Choices in Educational Computing." It first gives a brief history of computers in education, going back to the 1960's when talk of a forthcoming computer revolution in education was first heard. The authors then raise important questions for the current period of computers in education. These include: Where will good software come from? How will teachers get the needed training? Is the available hardware adequate? What kinds of social and educational changes might result from computers in the schools? The chapter ends with descriptions of four possible futures: computers turn out to be another educational fad; education becomes more centralized and systematic through the use of computers; schools become irrelevant; and educators become comfortable with computers and make good use of them.

The book also contains a large section on resources for educators. This 48-page list was compiled by Newton Key of Intentional Educations, and it will be updated in the 1983 *Classroom Computer News Magazine* Directory of Educational Computing Resources. Among the listings included are software directories and catalogs, associations, periodicals, resource centers, user groups and other sources of information. This is a valuable resource list. However, it will become dated quickly and annual updates are essential.

I recommend *Practical Guide to Computers in Education* to every educator who is starting to use, or considering using, computers. The chapters on evaluating software and introducing computers into schools, the real-life vignettes throughout the book, and the resource list and bibliography are especially valuable.

However, I have one general criticism of this book (in addition to the quibbles discussed above). It is, in many sections, negative in tone and shortsighted. I suspect the authors are over-reacting to those who claim that computers will cure all that ails education. Certainly, computers are not a panacea and a practical guide should be honest about the difficulties one may encounter. But I believe a book of this sort should also convey an understanding and appreciation of the potential of computers, and it should encourage educators to explore the ways computers can be used.

The authors limit themselves to describing existing software, without acknowledging recent improvements or looking ahead to what will be available soon. Little is said about computer graphics or music, two areas in which some excellent innovative software is already available. They mention the Source, but fail to convey the excitement of having enormous amounts of information readily accessible from your classroom or home. Nothing is said about the potential of video disks, computerized speech synthesis or speech recognition. They also neglect the ways computers will be able to aid handicapped students, such as automatic readers for the blind.

Perhaps the authors tried to limit the book to what is immediately practical for educators. However, with the rapid pace of change, what was not practical when the book was written may well be practical by the time many educators read it.

Software Catalogs

There is a lot of educational software available, but it is often difficult to find what you need and even more difficult to know whether it is worth purchasing. Several companies have put together educational software catalogs which should be very helpful. All the companies listed below have reviewed the software in their catalogs and, better yet, will allow you to return software within 30 days of purchase – a risk-free way to check whether a particular program meets your needs. All of these catalogs include Apple, TRS-80 and PET software for grades K-12, as well as books and computer supplies. Some also include software for Atari and TI computers.

K-12 Micromedia. P.O. Box 17, Valley Cottage, NY 10989, 914-358-2582.

Scholastic Microcomputer Instructional Materials. Scholastic, Inc. 904 Sylvan Ave., Englewood Cliffs, NJ 07632, 800-631-1586.

J. L. Hammett Microcomputer Catalog. Hammett Place, Braintree, MA 02184, 800-225-5467.

Opportunities for Learning Catalog. 8950 Lurline Ave., Chatsworth, CA 91311, 213-341-2535.

Periodicals on Computers in Education

There are now many periodicals on computers in education. Each of the following contains a variety of articles, announcements about new hardware and software, software and book reviews, and other information of interest to educators. Most of these have been publishing a short time and continue to change and develop. Judging from the past year, I recommend *Electronic Learning* and *Classroom Computer News* most highly, but all of the following have been useful.

Classroom Computer News. P.O. Box 266, Cambridge, MA 02138, 617-923-8595.

Electronic Learning. Scholastic, Inc. 902 Sylvan Ave., Englewood Cliffs, NJ 07632.

Educational Computer Magazine. P.O. Box 535, Cupertino, CA 95015.

Electronic Education. Suite 220, 1311 Executive Center Drive, Tallahassee, FL 32301.

The Computing Teacher. Dept. of Computer and Information Science. University of Oregon, Eugene, Oregon 97403. ©

Book Review:

Understanding Computer Science

Louis F. Sander
Pittsburgh

I picked up *Understanding Computer Science* while waiting in line at Radio Shack, and I couldn't put it down. Having brought it home and read it from cover to cover, I'm convinced that it's a little-known treasure that belongs on every computer owner's bookshelf. This book is an inexpensive overview of all the major areas of computing, written at a level which any interested person can understand, and the information it contains is perfectly packaged for home computerists. It is an ideal guide to self-paced individual learning of computer subjects.

The book was designed to build understanding step-by-step, and it succeeds. The first chapter reviews the development of computer hardware and software from the abacus to the microcomputer, and provides a non-technical background on the state of solid state technology. It familiarizes the reader with elementary terms and concepts, and charts the place of the personal computer in the world of the 1980's. Like all the subsequent chapters, this one concludes with a multiple-choice self-evaluation quiz. The questions are straightforward and there is an answer key in the back of the book.

The World Beneath The Keyboard

The second chapter, "Computer Architecture and Hardware," builds on the foundation laid by the first, providing a demystifying tour-de-force of busses, logic gates, binary numbers, and the other structures of the world beneath the keyboard. The secrets of tape and disk recording are also revealed here, and all of this is done with a minimum of jargon and a maximum of clarity. This chapter alone is worth the price of the book, but there's lots more to follow!

The "Programming" chapter starts with simple definitions, continues with examples of flowcharting and program design, and ends with a discussion of coding, translating, and debugging.

The "Languages" chapter gives an overview of BASIC – examples of some of the statements, a sample program, rules for evaluating expressions, etc. There's nothing remarkable here for the **COMPUTE!** reader who's past the beginner stage, but it's a good review of what BASIC is all about. What *is* remarkable is the rest of the chapter. It gives us similar, clear overviews of PASCAL, FORTRAN, COBOL, and PL/1. Did you ever wonder what these languages look like? Here's an excellent place to find out.

There are other chapters on Operating Systems, Resource Management (in large systems), Data Structures, Language Translators, and Systems Analysis. All are well-written, all are informative and useful. A view of the future of computers, a four-page glossary, and an excellent index complete the book.

Understanding Computer Science is a unique and valuable one-stop source of general information on computers. In a field where detail can overwhelm the beginner (and almost always does), this book offers a clear and understandable view of the big picture. In a field where even the expert can get lost in details, this $3 volume is an excellent overview.

Understanding Computer Science, by Roger S. Walker. Developed and published by Texas Instruments Learning Center. Distributed by Radio Shack (#62-1383). 267 pages, $2.95. ©

Review:

Flex File: A PET Data Base Manager

Donald C. Johnson
Renton, WA

Recent involvement with automatic test equipment has introduced me to a programming technique referred to as *application programming.* Application programming is very user oriented in that the operator initially defines his/her program needs by answering a series of computer screen prompