressing information like track, sector, volume number and sync data. The DATA mark is the data sector where the disk data is actually stored. Each data sector has 430 nybbles which correspond to the 256 data bytes which get stored on the diskette.

To write a sector, we must first *PRE-NYBBLIZE* our data into a form where it gets stored onto the diskette. the RWTS routines automatically do this so we never have to worry about this. Each chunk is then indexed by a table located at $BC9A in RAM. It then gets stored into a buffer located somewhere around $BAAA to $BB00. An *ADDRESS MARK* can be broken down into the following:

1. SYNC NYBBLE—This is a mark on the diskette which marks the beginning of the address mark.

2. Two special nybbles ($D5, $AA) are used to check for proper sync.

3. MARK TYPE NYBBLE—($B5) follows this special sync information.

4. MARK INFORMATION—Track, sector, volume number of this location.

5. CHECKSUM—An error detecting nybble which checks for possible bit errors.

6. END MARK—Marks the end of the address mark block of data.

THE DATA MARK is where 256 bits of data are kept. This accounts for approximately 430 nybbles of data.

From applesauce

CONVERTING BRAND X TO WORK WITH BRAND Y

by
Randall Hyde

In the beginning there was ALTAIR and MITS BASIC. And things were just fine and dandy. Then came IMSAI, and POLYMORPHIC, and North Star, and the PET, and. . . and the Apple. Along with each new computer came a brand new BASIC which was incompatable with most of the others. People got used to converting one BASIC to another, and in fact some books have been written on the subject!

In the beginning there was the mini-assembler. Only a few souls were hardy enough to use this thing. Very few people (other than the marketing people at Apple Computer) were very happy with it. As a result several people wrote their own assembler for the Apple II. Today, over 15 different (and incompatable) assemblers exist for the Apple II. Shades of BASIC! So now, when you pick up your favorite Apple or 6502 magazine and see a source listing for some neat new utility, or possibly some new game written in assembly language you cringe. After all, it's bad enough having to put up with all the different BASICS out there on all the different machines, but why do we have to worry about converting code written using different assemblers for the same machine?

Well, fear the thought of conversion no more! Converting assembly source code from one assembler for another is actually quite easy. And although very few assemblers use the exact MOS syntax for their mnemonics etc., by following a few simple rules you too can convert. The following tables present a guide for converting between the more popular assemblers.

RESERVING MEMORY:

The following discussion assumes you wish to reserve n bytes for data storage.

ASSEMBLER	MNEMONIC
SOME VERSIONS OF TED/II	DS n or RMB n

DEFINING A HEXADECIMAL STRING

LISA	HEX nnnnnnnn. . .
SC ASM/II	. HS nnnnnnnn. . .
ORIGINAL MICROPRODUCTS	. HS nnnnnnnn. . .
ASM/65	. BYTE $nn,$nn,$nn. .
TED/II	HEX nnnnnnnn. . .
SOFTWARE CONCEPTS	DFB $nn,$nn,$nn. . .
UCSD	. BYTE nn,nn,nn. . .

PATRONIZE APPLE ORCHARD ADVERTISERS

SPECIFYING THE PROGRAM ORIGIN

ASM/65	*=n
EVERYONE ELSE	ORG n -or- . OR n

SPECIFYING WHERE THE OBJECT CODE IS TO BE STORED

ASM/65	. OFFSET n
LISA	OBJ n
TED/II	OBJ n
SC ASM/II	. TA n

STORING AN ADDRESS IN MEMORY

ASM/65	. WORD nnnn,nnnn, . . .
LISA	ADR nnnn
TED/II	ADR nnnn (some versions use DFW)
SC ASM/II	.DA nnnn
ORIGINAL MICROPRODUCTS	. SA nnnn
SOFTWARE CONCEPTS	DFD nnnn,nnnn, . . .

STORING A STRING IN MEMORY

ASM/65	. BYTE "sssssssss"
LISA	ASC "sssss" -or- ASC 'ssssss'
SC ASM/II	. AS 'sssss' -or- . AS -'sssss'
ORIGINAL MICROPRODUCTS	. AS 'sssss'
SOFTWARE CONCEPTS	ASC 'sssss'
TED/II	ASC 'sssss'

STORING THE LOW ORDER BYTE OF AN ADDRESS IN MEMORY

SC ASM/II	. DA #nnnn
LISA	BYT nnnn

STORING THE HIGH ORDER BYTE OF AN ADDRESS IN MEMORY

SC ASM/II	. DA /nnnn
LISA	HBY nnnn

EQUATING A LABEL TO AN ADDRESS

ORIGINAL MICROPRODUCTS	. DL nnnn
SC ASM/II	. EQ nnnn
ASM/65	= nnnn
LISA	EQU nnnn -or- EPZ nnnn
EVERYONE ELSE	EQU nnnn

In addition to these "standard" operations there are several pseudo opcodes which are quite specialized and have no real equal in other assemblers. Some of these pseudo opcodes include:

DCI – Stores a string in memory whose last character has an inverted high order bit (TED & LISA).
.INV – Stores string in memory in the inverted format (LISA)
BLK – Stores string in memory in the blinking format (LISA)
.OPT – Allows user to specify some assembly time options (ASM/65)
.PAGE – Skips to top of form on printer (ASM/65)
PAG – Same as .PAGE (LISA & TED/II)
STR – Stores a string in memory with a leading length byte (LISA)
LST ON – Turns listing option on (TED/II)
LST – Turns listing option on (LISA)
LST OFF – Turns listing option off (TED/II)
NLS – No listing (LISA)

There are several other pseudo opcodes floating around, but their usage is so rare that there isn't any need to discuss them here. There are some syntax difference among the various assemblers, as well as some mnemonic changes and/or extensions. Some assemblers (such as TED/II, LISA, and the Original Microproducts) support the "extended mnemonics" BGE and BLT (for branch if greater or equal, and branch if less than respectively). If your assembler doesn't support these extended mnemonics simply substitute "BCS" for "BGE" and "BCC" for "BLT". Likewise, substitute "EOR" for "XOR", "BNE" for "BTR", and "BEQ" for "BFL" should you encounter these. Some assemblers also support the Sweet-16 instruction set (TED/II, SC ASM II, and LISA). If you encounter this type of code you're better off buying one of the above assemblers instead of trying to code it by hand.

There are some syntax differences among the various assemblers. One key area where almost everyone differs concerns the immediate addressing mode. Standard MOS syntax says if you want to use the low order byte of the 16-bit value in the operand field you specify this by preceeding the address with either "#" or "#<". Most people use the first version and do not allow the second version (TED/II and ASM/65 are the exceptions). The original Microproducts assembler does not allow the "#" at all! The only type of immediate addressing available is an eight bit hex constant. The user of the Microproducts assembler specifies the immediate addressing mode by simply placing a hex digit in the operand field with no leading, or otherwise special characters. To differentiate between a label (such as FF) and a hex value greater than 9F, the Microproducts assembler requires the hex value to begin with a zero.

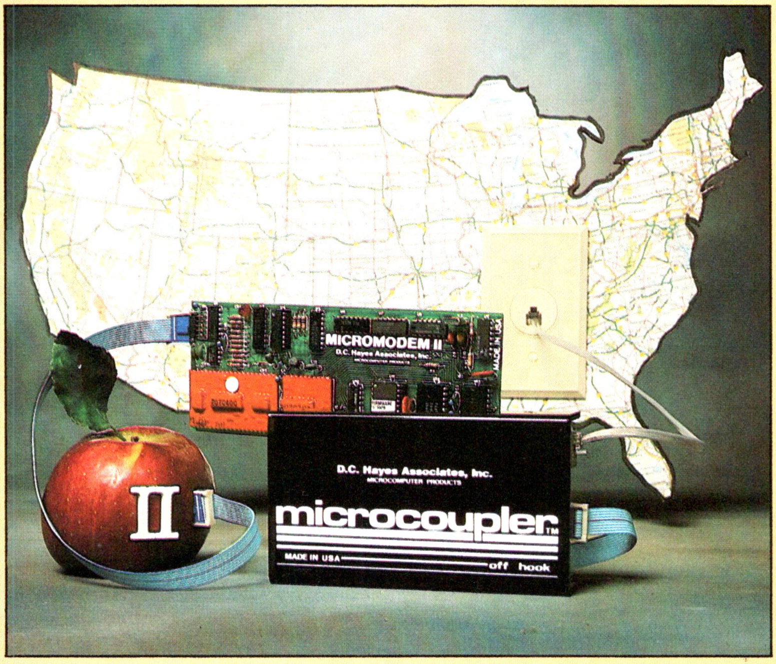
The next variation occurs when you wish to select the high order byte of a sixteen-bit address expression. Standard MOS syntax assemblers use "#>". The only two assemblers for the Apple II which use this mode are TED/II and ASM/65. The SC ASM II and LISA specify the high order immediate addressing mode by using "/" instead of the "#>". Finally, the Software Concepts assembler requires that you divide the value by 256 when you wish to use the high order byte.

As you can see, there is a considerable syntax variation from assembler to assembler. Generally however, programs published will use either the Original Microproducts assembler, TED/II, or LISA which simplifies the conversion requirement. Let us hope that 15 more incompatable assemblers for the Apple do not crop up!

```
>
>

>LIST
  10 LOMEM:3072
  50 REM SPACE TRIP BY MARK CROSS

 100 ZO=YO=COLR=0: REM  INTEGER BASIC

 200 INIT=2048:PLOT=2830
 210 REM    ABOVE CALLS FOR WOZPAK
            HI-RES ROUTINES

 300 COLR=255: CALL INIT: REM  SET UP
      BACKGROUND
 310 FOR I=1 TO 40:XO= RND (280)

 320 YO= RND (160): CALL PLOT: NEXT
     I
 330 DIM X(10),Y(10)
 340 XC=140:YC=80: REM  CENTER
 350 FOR I=1 TO 10
 360 X(I)= RND (21)-10:Y(I)= RND
     (21)-10
 370 IF ABS (X(I))<3 OR ABS (Y(I)
     )<3 THEN 360
 380 NEXT I: REM  LINES 350-370 SET U
     P MOVING STARS
 400 FOR I=1 TO 10: REM   THIS LOOP
     MOVES STARS
 410 COLR=0:XO=X(I)+XC:YO=Y(I)+YC:
      CALL PLOT
 420 X(I)=X(I)*4/3:Y(I)=Y(I)*4/3

 430 IF ABS (X(I))<140 AND ABS (
     Y(I))<80 THEN 460
 440 X(I)= RND (21)-10:Y(I)= RND
     (21)-10
 450 IF ABS (X(I))<3 OR ABS (Y(I)
     )<3 THEN 440
 460 COLR=255:XO=XC+X(I):YO=YC+Y(
     I): CALL PLOT
 470 NEXT I
 480 GOTO 400: REM   MOVE THEM ALL
     AGAIN
```


The Dawn of New Uses
for apple II*
with Micromodem II™
D.C. Hayes Associates, Inc.
MICROCOMPUTER PRODUCTS
microcoupler™
off hook
MADE IN USA

Direct Connect Your Apple II* To the Rest of America



Install Micromodem II™ and watch the dawn of new uses bring back the excitement of discovery in your Apple II. The Micromodem II package lets your Apple II communicate with any other on-line computer in North America. You can send and receive information — automatically — from across town or across the continent on standard telephone lines. Imagine the communication possibilities for business information, educational uses, scientific applications and computer games!

The Micromodem II changes computer signals into telephone compatible signals. The accompanying Microcoupler™ connects directly into the standard modular telephone wall plug. They can send or receive calls automatically whether you are calling the local Apple Bulletin Board or accessing an information service from across the continent. You can even take advantage of low evening and weekend rates on the telephone lines.

*Registered trademark of Apple Computer, Inc.

™**Micromodem II** and **Microcoupler** are trademarks of D.C. Hayes Associates, Inc.

Applecations

The following categories are presently available from remote computer services. Many of them are obtainable from a centralized service. Some are specialized services available from only one supplier. The Micromodem II and D.C. Hayes Associates provide you with the opportunity to access these programs. We do not provide the programs.

HOME AND HOBBY
Electronic Mail
Restaurant Listings & Reviews
Computer Game Library
Computer Graphics
Gourmet Meal Recipes
Syndicated Home Entertainment Features
Balancing Checkbook
Income Tax Assistance
Financial News and Commentary
Airline, Hotel, Motel and Rental Car Reservations
Biorhythms
Daily Horoscope
National Real Estate Locator Services
Nutrition Analysis

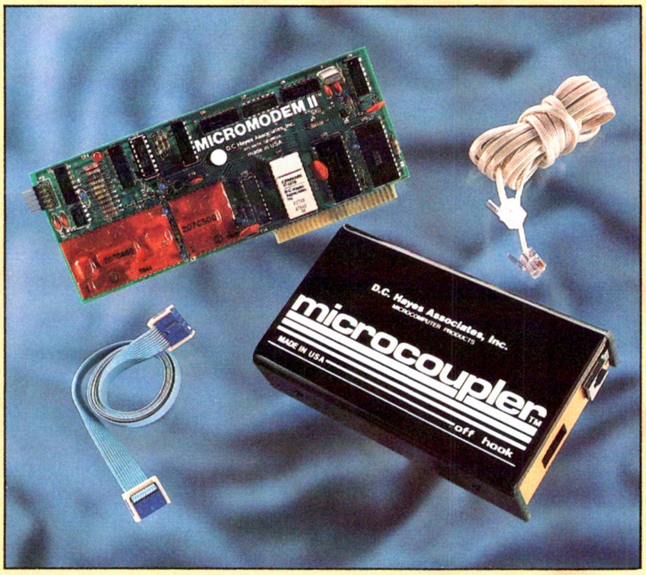
BUSINESS
Accounts Payable
Accounts Receivable
General Ledger
Payroll
Net Cash Flow
New York Stock Exchange
American Stock Exchange
Commodity Prices and Futures
Financial Commentary
Foreign Exchange Rates
Gold, Silver and Platinum Prices
UPI Wire Services
Airline, Hotel, Motel and Rental Car Reservations
Calculator
File, Editing and Sorting
Program Debugging
Programming in Extended BASIC, FORTRAN, COBOL, RPG, PASCAL

EDUCATION
Accessing College Computers
Languages: French, German and Others
Algebra
Social Sciences
Federal Financial Aid Programs
Typewriter Keyboard Drills
Installment Loan Annual Interest
Financial News
Reference Manuals And Programming Guides

SCIENCE AND INDUSTRY
Electrical Engineering
Mechanical Engineering
Computer Simulations
Statistics
Program Debugging
System Commands
Fuel Management
Thermal Hydraulics
Heat Transfer
Architectural Engineering
Circuit Analysis
Dependent Variables
Three Dimensional Analysis of Structures

There are many more specific programs available from the remote computer service companies. D.C. Hayes Associates, Inc. manufacturers the Micromodem II and the Microcoupler, and does NOT provide remote computer services.

The Micromodem II™ Package



The D.C. Hayes Associates Micromodem II™ package is a complete data communications system specifically designed for your Apple II* personal computer system. It consists of two microcomputer products and two connecting cords. The first unit, the Micromodem II plugs directly into your Apple II. It provides all the functions of a serial interface card plus programmable automatic dialing and answering. The on-board ROM firmware adds remote console, terminal mode and simplified implementation of more sophisticated applications with BASIC programs.

The Microcoupler is supplied with the Micromodem II. It is registered with the FCC and allows you to connect directly into the telephone system. It is compatible with all North American standard telephone lines and FCC approved for direct connection in the U.S.A. The unit is not approved for use on equipment originating calls in Canada; however, calls going into Canada and Mexico are accepted by their equipment. You need no additional equipment from the telephone company. Because this is a direct connection, there is none of the distortion associated with acoustic couplers. It was those distortions that caused losses in transmissions. Now your communications are processed with incredible accuracy.

A similar package is available for S100 bus computers.

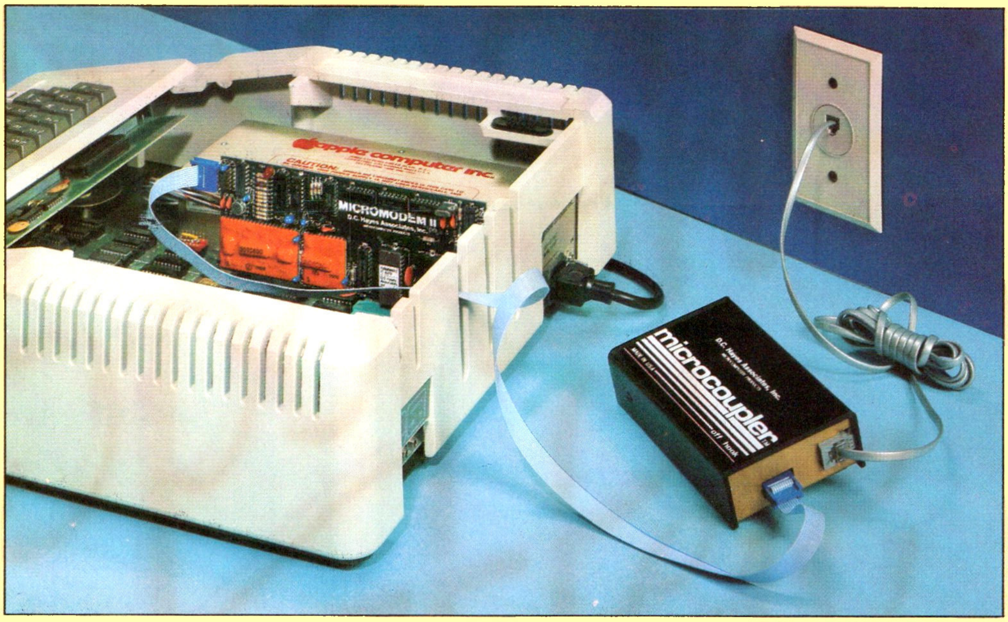
Features

- **Auto-Answer**
- **Auto-Dial**
- **Auto-Data Transfer**
- **Bell System 103 Compatible Modem**
- **Supports Originate Mode**
- **Supports Answer Mode**
- **FCC Registered**
- **Direct-Connect Microcoupler No acoustic coupler distortion or loss**
- **On-board ROM firmware**
- **Manufactured by D.C. Hayes Associates, The Micromodem Specialists**
- **Available through your local Computer Store**
- **Backed by the D.C. Hayes Associates Warranty**
- **Direct connects your Apple II™ with any time sharing computer in North America**
- **Greatly expands your present capabilities**
- **Extremely simple to connect**
- **Suggested retail of only $379**

Installing the Micromodem II™ in your Apple II*

The Micromodem II is very easy to install. Simply insert the Micromodem II circuit board into any Apple II expansion slot after slot 0. Then plug one end of the ribbon cable into the circuit board and the other end into the Microcoupler™. Snap one end of the telephone cable into the Microcoupler and the other end into your standard modular telephone jack. You are now completely installed. Just notify the telephone company that you are using an FCC registered device. There is no extra telephone company equipment needed.

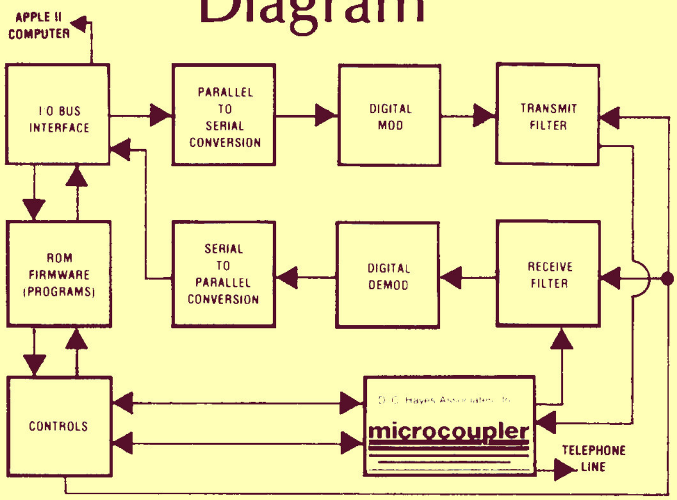
Enjoy the new worlds open to your Apple II!



Specifications

Data Format	Serial, binary, asynchronous 7 or 8 data bits, 1- or 2-stop bits odd, even, or no parity.
Lower case characters	Can be optionally converted to upper case, or can be passed through unmodified.
Firmware	1024 byte read only memory (ROM)
Power consumption	1.5 W Typical
Card size	7" x 3" including connector fingers
Microcoupler size	5-½" x 3-¼" x 1-⅜"
Modem compatibility	Bell System 103-compatible originate or answer mode, dial pulse dialing and auto-answer -50 dBm receive sensitivity -10 dBm transmit level 110 or 300 baud data rate
FCC registration	FCC Registration No. BI986H-62226-PC-E. Ringer equivalence 0.4B. Connects with modular jacks RJ11W or RJ11C.
Supplied with	Modem interface card, firmware in ROM, Microcoupler™, connector cables, owner's manual.
Suggested Retail	$379.00

Micromodem II Functional Block Diagram



All D.C. Hayes Associates, Inc. Products Are Available at Your Computer Store

D.C. Hayes Associates, Inc.

MICROCOMPUTER PRODUCTS

10 Perimeter Park Drive, Atlanta, Georgia 30341 (404) 455-7663

 Specifications subject to change without notice. Printed in U.S.A. 97-028

From Call-A.P.P.L.E.

GAME PADDLE PORT PRINTER DRIVER

by
Darrell & Ron Aldrich

Here is some valuable information that can perhaps save you up to $200.00, by eliminating the need for a printer interface card. This is subject to two conditions:

1. That the printer is not to be used in conjunction with a language card system, and
2. That it is a TTL Compatible serial mode printer such as the Integral Data series or the Heath H-14

The complete connection can be made with three wires as described below, and the printer can be operated at 1200 baud serial mode, with handshake, using the driver program on this page.

A CALL 768 will bring the printer up; PRINT D$"PR#0" will turn it off, Window width may be set as follows:

Upon initialization, POKE 777,width.
After initialization, POKE 940,width.

Two characters required to set character spacing on the Integral Data printers are not directly available from the Apple keyboard. This is provided for with two additional CALLs:

CALL 930 ($3A2) to set 8.3 characters per inch.
CALL 935 ($3A7) to set 16.5 characters per inch.

Baud rate may be found at $383.

PIN CONNECTIONS

	TO APPLE GAME I/O
IDS 225 pin 5 CLEAR TO SEND	pin 4 (SWITCH 2)
IDS 225 pin 7 SIGNAL GROUND	pin 8 (GROUND)
IDS 225 pin 3 RECEIVE DATA	pin 15 (ANNUNCIATOR 0)
IDS 440 pin 20 DATA READY	pin 4 (SWITCH 2)
IDS 440 pin 7 SIGNAL GROUND	pin 8 (GROUND)
IDS 440 pin 3 RECEIVE DATA	pin 15 (ANNUNCIATOR 0)

CAUTION: The Game I/O is not buffered. Make certain your corrections are correct before powering up.

```
WNDWDTH   EQU $21
CH        EQU $24
CV        EQU $25
CSWL      EQU $36
WNDSET    EQU $3AC
WNDSAV    EQU $3AD
DRTRN     EQU $3EA
YSAVE     EQU $778
COLCNT    EQU $7F8
MARK      EQU $C058
SPACE     EQU $C059
SW2       EQU $C063
VTAB      EQU $FC22
WAIT      EQU $FCA8
COUT      EQU $FDED
*
          ORG $300
          OBJ $300
*
TTINIT    TYA
          PHA
          LDY CSWL
          JSR CONNECT
          NOP
          LDA #40
          STA WNDSET
          LDA CH
          STA COLCNT
          PLA
          CPY #$F0
          BEQ NODOS
          TAY
          JMP DRTRN
NODOS     TAY
          RTS
*
TTOUT     PHA
          PHA
          LDA WNDWDTH
          STA WNDSAV
          LDA WNDSET
          STA WNDWDTH
TTOUT2    LDA #$48
          JSR WAIT
          LDA COLCNT
          CMP CH
          PLA
          BCS TESTCTRL
          PHA
          LDA #$A0
TESTCTRL  BIT RTS1
          BEQ PRNTIT
          INC COLCNT
PRNTIT    JSR DOCHAR
          PLA
          PHA
          BCC TTOUT2
          EOR #$0D
          ASL
          BNE FINISH
          STA COLCNT
          LDA #$8A
          JSR DOCHAR
          LDA #$58
          JSR WAIT
FINISH    LDA COLCNT
          BEQ SETCH
          SBC WNDWDTH
          SBC #$F7
          BCC RETURN
          ADC #$1F
SETCH     STA CH
RETURN    LDA WNDSAV
          STA WNDWDTH
          PLA
RTS1      RTS
*
DOCHAR    STY YSAVE
          PHP
          LDY #$0B
          CLC
TTOUT3    PHA
          BCS MARKOUT
          LDA SPACE
          BCC TTOUT4
MARKOUT   LDA MARK
TTOUT4    LDA #$14
DLY1      PHA
          LDA #$20
DLY2      LSR
          BCC DLY2
          PLA
          SBC #$01
          BNE DLY1
          PLA
          ROR
          DEY
          BNE TTOUT3
          LDY YSAVE
          PLP
END       RTS
*
CONNECT   LDA #<TTOUT
          STA CSWL
          LDA #>TTOUT
          STA CSWL+1
          RTS
*
CTRL\     LDA #$9C
          JMP COUT
*
CTRL_     LDA #$9F
          JMP COUT
PGMEND    BRK
```

From Call-A.P.P.L.E.

INTEGRAL DATA HI-RES SCREEN DUMP

by
Darrell & Ron Aldrich

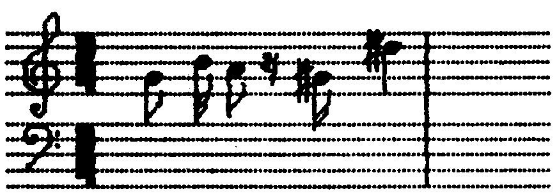
```
1            ORG $C00
2            OBJ $4000
3 *
4 COUT       EQU $FDED
5 *
6 HBASL      EQU $26
7 HBASH      EQU $27
8 *
9            LDA #$20
10           STA HPAG
11           LDA #$00
12           STA X
13           STA X+1
14 XLOOP     LDA #$00
15           STA CHR
16           LDA Y1
17           STA Y2
18 CHRLOOP   LDA Y2
19           LDX X
20           LDY X+1
21           JSR HPOSN
22           LDA HMASK
23           AND #$7F
24           AND (HBASL),Y
25           CMP #$00
26           BEQ OFF
27           LDA #$40
28 OFF       STA SCRATCH
29           LDA CHR
30           LSR
31           ADC SCRATCH
32           STA CHR
33           INC Y2
34           LDA Y2
35           SBC Y1
36           CMP #3
37           BNE CHRLOOP
38           LDA CHR
39           JSR COUT
40           LDA X
41           CMP #$17
42           LDA X+1
43           SBC #$01
44           INC X
45           BNE SKIP1
46           INC X+1
47 SKIP1     BCC XLOOP
48           RTS
49 *
50 HPOSN     STA Y0
51           STX X0L
52           STY X0H
53           PHA
54           AND #$C0
55           STA HBASL
56           LSR
57           LSR
58           ORA HBASL
59           STA HBASL
60           PLA
61           STA HBASH
62           ASL
63           ASL
64           ASL
65           ROL HBASH
66           ASL
67           ROL HBASH
68           ASL
69           ROR HBASL
70           LDA HBASH
71           AND #$1F
72           ORA HPAG
73           STA HBASH
74           TXA
75           CPY #$00
76           BEQ HPOSN2
77           LDY #$23
78           ADC #$04
79 HPOSN1    INY
80 HPOSN2    SBC #$07
81           BCS HPOSN1
82           STY HNDX
83           TAX
84           LDA MSKTBL-$F9,X
85           STA HMASK
86           TYA
87           LSR
88           LDA HCOLOR
89 HPOSN3    STA HCOLOR1
90           BCS CSHFT2
91           RTS
92 *
93 CSHFT2    ASL
94           CMP #$C0
95           BPL RTS1
96           LDA HCOLOR1
97           EOR #$7F
98           STA HCOLOR1
99 RTS1      RTS
100 INVERSE  LDA #$20
101          STA HPAG
102          STA HBASH
103          LDY #$00
104          STY HBASL
105 INVLOOP  LDA (HBASL),Y
106          EOR #$FF
107          STA (HBASL),Y
108          INY
109          BNE INVLOOP
110          INC HBASH
111          LDA HBASH
112          AND #$1F
113          BNE INVLOOP
114          RTS
115 MSKTBL   HEX 8182848890A0C0
116 Y1       HEX 00
117 X        HEX 0000
118 Y2       HEX 00
119 CHR      HEX 00
120 SCRATCH  HEX 00
121 X0L      HEX 00
122 X0H      HEX 00
123 Y0       HEX 00
124 HCOLOR   HEX 00
125 HNDX     HEX 00
126 HPAG     HEX 00
127 HMASK    HEX 00
128 HCOLOR1  HEX 00
```

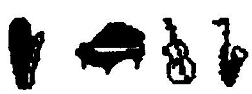
The Hi-Res screen dump programs as listed are for the IDS225. Two changes are required for them to operate correctly with the Paper Tiger (IDS440). Basic line 220 should be changed to read as follows:

```
220 FOR Y0=0 TO 186 STEP 6: POKE
    3302, Y0: CALL 3072
```

In the assembly program, change location $C48 from 03 to 05 (reference line 36 of source code.)

```
>LIST
  10 REM  HI-RES SCREEN DUMP ROUTINE
  20 REM        FOR INTEGRAL DATA
  30 REM        IP - 225 PRINTER

  40 REM   BY RON & DARRELL ALDRICH
 100 B$="":C$="":D$="":K$="": REM
      CTRL B, C, D & K
 110 X0=Y0=COLR: DIM A$(40)
 120 PRINT D$;"NOMON C,I,O"
 130 TEXT : CALL -936: POKE -16297
     ,0
 140 PRINT D$;"BLOAD IP225 DRIVER"
     : PRINT D$;"BLOAD HI-RES DUMP.B"
 170 CALL -936: INPUT "NAME OF PICTUR
     E ",A$: IF A$="" THEN 170
 180 PRINT D$;"BLOAD ";A$;",V,A$2000"
     : GR
 190 CALL -936: INPUT "INVERT IMAGE T
     O PRINTER ?",A$: IF A$="" THEN
     190: IF ASC(A$)=217 THEN 210
     : IF ASC(A$)#206 THEN 190
 200 CALL 3266
 210 CALL 768: CALL 935: PRINT :
      PRINT C$: REM  CALL PRINTER; SE
     T SMALL TYPE; SET GRAPHICS
 220 FOR Y0=0 TO 188 STEP 4: POKE
     3302,Y0: CALL 3072: REM   PRINT
     LINE
 230 PRINT C$;K$;: NEXT Y0
 240 PRINT C$;B$: PRINT D$;"PR#"

 250 END
```





PRINTED IN 2.5 MINUTES ON IP 225

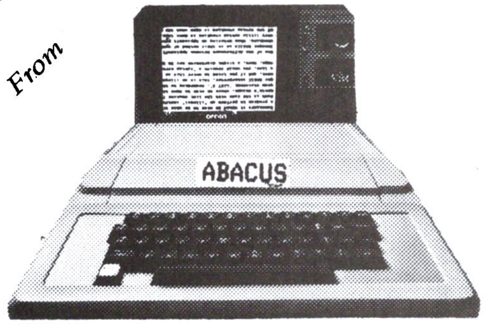
```
]LIST

10  REM  HOW TO USE VAL & STR$
         FUNCTIONS IN APPLESOFT

100 :A$ = "32767"
110 :A =  VAL (A$)
120  PRINT A
130 :B$ =  STR$ (A)
140  PRINT B$
150  END
```

*C00.CE7 Hi - Res Dump

```
0C00- A9 20 8D F1 0C A9 00 8D
0C08- E7 0C 8D E8 0C A9 00 8D
0C10- EA 0C AD E6 0C 8D E9 0C
0C18- AD E9 0C AE E7 0C AC E8
0C20- 0C 20 66 0C AD F2 0C 29
0C28- 7F 31 26 C9 00 F0 02 A9
0C30- 40 8D EB 0C AD EA 0C 4A
0C38- 6D EB 0C 8D EA 0C EE E9
0C40- 0C AD E9 0C ED E6 0C C9
0C48- 03 D0 CD AD EA 0C 20 ED
0C50- FD AD E7 0C C9 17 AD E8
0C58- 0C E9 01 EE E7 0C D0 03
0C60- EE E8 0C 90 A8 60 8D EE
0C68- 0C 8E EC 0C 8C ED 0C 48
0C70- 29 C0 85 26 4A 4A 05 26
0C78- 85 26 68 85 27 0A 0A 0A
0C80- 26 27 0A 26 27 0A 66 26
0C88- A5 27 29 1F 0D F1 0C 85
0C90- 27 8A C0 00 F0 05 A0 23
0C98- 69 04 C8 E9 07 B0 F8 8C
0CA0- F0 0C AA BD E6 08 8D F2
0CA8- 0C 98 4A AD EF 0C 8D F3
0CB0- 0C B0 01 60 0A C9 C0 10
0CB8- 08 AD F3 0C 49 7F 8D F3
0CC0- 0C 60 A9 20 8D F1 0C 85
0CC8- 27 A0 00 84 26 B1 26 49
0CD0- FF 91 26 C8 D0 F7 E6 27
0CD8- A5 27 29 1F D0 EF 60 81
0CE0- 82 84 88 90 A0 C0 00 00
```

*300.3AF Printer - Driver

```
0300- 98 48 A4 36 20 99 03 EA
0308- A9 28 8D AC 03 A5 24 8D
0310- F8 07 68 C0 F0 F0 04 A8
0318- 4C EA 03 A8 60 48 48 A5
0320- 21 8D AD 03 AD AC 03 85
0328- 21 2C 63 C0 10 FB AD F8
0330- 07 C5 24 68 B0 03 48 A9
0338- A0 2C 6F 03 F0 03 EE F8
0340- 07 20 70 03 68 48 90 E1
0348- 49 0D 0A D0 0D 8D F8 07
0350- A9 8A 20 70 03 A9 58 20
0358- A8 FC AD F8 07 F0 08 E5
0360- 21 E9 F7 90 04 69 1F 85
0368- 24 AD AD 03 85 21 68 60
0370- 8C 78 07 08 A0 0B 18 48
0378- B0 05 AD 59 C0 90 03 AD
0380- 58 C0 A9 14 48 A9 20 4A
0388- 90 FD 68 E9 01 D0 F5 68
0390- 6A 88 D0 E3 AC 78 07 28
0398- 60 A9 1D 85 36 A9 03 85
03A0- 37 60 A9 9C 4C ED FD A9
03A8- 9F 4C ED FD 28 28 00 02
```



APPLE BAY AREA COMPUTER USERS SOCIETY

MODIFICATIONS TO THE APPLE II DISPLAY UPPER AND LOWER CASE LETTERS

by
John Macdougall

This article describes a simple modification to the Apple II. which can be used to display, either upper/lower case letters, when using the Apple Writer Text Editor, or can be used to display an alternate character set. The modification consists of removing the existing 2513 character generator ROM and replacing it with a 2716 EPROM. The 2716 contains two character sets. The first is the standard duplicate of the 2513 and the second is a special set which, for example, works with the Text Editor characters.

Since the 2716 is not pin compatible with the 2513, an interconnect pattern is needed. In addition, certain connections must be made to the main board. To do this effectively, a small circuit board is used which holds the 2716 and plugs into the 2513 socket. Three wires from this board then go to "piggyback" socket extensions on the main board. By this means, the modification is simply plug-in and no modifications are required to the main board. A circuit diagram of this small board and its interconnections is presented in figure 1.

How the Circuit Works:

Imagine that your character generator ROM has two character areas. The first of these is an upper case area and the second is a lower case area. Switching between these two areas can be accomplished by using a high address bit. This turns out to be very appropriate to the Apple Text Editor since it in fact stores the characters such that upper case characters have the high bit set low so that they will display in inverse video. This bit is picked up from pin 6 of B13 and is used to select the ROM area from which the display character is selected. There is one problem with this method, and that is that the high bit set low tells the Apple hardware to set an inverse character. The result of this simple modification is that we now have lower case but the upper case is still in inverse video. The solution is to put into the ROM the inverse characters so that although the Apple thinks it is displaying an inverse character it is really displaying the inverse of an inverse.

There is still a problem when you come to observe the resulting characters. They have funny lines and extra information which is very distracting. This is solved by getting at the shift register parallel load inputs and setting them with a sixth bit from the ROM. To do this they must be lifted from ground and connected to the little board. Thus pins 3 and 14 are cut and the lead from the 2716 is connected to the 74166 pins.

A final refinement to the system is to make the selection of mode software selectable. So rather than put a switch on the circuit board, the mode select address pin is connected to the game socket at annunciator pin 3. The latch which provides this putput always comes up with a low output on power-on. The addressing is arranged so that this gives the normal character set in Apple. The result is that to the unsuspecting user, the system configuration looks exactly as he has always seen it and he will never know that there is lower case present. The case can be set and reset as follows:

PATRONIZE APPLE ORCHARD ADVERTISERS

For use with the text editor, the conversion to lower case can be made automatic by putting the lower case PEEK into the editor HELLO program as follows:

```
5 D$="": REM CONTROL-D
10 PRINT D$; "NOMON I,O,C": CALL -936
20 POKE 1010, 191: POKE 1011,157: POKE 1012,56
30 POKE -16289,0
40 PRINT D$; "BRUNTEDITOR"
50 END
```

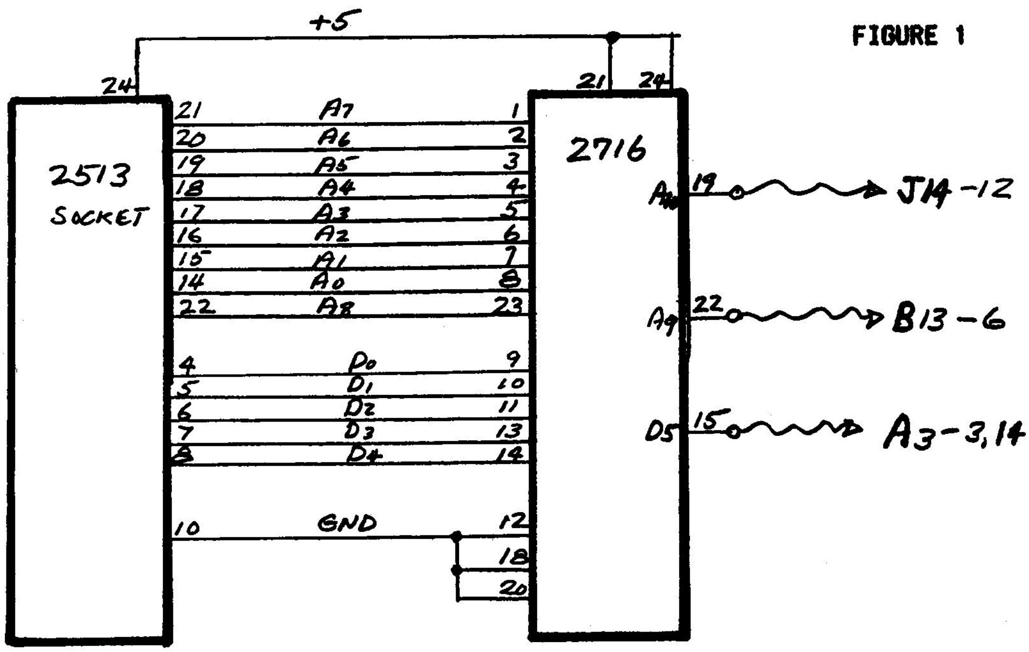
Other Features.

Because of the independent character sets with this system, it is possible to have additional characters. You may have noticed the odd brackets used above. The special characters, which can be accessed by this system, as currently implemented, are as follows:

[— esc-control-n
] — esc-shift-m
— control-n
— shift-m
~ — shift-n
^ — esc-shift-n

CIRCUIT FOR DISPLAYING UPPER/LOWER CASE LETTERS USING THE APPLE TEXT EDITOR.

FIGURE 1

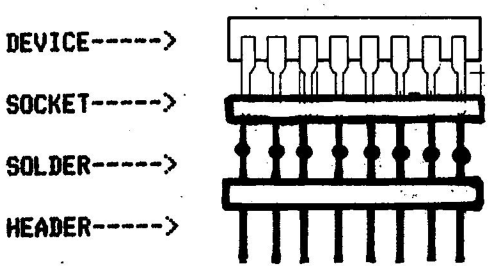


J14: USE A 16 PIN SOCKET,ATTACH PIN 12 TO THE 2716 at PIN 19. PLUG THIS SOCKET INTO GAME PADDLE SOCKET,GAME PADDLES MAY THEN BE PLUGGED INTO TOP OF THIS SOCKET

B13: AT LOCATION B13,REMOVE THE IC (74LS02) THEN TAKE A 14 PIN SOCKET AND ATTACH A WIRE TO PIN 6,CONNECT THE OTHER END OF THIS WIRE TO PIN #22 OF THE 2716. NOW INSERT THIS SOCKET INTO LOCATION B13,THEN REINSTALL IC (74LS02) INTO THIS SOCKET.

A3: AT LOCATION A3 REMOVE IC (74166) THEN TAKE A 16 PIN SOCKET AND CONNECT A WIRE TO BOTH PINS 3 AND 14,CONNECT THE OTHER END OF THIS WIRE TO PIN #15 OF THE 2716. BESURE TO CUT PINS 3 AND 14 SHORT SO THEY DO NOT GO THRU AND INTO THE BOARD SOCKET,HOWEVER ALL REMAINING PINS MUST CONNECT TO BOARD SOCKET. NOW PLUG SOCKET INTO LOCATION A3 THEN REINSERT IC (74166) INTO THIS SOCKET.

Example of a Piggyback Socket Mounting.



A listing of the hex code for the character ROM is presented on the following pages.

```
000- 0E 11 15 17 16 10 0F 00
008- 04 0A 11 11 1F 11 11 00
010- 1E 11 11 1E 11 11 1E 00
018- 0E 11 10 10 10 11 0E 00
020- 1E 11 11 11 11 11 1E 00
028- 1F 10 10 1E 10 10 1F 00
030- 1F 10 10 1E 10 10 10 00
038- 0F 10 10 10 13 11 0F 00
040- 11 11 11 1F 11 11 11 00
048- 0E 04 04 04 04 04 0E 00
050- 01 01 01 01 01 11 0E 00
058- 11 12 14 18 14 12 11 00
060- 10 10 10 10 10 10 1F 00
068- 11 1B 15 15 11 11 11 00
070- 11 11 19 15 13 11 11 00
078- 0E 11 11 11 11 11 0E 00
080- 1E 11 11 1E 10 10 10 00
088- 0E 11 11 11 15 12 0D 00
090- 1E 11 11 1E 14 12 11 00
098- 0E 11 10 0E 01 11 0E 00
0A0- 1F 04 04 04 04 04 04 00
0A8- 11 11 11 11 11 11 0E 00
0B0- 11 11 11 11 11 0A 04 00
0B8- 11 11 11 15 15 1B 11 00
0C0- 11 11 0A 04 0A 11 11 00
0C8- 11 11 0A 04 04 04 04 00
0D0- 1F 01 02 04 08 10 1F 00
0D8- 1F 18 18 18 18 18 1F 00
0E0- 00 10 08 04 02 01 00 00
0E8- 1F 03 03 03 03 03 1F 00
0F0- 00 00 04 0A 11 00 00 00
0F8- 00 00 00 00 00 00 00 3F
100- 00 00 00 00 00 00 00 00
108- 04 04 04 04 04 00 04 00
110- 0A 0A 0A 00 00 00 00 00
118- 0A 0A 1F 0A 1F 0A 0A 00
120- 04 0F 14 0E 05 1E 04 00
128- 18 19 02 04 08 13 03 00
130- 08 14 14 08 15 12 0D 00
138- 04 04 04 00 00 00 00 00
140- 04 08 10 10 10 08 04 00
148- 04 02 01 01 01 02 04 00
150- 04 15 0E 04 0E 15 04 00
158- 00 04 04 1F 04 04 00 00
160- 00 00 00 00 04 04 08 00
168- 00 00 00 1F 00 00 00 00
170- 00 00 00 00 00 00 04 00
178- 00 01 02 04 08 10 00 00
180- 0E 11 13 15 19 11 0E 00
188- 04 0C 04 04 04 04 0E 00
190- 0E 11 01 06 08 10 1F 00
198- 1F 01 02 06 01 11 0E 00
1A0- 02 06 0A 12 1F 02 02 00
1A8- 1F 10 1E 01 01 11 0E 00
1B0- 07 08 10 1E 11 11 0E 00
1B8- 1F 01 02 04 08 08 08 00
1C0- 0E 11 11 0E 11 11 0E 00
1C8- 0E 11 11 0F 01 02 1C 00
1D0- 00 00 04 00 04 00 00 00
1D8- 00 00 04 00 04 04 08 00
1E0- 02 04 08 10 08 04 02 00
1E8- 00 00 1F 00 1F 00 00 00
1F0- 08 04 02 01 02 04 08 00
1F8- 0E 11 02 04 04 00 04 00
200- 0E 11 15 17 16 10 0F 00
208- 04 0A 11 11 1F 11 11 00
210- 1E 11 11 1E 11 11 1E 00
218- 0E 11 10 10 10 11 0E 00
220- 1E 11 11 11 11 11 1E 00
228- 1F 10 10 1E 10 10 1F 00
230- 1F 10 10 1E 10 10 10 00
238- 0F 10 10 10 13 11 0F 00
240- 11 11 11 1F 11 11 11 00
248- 0E 04 04 04 04 04 0E 00
250- 01 01 01 01 01 11 0E 00
258- 11 12 14 18 14 12 11 00
260- 10 10 10 10 10 10 1F 00
268- 11 1B 15 15 11 11 11 00
270- 11 11 19 15 13 11 11 00
278- 0E 11 11 11 11 11 0E 00
280- 1E 11 11 1E 10 10 10 00
288- 0E 11 11 11 15 12 0D 00
290- 1E 11 11 1E 14 12 11 00
298- 0E 11 10 0E 01 11 0E 00
2A0- 1F 04 04 04 04 04 04 00
2A8- 11 11 11 11 11 11 0E 00
2B0- 11 11 11 11 11 0A 04 00
2B8- 11 11 11 15 15 1B 11 00
2C0- 11 11 0A 04 0A 11 11 00
2C8- 11 11 0A 04 04 04 04 00
2D0- 1F 01 02 04 08 10 1F 00
2D8- 1F 18 18 18 18 18 1F 00
2E0- 00 10 08 04 02 01 00 00
2E8- 1F 03 03 03 03 03 1F 00
2F0- 00 00 04 0A 11 00 00 00
2F8- 00 00 00 00 00 00 00 3F
300- 00 00 00 00 00 00 00 00
308- 04 04 04 04 04 00 04 00
310- 0A 0A 0A 00 00 00 00 00
318- 0A 0A 1F 0A 1F 0A 0A 00
320- 04 0F 14 0E 05 1E 04 00
328- 18 19 02 04 08 13 03 00
330- 08 14 14 08 15 12 0D 00
338- 04 04 04 00 00 00 00 00
340- 04 08 10 10 10 08 04 00
348- 04 02 01 01 01 02 04 00
350- 04 15 0E 04 0E 15 04 00
358- 00 04 04 1F 04 04 00 00
360- 00 00 00 00 04 04 08 00
368- 00 00 00 1F 00 00 00 00
370- 00 00 00 00 00 00 04 00
378- 00 01 02 04 08 10 00 00
380- 0E 11 13 15 19 11 0E 00
388- 04 0C 04 04 04 04 0E 00
390- 0E 11 01 06 08 10 1F 00
398- 1F 01 02 06 01 11 0E 00
3A0- 02 06 0A 12 1F 02 02 00
3A8- 1F 10 1E 01 01 11 0E 00
3B0- 07 08 10 1E 11 11 0E 00
3B8- 1F 01 02 04 08 08 08 00
3C0- 0E 11 11 0E 11 11 0E 00
3C8- 0E 11 11 0F 01 02 1C 00
3D0- 00 00 04 00 04 00 00 00
3D8- 00 00 04 00 04 04 08 00
3E0- 02 04 08 10 08 04 02 00
3E8- 00 00 1F 00 1F 00 00 00
3F0- 08 04 02 01 02 04 08 00
3F8- 0E 11 02 04 04 00 04 00
400- F1 EE EA E8 E9 EF F0 FF
408- FB F5 EE EE E0 EE EE FF
410- E1 EE EE E1 EE EE E1 FF
418- F1 EE EF EF EF EE F1 FF
420- E1 EE EE EE EE EE E1 FF
428- E0 EF EF E1 EF EF E0 FF
430- E0 EF EF E1 EF EF EF FF
438- F0 EF EF EF EC EE F0 FF
440- EE EE EE E0 EE EE EE FF
448- F1 FB FB FB FB FB F1 FF
450- FE FE FE FE FE EE F1 FF
458- EE ED EB E7 EB ED EE FF
460- EF EF EF EF EF EF E0 FF
468- EE E4 EA EA EE EE EE FF
470- EE EE E6 EA EC EE EE FF
478- F1 EE EE EE EE EE F1 FF
480- E1 EE EE E1 EF EF EF FF
488- F1 EE EE EE EA ED F2 FF
490- E1 EE EE E1 EB ED EE FF
498- F1 EE EF F1 FE EE F1 FF
4A0- E0 FB FB FB FB FB FB FF
4A8- EE EE EE EE EE EE F1 FF
4B0- EE EE EE EE EE F5 FB FF
4B8- EE EE EE EA EA E4 EE FF
4C0- EE EE F5 FB F5 EE EE FF
4C8- EE EE F5 FB FB FB FB FF
4D0- E0 FE FD FB F7 EF E0 FF
4D8- E0 E7 E7 E7 E7 E7 E0 FF
4E0- FF EF F7 FB FD FE FF FF
4E8- E0 FC FC FC FC FC E0 FF
4F0- FF FF FB F5 EE FF FF FF
4F8- FF FF FF FF FF FF FF C0
500- FF FF FF FF FF FF FF FF
508- FB FB FB FB FB FF FB FF
510- F5 F5 F5 FF FF FF FF FF
518- F5 F5 E0 F5 E0 F5 F5 FF
520- FB F0 EB F1 FA E1 FB FF
528- E7 E6 FD FB F7 EC FC FF
530- F7 EB EB F7 EA ED F2 FF
538- FB FB FB FF FF FF FF FF
540- FB F7 EF EF EF F7 FB FF
548- FB FD FE FE FE FD FB FF
550- FB EA F1 FB F1 EA FB FF
558- FF FB FB E0 FB FB FF FF
560- FF FF FF FF FB FB F7 FF
568- FF FF FF E0 FF FF FF FF
570- FF FF FF FF FF FF FB FF
578- FF FE FD FB F7 EF FF FF
580- F1 EE EC EA E6 EE F1 FF
588- FB F3 FB FB FB FB F1 FF
590- F1 EE FE F9 F7 EF E0 FF
598- E0 FE FD F9 FE EE F1 FF
5A0- FD F9 F5 ED E0 FD FD FF
5A8- E0 EF E1 FE FE EE F1 FF
5B0- F8 F7 EF E1 EE EE F1 FF
```

```
5B8- E0 FE FD FB F7 F7 F7 FF
5C0- F1 EE EE F1 EE EE F1 FF
5C8- F1 EE EE F0 FE FD E3 FF
5D0- FF FF FB FF FB FF FF FF
5D8- FF FF FB FF FB FB F7 FF
5E0- FD FB F7 EF F7 FB FD FF
5E8- FF FF E0 FF E0 FF FF FF
5F0- F7 FB FD FE FD FB F7 FF
5F8- F1 EE FD FB FB FF FB FF
600- 08 04 02 00 00 00 00 00
608- 00 00 0E 01 0F 11 0F 00
610- 10 10 1E 11 11 11 1E 00
618- 00 00 0F 10 10 10 0F 00
620- 01 01 0F 11 11 11 0F 00
628- 00 00 0E 11 1F 10 0F 00
630- 06 09 08 1E 08 08 08 00
638- 00 00 0E 11 11 0F 01 0E
640- 10 10 1E 11 11 11 11 00
648- 04 00 0C 04 04 04 0E 00
650- 02 00 06 02 02 02 12 0C
658- 10 10 11 12 1C 12 11 00
660- 0C 04 04 04 04 04 0E 00
668- 00 00 1B 15 15 15 11 00
670- 00 00 1E 11 11 11 11 00
678- 00 00 0E 11 11 11 0E 00
680- 00 00 1E 11 11 1E 10 10
688- 00 00 0F 11 11 0F 01 01
690- 00 00 17 18 10 10 10 00
698- 00 00 0F 10 0E 01 1E 00
6A0- 08 08 1E 08 08 09 06 00
6A8- 00 00 11 11 11 13 0D 00
6B0- 00 00 11 11 11 0A 04 00
6B8- 00 00 11 11 15 15 1B 00
6C0- 00 00 11 0A 04 0A 11 00
6C8- 00 00 11 11 11 0F 01 0E
6D0- 00 00 1F 02 04 08 1F 00
6D8- 07 0C 0C 18 0C 0C 07 00
6E0- 04 04 04 04 04 04 04 04
6E8- 1C 06 06 03 06 06 1C 00
6F0- 0D 16 00 00 00 00 00 00
6F8- 7F 7F 7F 7F 7F 7F 7F 7F
700- 00 00 00 00 00 00 00 00
708- 04 04 04 04 04 00 04 00
710- 0A 0A 0A 00 00 00 00 00
718- 0A 0A 1F 0A 1F 0A 0A 00
720- 04 0F 14 0E 05 1E 04 00
728- 18 19 02 04 08 13 03 00
730- 08 14 14 08 15 12 0D 00
738- 04 04 04 00 00 00 00 00
740- 04 08 10 10 10 08 04 00
748- 04 02 01 01 01 02 04 00
750- 04 15 0E 04 0E 15 04 00
758- 00 04 04 1F 04 04 00 00
760- 00 00 00 00 04 04 08 00
768- 00 00 00 1F 00 00 00 00
770- 00 00 00 00 00 00 04 00
778- 00 01 02 04 08 10 00 00
780- 0E 11 13 15 19 11 0E 00
788- 04 0C 04 04 04 04 0E 00
790- 0E 11 01 06 08 10 1F 00
798- 1F 01 02 06 01 11 0E 00
7A0- 02 06 0A 12 1F 02 02 00
7A8- 1F 10 1E 01 01 11 0E 00
7B0- 07 08 10 1E 11 11 0E 00
7B8- 1F 01 02 04 08 08 08 00
7C0- 0E 11 11 0E 11 11 0E 00
7C8- 0E 11 11 0F 01 02 1C 00
7D0- 00 00 04 00 04 00 00 00
7D8- 00 00 04 00 04 04 08 00
7E0- 02 04 08 10 08 04 02 00
7E8- 00 00 1F 00 1F 00 00 00
7F0- 08 04 02 01 02 04 08 00
7F8- 0E 11 02 04 04 00 04 00
```

After construction is complete, carefully check all connections. Then install the board, making doubly sure the proper locations are selected.

Be sure to add the previously mentioned POKES to the Apple-writer program. Then resave for future use.

You now have a very sophisitcated text editor for your Apple. You will now enjoy typing letters, since what you see on your screen is what you get on the printer.

good luck . . .

PAGE FLIP

by
Glen Hoag
From: The Cider Press

The following brief program is a demonstration of how to move from TEXT page 1 to page 2 from BASIC.

```
10  REM (1) SET UP PAGE 1 SCREEN
11  REM (2) SET UP POINTERS FOR PAGE 1 START
12  REM (3) SET UP POINTERS FOR PAGE 1 END
13  REM (4) SET UP POINTERS FOR PAGE 2 START
14  REM (5) CALL MONITOR MOVE ROUTINE
15  REM (6) POKE THE SWITCH TO DISPLAY PAGE 2

20  REM STEP (2)
21  POKE 60,0: POKE 61,4: REM POKE VALUES FOR A1
30  REM STEP (3)
31  POKE 62,255: POKE 63,7: REM POKE VALUES FOR A2
    (CONTAINS $07FF)
40  REM STEP (4)
41  POKE 66,0: POKE 67,8: REM POKE VALUES FOR A4
    (CONTAINS $0800)
50  REM STEP (5)
51  CALL -468: REM MONITOR MOVE ROUTINE LIVES AT
    $0FE2CH
60  REM STEP (6) 61 CALL -936: REM CLEAR PAGE 1

61  CALL -936: REM CLEAR PAGE 1

62  POKE -16299,0: REM TURN ON PAGE 2
70  PRINT "THIS IS PAGE 1"
71  FOR I=0 TO 1000: NEXT I
72  POKE -16300,0: REM TURN ON PAGE 1
73  FOR I=0 TO 1000: NEXT I
74  GO TO 62
```

From **the michigan APPLE-gram**

APPLE TYPER

by
W. Curt Deegan

One thing I like to see in an article is an example of how to use the material being described. While it may be obvious to the author, or even everybody but me, I still feel it is far easier for the author to describe his work than for me to translate it. While this article will be brief, I trust it will not be incomplete.

APPLE TYPER is my description mechanism, while the actual topic of discussion will be TRIM PRINT. TRIM PRINT is an Integer Basic utility program with its use and interface demonstrated in the simple application program, APPLE TYPER.

It is always more impressive if the instructive or textual output of a program is neatly spaced on the display or print line such that words are not split one line to the next. This is precisely what TRIM PRINT does. By including one routine in your programs and passing output to it rather than using the PRINT command directly, you can have automatically spaced print/display lines.

APPLE TYPER is an extra expensive electric typewriter program which uses TRIM PRINT to keep things neat.

First to look at TRIM PRINT.

The TRIM PRINT routine is line numbered from 32000. There are five lines that are included to demonstrate a printer interface using the APPLE-II serial card. These lines, 32210 through 32240 and 32260 can be removed if not required for your application. The rest of the routine has been kept small and uses as few variable names as possible to avoid conflict with the main program in which it is embedded. Also, the interfact to TRIM PRINT has been kept simple for the same reason.

Following is a table of variable usage:

- 0$ — output passed to TRIM PRINT
- L$ — current print line being built or partial line carried over
- L0 — unused
- L1 — length carried over
- L2 — pointer to current position in 0$
- L3 — length of output passed to TRIM PRINT in 0$
- L4 — amount of 0$ to append to L$
- L5 — used for controlling blanks
- L6 — position where line should be broken
- L7 — unused
- L8 — print line length
- L9 — unused

The first part of the routine, through line 32090, determines if a null line was passed to instruct the printing of a short line. Also checked for is the need for a filler blank between the end of the last line passed and the beginning of the current line. The next few program steps append sufficient characters from 0$ to fill out L$ to the full L8 line length. The FOR loop from line 32170 backsteps in L$ to find the first occurrence of a blank at which the output line can be broken. The printer control follows and the line is then either printed or displayed by line number 32250. The last bit of code determines how much is left to be printed and if it is sufficient for another print line, goes through all the above again. The pointers will be properly positioned at this time and TRIM PRINT returns to the using program.

With all that said, now to the example, APPLE TYPER.

The use of 0$ to pass the output with a GOSUB 32010 activating TRIM PRINT is shown. The variable N$ is used by APPLE TYPER as an End-of-Session test character and is not related to TRIM PRINT operation. The REMARKS in APPLE TYPER should be sufficient to describe the remainder of its function.

Note that 0$ has been DIMensioned to the maximum input character string length. By so doing, the APPLE-II Monitor will signal excessive line length without any user programming required.

Obviously, much more could be done within APPLE TYPER to spiffy up its function and operation, but that was not necessary to demonstrate the TRIM PRINT interface, and is therefore left as an exercise to the reader.

I trust the TRIM PRINT utility, if not APPLE TYPER, will be of use, instructive, or both.

```
>LIST
  100 REM   ******************
  110 REM
  120 REM
  130 REM    ---APPLE TYPER---
  140 REM
  150 REM    BY:
  160 REM       W.C. DEEGAN
  170 REM       17 MAY 1979
  180 REM       DETROIT, MI
  190 REM
  200 REM    ================
  210 REM    =ALL  COMMERCIAL=
  220 REM    =RIGHTS RESERVED=
  230 REM    ================
  240 REM
  250 REM   ******************
  260 REM   ---
  270 REM   ...VARS FOR TRIM RTN
  280 REM   ---
  290 L8=80: REM  LINE LENGTH
  300 DIM O$(255): REM  INPUT VAR
  310 DIM L$(L8): REM  PRINT VAR
  320 REM
  330 REM   ---
  340 REM   ...SET PRINTER FOR 600BPS
  350 REM   ---
  360 POKE 1145,32: REM OVE
  370 REM
  380 REM   ******************
  390 REM   ---
  400 REM   ...INITIALIZATION
  410 REM   ---
  420 REM   CLEAN THE SCREEN
  430 TEXT : CALL -936
  440 REM
  450 REM   END-OF-INPUT CHARCTER
  460 N$="": REM  <CNTL>N
  470 REM
  480 REM   ******************
```

```
  490 REM
  500 REM  <TYPER> INPUT ROUTINE
  510 REM
  520 TAB 10: PRINT "A P P L E   T Y P
       E R"
  530 VTAB 4
  540 REM  PROMPT THEN ACCEPT INPUT
  550 INPUT "=>",O$
  560 REM  IF EMPTY LINE,FORCE OUTPUT
  570 IF LEN(O$)=0 THEN 660
  580 REM  IF <CNTL>N FORCE OUTPUT
  590 REM     AND END PROGRAM
  600 IF O$(1)#N$ THEN 660
  610 O$="": REM  NULL LINE
  620 GOSUB 32010
  630 END
  640 REM  GIVE INPUT TO "TRIM"
  650 REM     AND GO AGAIN
  660 GOSUB 32010
  670 GOTO 550
32000 END
32010 REM  *******************
32020 REM
32030 REM  <TRIM> PRINT ROUTINE
32040 REM
32050 L2=0:L3= LEN(O$)
32060 IF L3=0 THEN 32310
32070 IF L1=0 THEN 32100
32080 IF O$(1,1)=" " OR L$(L1)=" "
       THEN 32100
32090 L1=L1+1:L$(L1)=" "
32100 L4=L8-L1
32110 IF L2+L4>L3 THEN L4=L3-L2
32120 L$(L1+1)=O$(L2+1,L2+L4)
32130 L1= LEN(L$)
32140 IF L1=L8 THEN 32160
32150 IF L3=0 THEN 32200: GOTO 32320

32160 L5=1
32170 FOR L6=L1 TO 1 STEP -1
32180 IF L$(L6,L6)=" " THEN 32210

32190 NEXT L6:L5=0
32200 L6=L1
32210 POKE 54,7: REM OVE
32220 POKE 55,193: REM OVE
32230 POKE 1785,L8+1: REM OVE
32240 POKE 2041,1: REM OVE
32250 PRINT L$(1,L6-L5)
32260 PR#0: REM OVE
32270 IF L6=L1 THEN 32290
32280 L$(1)=L$(L6+1,L1)
32290 L1=L1-L6:L2=L2+L4:L5=0
32300 IF L2<L3 THEN 32100
32310 IF L1>0 AND L3=0 THEN 32200

32320 RETURN
```

```
]LIST

5  REM      " COLOR TWENTYONE "

        BY  DARRELL ALDRICH

10  GR : HGR : HOME : PRINT "TWEN
     TY-ONE COLORS"
20  DATA  GREEN,VIOLET,WHITE,BLAC
     K,ORANGE,BLUE
30  FOR I = 1 TO 6: READ A$(I): NEXT

40  FOR A = 1 TO 6: FOR B = A TO
     6
50  VTAB 23: PRINT A$(B)"-"A$(A)"
     "
60  FOR C = 29 TO 119 STEP 2
70  HCOLOR= A: HPLOT 40,C TO 240,
     C
80  HCOLOR= B: HPLOT 40,C + 1 TO
     240,C + 1
90  NEXT C,B,A: GOTO 40
```

From
the michigan
APPLE-gram

For those who enjoyed APPLE TYPER, or found its TRIM PRINT routine useful, APPLE TYPER-II is just for you.

With a minimum of modification to the original program and the addition of new routine, the APPLE TYPER-II program gives trimmed and right justified lines of print. Use APPLE TYPER-II where you would normally use a print statement and all of your program output will have a cleaner more professional appearance.

As with the original APPLE TYPER, the variable names used in RIGHT ADJUST are rather cryptic to make integration of these routines into other programs simpler. The new variables used in the RIGHT ADJUST routine are:

- LO — total spaces in line
- L7 — #blanks to add at each space in line
- L9 — # of line spaces where 1 more blank is added
- L11 — loop index
- L12 — loop index

To keep the record straight, some variables are also used for loop indices or temporary storage as well as for the function listed.

The listing which follows shows only those lines from the original APPLE TYPER which are changed or new for the APPLE TYPER-II. The change to line 310 is required only to please the perfectionists. If L8 (the line length) is set to one — because no one said not to — L$ must be at least 2, hence this change. Line 315 — all new lines are numbered ending in 5 — sets up a variable used to move in the spaces druing justification. Lines 32150 and 32180 are changed and line 32205 added to provide the linkage to the new RIGHT ADJUST routine. Line 32265 is added to allow passing back to TRIM PRINT from RIGHT ADJUST the actual length of the unjustified print line. 32280 has been changed to use the new variable LL$ to build the next print line. Line 32310 has been changed and 32325 added to properly handle line length adjustment in light of the fact the printed line is usually equal to L8 while the length of the line passed to TRIM PRINT and RIGHT ADJUST and used by them is something less than L8 due to the padding with blanks for justification purposes.

The RIGHT ADJUST routine is lines 32330 through 32690. The program is structured as follows:

- 32400—32410 determine number of fill blanks required
- 32420—32440 determine number of spaces in the line
- 32450—32460 set number of blanks to add at each space in the line spaces where 1 more blank is needed to fill out line
- 32480—32630 move through line adding blanks as each space is found, first add one based on L7, then as appropriate, add 1 more according to L9

There are a few tests included in the RIGHT ADJUST routine that are not specifically called out above. They generally are to determine if a special case exists; i.e. line with no blanks, line requiring no justification etc.

That's all there is to APPLE TYPER-II. It should be noted that the Integer BASIC implementation of this program means a somewhat slow execution of the filling process. If you are interested in speeding things up a bit, consider removing the REMarks, move the RIGHT ADJUST routine to the beginning of the program followed by TRIM PRINT and then the TYPER routine at the end. Also combine as many lines as possible — while avoiding any changes to the logic of the program.

Happy typing, trimming, and/or justifying.

APPLE TYPER - II

by
W. Curt Deegan

```
>LIST
  310 DIM L$(L8+1): REM  PRINT VAR ***
      CHANGED***
  315 DIM LL$(L8+1): REM  JUSTIFY VAR
      ***ADDED***
32150 IF L3=0 THEN 32200: RETURN
      : REM ***CHANGED***
32180 IF L$(L6,L6)=" " THEN 32205
      : REM ***CHANGED***
32205 GOSUB 32400: REM ***ADDED***
32265 L6=L7: REM ***ADDED***
32280 L$(1)=LL$(L6+1,L1): REM ***CHANG
      ED***
32310 IF L1<=0 OR L3<>0 THEN RETURN
      : REM ***CHANGED***
32325 L6=L1:L7=L6: GOTO 32210: REM ***
      ADDED***
32330 REM
32340 REM ************************
32350 REM *RIGHT JUSTIFY
32360 REM * SUBROUTINE
32370 REM *W.CURT DEEGAN
32380 REM *19 JUNE 1979
32390 REM ************************
32400 L7=L8-(L6-L5):LO=0
32410 IF L7=0 THEN 32630
32420 FOR L9=1 TO L6-L5
32430 LO=LO+(L$(L9,L9)=" ")
32440 NEXT L9:LL$=L$
32450 IF LO=0 THEN 32630
32460 L9=L7 MOD LO:L7=L7/LO
32480 L11=1
32500 FOR LO=1 TO L6-L5
32510 L$(L11)=LL$(LO,LO)
32520 IF L$(L11,L11)#" " THEN 32610

32530 IF L7=0 THEN 32580
32540 FOR L12=1 TO L7
32550 L11=L11+1:L$(L11)=" "
32570 NEXT L12
32580 IF L9=0 THEN 32610
32590 L11=L11+1:L9=L9-1
32600 L$(L11)=" "
32610 L11=L11+1: NEXT LO
32620 L7=L6:L6=L8:L5=0: RETURN
32630 L7=L6:L5=0: RETURN
32640 REM *********************
32650 REM *ALL COMMERCIAL RIGHTS
32660 REM *  ---RESERVED---
32670 REM * BY: W.CURT DEEGAN
32680 REM *     19 JUNE, 1979
32690 REM *********************
```

R & R FOR DECIMAL DUMPS

by
Max J. Nareff
From: The Cider Press

Calculations in Applesoft usually result in long multidecimal numbers. While the accuracy of the numbers is commendable, long mantissas are often not necessary; frequently they are disruptive of the three - columns - via - .commas screen format the Apple provides.

Following is a simple function for reducing post-decimal numbers and for rounding off the residuals (R&R).

One of the many functions preprogrammed in Applesoft is INT, used to drop the fractional part of a number. The excision is sharp and clear.

Y=INT (3.4729): PRINT Y

Will result in "3". It doesn't round off; it truncates. Thus:

Z=INT (6.87563): PRINT Z

Yields "6", with everything to the right of the decimal ignored. In order to retain numbers to the right of the decimal and to "round them off", we must define a function ourselves. We create this special action by means of the DEF (ine) F (unctio) N command.

The function reads DEF FNA(X)=INT (X) where FN indicates function and the following letter can be any alphabetic character merely serving to define that particular function at the time of its use. The (X) is the value to be acted upon.

The following simple program tells the story:

```
20 DATA 3.4678, 19.2062, 11.562, 141.45917, 1000
30 READ X
40 IF X-1000 THEN 70
50 PRINT X, INT (X)
60 GOTO 30
70 END
```

Note that the standard integer function, INT (X), drops the decimals. Now add:

```
5 DEF FNA(X)=INT (X*100 +.5)/100
```

and rewrite line 50 to read:

```
50 PRINT X, INT(X), FNA (X)
```

Note how the DEF FNA(X) limits the post-decimal numbers to two and "rounds them off". The latter is due to the addition of .5, which increases by 1 those decimal numbers .5 or more; anything less is not propelled over the carry-over cliff. When dealing with several variables (X, Y, Z), just plug them between the parentheses.

APPLE-II HEX and the TI Programmer's Calculator

by
Curt Deegan

If you have a Texas Instruments programmer's calculator that does both decimal and hexadecimal computations, here's how to compute those negative APPLE Integer BASIC POKEs, PEEKs, and CALLs; setp by step.

1. Turn it on.
2. Push the HEX key.
3. Enter the HEX address.
4. Push the - key.
5. Enter 10000.
6. Push the = key.
7. Push the DEC key.

And presto, one decimal address, ready to go.

NOW PRESENTING...

Software for Apple II

for your Entertainment · Business · Education

Star Attractions:

WRITE-ON Professional Word Processing lets you edit, move, delete, find, change and repeat any body of text, merge and save on disk. Does right-justified margins, centering, page numbering. You can enter name & address into form letters when printing. Edit and merge any text disk file—even files not created by WRITE-ON—and spool text to disk for letter printing or editing. Chain files when printing for an infinite number of pages in a single printer run. Needs Applesoft and 32K.
On Disk with operating manual. $99.50

FILEMASTER 2 programs: *FORMAT & RETRIEVAL* comprise a powerful data file manager. Great for everything from phone lists to legal abstracts. Needs 32K. Design your own data structure. Up to 500 characters per record. Up to 15 searchable fields in any combination. **On Disk. $34.95**

SPACE Multi-faceted simulation of life in interstellar society. You and opponents must make life & death decisions. Keeps track of your progress from one game to next. Needs 48K and Applesoft ROM. **Disk $29.95**

Pot O'Gold I or our All New Pot O' Gold II A collection of 49 programs for 16K Apple. Everything from Logic to action games. Only a buck a game. Specify I or II. Price each: **Tape $49 . . . . Disk $54**

ADVENTURE Fight off pirates and vicious dwarfs. 700 travel options, 140 locations, 64 objects. Needs ROM & 48K. **Disk. . $29.95**

16K CASSETTE INVENTORY Use item number, description, stock amount, reorder amount, restock date, cost & sell price. Holds up to 140 items. **Tape $35**

32K DISK INVENTORY: Use stock numbers description, vendor, record of purchase and sales date, amount on hand, cost & sell price, total value. Holds up to 300 items. **Disk $40**
With Parts Explosion: Disk $50

32K DATA BASE Cross file for phone lists, bibliographies, recipes. Run up to 9 lines of 40 columns each. Search by item anywhere. **Disk. $20**

24K HI-RES LIFE SIMULATION Conway's equations on 296x180 screen. A mathematical simulation to demo population growth with birth, death and survival as factors. **Tape $10**

16K CIRCUIT LOGIC DEVELOPMENT AID Evaluate circuits of up to 255 gates, including AND, OR, NOR, NAND, XOR, XNOR and INVERTER. **Tape. $10**

16K MORSE CODE TRAINER Learn Morse Code, and transmit or receive over radio. **Tape $10**

16K PACIFICA: Discover the floating island and rescue the beautiful princess. To win you must recover the enchanted crown, but you face the threat of magic spells and demons. **Tape. $9.95**

RAINBOW'S CASINO 9 gambling games: Roulette, Blackjack, Craps, Horserace, and a few originals that Vegas hasn't heard about. Needs 16K. **Tape $29.95**

16K SPACE WAR: You in your space capsule battle against the computer's saucer . . . in hi-res graphics. **Tape $12**

16K MEMORY VERIFY Diagnostic routine to check range of memory. Indicates faulty addresses, data in memory cell, and faulty data. **Tape $5**

16K APPLEODION Music synthesis composes original Irish jigs. Enter your own music and save on tape or disk. Includes 3 Bach fugues. **Tape. $10**

16K APPLEVISION Demo for Hi-Res graphics and music. **Tape $10**

32K COMPU-READ 5 programs to teach you speed reading, in stages. Includes synonym and antonym identification. You control your rate of speed, or keep up with the computer's pace. **Disk. $24.95**

48K PERCEPTION I, II, III random shapes and sizes must be matched. In III, you control format and display time and get weighted scores. Needs ROM. **Each Disk. $24.95**

24K POLAR PLOT Plot polar equations in Hi-Res Graphics. **Tape $10**

32K SHAPE SCALER Utility to generate and animate Hi-Res graphic shapes. Simple routine provided to inspect position of shapes, and specify precise X/Y coordinates and scale. Needs ROM. **Disk. $13.95**

APPLE MONITOR PEELED Everything you wanted to know about the Apple Monitor but couldn't figure out. User-written manual in plain English clears your confusion. **Only $9.95**

Don't see what you've been looking for, here? Then write for our FREE SOFTWARE CATALOG. We're saving one just for you!

To order, add $2 shipping. California residents add 6% sales tax. Sorry, we can not ship to P.O. Boxes. VISA/ MASTERCHARGE and BANKAMERICARD Welcomed!

VISA

master charge

RAINBOW COMPUTING INC.

Garden Plaza Shopping Center, Dept. 1A
9719 Reseda Blvd., Northridge, Ca 91324
Telephone: (213) 349-5560

From applesauce

CONVERTING INTEGER BASIC PROGRAMS TO ASSEMBLY LANGUAGE

by
Randall Hyde

Programming in Integer Basic is easy, right? (if not, put this article down and go learn BASIC *FIRST!)* Programming in assembly language is hard, right? (If not, don't even bother reading this!) On the other hand, programs written BASIC are very slow whereas programs written in assembly language are very fast. Oh, the cruel world is full of such compromises. One of the main reasons that BASIC is so slow is because it is interpreted. This simply means that the BASIC interpreter has to figure out what each BASIC statement means *WHILE THE PROGRAM IS RUNNING.* The obvious disadvantage here is that the interpreter spends a lot of time just figuring out what the current statement is (this action is often called "Parsing"). Some systems, such as Apple Pascal do not interpret the source code but rather "Compile" it into a type of machine code. This has the advantage that we can enter the source code into the system in a language not too far removed from English and yet enjoy the benefits of a pseudo-compiled code (Pascal actually generates a "pseudo-code" which is then interpreted). The disadvantage to a compiled language is the fact that you don't get nice run-time error messages (complete with line number telling you where the error occurred). Typically you get something like "FLOATING POINT OVERFLOW" and that's it. You, the programmer, must figure out where in the program the error actually occurred. This type of debugging can take hours (in fact it can even take several days). Obviously, for debugging purposes, an interpretive language such as BASIC is preferred.

Nevertheless, the day will come when all of the bugs are removed from the program (or at least most of them!) and the programmer begins to prefer speed over the interaction. The obvious solution is to write a BASIC compiler, which would take your error-free Integer Basic code and convert it to 6502 machine language. Although it would be very easy to write an Integer Basic compiler for standard Integer Basic, one problem arises. Many people have used non-standard techniques to simulate missing functions such as CHR$, STR$, VAL, etc. Most of these "bastardized" functions depend on the fact that Integer Basic stores its variables in a very standard format. Unfortunately, an Integer Basic compiler would not take this into account, and as a result it would not compile such programs correctly.

All is not lost however. Although the computer is not smart enough to compile such Integer Basic programs correctly, the human programmer (by following a few simple rules) is perfectly capable of "hand compiling" such programs. Although hand compilation is not trivial, it is not all that hard and any intermediate programmer who knows Integer Basic fairly well and assembly language moderately well should be capable of hand compiling his Integer Basic programs.

The first rule is *"IF SOMEONE ELSE HAS ALREADY DONE SOMETHING FOR YOU, DON'T DUPLICATE THEIR EFFORTS."* This rule particularly applies to input and output in assembly language. There are several output packages available ranging from the primitive I/O routines in the Apple monitor to more sophisticated routines provided by several vendors. In the examples presented in this article, I will use the LZR IOS input / output routines which are provided with LISA v1.5.

SIMULATING THE "PRINT" STATEMENT

There are actually three forms of the print statement which we must consider.

1) Printing a string constant: PRINT "HELLO THERE"
2) Printing an integer variable: PRINT I
3) Printing a string variable: PRINT A$

Any other form of the PRINT statement can be simulated by using one of the above three forms. For example, PRINT "A=", " I=, I can be simulated by:

```
PRINT "A=";
PRINT A$;
PRINT " I=";
PRINT I
```

The LZR IOS subroutine package includes routines to realize these three functions. The subroutine "PCIM" (for Print Characters IMmediate) allows us to print any string constant, the subroutine "PINT" (for Print INTeger) allows us to print an integer variable, and "PCIA" (for Print Characters INdirect) allows us to print string variables.

Let's consider PCIM first. To use this subroutine simply follow the JSR with the ASCII string to be printed terminated by a HEX OO.

EXAMPLE:

```
JSR PCIM
ASC "I="
HEX OO
```

Prints "I-" onto the video screen.

Using PINT is just as easy, simply follow the JSR PINT with the address of the integer you wish printed.

EXAMPLE:

```
JSR PINT
ADR I
```

Finally, to print a string variable we use the PCIA subroutine in a manner identical to PINT.

EXAMPLE:

```
JSR PCIA
ADR A$
```

So to convert PRINT "I=", I," A=",A$ to assembly language you would use:

```
JSR PCIM
ASC "I="
HEX OO
JSR PINT
ADR I
JSR PCIM
ASC "A="
JSR PCIA
ADR A$
LDA #$8D
JSR COUT
```

Note that you must explicitly output the return. Sometimes, when using PCIM, you can include the carriage return as part of the string you are printing.

EXAMPLE:

```
JSR PCIM
ASC "HELLO THERE"
HEX 8DOO
```

The last thing to remember is that string must always be terminated with a hex OO.

SIMULATING THE "INPUT" STATEMENT

This statement has two forms, you may either input an integer or you can input a string. If a prompting string appears in the INPUT statement use the PCIM subroutine to print it.

INPUTTING A STRING:

This is simple, use the "GETLN" routine in the Apple monitor or the "ROLN" routine iin the LZR IOS package. When these routines are called they will read a line from the Apple keyboard and store the resulting code in page 2 ($200-$2FF). Upon returning to your program your code can move this data into the desired string locations.

INPUTTING AN INTEGER:

This one is not quite so simple. First read a line of text from the keyboard (using GETLNZ or equivalent) Next check in page two for the first (or next) occurrence of a decimal digit (i.e., move a pointer past all the blank characters). If a digit is not found report some kind of error and make the user re-enter the number. If a digit is found store the index into page two in the Y register. Next, call the subroutine "CVHD" (provided in the LZR iIOS routines). This subroutine will look at the location pointed at by the Y register in page two and convert the following characters to the internal binary representation required by the 6502 microprocessor. The resulting 16-bit integer is stored in location ADDR (see the LZR IOS source listing provided with LISA for the exact address). After the call to CVDH is made you can move the data in ADDR and ADDR+1 to the desired location.

. . . . so much for input & output . . . .

Program Constructs, or "So That's How They Do It!"

THE GOTO STATEMENT:

The GOTO statement has an identical partner in assembly language. The assembly language equivalent is the JMP (jump) instruction. The only difference is that GOTO always references a line number whereas JMP always references an absolute address or a label. Because you can jump to labels (and not have to worry about line numbers which haven't been defined yet) The 6502 JMP instruction is actually EASIER to use than the BASIC GOTO instruction.

```
EXAMPLE        BASIC          ASSEMBLY
               10 GOTO 100    JMP L100
               ,              ,
               ,              ,
               ,              ,
               100 END        L100 BRK  ; ETC.
```

THE IF STATEMENT:

The IF statement, because of its many variations, is not a trivial instruction to "hand-compile". Before discussing assembly language equivalents it is necessary to discuss how to break a complex IF statement into a group of simple IF statements. A complex IF statement contains the OR, AND, and NOT operators, a simple IF statement does not contain any of these operators.

example 1

```
IF (COND1) AND (COND2) THEN (STATEMENT)
                  is the same as
IF NOT (COND1) THEN LINE #
IF NOT (COND2) THEN LINE #
(STATEMENT / 1K
(LINE) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Note that if either condition is false you will skip over (statement #1) as is the case in the "AND" version.

example 2

```
IF (COND1) OR (COND2) THEN (STATEMENT)
                  is the same as
IF (COND1) OR (COND2) THEN (STATEMENT)
                  is the same as
IF (COND1) RHEN (LINE)#1)
IF NOT (COND2) THEN (LINE #2)
(LINE #1)
(LINE#2) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

If for instancewe execute (LINE#1) unless both (COND1) AND (COND2) are false.

Now that AND and OR are out of the way, let's discuss the various conditions. BASIC supports the logical operators "#", "=", ">=", ">", "<", and "=". Assembly language, when using the CMP instruction, only supports "#", "=", "<", and ">=". This presents only a minor problem since "<=" can be synthesized using "<" and "=", and ">" can be synthesized by using ">=" and "#".

16 Bit Comparison Synthesis

X=Y	X#Y	X>= Y	X<Y
LDA X	LDA X	LDA X	LDA X
CMP Y	CMP Y	CMP Y	CMP Y
BNE NTEQ<	BNE NTEQL	LDA X+$1	LDA X+$1
LDA X+$1	LDA X+$1	SBC Y+$1	SBC Y+$1
CMP Y+$1	CMP Y+$1	BLT LSTHAN	BGE GTREQL
BNE NTEQL	BEQ EQUAL		
NTEQL:			

In each case the program drops all the way through if the condition is met, and branches off to some other location if the condition is not met. So a statement such as

if I<10 then 100

is converted as:

```
LDA I
CMP #!10   ;DECIMAL 10
LDAI+$1
SBC /!10  ;H.O. BYTE OF DECIMAL 10
BLT LIN100
```

Obviously the h.o. byte of 10 is 0, but "/!10" conveys the meaning much better. As another example:

IF (J <K) AND (I #10) THEN 100

is converted to:

```
LDA J
CMP K
LDA J+$1
SBC K+$1
BGE NOTLS
LDA I
CMP #!10
BNE NOTLS
LDA I+$1
CMP /!10
BEQ LIN100
```

NOTES:

Admittedly, this is a lot of code just to translate one BASIC statement. Maybe now you can appreciate BASIC's efficiency (code efficiency that is!) a little better.

Comparing strings is simply beyond the scope of this article, but the idea is still the same. It just takes more code is all.

THE FOR/NEXT LOOP

This one is fairly easy. First, let's break the for-look down in BASIC and simulate it using IF statements.

```
FOR I=1 TO 1000
        (PGM CODE GOES HERE)
NEXT I
```

- becomes-

```
I=1
10 IF  I >1000 THEN nnnn (NNNN is some four digit line #)
20 (PROGRAM CODE GOES IN HERE)
nnnn-2    I=I+1
nnnn-1    GOTO 10
nnnn (next statement after the loop)
```

Since we already know how to simulate an IF statement and the GOTO statement, half of our work is done. Now all we need to do is translate the BASIC statements "I=1" and "I=I+1", I=I+1 is easy, simply use the following code:

```
LDA #!1
STA I
LDA /!1
STA I+$1
```

For I=I+1 we could load the accumulator with I, add one to the accumulator and store into I, etc. But a better way is to use the 6502 INC instruction as follows:

```
  INC I
  BNE THERE
  INC I+$1
THERE:
```

This code performs a 16-bit increment on location I.

One last thing to note about our loop to improve efficiency. The 6502 does not have a "branch if less than or equal" instruction. As a result we have to simulate this instruction by combining the "BLT" instruction with the "BEQ" instruction. By doing this the computer has to execute more instructions which not only takes longer but uses more code as well. A much better way to do this is to compare I with the value 1001 instead of the value 1000. By doing this we can get by with using just the "BLT" instruction since 1000 is definitely less than 1001. The previously described FOR loop gets coded as:

```
        LDA #!1
        STA I
        LDA /!1
        STA I+$1
FORLP   LDA I
        CMP #!1001
        LDA I+$1
        SBC /!1001
        BLT FORLPO
        JMP FORXIT
FORLPO:
        (PGM CODE GOES HERE)
        INC I
        BNE FORLP1
        INC I+$1
FORLP1  JMP FORLP
FORXIT:
```

Note the extensive use of the jump instruction where the branch instruction looks like it should suffice. This is due to the fact that the branch instructions use the relative addressing mode and as such have a limited range. If the program code which goes inside the FOR/NEXT loop is very short you may use the following code:

```
        LDA #!1
        STA I
        LDA /!1
        STA I+$1
FORLP   LDA I
        CMP #!1001
        LDA I+$1
        SBC /!1001
        BGE FORXIT
        (PGM CODE GOES HERE)
        INC I
        BNE FORLP
        INC I+$1
        JMP FORLP
FORXIT: (REMAINDER OF PGM FOLLOWS)
```

The previous discussion assumes that constants are used and the stepsize is one. What happens if you have a statement of the form:

```
FOR I=J TO K
                   OR POSSIBLY-
FOR I= 1000 TO STEP -1
                   -OR EVEN-
FOR I= 1 TO 1000 STEP 2
```

In the first example here, where we have a variable for the initial value and a variable for the final value (or any permutation thereof), we have to alter our original program only a little bit. First, we would initialize I to J instead of the constant 1. This is accomplished as follows:

```
LDA J
STA I
LDA J+$1
STA I+$1
```

Next we have to worry about the comparison. Remember, the comparison is a less than or equal compare which doesn't exist on the 6502 processor. In order to make life convenient for us it would be nice if we could add one to K before making the comparison. Unfortunately we do not want to disturb the value currently in K because other parts of the program (even within the loop) may need to access K. What we can do is create a temporary location, poke the value of K+1 into it, and then use this temporary location for our comparisons. The resultant code looks like the following:

```
        CLC
        LDA K
        ADC #$1
        STA TEMPK
        LDA K+$1
        ADC #$0
        SBC TEMPKL+$1
FORLP   LDA I
        CMP TEMPK
        LDA I+$1
        SBC TE¼L+$1
        BGE FORXIT

        ETC.
```

The next problem on our agenda is that of a stepsize other than one. If the stepsize is a positive value other than one we can simply replace the INC sequence with:

```
        CLC
        LDA I
        ADC STPSIZ
        STA I
        LDA I+$1
        ADC STPSIZ+$1
        STA I+$1
        JMP FORLP
```

(This, of course, may be optimized if the actual constant is known at assembly time)

If the stepsize is negative, things are not quite so simple. The problem here is that we don't exit the loop when the index is greater than the final value, but rather we exit the loop when the index is less than the final value. The nice thing, of course, is that we can use the BLT instruction to test for the end of the loop and we don't have to worry about adding one before performing the test. For the loop:

```
        FOR I=1000 to 1 STEP -1
```

we could use the initialization and testing code:

```
        LDA #!1000
        STA I
        LDA /!1000
        STA I+$1
FORLP   LDA I
        CMP #!1
        LDA I+$1
        SBC /!1
        BLT FORXIT

        ETC.
```

If the stepsize is -1 we can use the special code sequence:

```
        LDA I
        BNE FORLP1
        DEC I+$1
FORLP1  DEC I
        JMP FORLP
```

otherwise you must use:

```
        SEC
        LDA I
        SBC STPSIZ
        STA I
        LDA I+$1
        SBC STPSIZ+$1
        STA I+$1
        JMP FORLP
```

naturally this can be improved upon if the stepsize is a constant known at assembly time.

This guide to coding FOR/NEXT loops does not consider all possible combinations, but it does present a guideline which can be used to code FOR/NEXT loops in assembly language. One thing to be aware of: if the stepsize is a variable which takes on both positive and negative values during the course of a program, things get very *HAIRY!* Thank God this almost never occurs.

CONVERTING PEEKS, POKES, AND CALLS

These are the easiest of all. A PEEK is simply a LDA instruction in disguise. Likewise a POKE instruction is simply a STA instruction in disguise. A CALL is simply a JSR in disguise.

```
POKE I,J  LDA I
          STA J (SORTOF)
I=PEEK(J) LDA J
          STA I
          LDA #$0 ;MUST ZERO H.O. BYTE
          STA I+$1
CALL -936 JSR $FC58 ;DECIMAL CONVERTED TO HEX
```

CONVERTING GOSUB & RETURN

Just like the GOTO & JMP Instructions the GOSUB gets translated straight across to a JSR instruction. Likewise, the BASIC RETURN statement gets translated directly to an RTS instruction.

CONVERTING THE DIM STATEMENT:

DIM is used to allocate storage in a BASIC program. BASIC allocates its storage dynamically. That is, while the program is running you can choose the size of the array by using a statement of the form:

DIM X(I)

where I is some variable which has been previously defined to be some value. Although it is possible to allocate storage dynamically in assembly language, this process is by no means trivial. As a result, we must make the restriction that all arrays dimensioned be dimensioned with a constant value as opposed to a variable.

With this in mind there are two basic forms of the DIM statement. Dimensioning an integer array, and dimensioning a string.

The statement which corresponds to the DIM statement in assembly language is the DFS (or define storage) pseudo opcode. In reserving memory for your arrays you must keep in mind that integer arrays require two bytes per array element and string arrays require one byte per array element plus one byte for the length.

so DIM X(100), A$(20) would be converted to:

```
X  DFS  !200
A$ DFS  !21
ETC.
```

THE ASSIGNMENT STATEMENT (LET)

The assignemnt statement, because there are so many variations on it (in fact there is almost an infinite number of variations), is easily the hardest statement to convert to assembly language.

Rather than getting involved in a long discussion of operator precedence and RPN and so on and so forth, I will avoid these topics all together and let you worry about it. I will concentrate here on how you perform such operations as addition, subtraction, multiplication, division, etc. I am going to assume that all assignment statements have been broken down into one of two forms:

1) variable = term
or
2) variable = term op term

where term is either a variable or a constant and "op" is either "+", "-", "/", or "*" (notice how I avoid "^", AND, OR, NOT, >=, ETC. but the ideas presented still apply to these operators).

Any normal BASIC assignment statement can be broken down into this format. For instance, the BASIC statement:

I= (J+K)*(L+5*M)

can be broken down into:

```
TEMP1 = J+K
TEMP2 = 5*M
TEMP2 = TEMP2+L
I = TEMP1*TEMP2
ETC.
```

which corresponds to the desired format.

Now, performing the assignment statement is easy. We simply perform the operations one at a time and store the final result in I. The only problem to this scheme is the fact that the 6502 only supports addition and subtraction. There is no multiply or divide instruction. As a result we have to write a subroutine to perform the mulitplication and division subroutines. Unfortunately, the Auto-start ROM does not. So to make sure that your program is portable, I would suggest that you copy the multiply and divide routines out of the red book (or the "white book" if you have the new reference manual!) and incorporate them directly into your program.

The previous code is converted as follows:

```
CLC
LDA J
ADC K
STA TEMP1
LDA J+$1
ADC K+$1
STA TEMP+$1

LDA #!5
STA MULOP1
LDA /!5
STA MULOP1+$1
LDA M
STA MULOP2
LDA M+$1
STA MULOP2+$1
JSR MLTPLY  ; COMPUTES MULOP1 * MULOP2

CLC
LDA PRODCT  ; GET PRODUCT OF PREVIOUS MULTIPLY
ADC L
STA TEMP2
LDA PRODCT+$1
ADC L+$1
STA TEMP2+$1

CLC
LDA TEMP2
STA MULOP2
LDA TEMP2+$1
STA MULOP2+$1
LDA TEMP1
STA MULOP1
LDA TEMP1+$1
STA MULOP1+$1
JSR MLTPLY

LDA PRODCT
STA I
LDA PRODCT+$1
STA I+$1
```

VOILA!

OTHER STUFF

Most of the other functions in integer Basic (such as TAB, VTAB, TEXT, PR#, IN#, PLOT, HLIN, VLIN, COLOR=, ETC. are handled by monitor calls. To implement these functions I refer the reader to the Apple monitor.

WARNING! Throughout my discussion I have assumed that you are using *UNSIGNED INTEGER VARIABLES.* The rules for signed integer variables (for comparisons etc.) are different. Unfortunately due to space requirements I cannot describe hoe how to manipulate string variables, nor can I discuss how array elements are accessed. These two subjects warrant as much discussion as is present in this entire article. This information will be published in a future issue of Applesauce or the Apple Orchard.

MicroNET

Dear Apple II User:

As more and more Apple II users are finding out every day, you can expand the potential of your Apple II computer with MicroNET. The MicroNET Personal Computer Service can give your personal computer the additional capability of handling more complex computing tasks, and MicroNET can offer you a variety of computing services.

And now we are pleased to announce the additional offering of the MicroNET Executive designed specially for Apple II users. The MicroNET Executive is a special program which enables your personal computer to function as an intelligent terminal. As such, it opens up the entire range of MicroNET services and programs for your use, such as:

-- use of the MicroNET Software Exchange to purchase programs and have them down-line loaded into your Apple II computer
-- the use of special characters such as left and right brackets for file transfer operations
-- the ability to transfer files to and from the large MicroNET time sharing computers
-- the ability to receive output longer than the 40 character line length without broken words at the end of lines
-- automatic error detection and retransmission during file transfer
-- automatic dial-up with a D.C. Hayes Micromodem II, and more.

For Apple users who write practical software programs, we offer you the opportunity to sell your programs nationwide using the MicroNET Software Exchange. Serious authors should contact us to discuss the specifics and to arrange for testing your program.

Thousands of personal computer users across the country are using the MicroNET service because they know experience counts. Since MicroNET is a service of CompuServe, the well-known nationwide computer services company, we can offer the most reliable service available in the world.

Apple II users, don't delay. Write to MicroNET today.

Sincerely,

John E. Meier
MicroNET Marketing Manager

CompuServe

Personal Computing Division
5000 Arlington Centre Boulevard
Columbus, Ohio 43220
Telephone 614/457-8600

From WASHINGTON APPLE PI

THE LANGUAGE SYSTEM -- THE APPLE GROWS UP

by
Bill Wurzel

The new Apple Language System greatly expands the capabilities of the Apple II, giving the user a versatile text editor. more usable diskette space, and the ability to write programs in a powerful, high-level language. The next few pages will briefly describe the hardware and software that comprise the Language System.

WARNING!!

Despite advertising claims to the contrary, I was unable to use all the features of the Language System with a single disk drive. This may reflect my own lack of familiarity with the System rather than a deficiency in the System itself, but if you're interested in the single drive system, I'd advise you to visit your friendly computer store for a complete demonstration first!

WHAT'S IN THE BOX?

The Language System package contains:

1. The Language card: This is a PC board about the size of the Applesoft ROM card and plugs into slot zero on the motherboard. Installation requires removing one of the RAM chips and plugging in a 16 pin connector to the card (The card seems to require the dynamic memory refresh signal from the on-board RAM socket. There is a 16K x 1 RAM chip on the card which apparently does the work of the chip you removed.)
2. Two PROMs: The increased disk space (about 40K per disk – I think the system accomplishes this by writing shorter headers, not by changing hard density) requires substituting two new PROMs for the old two on the disk controller board.
3. An IC puller: Worked very well – changing the chips was no problem.
4. Installation and Operating Manual: Well written. Also documents the Autostart ROM feature (including the stop-list feature and improved cursor control that come with the Autostart ROM).
5. Apple Pascal Reference Manual: This is sort of the equivalent of the Big Red Book. Like the Big Red Book, it is not a tutorial. In its own words: ". . .this manual is most definitely not intended for beginners at using computers and Pascal . . ." Nonetheless, for *preliminary documentation,* the Reference Manual is well written and comprehensive.
6. Pascal User Manual and Report: This book is written by Jensen and Wirth, creators of Pascal and appears to be the definitive documentation of the Standard language. Its 165 pages present the features of Standard Pascal briefly and include many helpful examples. It also documents a sample implementation (on a CDC 6000 series machine) and presents the language's syntax in both Backus-Nauer and railroad track formats. The second part of the book is the *Report* which describes the language more succinctly.
7. Problem Solving Using Pascal: Written by Kenneth Bowles. one of the architects of the UCSD variant of Pascal, this book attempts to teach the major features of the language to a reader who already has some knowledge of programming fundamentals. It seems to be an adequate introduction to some of the more straightforward capabilities of Pascal, but the language's more powerful features are inadequately presented or not discussed at all. Although the language described is UCSD Pascal (the variant supported by the Apple System) the example programs using graphics and some using disk I/O require minor modifications to run properly. All in all,I don't think this book is one of the better "introductions" to Pascal.

WHAT'S IN THE SYSTEM?

The Language System operating system and component routines are completely different from the familiar Monitor/DOS. From the outer, or command level you can enter the file manager, compiler, assembler, linker or your own compiled program. Each of these system programs is described below.

HOW DOES IT WORK?

The language card essentially overlays the upper 12K of ROM with 16K of addressable, write-protectable RAM. It crams 16K into 12K by switching between two banks of RAM which share the DOOO-DFFF address space. Thus the space C000-CFFF continues to be used for I/O and internal circuit switches.

This new 16K of RAM can be loaded from disk and then by program control can be write-protected – making it a ROM! So essentially what you have is 16K of erasable programmable read-only memory – instant EPROM! Load this EPROM with the old Apple Monitor and you have your old machine – with Integer and Applesoft Basics instantly available (Both are "in ROM" so the Applesoft ROM card is no longer necessary). DOS, of course, runs under the Monitor too, but must be loaded into high RAM from disk – exactly like the old Monitor system.

In "Pascal mode," the 16K "EPROM" is loaded with a P-code interpreter. P-code (the P is for "pseudo-" I guess . . .) is the "object code" to which all Pascal programs are compiled. Some back issues of *BYTE* explain this very well. The component programs (compiler, editor, etc.) are written for the most part in Pascal and are loaded into high RAM and "executed" by the "ROM-resident" P-code interpreter as necessary.

So, with this 16K "EPROM" you can even write your own monitor or make changes to the present one. I'm sure other languages like FORTRAN, Lisp, etc. can be implemented in this address space. Keep your eye on the marketplace!

WHAT IS PASCAL?

It's not the purpose of this article to describe Pascal completely. Nevertheless, certain points are relevant to a decision to purchase the Language System or not.

Pascal is a high level language developed in the late 1960's by Kathleen Jensen and Niklaus Wirth (pronounced "veert") at the ETH* in Zurich, Switzerland. It was developed to be used in *teaching* programming, but its strengths (discussed below) gradually led to its use as a general-purpose programming language in its own right.

There are two flavors of Pascal referred to in Apple's documentation: Standard Pascal as defined by Jensen and Wirth and documented in reference 6 above, and UCSD Pascal, an extension of the Standard, described in references 5 and 7 above. The variants are very similar; differences lie primarily in the areas of file types, string-handling ability and program flow. In the main, UCSD extends Standard Pascal, although one Standard Feature, FUNCTION types in parameter lists, is not supported in the UCSD version.

The main strengths of Pascal are its data structuring capabilities and its "top down – structured programming" format. Data structuring capabilities include extensible data type definition, set structures (in the mathematical sense of "set"), multi-dimensional arrays of any data type, collections of non-like data types called "records" (whose components may themselves be records), and LINK type variables (making list processing possible). Procedures and Functions (similar to those in FORTRAN) are also implemented; the parameters may be call-by-name or call-by-value. Procedures and functions may be multi-nested and contain their own local variables. Apple Pascal supports Hi-res graphics only (and does not support the second graphics screen), but offers powerful graphics subroutines enabling, among other things, the mixing of Hi-res text characters (including lower case) and line graphics. Oh yes, I almost forgot: procedures and functions are completely recursive!

On the negative side of the ledger: Pascal does not appear to be a good scientific language. Features such as exponentiation (i.e. "to the power of" primitive), transcendental functions (like trig functions, logarithms etc.), and primitives for matrix and complex algebra are not included in the design of Pascal. The trancendental functions, however, are implemented in Apple Pascal, but they're implemented as subroutines written in Pascal and consequently run as slow as or slower than the corresponding Applesoft functions. Furthermore, passing functions in parameter lists is (at least in my experience) primarily used in scientific programming; this feature is not supported in UCSD Pascal.

On balance, however, Pascal is a powerful high-level language. For short, quick-and-dirty programs, BASIC is probably an easier faster language to use. But for long, involved programs or ones which require complicated data structures, there is just no fair comparison!

The following paragraphs discuss the software components of the Language System individually.

THE SYSTEM FILER

Pascal files are composed of 512-byte blocks of data. The file name consists of the disk volume (or drive) on which it resides (with appropriate default values), the file name identifier itself, and an optional extension. The extension usually identifies what kind of file it is (source code or readable text, object code, data for program manipulation, etc.) and is required for some system functions.

The Filer commands are as follows:

B(ad blocks): Tests all 280 blocks on the specified diskette to see that information has been recorded consistently. (see X(amine) below.)

C(hange): Renames a diskette or a diskette file.

D(ate): Sets a new current date for the system. Every time a file is saved, the current date is saved along with it and is displayed when you "list the catalog".

E(xtended list): Shows the contents of a diskette, displaying extra information about the files and unused portions.

G(et): Designates a specifed diskette file as the workfile.

K(runch): Packs the files on a diskette so that unused portions of the diskette are combined into one area. Apple DOS allocates disk space differently so it doesn't need this function. But the Language System's contiguous storage scheme keeps disk head activity to a minimum.

M(ake): Creates a diskette directory entry and "dummy" file.

N(ew): Clears the workfile.

P(refix): Changes the current default volume name.

R(emove): Removes the specified file from the directory.

S(ave): Saves the workfile under the specified name.

T(ransfer): Transfers information from one file to another file: can be used to move or save diskette files, copy entire diskettes, or send files to a printer or other device.

V(olumes): Shows the devices and diskettes currently in the system.

W(hat): Tells the name and state of the workfile.

X(amine): Attempts to fix diskette blocks reported bad by the B(ad blocks) command. Marks blocks which can't be fixed as "do-not-use."

Z(ero): Erases the directory of a diskette; clears the diskette directory without having to reformat the disk.

THE SYSTEM EDITOR

It is difficult to do justice to the text editor in a verbal description. You really have to see it to appreciate it. But briefly . . . the editor is designed to be used either for entering programs (where you'd like it to behave one way), and entering natural language text (where you'd like it to behave completely differently). The editor's personality is determined by the "Environment" block as follows:

Auto indent: If true, positions the cursor to the first non-blank character of the previous line after each carriage return. By convention, Pascal is multi-level indented to make it easier to follow program logic.

Filling: If true, the editor "looks ahead" to see if the word you have typed will fit on the current line. If not, it does an automatic return and starts the word on the next line.

Left Margin: Sets it.

Right Margin: Can range from 1 to 79 (possibly higher). The System supports an 80-character line on the standard TV screen. It does this by splitting the line in half and permits the user to view one half or the other. During text entry, there is an option for automatic horizontal scrolling to follow the cursor.

Paragraph margin: For the M(argin) command.

Command char: Specifies the character which protects lines from being marginated.

Token default: Used in F(ind) and R(eplace) commands. Determines whether any occurrence of the specified string is to be found or only those occurrences surrounded by blanks.

Markers: The System permits ten named "markers" to identify ten different locations in the file. This is useful for jumping to certain locations in a long file quickly or copying only portions of a file.

Cursor commands, inserting, deleting and changing text are all pretty standard from one text editor to another – nothing new or exciting here! But text, as it is inserted or deleted, is placed into a special *copy buffer* from which it may be rapidly copied into another location in the file. Thus the same text may be inserted many times into the file quickly. Also, using a sequence of delete-move cursor-copy, you can implement an easy-to-use and *manageable* move-text function. (Most move-text functions, if they exist at all, require you to specify from-line, to-line, number-of-lines, etc. This is awkward and prone to easy errors. Moving text with Apple's editor (at least for me), is much easier!)

One final feature which makes the editor a joy to use is the F(ind) and R(eplace) functions. These let you find the first, the nth, or all instances of a specified string, either bounded by blanks or not, and replace any or all such instances with another text string. As an option, the editor will ask you to verify each substitution before it is made, letting you determine where you do and don't want replacements.

There are several other features, too! See the editor in action for yourself!

THE SYSTEM COMPILER

The compiler is a program which translates a Pascal source program into machine-interpretable P-code. It has several options which allow you to:

1. Place a character string directly into the codestream.
2. Permit or prohibit use of the GOTO command. The GOTO controversy is a long and interesting one. The command generally tends to make code harder to follow. The branch of Computer Science concerned with *proving* programs correct generally forbids its use.
3. Generate code which automatically checks for I/O errors.
4. Include other source text from a disk file into the text being compiled.
5. Send a listing of the compilation to a printer or disk file.
6. Page the listed output (i.e. skip over perforations in paper).
7. Suppress compiler progress information from TV screen.
8. Generate code which will check subscript ranges. BASIC does this automatically, but you pay for it with increased running time. If you're sure you won't exceed array dimensions (thus clobbering other P-code), Pascal lets you save some run time and program storage space.
9. Accommodate large programs or large symbol tables by making segments of the compiler swap themselves in and out. The space this swapping saves is available to store program or symbol table.

These look like pretty nice options, and they are! But they must all be included in the source program as pseudo-comments. This means that to change one of the options (like whether you list the program on a printer) you have to re-edit the source text. Obviously, a better way to implement compile time options is to specify them at compile time! This is a small deficiency in the Language System which I hope will be corrected in later releases.

THE SYSTEM ASSEMBLER

The Language System also includes a macro-assembler with a few extra capabilities for use especially with Pascal. Some of the Assembler features include:

1. Free-format line. Labels and operands do not have to begin in specified columns.
2. Macro facility. The user can define macro statements which can invoke other macro statements to a depth of five. The macro facility is really rudimentary; variables in the macro definition statement are assigned position-dependent values from the macro invocation statement. These strings cannot be "substrung" or concatenated. Also there is no provision for macro-defined variables (at least none is documented) and, I think, this really emasculates (effeminates?) the full power of a macro assembler. A conditional assembly capability is supported but limited by the lack of macro-defined variables.
3. .ORG, .ASCII, .BYTE, .WORD, .BLOCK, .EQU, . ABSOLUTE all do about what you'd suppose they do.
4. The special labels .CONST, .PRIVATE and .PUBLIC give the assembler programmer access to any or all *global* constants and variables.
5. The .DEF and .REF pseudo-ops permit communication between independently assembled routines.
6. Local labels. Local labels are labels which are placed in a temporary stack, not entered into the symbol table. They are very useful for short branches. Any regular label purges the local label stack so these short, numeric labels may be re-used.

All in all, the assembler is adequate and easy to use. The macro facility could really be improved – but I guess most of us shouldn't need it. After all, if the application is so complex that it needs powerful macros, it should probably be written in Pascal!

THE SYSTEM LINKER

Describing the operation and options of the Linker gets pretty complicated pretty fast. In general, the linker links together individually compiled or assembled pieces of code into one intercommunicating module. If you use an assembly language subroutine in BASIC, for example, you have to plan where you want to put it, maybe fool BASIC into thinking it isn't there, and get at it with constant-valued PEEKs and POKEs. The Language System relieves you of this drudgery. It puts your assembly language program wherever it wants and then updates the appropriate addresses so that the Pascal program can call it. Also, the linker resolves the .CONST and .PUBLIC addresses mentioned earlier, as well as .DEFs and .REFs between assembly language routines.

CONCLUSION

There are several other features of the Apple Language System (such as the System Librarian) which you can mess with if you want, but can usually leave alone. They're not used too much and are difficult to describe!

The Apple Language System (and incidental goodies like the Autostart ROM and Applesoft-in-ROM) really represents a quantum jump in the computing power of the Apple II. Furthermore, the "virtual EPROM" in which it is implemented opens the door for hundreds of other applications, of which operating systems and languages are just a part.

Now if the Pascal and DOS disks were just *compatible* . . .

EDITOR'S NOTE: *Watch for this to come.*

EPILOG

Finally, you supposedly can do everything with a one-drive system that you can with a two-drive one. *BUT* it requires transferring files and switching disks with a frequency that I find unacceptable. Your source files must be few and small. So let me repeat: see the one drive system do everything you want to do before you buy!

Also of interest, the August 1978 issue of *BYTE* has some good articles on PASCAL.

THE 15 SECOND 15 CENT RESET FIX

by
Ken Silverman
From: The Cider Press

Does this look familiar to you?

10 PRINT "I AM GOING TO ACCIDENTALLY PRESS THE RESET KEY"

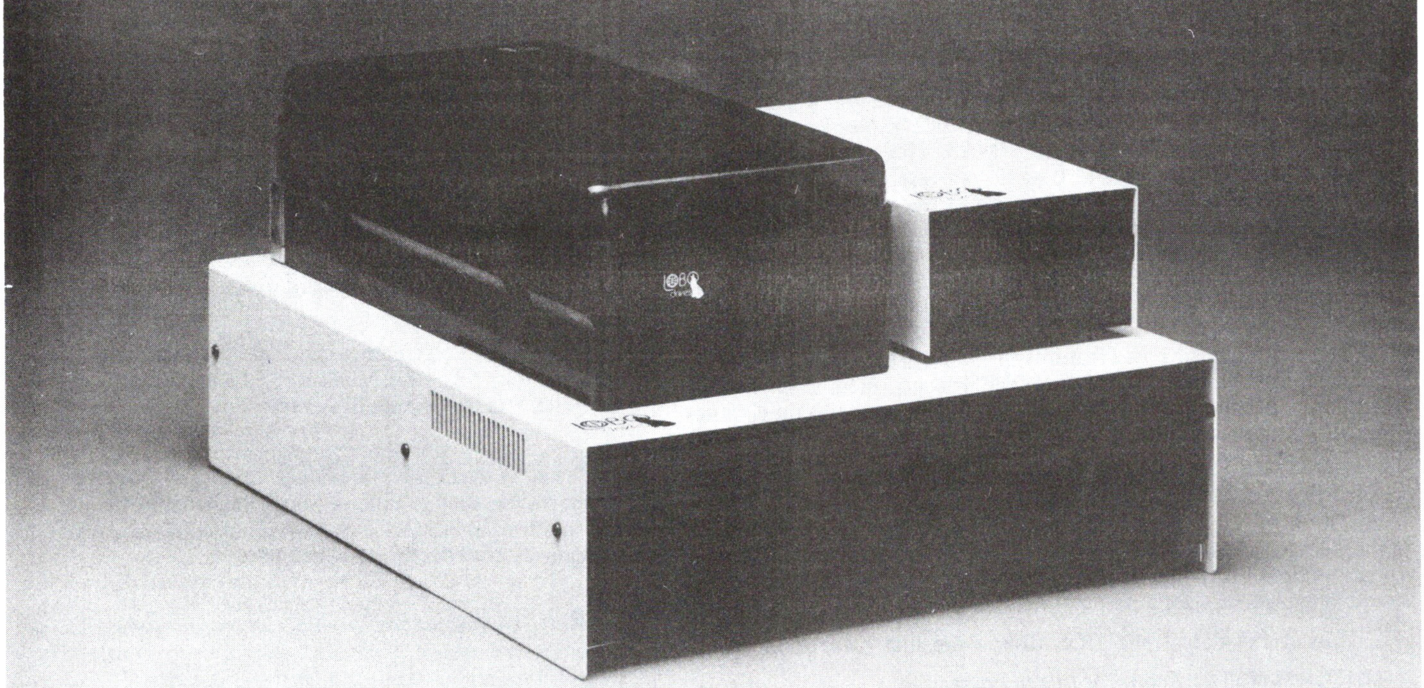
Here is the 15 second 15 cent fix.

1. Take a small screw driver and lift off the Key Cap of the RESET key (Picture 1).
2. Then place a 7/16 in. I.D. by 5/8 in. O.D. O Ring over the stem of the key. Sears model 42-22517 O Ring (Picture 3).
3. Place the Key Cap back on the key.

Now if you accidentally hit the RESET key it won't depress. Please note the RESET Key can still be used but it will take a few pounds of pressure.

NOTE: This modification might void your Apple II Warranty.

NEW FROM LOBO:

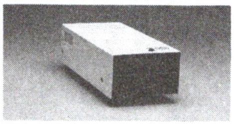


An Entire Family of Disk Drives for APPLE, TRS-80*, and S-100 Computers

Only LOBO DRIVES offers you an entire family of fully-compatible disk drives to select from. Whatever computer you're using, APPLE, TRS-80, or S-100, you can add a LOBO drive now, with the peace-of-mind of knowing there's a whole family of drives available when you're ready to expand.

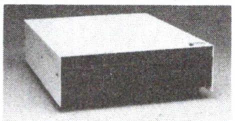
And every drive you order comes complete with chassis and high reliability power supply. Each drive is 100% calibrated, burned-in, and performance tested on either an APPLE, TRS-80, or S-100 computer before it's shipped. We are so proud of our drives . . . our quality, reliability, and performance, that we back-up every drive with a one year, 100% parts/labor warranty.

400 SERIES FLOPPY DISK DRIVES



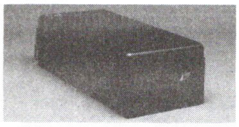
Meet our low-cost 5.25-inch mini drive that records data in either hard or soft sectored format. It is available in single or double density configurations, with a total storage capacity of 220K bytes.

800/801 SERIES FLOPPY DISK DRIVES



Here is our dual 8-inch Floppy disk memory unit. It records and retrieves data on standard 8-inch diskettes to provide 800K bytes of data storage unformatted, or 512K bytes in IBM format per drive. It is also available with double-sided, double-density capabilities, for a maximum storage capacity of 1.6 Megabytes.

7000 SERIES HARD DISK DRIVES



The latest member of our drive family, the Series 7000 is an 8-inch, 10 Megabyte Winchester Technology, hard disk drive. It is fully hardware/software compatible and comes complete with disk controller. Now you can have the convenience, speed, reliability, and all the storage capacity you need.

Call or write for the complete LOBO DRIVES story. Find out just how competitively priced a quality drive can be.

Quantity discounts available –
Dealer inquiries invited.

Yes, I want to know more about LOBO Drives and what they can do. Send me information on:
☐ TRS-80 ☐ APPLE ☐ S-100

☐ 5 1/4-in. Floppy drive
☐ 8-in. Winchester hard disk, 10 Mbyte drive
☐ 8-in. Floppy drive Single sided Double sided
☐ Double density expansion interface

Name ____________
Company ____________
Address ____________
City ________ State ________ Zip ________
Phone No. ____________
If dealer, provide resale no. ____________

LOBO drives INTERNATIONAL

935 Camino Del Sur
Goleta, California 93017
(805) 685-4546

"CAN YOU REALLY AFFORD TO PAY LESS?"

*TRS-80 is a registered trademark of Radio Shack, a Tandy Company.

From

the michigan APPLE-gram

HEX – ASCII MEMORY DUMP

* * *

by
Curt Deegan

There are some quite useful features in the APPLE monitor that support the efforts of both the BASIC and assembly or machine language programmer. One, the memory dump, will display the contents of any range of memory locations. With the program shown below, this valuable utility is extended somewhat to display not only the HEX contents of memory locations, but also the ASCII equivalent of those HEX values. This added information can prove to be a real aid when dissecting the internal representation of BASIC programs as well as for inspecting data and machine language in memory.

Operation of this routine is in two steps. The first step establishes the 'CTL'Y branching vector at HEX memory locations $3F8-$3FA. This step is accomplished by entering the following machine language execute command:

*300G

The second step actually requests the display of memory. The procedure for htis second step is identical to that used for a normal memory display to the point where the carriage return would be entered. Instead, first enter the 'CTL'Y character and then the carriage return. This would look something like this:

*2000.202F'CTL'Y'RETURN'

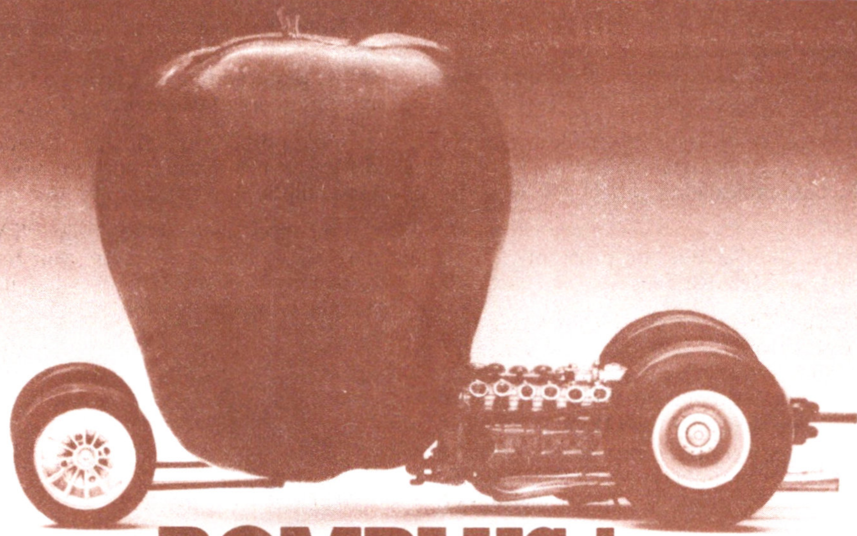
Where 'CTL'Y means to hold down the key marked 'CTRL' while pushing the Y key, and 'RETURN' means to push the key that is marked 'RETURN'. Having done this the portion of memory will be displayed as usual with the additional ASCII representation of the HEX data appearing on the left of each display line.

A few considerations. While this routine will only be used when in monitor mode (i.e. the * prompt), the language from which monitor mode was entered can affect its operation. If 'RESET' is used to enter the monitor mode then no further considerations are necessary for proper operation. However, with Applesoft ROM active, the routines in the APPLE monitor are not accessible to this routine even after a]CALL –151 has been executed. First the ROM card must be turned off. Of course, upon return from monitor mode it would be desirable to restore the last active language system. This means turning back on the ROM card when appropriate, and, regardless of which Applesoft is being used (ROM or RAM), the low page one addresses used by both the Sweet 16 interpreter, called from this routine, and by Applesoft as the return vector for an eventual user restart, must be restored to their ocntents before this routine was executed. A look through the program listing will show these points have been taken into account. APPLE Integer BASIC requires no such special handling and is not sensitive to these unique Applesoft provisions. Internals of this routine are described in the comments of the assembly language listing. – Curt

```
;********************
;
;  HEX-ASCII MEMORY
;    DUMP ROUTINE
;
;        -=*=-
;
; COPYRIGHT (C) 1979
; BY: W. CURT DEEGAN
;
;   ALL COMMERCIAL
;  RIGHTS  RESERVED
;********************
;
;EQUATES
;
ORIGIN EQU $0300
OBJECT EQU $0800
;
;  MONITOR ROUTINES
COUT    EQU $FDED
XAM     EQU $FDB3
NXTA1   EQU $FCBA
PRBL3   EQU $F94C
SW16    EQU $F689
;   I/O CNTRL LOCS
FP      EQU $C080
INT     EQU $C081
;   'CTL'Y VECTOR LOC
USRADR EQU $03F8
;   FP ADDRS TO SAVE
;   FROM SWT16 AREA
ZERO    EPZ $00
;   HRZNTL CURSOR POS
CH      EPZ $24
;   ADDR RANGE TO DUMP
A1L     EPZ $3C
A1H     EPZ $3D
A2L     EPZ $3E
A2H     EPZ $3F
;   TEMP STORE LOCS
T1L     EPZ $60
T1H     EPZ $61
T2L     EPZ $62
T2H     EPZ $63
;
;SET ORIGIN ADDRESS
;
        ORG ORIGIN
        OBJ OBJECT
;
;*********************
;
;SET UP 'CTRL'Y
;
START   LDA 4C
        STA USRADR
        LDA #DECODE
        STA USRADR+$1
        LDA /DECODE
        STA USRADR+$2
        RTS
;
;*********************
;
;DATA SAVE AREA
;
;   FP ROM STATUS SAVE
ROMSWT HEX 00
;   FP VECTOR SAVE AREA
ZSAVE   HEX 000000000000
;
;*********************
;
```

```
75 ; MAIN ROUTINE
76 ;
77 ;  CHK IF FP ROM IS ON
78 DECODE LDA SW16
79        STA ROMSWT
80 ;  SAVE FP VECTORS FROM
81 ;  SWT16 EXECUTION
82        LDX 05
83 JMPSAV LDA ZERO,X
84        STA ZSAVE,X
85        DEX
86        BPL JMPSAV
87 ;  RESET FP ROM SWITCH
88        LDA INT
89 ;  SAVE DUMP ADDR RANGE
90        LDA A2L
91        STA T2L
92        LDA A2H
93        STA T2H
94 ;--------------------
95 ;DUMP HEX ROUTINE
96 ;
97 ;  THIS ROUTINE WILL
98 ;  COMPUTE THE RANGE
99 ;  OF ADDRS NEEDED TO
100 ;  PRINT 1 LINE OF HEX
101 ;  DUMP.  THE START IS
102 ;  EITHER THE INITIAL
103 ;  ADDRESS OR THE LAST
104 ;  ADDR IF THIS IS NOT
105 ;  THE FIRST LINE OF
106 ;  THE DUMP OUTPUT
107 ;
108 LNLOOP LDA A1H
109        STA T1H
110        STA A2H
111        LDA A1L
112        STA T1L
113        ORA 07
114        STA A2L
115        JSR SW16
116        SET R2,A2L
117        SET R3,T2L
118        LDD @R2
119       !STO R4
120        LDD @R3
121        CPR R4
122        BIC MORE
123        SET R2,A2L
124        SET R3,T2L
125        LDD @R3
126        STD @R2
127 MORE   RTN
128 ;  JMP TO MONITOR HEX
129 ;  DUMP ROUTINE FOR 1
130 ;  LINE OF OUTPUT
131 PART   JSR XAM
132 ;----------------------
133 ;DUMP ASCII ROUTINE
134 ;
135 ;  THIS PART DUMPS THE
136 ;  ASCII CODE THAT GOES
137 ;  WITH THE HEX CODE
138 ;  PRINTED ABOVE
139 ;
140 ;  FIRST SET ADDR RANGE
141        LDA T1L
142        STA A1L
143        LDA T1H
144        STA A1H
145 ;  PUT COLON AT SCRN
146 ;  POS 29
147        LDA 1D
148        STA CH
149        LDA 0BA
150        LDX 02
151        JSR PRBL3
152 ;  LOOP PRINTING ASCII
153 ;  EQUIV OF HEX UNTIL
154 ;  END OF 1 LINE
155        LDY 00
156 CHLOOP LDA (A1L),Y
157        BPL PRINT
158 ;  CHECK FOR CTL CHAR
159 ;  AND SUBSTITUTE A '
160 ;  IF HEX IS BETWEEN
161 ;  $80 AND $9F
162        AND 60
163        BNE REGET
164        LDA 0AE
165        BNE PRINT
166 REGET  LDA (A1L),Y
167 ;  PRINT ASCII EQUIV OF
168 ;  HEX VALUE OR '.' IF
169 ;  A CTL CHAR
170 PRINT  JSR COUT
171 ;  STEP TO NEXT LOC
172        JSR NXTA1
173 ;  END OF LINE?
174        BCC CHLOOP
175 ;-------------------
176 ;TEST FOR END ROUTINE
177 ;
178 ;  HERE THE INITIAL
179 ;  ADDR RANGE IS CHKED
180 ;  AGAINST THE CURRENT
181 ;  POINTERS TO SEE IF
182 ;  WE ARE FINISHED
183 ;
184        JSR SW16
185        SET R1,A1L
186        SET R2,A2L
187        SET R3,T2L
188        LDD @R1
189        STO R4
190        LDD @R3
191        STD @R2
192        CPR R4
193        BNC NOMORE
194        RTN
195        JMP LNLOOP
196 NOMORE RTN
197 ;---------------------
198 ;RESET POINTERS ROUTINE
199 ;
200 ;  WE ARE THRU. NOW TO
201 ;  REPLACE THE FP ADDR
202 ;  VECTORS AND RESET
203 ;  THE FP ROM IF IT
204 ;  WAS ON WHEN THIS
205 ;  DUMP S STARTED
206 ;
207        LDA ROMSWT
208        CMP 20
209        BEQ QUIT
210        LDA FP
211 QUIT   LDX 05
212 JMPRST LDA ZSAVE,X
213        STA ZERO,X
214        DEX
215        BPL JMPRST
216        RTS
217 ;
218 ;*********************
219 ;PERMISSION IS GRANTED
220 ;TO REPRODUCE FOR THE
221 ;NON-PROFIT USE AND
222 ;DISTRIBUTION OF CLUBS
223 ;HOBBYISTS AND EDUCA-
224 ;TIONAL INSTITUTIONS,
225 ;SO LONG AS CREDIT IS
226 ;GIVEN THE MICHIGAN
227 ;APPLE-GRAM AND THE
228 ;AUTHOR: W.CURT DEEGAN
229 ;*********************
230 ;
231 ;
*300.3B2

0300- A9 4C 8D F8 03 A9 17 8D
0308- F9 03 A9 03 8D FA 03 60
0310- 00 00 00 00 00 00 00 AD
0318- 89 F6 8D 10 03 A2 05 B5
0320- 00 9D 11 03 CA 10 F8 AD
0328- 81 C0 A5 3E 85 62 A5 3F
0330- 85 63 A5 3D 85 61 85 3F
0338- A5 3C 85 60 09 07 85 3E
0340- 20 89 F6 12 3E 00 13 62
0348- 00 62 34 63 D4 03 08 12
0350- 3E 00 13 62 00 63 72 00
0358- 20 B3 FD A5 60 85 3C A5
0360- 61 85 3D A9 1D 85 24 A9
0368- BA A2 02 20 4C F9 A0 00
0370- B1 3C 10 0A 29 60 D0 04
0378- A9 AE D0 02 B1 3C 20 ED
0380- FD 20 BA FC 90 EA 20 89
0388- F6 11 3C 00 12 3E 00 13
0390- 62 00 61 34 63 72 B4 02
0398- 04 00 4C 32 03 00 AD 10
03A0- 03 C9 20 F0 03 AD 90 C0
03A8- A2 05 BD 11 03 95 00 CA
03B0- 10 F8 60
ARDORESEWE ARE THRU.  ;  REPL
 203 ;  TE04 ;  WAS ON WHEN T
AS STARTED
 206 ;
2SWT
 0O8T        CMP
5210         LDA
 212 JMPRST LDA ZSAVX ZERO,X
 214        DBPL JMPRSTC
 216
 218 ;*****************ERMISSION
 221 ;NON-02 ;DISTRIBUTION OF
 224 ;TIONAL INI LONG AS CRED
2IGAN
 227 ;APPLE-GRMUTHOR: W.CURT
*****************
 230 ;
```

NEW POWER FOR YOUR APPLE FROM MOUNTAIN HARDWARE.



ROMPLUS+

NEW EXPANDABILITY.
ROMPLUS+ is a peripheral board whose added features can turn the Apple * computer into the most powerful personal computer available today.

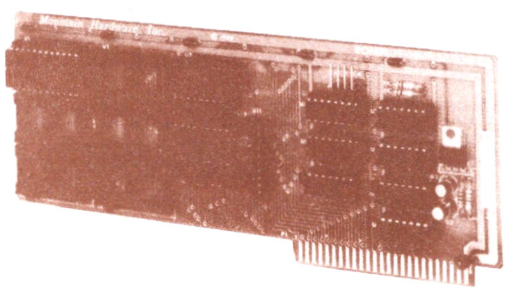
NEW POWER.
ROMPLUS+ provides six sockets to accept individually addressable 2K ROM's or EPROM's. Keyboard Filter™, a 2K ROM program, comes installed on the ROMPLUS+ board and adds many useful features to your Apple, including:

- Upper and lower case letters. The only system that offers keyboard input and standard shift key operation.
- Multiple user-defined character sets.
- Colored or inverse-colored letters.
- Keyboard macros—two key-stroke, automatic typing of multiple, user-defined words or phrases. Including BASIC and DOS commands.
- Mixed text and graphics.
- Improved cursor control.
- STOP LIST and END LIST.
- Works with Integer BASIC, RAM or ROM Applesoft, and DOS.
- And more...quick to learn. Easy to use.
- Software support provided on disk includes demonstration programs and two Editors that allow you to define your own characters or keyboard macros.

SOPHISTICATED FIRMWARE.
In addition to the Keyboard Filter ROM, ROMPLUS+ offers five sockets for ROM or EPROM, plus "scratch-pad" RAM. And, sophisticated firmware on ROMPLUS+ allows one, two, or more of its chips to be used simultaneously for programs longer than 2K.

EXPANDED UTILITY.
Many software programs really ought to be utilized as firmware. ROMPLUS+ makes that an actuality for the Apple by providing six additional ROM sockets.

AT YOUR DEALER. NOW.
$169. Complete and tested. Including the powerful Keyboard Filter ROM and full documentation. Ask your dealer for a demonstration.



Mountain Hardware, Inc.
LEADERSHIP IN COMPUTER PERIPHERALS
300 Harvey West Blvd., Santa Cruz, CA 95060
(408) 429-8600

Sounds great.
☐ Please send me all the details on ROMPLUS+ and Keyboard Filter.

Name ______________________________

Address ______________________________

City ________________ State __________ Zip ______

Phone ______________________________

*Trademark of Apple Computer Inc.

From the michigan APPLE-gram

DOS TIDBITS

by

Jerry Rivers

In this article we will be discussing the different filetypes that the APPLE DOS can store, some rudimentary definitions for the various disk file structures being used in the computing industry, plus an illustrative example or two.

The DOS is equipped to handle four main file types: INTEGER and APPLESOFT BASIC programs, BINARY (machine language graphics pages, data tables, etc.) and TEXT (data files of all types).

Since we have already talked about the program– style file, we will begin this article with BINARY files.

As you know, computers are only capable of working with binary information – on and off, or 0 and 1. However, a binary representation of most numbers is very unwieldy at best. For example, the number 256 decimal is represented (in 16 'bit' format) as 0000000100000000,1234 decimal is 0000010011010010. Clearly, a better way must be found. On our APPLEs binary information is normally given in a numbering system called HEXADECIMAL, or base 16. In this system the number digits are 0, 1, 2,3,4,5,6,7,8,9,A,B,C,D,E,F.

This still leads to confusion for people, but is easy to work with in computer programming. For example, the two numbers listed above are shown as $0100 (256) and $04D2 (1234). While this can hardly be construed as a complete class in the hexadecimal numbering system, it can serve as background to our discussion on the storage of BINARY data files using the APPLE DOS.

The APPLE computer presents numbers in increments of 8 Binary DIGITS or BITS, called BYTES. Each byte is placed into its own space called an address. Since addresses in the APPLE are two bytes (16 bits), the largest integer number that can be represented is 65535 (64K), What you see if you go into the MONITOR and list out some data, is what is known as a MEMORY (or CORE) DUMP. Some data is classified as machine language programs some as misc. data, some may even be a part of a graphics display or text.

The point in this, you may save on disk a piece of memory (sometimes called a core image or 'snapshot') very easily.

The DOS command to save binary data to disk is BSAVE. Unfortunately, it's someshat complicated to do so. As stated in the DOS Manual, the syntax of the BSAVE command is: BSAVE f, Aa, LJ where f is the file name you want the data saved in, 'Aa' is the starting address of the data, and 'Lj' refers to the number of bytes of data you wish to BSAVE.

Addresses and byte lengths can be given either in decimal or in hexadecimal (is a $ is added to the address). Therefore if you wanted to BSAVE the HIRES page 2 area of memory which runs from $4000 to $5FFF hex (16384 to 24575 decimal) to a file called HGR PAGE2, either of the 2 statements below would be valid:

BSAVE HGR PAGE2,A$4000,L$2000 –or–
BSAVE HGR PAGE2'A16384,L8192

Note that the length is calculated by subtracting the starting address (16384) from the ending address (24575) plus 1. The + 1 is required to avoid saving one byte too few (how many numbers are there between 1 and 9 ??).

(Naturally, you will have to add on the disk drive slot and drive number if required, plus you may want to add a disk volume number is not set properly).

Once you have successfully BSAVEd data to a diskette, that part of the task is complete. But how do you get the binary data back into memory ?? and, what if a new location to store the data in memory is desired ??

The command to retrieve binary data is BLOAD. It too has a special set of optional parameters to worry about, which we'll cover by example.

Suppose it is desired to restore the HIRES graphics page we BSAVEd above. We can bring it back into memory with:

BLOAD HGR PAGE2

That statement is all that is required to bring back the data in HGR PAGE2 to the same memory location it was before. Now, let's stipulate that the graphics data must be restored to 'page 1', not 'page 2'. In this case we would use:

BLOAD HGR PAGE2,A$2000 –or–
BLOAD HGR PAGE 2,A8192

These two statements perform an identical function, They differ only in how the address information is presented.

It is important to note, however, that some types of data CANNOT be relocated in this manner. Machine language programs, for example, ordinarily CANNOT be relocated this way or they won't run!!

APPLE DOS TEXT FILES

Everything stored on a diskette which is not a BOSIC program and not a BINARY data file, is considered to be TEXT. A data file of all numbers for use by a program is considered TEXT. A mailing list is considered text. Every thing but SAVE or BSAVE files are TEXT files.

In the most general sense, a FILE is an orderly collection of data referred to as one unit, normally under one name, the FILE NAME (in APPLE DOS, this name can be from 1 to 30 characters).

Inside a FILE are one or more sub-divisions of data known as RECORDS. Ordinarily one RECORD is synonomous with a line of text string or numeric data. Further sub-divisions are possible. For instance, a FIELD is a part of a RECORD and a SUB-FIELD is a sub-set of a FIELD.

Let's set up a realistic example: Suppose we wanted to define a name address, and phone number file. The file might be defined like below –

```
FILE NAME   :   MAIL LIST
RECORD      :   ONE PERSON'S DATA

FIELD 1     :   LAST NAME
FIELD 2     :   FIRST NAME, MIDDLE INITIAL
    SUBFIELD 1 : FIRST NAME
    SUBFIELD 2 : MIDDLE INITIAL
FIELD 3     :   ADDRESS
    SUBFIELD 1 : STREET NUMBER
    SUBFIELD 2 : STREET NAME
    SUBFIELD 3 : APARTMENT NUMBER
FIELD 4     :   CITY
FIELD 5     :   STATE
FIELD 6     :   ZIP CODE
FIELD 7     :   PHONE NUMBER
    SUBFIELD 1 : AREA CODE
    SUBFIELD 2 : PHONE NUMBER
```

Even though a computer demands a highly structured way of defining and storing data, it can be done in a way conducive to good understanding by the PEOPLE the program is supposed to benefit.

Once you have decided WHAT you want to tabulate and record, you must then find a way HOW to store the data. This requires you to know something about FILE STRUCTURE. I know you have all heard of buzzwords for file structure, like sequential and random. But what do these words mean ???

In a SEQUENTIAL file, all information is physically stored in the file IN THE ORDER IT IS WRITTEN TO THE FILE. An example of this is an ordinary music tape recorder. When you play back the songs, you must listen to them in the order in which you first recorded them. If that isn't what you want to hear, your only choice is to 'fast-forward' over songs you want to by-pass. But, YOU MUST PASS OVER EVERY SONG ONE WAY OR ANOTHER.

Contrasted to a tape recording is the LP phonograph record. Here you CAN listen to each song in sequence, OR skip a song or group of songs by lifting the stylus and putting it down at the location of the song you want to hear next. What you did was select a song RANDOMLY!! A cassette tape with your programs or data on it is another type of sequential access file structure. A diskette is an example of RANDOM ACCESS.

Within the major grouping of random file structures, other sub-groups have been defined such as indexed sequential work addressable, keyed sequential, actual key, direct access and many others.

On the APPLE, we'll have to stick with straight SEQUENTIAL, and RANDOM access by RECORD number. First we'll cover the sequential file method.

In a sequential file, you put data on a disk file with a regular PRINT statement without worrying about how long a line or RECORD of data is. That is, it is a random line length file.

To take advantage of RANDOM file access you must use a FIXED LENGTH RECORD. The upshot of this is that YOU must make an effort in your program to keep any line PRINTed to the disk the same length. A good way to do this is to 'pad' unused positions in the record with blanks.

(More on this in the RANDOM ACCESS example program presented later on). A word or two is now in order on the 'format' of TEXT FILE data on the diskette. As you would surmise, since all data stored within the APPLE is encoded using ASCII numbers for each character, this would be a logical way to put data onto a diskette. In fact this is exactly how it is done (however, even though REAL and INTEGER numbers are stored in internal binary format, they too go out on the diskette as ASCII CHARACTERS).

A little known 'feature' of APPLE DOS is that all characters are 'packed' together as they are written to a disk file. Ordinarily this is good, since it wastes no space on the diskette. But, a problem can arise if you don't take the packing into account.

To illustrate this, let us define three variables X, Y, and Z: X=1:Y=2:Z=3.

If you PRINTed them with PRINT X, Y, Z you would expect the output to be:

```
1                2                3
```

But if you were sending these numbersto a disk file, the record would be:

```
123                    (How'd that happen ???)
```

What happened was this: the numbers were 'packed' together into one string !!!

Can this 'feature' be worked around ??? YES. There are two main ways to solve a dilemma of this kind. First, we can always print commas (,) between each number or we can print each on a separate line of data. Let's look at both ways:

PRINT X; ","; Y; ","; Z gives: X,Y,Y

PRINT X: PRINT Y: Print Z puts each of the variables on its own line.

Anybody care to guess how the APPLE can figure out which way we did it?? As is normal for INPUT statements, a comma is considered a 'separator', so the first method works OK. Lines put on a disk by themselves work OK so long as you use a separate INPUT statement for each one.

(What is actually going on is this: at the end of each line of data written to a SEQUENTIAL disk file, a carriage return character (ASCII 13) is appended to the end of each line. In this way, you can use lines of variable length).

Now that we've specified the structure of our file, all that's left is to OPEN the file to WRITE to it and we're done.

Wait a minute!!! What's this OPEN and WRITE business all about???

On large mainframe computers, programs are normally available to automatically take care of all the disk files whether you are sending data to them or taking data from them. In order for a file to be available to you it must be OPENed. just like you must open a door to enter your house. The big computers take care of this for you but with APPLE DOS, we are on our own.

Once a file is OPEN, we must then tell the DOS if we are putting data onto our disk (WRITE) or getting data back that is already there (READ). Lastly, when we have finished with the file, we always CLOSE it to prevent loss of data.

Let's examine the syntax for each:

```
OPEN  F    (,Lj)  (,Ss)  (,Dd)  (,Vv)
READ  F    (,Rr)  (,Bb)
WRITE F    (,Rr)  (,Bb)
CLOSE (F)  (,Ss)  (,Dd)  (,Vv)
```

(I will cover two more commands, APPEND and POSITION in a later DOS article)

In each of the statements above:

F is the FILE NAME
L is the record character length
R is the record number
B is the byte number
S is the disk controller slot #
D is the disk drive no. 1 or 2
V is the diskette volume number

Parentheses ( ) indicate optional parameters you may use if needed.

Looks awfully complicated. Not really.

First off, the parameters L, R, and B are only used in RANDOM access files. S and D are only required if you have two or more disk drives and controllers. The volume number is no problem: just put a number in for the volume number you are using or simply use V0.

Below is a very small program to WRITE three variables to disk then READ them back again (APPLESOFT PROGRAM).

```
10      D$=CHR$(4) : REM CTRL D
25  INPUT "WHAT FILE NAME ?";F$
30  PRINT D$; "OPEN ";F$: REM OPENFILE
35  PRINT D$;"WRITE ";F$. REM WRITE SET
40  S=1:Y=-2:Z=3.3:REM DEFINE VARIABLES
45  REM
50  REM FILE OPEN, PRINT DATA
55  REM
60  PRINT X:PRINT Y:PRINT Z
65  PRINT D$; "CLOSE ";F$:REM CLOSE FILE
70  REM
75  X=0:Y=0.Z=0. REM CLEAR VARIABLES
80  REM
85  REM RETREIVE DATA FROM DISK
90  REM
95  PRINT D$;"OPEN ";F$:REM RE-OPEN FILE
100 PRINT D$; "READ ";F$:REM READ SET
105 REM
110 REM NOW INPUT DATA INTO MEMORY
115 REM
120  INPUT X:INPUT Y:INPUT Z
125 PRINT D$; "CLOSE ";F$:REM CLOSE FILE
130 PRINT  "VARIABLES  READ  FROM  DISK"
135 PRINT X,Y,Z:REM PRINT RESULTS
140 END
```

You may remember from last month's discussion on the DOS that a special character (CTRL D) is required to tell the DOS that a disk command is coming. This program uses a string variable D$ as a convenient way to get the required CTRL D into the PRINT statements. Notice too that no NOMON command was issued. This was left out intentionally so that all data going to and from the disk would be visible to the user.

Let us now define our RANDOM ACCESS example problem. We'll write to disk the names and titles of the officers of our club. The RECORD structure is explained in the program. One RECORD is 32 characters long, so we must OPEN the RANDOM file with a length of L33 (to allow for the carriage return character (ASCII 13) that is put at the end of each line.

```
50  REM RANDOM FILE EXAMPLE
60  HOME:PRINT :PRINT
80  PRINT "DO YOU WANT TO SET NOMON?";
90  INPUT "Y/N ";A$:HOME
95  D$=CHR$(4):REM 'CTRL D'
100 PRINT D$; "MON IcD.)"
105 IF A$="Y" THEN PRINT D$;"NOMON I,C,D"
110 HOME: PRINT "DEFINING DATA . . . :PRINT
120 DATA DONETH, JOE, PRESIDENT
130 DATA VELASCO, AL, V.P. (ADMIN)
140 DATA RIVERS, JERRY, V.P. (TECH INFO)
150 DATA AYALA, JIM, SECRETARY
160 DATA SOPALA, HOHN, TREASURER
200 REM
210 REM RECORD STRUCTURE IS –
220 REM
230 REM FIRST NAME – 6 CHARACTERS
240 REM LAST NAME – 10 CHARACTERS
250 REM TITLE – 16 CHARACTERS
260 REM TOTAL – 32 CHARACTERS
300 DIM S$(5,3): REM 5 RECORDS, 3FIELDS
```

```
310 D$=CHR$(4):REM CTRL D
320 REM
330 REM 'READ' IN NAMES & TITLES
340 REM
350 FOR I=1 TO 5:REM # OF RECORDS
360 FOR J=1 TO 3:REM # OF FIELDS
370 READ S$(I,J): REM READ NAMES
375 PRINT S$(I,J);" ";
380 NEXT J:PRINT :NEXT I
390 REM
400 REM PAD DATA FIELDS
410 REM
420 FOR I=1 TO 5
440 L=LEN(S$(I,1): REM LAST NAME LENGTH
460 FOR K=L+1 TO 10:S$(I,1)=S$(I,1)+" "
480 NEXT K:REM PAD LAST NAME
500 L=LEN(S$(I,2)):REM 1ST NAME LENGTH
520 FOR   K=L+1   TO   6:S$(I,2)=S$(I,2)+   "   "
540 NEXT K:REM PAD FIRST NAME
560 L=LEN(S$(I,3)):REM TITLE LENGTH
580 FOR K=L+1 TO 16:S$(I,3)=S$(I,3)+ " "
600 NEXT K:REM PAD TITLE
620 NEXT I:REM DO 5 RECORDS
640 REM
660 PRINT :PRINT "CREATE RANDOM FILE..."
680 REM
700 PRINT D$;"OPEN RNDFIL,L33"
720      REM
740 FOR R=1 TO 5
760 PRINT D$; "WRITE RNDFIL,R";R
780 PRINT S$(R,1)+S$(R,2)+S$(R,3)
800 NEXT R:REM WRITE 5 RECORDS
820 PRINT D$; "CLOSE":REM CLOSE FILE
840 REM
860 PRINT :PRINT "READ DATA BACK IN";
865 PRINT " REVERSE ORDER . . .":PRINT
870 PRINT " # # # MICHIGAN APPLE OFFICERS#"
880 PRINT :REM OPEN FILE FOR READ
900 PRINT D$;"OPEN REDFIL,L33"
920 FOR I=5 TO 1 STEP -1
940 PRINT D$; "READ RENFIL,R";I
945 INPUT A$:REM GET RECORD FROM DISK
950 REM
955 REM RE-ARRANGE DATA FIELDS
960 REM
980 PRINT MID$(A,11,6);:REM 1ST NAME
985 PRINT LEFT$(A$,10);:REM LAST NAME
990 PRINT MID$(A,17,16):REM TITLE
995 NEXT I:REM READ 5 RECORDS
1000PRINT D$;"CLOSE":REM CLOSE FILE
1020PRINT D$;"MON I,C,0"
```

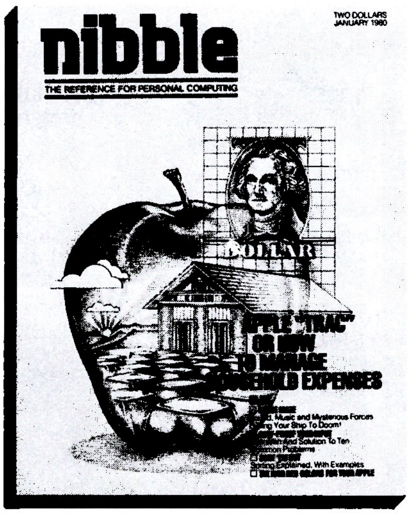
As can easily be seen by looking at the code in lines 180-220, the data set is created with last name first, and that is how it is stored on the disk. Lines 440-520 take care of 'padding' each of the fields to the same number of characters to preserve the required fixed length record size.

Note that a RANDOM file generally must be created the FIRST time by writing the data sequentially using the RECORD NUMBER as the 'KEY'. Once created, data may be READ in any order desired, as witnessed by lines 700-790 which READ the data in reverse order.

By now, you've probably figured out the major drawback to RANDOM files in APPLE DOS: YOU must at all times be aware of the RECORD number of all your data if it is to be retrieved non-sequentially.

(How the record numbers are kept track of is the subject of later DOS articles on a technique known as INDEXING).

INTRODUCING . . . NIBBLE
THE REFERENCE
FOR APPLE COMPUTING



NIBBLE IS:
A SOFTWARE GUIDE for high quality Applications Programs for your Home and Business.

NIBBLE IS:
A REFERENCE GUIDE to new Programming Methods.

NIBBLE IS:
A BUYERS GUIDE for making purchase decisions on new products.

NIBBLE IS:
A CONSTRUCTION PROJECT COOKBOOK for adding function and value to the system you already own.

NIBBLE IS:
A COMMUNICATIONS CLEARING HOUSE for users, vendors, and associations.

Each issue of NIBBLE features at least one significant new application program of commercial quality. The programs in NIBBLE are surrounded with articles which show how to USE the programming methods in your OWN programs.

Examples of upcoming articles:
☐ Modeling and Forecasting Your Business ☐ Build a Two-Tape Controller for $12 ☐ Arcade Shooting Gallery — Save Your Quarters! ☐ Data Base Management System I, II, III

And many many more! NIBBLE will literally "Nibble Away" at the mysteries of your system to help you USE IT MORE. In 1980, the principal featured system is the Apple II.

Try a NIBBLE

nibble
BOX 325 Lincoln, Mass. 01773

I'll try NIBBLE!
Enclosed is my $15 for 8 issues.
☐ **check** ☐ **money order**

Name ____________________

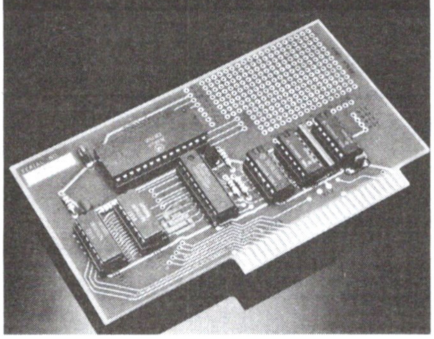
Address ____________________

City ____________________

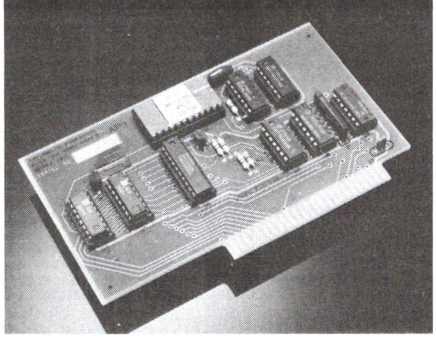
State ______________ Zip ________

TOOLS TO EXPAND THE APPLE.

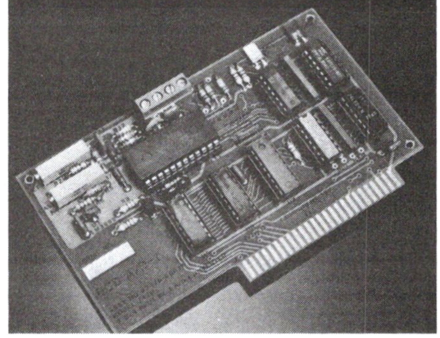
Programmable Interrupt Timer Module #7440A



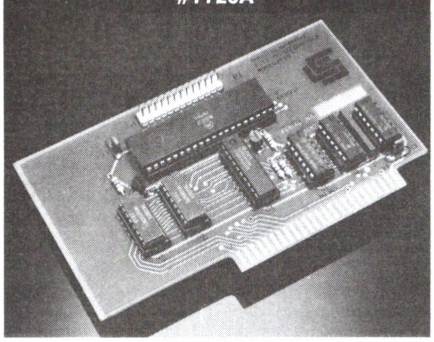
Arithmetic Processor #7811B



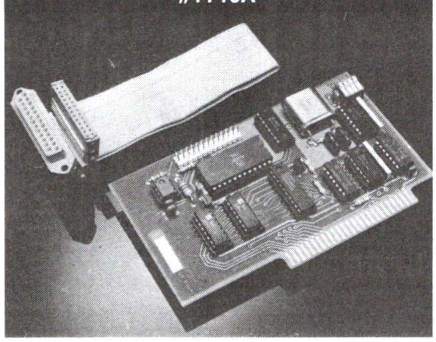
3¾ BCD A/D Converter #7470A



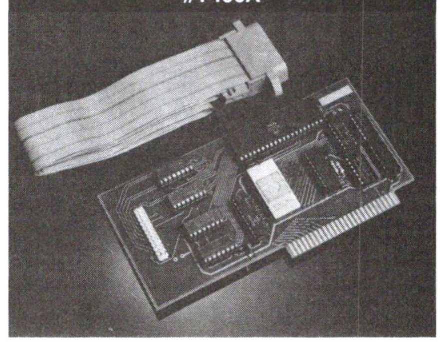
Parallel Interface #7720A



Asynchronous Serial Interface #7710A



GPIB IEEE 488 Interface #7490A



Programmable Interrupt Timer Module. Time events in four operating modes—continuous, single shot, frequency comparison, and pulse width comparison. Includes three 16-bit interval timers, plus flexible patch area for external interface. Programmable interrupts, on-board ROM, and much more.
Suggested list **$119.95**

Parallel Interface. Two bi-directional 8-bit I/O ports will connect your Apple to a variety of parallel devices, including printers, paper tape equipment, current relays, external on/off devices. Full featured, programmable interrupts, supports DMA daisy chaining.
Suggested list **$119.95**

Arithmetic Processor. Interfaces with Applesoft, so you just plug in and run. Based on the AM 9511 device, provides full 16/32-bit arithmetic, floating point, trigonometric, logarithmic, exponential functions. Programmed I/O data transfer, much much more. Not available in kit. (Not currently compatible with Apple II Plus.)
Suggested list **$399.95**

Asynchronous Serial Interface. Fully RS-232-C A thru E 1978 standard, this card will drive a variety of serial devices such as CRT terminals, printers, paper tape devices, or communicate with any standard RS-232 device, including other computers. Full handshaking, and fully compatible with Apple PASCAL!
Suggested list **$159.95**

3¾ BCD A/D Converter. Converts a DC voltage to a BCD number for computerized monitoring and analysis. Typical inputs include DC inputs from temperature or pressure transducers. Single channel A/D, 400 ms per conversion. Lots of potential.
Suggested list **$149.95**

GPIB IEEE 488 Interface. A true implementation of the IEEE 488 standard—the standard protocol for instrumentation and test devices. Control and monitor test instruments such as digital voltmeters, plotters, function generators, or any other device using the IEEE 488 protocol.
Suggested list **$300.00**

Apple II is a trademark of the Apple Corporation. Prices shown are for assembled and tested components. Kits are available.
CCS manufactures many other Apple components, plus components for the S-100 bus.

For serious users with serious uses for the Apple II computer, CCS manufactures a growing line of powerful card components. To find out just how much computer an Apple II can be, check out the whole California Computer Systems line at your nearest computer dealer, or send for literature.

California Computer Systems • 250 Caribbean • Sunnyvale, California 94086 • (408) 734-5811

California Computer Systems

From Call-A.P.P.L.E.

FLASH CARDS – A TUTORIAL PROGRAM

by

Rick Williams and Val J. Golding

The original program FLASH CARDS and an accompanying article by Rick Williams originally appeared in the September, 1979 Call –A.P.P.L.E., and a subsequent modification which provided a data base system for it appeared in October, as the result of a telephone call asking for a means of using a data base with it.

We have found through experience that a new routine almost always leaves something to be desired, and the Flash Cards Data Base proved to be no exception. In the instant case we have taken the concept of creating a data base for Flash Cards and started from scratch. It is a hard lesson, but we have learned, and wish to also pass on to our readers, that the user should do *NOTHING* that the computer can do for him. This left us with two design parameters (read problems):

1. That the user need not actually enter more than one line number nor the word DATA,
2. The menu must be so constructed as to allow it to be added to by the program.

The former was easy to accomplish, as it required only formatting the data elements before they were written to disk. The latter required considerably more thought. As it turns out, the menu is written so that the strings that compose it each have a length of one, unless a data base already exists for that item. Therefore the program searches for the first string that does not have a length greater than one, and after finding it, opens a temporary text file, writes the name of the new data base to it, and EXEC's it back into memory as a program line. The program then goes on to write the actual data base.

The reader is also alerted that lines 100 to 104, which contain six French vocabulary items as a short demonstration, must be actually written to a text file if their further use is desired. At the same time, line 8011 must be changed to read:

8011 F$(1) = "1"

so that the program will write the name of the COLORS data base in the correct line of the menu.

To create a new data base, simply select "0" from the menu and you will be prompted as to when to enter your data, which will be in the form of question and answer pairs. When all of the data has been entered, the program will then complete writing the new data base to disk, reSAVE the Flash Cards program (in order to retain the newly added menu item) and return the user to the menu where he or she may select and run any of the existing data bases, including the one just added.

For those who may not be familiar with the EXEC command, it is one of the most powerful tools you can use with your Disk II. In a nutshell, an EXEC file is a text file which is written in such a manner as to simulate the direct keyboard entry of commands and program lines, etc. When DOS receives an EXEC command it then reads the file, one record at a time onto the screen and into the keyboard buffer, where they are executed exactly the same as if the user had typed them in directly from the keyboard.

Anyone who had had to memorize a foreign language vocabulary, the capitals of the fifty states, or any series of question-answer pairs, has probably used flash cards. The applicable "question" on one side of a 3x5 card, and the "response" on the other. In spare moments one shuffles the cards and goes through them, one by one, reading one side and trying to guess – and eventually remember – what is written on the other. The cards are then shuffled again and the process is repeated.

Here is a short program that simulates such a deck of flash cards. The question-answer pairs appear in lines 100-998 as data statements in the form:

100 DATA question, answer, question, answer . . . You may use as many pairs as you wish, subject to available memory, and the data items may be of any convenient length. The program counts the number of pairs and drills you on them in random order by presenting the question, pausing, and then presenting the answer. Each pair is presented once and only once until the entire "deck" of flash cards is read. A bell then signals you that the cycle is complete and begins again in a different random order.

The main virtue of this tutorial is that, unlike many, the entry of your own material is neither fussy nor limited to a certain number of items in a certain format. The program is as easy to use and as effective as flash cards themselves.

Lines 1080 and 1100 may be modified to create any convenient pause. For keyboard control, substitute "GET Z$" in these two lines.

```
]LIST

10  REM        FLASH CARDS

          BY RICK WILLIAMS

20  CLEAR : GOSUB 8000: HOME : VTAB
     4: PRINT "THE FOLLOWING FLAS
     H CARD DATA BASES ARE AVAILA
     BLE"
30  VTAB 8: HTAB 4: PRINT F$(1): HTAB
     4: PRINT F$(2): HTAB 4: PRINT
     F$(3): HTAB 4: PRINT F$(4): HTAB
     4: PRINT F$(5): HTAB 4: PRINT
     F$(6): HTAB 4: PRINT F$(7): HTAB
     4: PRINT F$(8): HTAB 4: PRINT
     F$(9)
40  VTAB 20: INVERSE : HTAB 4: PRINT
     " SELECT DATA BASE FROM MENU
      ": HTAB 4: PRINT "OR 0 TO C
     REATE NEW DATA BASE"
50 D$ =  CHR$ (13) +  CHR$ (4): GET
     A$
60  NORMAL : IF A$ = "0" THEN 900
     0
70 A =  VAL (A$):FILE$ =  RIGHT$
     (F$(A), LEN (F$(A)) - 3)
80  PRINT D$"EXEC";FILE$: END : REM


100  DATA  ROUGE,RED,NOIR,BLACK
102  DATA  JAUNE,YELLOW,VERT,GREE
     N
104  DATA  ARGENT,SILVER,BLANC,WH
     ITE
999  DATA   *,*: REM

1000  READ Q$,A$:N = N + 1: IF Q$
      < > "*" THEN 1000
1010  RESTORE :N = N - 1: DIM A(N
      ),Q$(N),A$(N)
1020  HOME : PRINT
1030  FOR I = 1 TO N:A(I) = I: NEXT

1040  FOR J = 1 TO N:X =  INT ((N
      - J + 1) *  RND (1) + 1): READ
     Q$(A(X)),A$(A(X)):A(X) = A(N
      - J + 1): NEXT : RESTORE
1050  FOR J = 1 TO N
1060  PRINT
1070  HTAB 6: PRINT Q$(J);
1080  FOR Q = 0 TO 999: NEXT
1090  INVERSE : HTAB 18: PRINT A$
     (J): NORMAL
=100  FOR Q = 0 TO 999: NEXT
1110  NEXT
1120  PRINT  CHR$ (7)
1130  GOTO 1030: REM

8000  REM  DATA BASE WRITER FOR
            FLASH CARDS

           BY VAL J GOLDING

8010  DIM F$(9): REM MENU STRINGS
8011 F$(1) = "1  COLORS"
8012 F$(2) = "2"
8013 F$(3) = "3"
8014 F$(4) = "4"
8015 F$(5) = "5"
8016 F$(6) = "6"
8017 F$(7) = "7"
8018 F$(8) = "8"
8019 F$(9) = "9"

8100  FOR I = 1 TO 9: IF  LEN (F$
     (I)) = 1 THEN A = I: GOTO 81
     20: REM  WRITE NEW ITEM
             TO MENU
```

```
8110  NEXT
8120  PRINT D$"OPENTEMP"D$"WRITET
     EMP"
8140  PRINT 8010 + A" F$("A")=" CHR$
     (34);A"  "A$; CHR$ (34)
8150  PRINT "RUN8180"
8160  PRINT D$"CLOSE"
8170  PRINT D$"EXECTEMP": END
8180  GOSUB 8000: FOR I = 1 TO 9:
      IF  LEN (F$(I)) = 1 THEN A$
      =  RIGHT$ (F$(I - 1), LEN (
     F$(I - 1)) - 3): GOTO 9020
8190  NEXT : REM

9000  INPUT "WHAT NAME FOR NEW DA
     TA BASE ";A$
9010 D$ =  CHR$ (13) +  CHR$ (4):
      GOSUB 8100: REM  GO ADD TO
     MENU

9020 LINE = 100:INCR = 2:D$ =  CHR$
     (13) +  CHR$ (4): PRINT D$"D
     ELETETEMP"D$"OPEN";A$: REM
     OPEN NEW DATA BASE

9030  HOME : VTAB 2: PRINT "INPUT
      TWO DATA PAIRS (FOUR WORDS)
      "
9040  FOR I = 1 TO 4: INPUT DTA$(
     I): NEXT : PRINT D$"WRITE"A$

9050  IF LINE = 100 THEN  PRINT "
     DEL 100,998": REM SCRAP OLD
     DATA STATEMENTS

9060  PRINT "  "LINE"  DATA ";: FOR
     I = 1 TO 4: PRINT DTA$(I);: IF
     I <  > 4 THEN  PRINT ",";: REM
      ADD BASIC LINE FORMAT THEN
     WRITE TO DISK

9070  NEXT : PRINT : PRINT D$"CLO
     SE": PRINT "WRITE MORE DATA
     PAIRS ?": GET Y$: IF Y$ <  >
     "Y" THEN 9090
9080 LINE = LINE + INCR: PRINT D$
     "CLOSE"D$"APPEND"A$: GOTO 90
     40: REM WRITE NEXT GROUP OF
     DATA

9090  PRINT D$"APPEND"A$;D$"WRITE
     "A$: PRINT "RUN100": PRINT D
     $"CLOSE": HOME : VTAB 8: HTAB
     6: PRINT A$" DATA BASE COMPL
     ETED.": PRINT : PRINT "SAVIN
     G FLASH CARDS TO DISK": PRINT
     D$"SAVE FLASH CARDS": GOTO 2
     0: REM WRAP IT UP
```

APPLESOFT MEMORY MOVE

Much has been said in recent articles about how to do a memory block move from Integer Basic. It's almost as simple to do it from Applesoft. You just have to poke a short machine language routine into memory first.

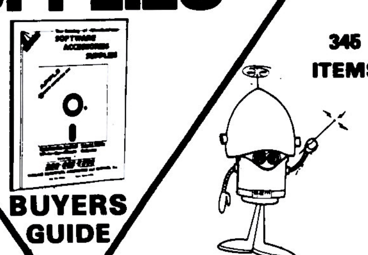
```
]LIST 100-120
100 POKE 768,160: POKE 769,0
110 POKE 770,32: POKE 771,44
120 POKE 772,254: POKE 773,96
```

This routine does a couple of things when called. It loads the "Y" register with a zero (LDY $ #0), calls the monitor memory move routine, then returns to Applesoft.

Here's a short subroutine that will move the contents of text page 2 into text page 1:

```
]LIST 200-250
200 REM
MOVE PG2 TO PG1
210 POKE 60,0: POKE 61,8
211 REM START OF BLOCK
220 POKE 62,255: POKE 63,11
221 REM END OF BLOCK
230 POKE 66,0: POKE 67,4
231 REM START OF OBJECT MEMORY
240 CALL 768: RETURN
241 REM DO THE MOVE
```

That's all there is to it. Have fun.



TEXT EDITOR

The Text Editor is a fast, simple to use program designed for the small business or personal user. *Text is processed in memory on a line by line basis. When the user is finished with the text, he may print it and/or save all or part of it to disk for later use. Upper and lower case operation is fully supported. If the computer has a hardware adapter (such as the Dan Paymar board) true upper/lower case is displayed.*

MAILING LIST

500 Names and Addresses per 5¼ inch diskette. Provisions are included to split files between diskettes giving virtually unlimited storage capacity. Any name on a diskette can be retrieved within 3 seconds. Names and addresses can be sorted in either Alphabetical or Zip code order. The sort typically requires less than 2 seconds to process 500 names.

FORM LETTER

The Form Letter package is designed to combine files from our Text Editor and Mailing List programs to enable the user to print out customized letters—such as advertising, notices, and so on. A model letter is created using the Text Editor. The user embeds special character strings in the body of the letter. When the Form Letter is run, these special strings are replaced by information contained in the Mailing List records.

DAN'S DISK UTILITIES

The Dan's Disk Utility Program [DDU] allows the user to directly examine and/or modify data on any trace and sector of a diskette. This program can be used to:

- *Enter patches to machine language programs on diskette.*
- *Recover a file that was accidentally deleted.*
- *Determine correct file sizes.*
- *Examine text files to check program operation.*

INTEGER

PASCAL UTILITY MICROMODEM PACKAGE [PUMP]

The Pascal Utility Micromodem Package is a set of intrinsic functions designed to facilitate the use of the D.C. Hayes Micromodem II with Apple/UCSD pascal. All the functions and procedures have been placed into an intrinsic unit and the unit has been placed in the SYSTEM.LIBRARY on the diskette we supply. We also include a routine for those users that have Dan Paymar's Lower Case Adapter, that modifies BIOS to allow the display of lower case characters.

MEMO CALENDAR

The Memo Calendar is designed to perform the functions of a diary and a memo pad/calendar. The user enters an 80 byte "memo," and assigns a date to it. You can call up any dates memos on the screen at any time.

APPLESOFT

APPLE BULLETIN BOARD SYSTEM

The Apple Bulletin Board System [ABBS] implements a personal computer based message storage and retrieval system on a 48K Apple II computer. The system is configured to automatically answer incoming phone calls through the use of a D.C. Hayes Micromodem II. Once it has answered the phone, and connection has been made with a computer or terminal at the other end of the line, the program determines the baud rate of the caller, and leads him through a small sign-on routine. After the caller is logged-in, he is welcomed to the system, and shown any announcements that the SYSOP wishes to place on the ABBS. He is then given limited control of the system, and can leave messages, or scan and read messages left by others.

SINGLE DISK COPY

The single disk copy program is intended for those Apple users who do not have two disk drives, but still need to copy diskettes for back-up purposes. The single disk copy program operates by reading as many sectors as your Apple's memory can hold, and then writing the sectors back out onto the copy diskette.

INTEGER

PERIPHERALS UNLIMITED INC.

2633 E 28th St STE 622

SIGNAL HILL, CA 90806

(213) 595-6858

Source ID TCA 099

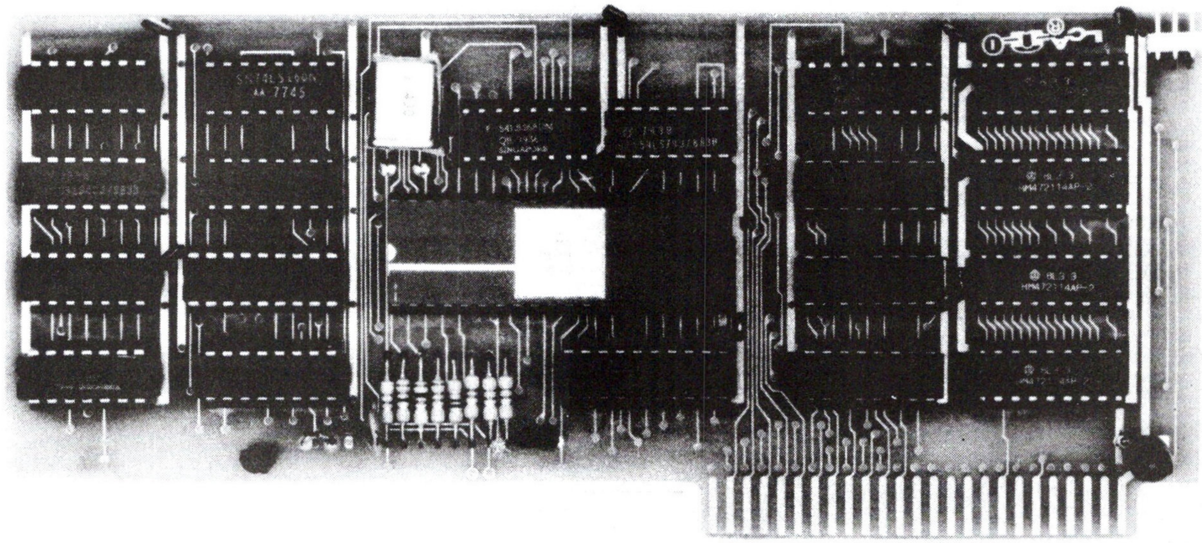
We will gladly accept orders over the phone, and usually ship the same day. On prepaid orders, we pay shipping and handling. We ship via United Parcel Service, Blue Label (air freight) if out of state. COD, Master Charge and Visa orders accepted. On orders from outside the United States, please use International Postal Money Orders drawn for US currency for payment.

Text Editor	$64.95
Mailing List	$39.95
Form Letter	$24.95
Memo Calendar	$29.95
Single Disk Copy	$19.95
Dan's Disk Utility	$24.95
Apple Bulletin Board System	$49.95
Pascal Micromodem Utilities	$24.95

All programs are supplied on a 5¼-inch diskette. Unless otherwise stated programs are a machine language Applesoft hybrid.

apple® computer apple® computer

DOUBLEVISION™



80 x 24 Video Display
COLUMNS LINES

with Upper and Lower Case

- Works with Apple II*, Apple II Plus*, and PASCAL
- Full 96 ASCII character set
- Fully programmable cursor:
  - 1-9 lines any position
  - Blinking (2 speeds) and non-blinking
- All software included for BASIC (optional for PASCAL)
- Shift Lock Feature
- Built in Light Pen capability
- Inverse video
- Full cursor control
- 50/60 Hz operation
- No conflict with other boards using $C800 to $CFFF

Introductory Sale Price .... $195°° **

- Compatible with the latest in word processing software "Apple-Pie 2.0"

►PASCAL software interface available for $25.00 additional
**For all orders received before midnight, April 30, 1980
Deliveries start end of February • Allow up to 4 weeks for shipment.
All Mail orders add $3.00 for postage, insurance and handling

Calif. Residents add 6% Sales Tax

*Apple is a Registered TM of Apple Computers, Inc.

The Computer Stop
16919 Hawthorne Blvd.
Lawndale, CA 90260
(213) 371-4010

MON. - SAT.
10-6

DISK BOOTSTRAP WITHOUT ROM

by
Richard F. Suitor

This note will discuss ways of "bootstrapping" the Apple Disk software without using the ROM that is built into the disk controller card. Whatever for, you ask? Those who have the language card and who have heavy investments in Integer and Applesoft software may find it useful. A person who has the standard language card may be doing program development in BASIC, and may, during the course of the debugging process, cause fatal changes in DOS. If a BASIC loader is stuck away in a portion of the language card that BASIC does not use, one can reboot directly, without using the Pascal BASIC disk. My personal interest is the converse. My family uses BASIC programs extensively, and I do not wish to insert another step in the startup procedure. If I decide to get Pascal, I would like to keep my BASIC loader on the disk controller, and boot Pascal with a loader in RAM. I don't know yet whether this is possible, but the initial trials were promising.

Both BASIC and Pascal disks can be booted from the loaders described here. I have not used Pascal enough to know whether extensive use after such a boot would cause later problems. Anyone who plans to try this should realize that only one of the two ROMs on the controller is a program ROM. That is the one that is low on the board with the lines running to the data pins on the connector. The other ROM is used internally on the controller, and is probably important to the increased data density that Pascal disks enjoy. That should remain the Pascal version. Will it be compatible with the BASIC software? Yes, it is used with BASIC software now. DOS speaks directly to the controller with RWTS; it does not use any of the routines on the controller card.

The following instructions will produce a loader that will be completely relocatable but will boot only from slot 6. The slot can be easily changed if desired.

Pascal Loader

The first step is to obtain a RAM copy of the Pascal loader by going to the monitor and moving the loader to a convenient spot, say $6000, with *6000<C600.C6FFM Now some changes must be made:

```
6028:85 1F A9 60
60F8:A5 1F 8D 18 08 4C 01 08
```

Return to BASIC with 3DOG and BSAVE PASCAL RELOAD, A$6000, L$100.

What do the changes do? The first four replace four shifts that change $C6 to $60. Instead, $C6 is stored in $1F and the A register is loaded explicitly with #$60. (This is the number that should be changed if another slot is used). At the end of the loader, the $C6 is retrieved from $1F and stored in $818 where it will provide a subroutine linkage for the first sector loader. Note that if the loader routine that is brought in from the first sector of the disk is ever changed, then there will have to be a corresponding change in this loader.

BASIC Loader

Again, first obtain a RAM copy of the standard loader by going to the monitor and

```
*6000<C600.C6FFM
```

Then make the following changes:

```
6026:85 1F A9 60
60F8:0D A9 A9 8D 1F 03 A5 1F 8D 20 03 4C 01 03 4C 2D FF
```

Then return to BASIC, and BSAVE BASIC LOADER,A$6000, L$109.

The nature of these changes is similar to those for the Pascal loader. This also changes the data loaded in from the first sector, and thus may not work on a special purpose disk with its own DOS and/or loader. For those, you'll just have to use your BASICS disk, which you'll have to use to get going anyway.

The Pascal loader would normally be stored on a BASIC disk, and BLOADed when it was desired to run Pascal. (Although the starting address is at the beginning, don't BRUN it unless from another slot or drive.) Drive one of slot 6 should have PASCAL3: in it when the loader is started. I have not used this method much (I don't have Pascal yet), but know that it will get one going. Another little chore: you must get a copy of either Integer or Applesoft to store in your language card. It is particularly important that a monitor be in the $F800-$FFFF region before the language card is read from. That can be accomplished by going to the monitor and accessing C081 twice. This reads from the ROM but write enables the RAM on the language card. Then

```
*F800<F800.FFFFM
```

will move the monitor. If you have a program that needs the old monitor (many used the multiply-divide routines which Apple so heartlessly discarded in the new version), get a disk copy of the old monitor from a friend or dealer with an old Apple. BLOAD it, and move it into the monitor region with above command. You may be out of luck though, if your program requires that the BASIC version that you have in ROM be active. That will enable the new monitor.

```
10 REM  XYZZY BY PETER SCHUG

20 REM  IN INTEGER BASIC

30 REM THIS PROGRAM DEMONSTRATES
   HOW A TWO LINE GRAPHICS PROGRAM
   CAN BE BOTH EDUCATIONAL AND IN-
   TERESTING

100 GR
110 COLOR= RND (16)+1:A= RND (1280
    ):X=A MOD 32:Y=A/32: PLOT X,
    Y: GOTO 110
```

THE APPLE RUMOR MILL

COURTESY OF PAUL KNEVELS

FROM The Michigan Apple-Gram

It has been rumored that APPLE is working on an APPLE III computer to be released shortly. At present the only advance information available is that a new microprocessor will be incorporated into the unit — the 6503.

Our research staff has been able to uncover a list of new op-codes that distinguish the 6503 as a breakthrough in computer technology.

The list is presented here for your information (and enjoyment).

AGB	ADD GARBAGE
BBL	BRANCH ON BURNED OUT LIGHT
BAH	BRANCH AND HANG
BLI	BRANCH AND LOOP INFINITE
BPB	BRANCH ON PROGRAM BUG
BPO	BRANCH IF POWER OFF
CPB	CREATE PROGRAM BUG
CRN	CONVERT TO ROMAN NUMERALS
DAO	DIVIDE AND OVERFLOW
ERS	ERASE READ-ONLY STORAGE
HCF	HALT AND CATCH FIRE
IAD	ILLOGICAL AND
IOR	ILLOGICAL OR
MDB	MOVE AND DROP BITS
MWK	MULTIPLY WORK
PAS	PRINT AND SMEAR
RBT	READ AND BREAK TAPE
RPM	READ PROGRAMMER'S MIND
RRT	RECORD AND RIP TAPE
RSD	READ AND SCRAMBLE DATA
RWD	REWIND DISK
SRZ	SUBTRACT AND RESET TO ZERO
SSD	SEEK AND SCRATCH DISK
TPR	TEAR PAPER
WED	WRITE AND ERASE DATA
WID	WRITE INVALID DATA
XIO	EXECUTE INVALID OP CODE
XOR	EXECUTE OPERATOR
XPR	EXECUTE PROGRAMMER

P.S. — There is no word from APPLE as to when we might expect these improvements. Perhaps in the next CONTACT.

P.K.

```
]LIST

10  REM  HEX-DEC CONVERTER

         BY VAL J GOLDING

100  HOME : GOSUB 500
110  VTAB 8: HTAB 8: PRINT "SELEC
     T MODE": PRINT : HTAB 10: PRINT
     "1 DECIMAL TO HEX": HTAB 10:
      PRINT "2 HEX TO DECIMAL": HTAB
     10: PRINT "3 EXIT": PRINT : INVERSE
     : HTAB 12: PRINT "SELECT": NORMAL

120  GET A: ON A GOTO 200,300: END
     : REM
200  HOME : VTAB 8: INPUT "ENTER
     DECIMAL NUMBER ";NBR
210 MOD256 =  FN MOD( NBR)
220  POKE 1,MOD256: POKE 0,NBR /
     256: REM POKE DATA INTO $0,1

230  POKE 60,0: POKE 61,0: POKE 6
     2,1: POKE 63,0: REM  POKE
          DATA ADDRESSES INTO X
          AND Y REGISTERS

240  CALL  - 936: VTAB 7: PRINT "
     DECIMAL = ";NBR: CALL  - 589
     : REM PRINTS CONTENTS OF
     A1L,H THRU A2L,H IN MONITOR

250  POKE 1064,160: POKE 1065,200
     : POKE 1066,197: POKE 1067,2
     16: POKE 1068,189: POKE 1069
     ,160: REM POKE ASCII INTO
               SCREEN

260  GOTO 400: REM

300  HOME : VTAB 8: INPUT "ENTER
     HEX NUMBER ";NBR$
310 HI$ =  LEFT$ (NBR$,2):LO$ =  RIGHT$
     (NBR$,2)
320  IF  LEN (NBR$) < 3 THEN HI$ =
     ""
330 HEX$ = "0:" + LO$ + " " + HI$
     + " N D823G"
340  FOR I = 1 TO  LEN (HEX$): POKE
     511 + I, ASC ( MID$ (HEX$,I,
     1)) + 128: NEXT : POKE 72,0:
      CALL  - 144
350  HOME : VTAB 7: PRINT "HEX =
     ";NBR$: PRINT "DECIMAL = "; PEEK
     (0) +  PEEK (1) * 256: REM

400  VTAB 20: INVERSE : HTAB 6: PRINT
     "HIT ANY KEY TO RESUME": NORMAL
     : GET A$: POKE 0,0: POKE 1,0
     : GOTO 100: REM

500  DEF  FN MOD(NBR) = (NBR / 25
     6 -  INT (NBR / 256)) * 256:
      RETURN : REM  DEFINE "MOD"
                    FUNCTION
```

MICROPRODUCTS/APPLE II*

6 CHARACTER LABEL EDITOR/ASSEMBLER

The **MICROPRODUCTS/APPLE II 6-Character Editor/Assembler** follows most of the basic rules and conventions and uses the same opcodes as developed by MOS Technology for the 6502 mircoprocessor, which is used in the APPLE II. This assembler, however, is much more powerful than the Apple factory supplied assembler, and incorporates a powerful text editor.

ADVANTAGES — An assembler with text editor immeasurably improves the user's ability to develop assembly language programs. It is approximately as easy to originate a machine language program with this assembler as it is to write a program in BASIC. Suppose it is necessary to add an instruction between previously written instructions. Without the assembler, it would be necessary to rewrite all of the instructions following the added instruction in order to relocate them in order. However, with this assembler, it merely requires inserting the new instruction with an intermediate line number and type "ASM" and a carriage return. This automatically relocates all successive code. This feature permits deletion and rearrangement as well as addition of instructions. When used in conjunction with our Disassembler/Text File Manager, it can insert portions of any text file into the main text file; thus providing features as a Macroassembler.

The assembler can assign a six-character mnemonic label to memory locations used as temporary storage registers or assigns a six-character mnemonic label to subroutines. The advantage here is that it is easier to remember a word related to what the subroutine does than to remember hexidecimal addresses for 20 or 30 subroutines or temporary storage registers. Any detected errors are immediately displayed in English along with the line number of the error.

FEATURES: The MICROPRODUCTS/APPLE II 6-Character Coresident Editor/Assembler, for high speed program development, is available on APPLE II compatible floppy diskette with instructions. It has two pass implementation and incorporates a text editor. This assembler incorporates provisions for calling any printer driver from any location in memory: ROM, RAM, PROM or EPROM. It can operate with any printer to provide hard copy records of programs when desired. This assembler also directly supports the MICROPRODUCTS/APPLE II EPROM Programmer and assembles code at over 3000 lines per minute. When data lines (line number followed by assembly instructions) are entered a syntax check is performed on each input line before that line is stored in the text file. Twenty text editor commands are available: a) **LINKING LOADER**, (resolves external label references), b) Delete portions of text record, c) Execute a DOS command, d) Select any increment for renumbering, e) Load previously saved text file from cassette, f) List text file, g) List any portion of text file, h) Initialize text line pointers prior to creation of new text file, i) Enter location of printer driver routine, j) Return to monitor, k) Renumber lines, l) Execute code without entering monitor, m) Save Text file on cassette, n) Scan text file for certain label, o) Assemble data in text file, p) Load text file from disk, q) Save text file on disk, r) Tab function, s) Concatenate text files, t) Restore text file pointers if destroyed.

Part No. 1013 .... on diskette $29.95

6 CHARACTER LABEL DISASSEMBLER/ TEXT FILE MANAGER

MICROPRODUCTS announces a powerful new two pass Disassembler/Text File Manager for the APPLE II microcomputer. This very useful programming tool disassembles any machine language program which resides in the APPLE II, such as BASIC, the Disk Operating System and printer driver routines, etc. and creates a text file for the MICROPRODUCTS 6-Character Label Editor/Assembler.

This disassembler will be extremely valuable to any programmer who wants to rewrite, debug, modify, analyze and understand the workings, functions and operation of inadequately documented programs for which there is no source listing available.

The Text File Manager portion of this program has the following features. • The Text File can be listed in toto • A range of line numbers can be listed • The start address of any printer driver routine can be specified • The Text File, or portions thereof can be saved on cassette or disk (very useful in generating subroutines) • The text file created starts at the same location as text files created by the MICROPRODUCTS 6-Character Label Assembler and is therefore completely compatible with that assembler.

Part No. 1015 .... diskette $29.95

APPLEBUG

APPLEBUG is a powerful programming aid that will assist in developing, debugging and testing machine language code on the APPLE II. APPLEBUG will also facilitate tracing logic of existing machine language programs such as the monitor, DOS and Applesoft. Since the Trace and Single Step functions have been deleted in the APPLE II Plus, APPLEBUG can replace those functions and has the capability for virtually an unlimited number of trace addresses and break points. Contents of trace addresses are displayed in Hex and ASCII. APPLEBUG has been designed to operate as a "stand-along" debug package, or in conjunction with the MICROPRODUCTS 6-Character Label Editor Assembler. In either environment, all modes and options are available. I/O management commands are included to facilitate the saving and/or loading the Label Table to or from disk or tape.

Part No. 1028 .... $29.95

ASSEMBLER/DISASSEMBLER/APPLEBUG COMBINATION

The Assembler, Disassembler and APPLEBUG are available on a single diskette along with 17 useful subroutines and the book "How to Program Microcomputers."

Part No. 1035 .... $89.95

4-CHARACTER ASSEMBLER TO 6-CHARACTER ASSEMBLER TRANSLATOR

The MICROPRODUCTS translates text files which were originated on the original 4-Character Label Co-resident Assembler into a format which permits them to be assembled by the new 6-Character Label/Editor Assembler.

Part No. 1023 .... $9.90

THERE ARE NO TRICKS TO PREVENT YOU FROM LOADING, LISTING AND MAKING A BACKUP COPY OF YOUR DISASSEMBLER OR ASSEMBLER.

* APPLE II is a trademark of Apple Computer Co.

APPLE II EPROM / Expand your ROM Software

Add capability to your system monitor or BASIC for business or other applications. Add to or replace existing APPLE II ROM software with operating systems of your own design. Other software systems similar to PASCAL, FORTH, LISP, APL, FORTRAN, COBOL, ALGOL, other BASIC's, etc. may be incorporated into your APPLE II ROM space. New operating systems can be put into EPROM memory with our EPROM programmer and plugged directly into your APPLE II board with our EPROM socket adaptor. The MICROPRODUCTS EPROM Programmer will program INTEL 2716s, 2758s and other 5-volt replacements for 2716s.

The EPROM Programmer looks just like memory to the computer and can be configured to program memory locations from 8000 to FFFF for a total range of 32K bytes. This means that the EPROMs can be used in computer applications other than the APPLE II, i.e., the MICROPRODUCTS Superkim, etc. This turns your APPLE II into a very low cost, powerful software development system.

FEATURES:
- Fully assembled
- Completely self-contained
- Onboard 25 volt power supply
- Textool Zero insertion force socket for EPROM
- Double sided plated through holes on fiberglass PC board
- Gold plated edge connector
- Fully socketed
- Latest low-power Schottky IC's
- Solder mask

ADVANTAGES:
- Put memory in two empty ROM slots in APPLE II
- Replace memory in existing APPLE II ROM slots
- Add new operating systems to APPLE II
- Programs INTEL 2716 2K byte EPROM's, 2758 1K byte EPROMs and other compatible 5 volt EPROMS
- Put peripheral drivers in permanent memory
- Use APPLE II to program EPROMs for other computers.

Part No. 1008 .... $99.95

APPLE II EPROM SOCKET ADAPTER

Since the 5-volt EPROMs on the market today are not pin compatible with the APPLE II ROMs, they will not work when plugged directly into the APPLE II ROM sockets. MICROPRODUCTS makes an EPROM socket adapter into which your 5-volt EPROM is inserted before insertion into your APPLE II. One MICROPRODUCTS socket adapter is required for each EPROM to be inserted into an APPLE II ROM socket.

Part No. 1009 .... $14.95

APPLE II PRINTER INTERFACE

Add a Printer to Your Apple II

With our Parallel Output Port Card that interfaces with any parallel printer. Printer driver routines available for Centronics 779, Anadex, OKIDATA 110 and PR-40 with others under development. Driver routines are supplied on cassette but may be ordered on EPROM (Interface brain).

Parallel Output Port Card is completely assembled, tested and guaranteed. Including Interconnecting cable, software stored on audio cassette and PC board which plugs directly into your APPLE II.

SPECIFICATIONS: Interface hardware consists of • an epoxy fiberglass PC board • double-sided • plated through holes • silk screen printed legends • gold plated edge card connector.

Printer Interface, Part No. 1001 .... $49.95

Parallel Output Port Card, Part No. 1005 .... $39.95

The MICROPRODUCTS Parallel Output Card can also enable your APPLE II computer to communicate with the outside world. Applications include power controller, tone music generator, plotter driver.

Features include 8 bits output, 15 ma output, current sink or source (can drive LEDs directly), TTL or CMOS compatible, will go in any slot on the APPLE II. Data available strobe.

GENERAL INFORMATION: Data can be transferred to an external device by a STA, STY, or STX from assembly language, or a POKE from BASIC. The 8 bits output can drive two 7-segment LED displays, relays, SCRs, printer, or anything which requires up to 8 bits of data.

APPLE INTERFACE BRAIN

MICROPRODUCTS announces the Interface Brain. This device plugs directly into your APPLE II computer to provide permanent memory intelligence for versatile, flexible and inexpensive so-called "dumb" peripheral interfaces. It supplies the permanent full-time availability of firmware drivers for the Centronics 779, PR-40 and OKIDATA printers as well as the MICROPRODUCTS EPROM Programmer the instant your computer is switched on. It allows the flexibility of a user changeable EPROM where situations of software or hardware update indicate a change is desirable or necessary. The Interface Brain is supplied on an EPROM set in a MICROPRODUCTS EPROM Adapter Socket, to permit direct insertion into the D8 ROM slot in your APPLE II, along with the necessary documentation for its operation.

Part No. 1004 .... $54.95

These products are available at your local computer store or directly from MICROPRODUCTS.

COMING SOON FOR APPLE II

• 6809 CONVERSION KIT •

Video Board to Enable You to Display 80 Characters, Upper & Lower Case, on the Screen of Your Monitor

AC Power Controller with EPROM Intelligence

Printer Interface with EPROM Intelligence

Software Downloading Card with EPROM Intelligence

microproducts

2107 ARTESIA BLVD./REDONDO BEACH, CA 90278 (213) 374-1673

From Call-A.P.P.L.E.

Bob Denison tells us: "I accepted the responsibility of reviewing the programs because I wanted an excuse to really take them apart. I have been in the investment business since 1964, and a general partner of First Security Company for a dozen years. I am also a Professor at the Columbia Business School where I teach the security analysis course. Consequently, my primary interest in my Apple is in its investment applications."

SOFTWARE REVIEWS

by
Robert Denison

Program: ANA1 (ALALYSIS 1)
Author: Galaxy
P.O. Box 22072
San Diego, CA 92122
Purpose: Statistical analysis of Dow Jones data
Language: Basic-48K,Applesoft ROM card, and disk required
Price: Disk & Manual . . . $49.95

Ratings:
Speed – 85
Ease of Use – 85
Documentation – 80
Error Comments – 75
Screen Display – 95
Purpose/usefulness – 90
Reliability – ?
Technical program level – 90

ANA1 is a set of Basic programs which provides a remarkably flexible approach to the analysis and "hires" plotting of any time series data. As delivered, the program is set up with history for the Dow Jones Industrial Average since 1904.

Up to 260 points can be plotted on the screen in any of five user selected colors. Scaling of the graphs is done automatically, subject to user change. Having loaded the initial data (as supplied on disk or your own), various mathematical transformations can be easily performed and then plotted. Built in functions include moving averages, least squares linear fit, and relationships to any constant using +, -, x, / operators. The results of such computations can be plotted either individually or in an unlimited series of overlays.

Additionally, straight lines can be drawn between any set of two point on the screen to portray trend lines. The program also maintains the data on a separate "page" so the user is able to flip back and forth between the graphs and the underlying numerical values. Other features include the ability to "filter" the data for user selected changes in absolute or percentage magnitude or time.

While my description may sound confusing, the program is in fact easy to use. The documentation and disk include a built-in programmed tutorial which quickly demonstrates the capabilities and commands of the system.

My only substantive reservation about this program is that there is no specific labeling of the multiple plots beyond the predictable sequence of five colors. Consequently there might be problems with the use of either black and white monitors or the growing number of printers which can perform "hires dumps." The author told me he was considering a modification to cover this, so I would suggest you contact Galaxy before purchase if you require such a change.

In summary, I consider ANA1 to be an excellent plotting package (the best I've seen) whether or not you are interested in the Dow Jones Averages.

Program: STOCK MARKET SYSTEM
Author: RTR SOFTWARE INC.
P.O. Box 12351
El Paso, Texas 79912
Purpose: Stock market charting
Language: Basic-32K/Applesoft ROM card or 48K and Disk required
Price: Disk & Manual . . . $79.95
One year's data per stock . . . $9.95

Ratings:
Speed – 85
Ease of use – 85
Documentation – 70
Error Comments – 85
Screen Display – 90
Purpose/usefulness – 75
Reliability – ?
Technical program level – 75

RTR'S Stock Market System is a highly specific and dedicated plotting program for the technical analysis of security prices, providing two types of "hires" graphic display. Firstly, individual stocks can be charted with the traditional display of "high-low-close" bars at the top of the screen with daily and average volume shown underneath. In this mode user selected moving averages may by overlayed, and grid scaling is automatic.

Secondly, up to five stocks can be plotted at once to show relative performance over the user selected time period. The securities are graphed in different colors which are labeled at the bottom. No provision is made for hard copy.

RTR offers to provide one year's weekly data on any listed security (minimum of two) at $9.95 each. Weekly updating can be continued by the user. Of course, initial data can be user entered with any chosen time frequency.

The user's manual is 16 pages long, and covers in detail such functions as creation and update of files, data transfer to other disks, automatic adjustment for stock splits, and charting analysis. While I found that the documentation required several readings, the program itself was very much easier to use. It is largely self prompting, and has good error recovery routines.

While both ANA1 and STOCK MARKET SYSTEM are advertised as securities charting programs, they are quite different. ANA1 offers far more flexible and sophisticated options for the analysis and plotting of stock prices or any other time series data. It is also much less expensive. The STOCK MARKET SYSTEM does, however, perform its specific objectives very well and the availability of historical data on disks may well be very important to people who lack either the time or inclination to key it in themselves.

Finally, I would like to point out that while there is great controversy in academic circles as to the validity and statistical significance of traditional stock market charting systems, nevertheless many investment firms at least consider such technical analysis to be one of many approaches to the problem.

ADVERTISER'S INDEX

A.P.P.L.E. 100
Apple Computer Co. .... Insert
Apple Orchard 69
Apple Shoppe, The 75
Andromeda Computer Systems 57
California Computer System 88
Compress, Inc. 61
CompuServe .... 72 & 73
Computer Case Co. 100
Computer Services 85
Computer Stop 94
Cook's Computer Co. 89
Corvus 2
Creative Computer 90
Data Cope 41,91
D.C. Hayes 49-52
Eastern House Software 63
Escon Products., Inc. 60
Galaxy 68
Hayden Book Co. Inc. 77
Heuristics, Inc. 92
Information Unlinited 24
Integral Data Systems .... Inside Back Cover
Lobo Drives 78
Microproducts 97
Microsoft 34
M & R .... Inside Front Cover
Mountain Hardware, Inc. 19, 20, 81, 82
Muse Software 84, 99
Pacific Exchange 10
Peripherals Unlimited 93
Personal Software 1
Powersoft, Inc. 56
Programma International .... Back Cover
Rainbow Computing 66
RFC/Sirius 46
RTR Software, Inc. 8
Small Business Computer System 36
Softagon 30
Sorrento Valley Association 91
Southwestern Data Systems 39
S.P.A.R.C. (Nibble) 87
Special Systems Design 100
SSM 3
Stoneware 40
Sublogic 38, 65
Synergistic Software 11
TCA (Source) 6
Telephone Software Connection 33
Vitek 43
Wallace Computers 92
Widl 86

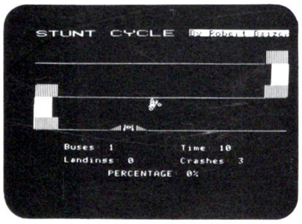
The Paper Tiger.™

will take bytes from your Apple.

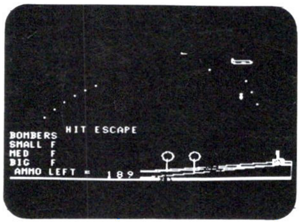
for the name of the Paper Tiger dealer
nearest you, call toll-free 800-343-6412

Integral Data Systems, Inc.
14 Tech Circle
Natick, Mass. 01760
(617) 237-7610

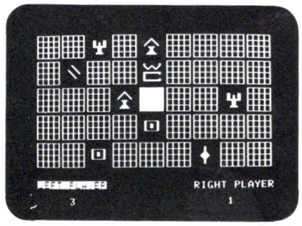
NEW APPLE II SOFTWARE



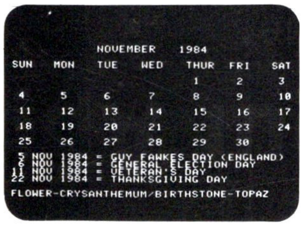
STUNT CYCLE $15.95



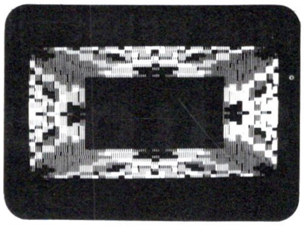
BLITZKRIEG $15.95



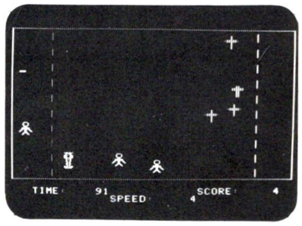
FRUSTRATION $ 9.95



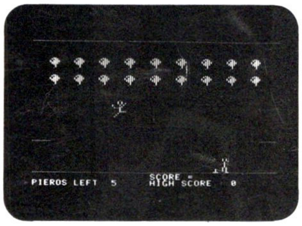
PERPETUAL CALENDAR $ 9.95



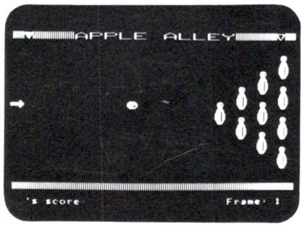
LORES HYPERPAK $ 6.95



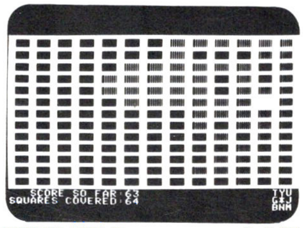
DEATH RACE $15.95



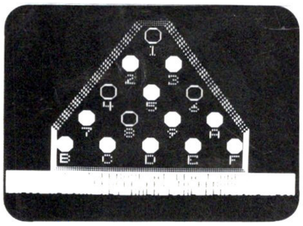
CLOWNS & BALLOONS $15.95



APPLE ALLEY $ 6.95

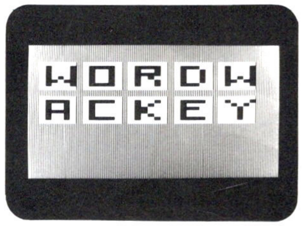


MOUSE HOLE $ 6.95

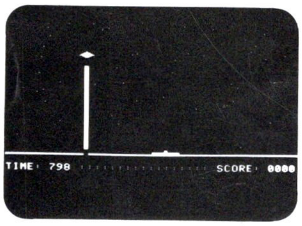


PEG JUMP $ 9.95

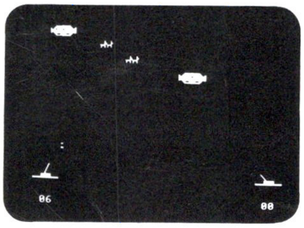
I-CHING $15.95



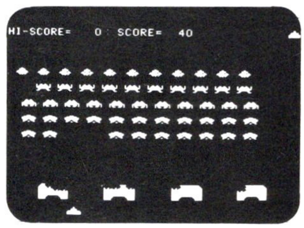
WORDWACKEY $ 6.95



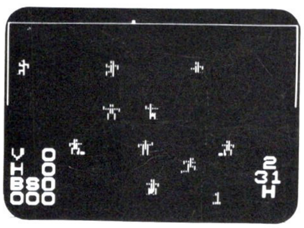
ALIEN INVASION $ 9.95



GUIDED MISSILE $15.95



APPLE INVADER $15.95

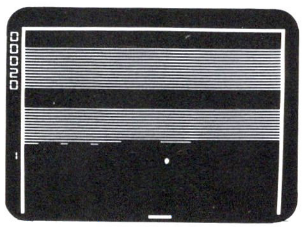


BASEBALL $15.95

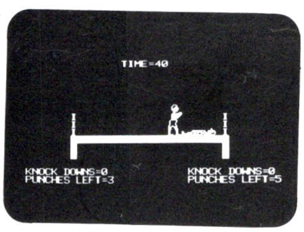
All orders must include 3% postage and handling with a minimum of $1.00. California residents include 6% sales tax.

VISA MASTERCHARGE

Apple II is a trademark of Apple Computers, Inc.



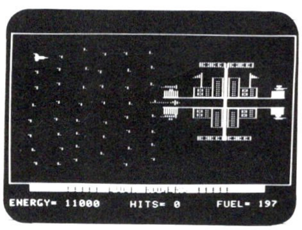
BREAKTHRU $ 9.95



BOXING $ 9.95

PROGRAMMA
INTERNATIONAL, Inc.
3400 Wilshire Blvd.
Los Angeles, CA 90010
(213) 384-0579
384-1116
384-1117

Dealer Inquiries Invited



STAR VOYAGER $15.95

PROGRAMMA

Software Program Products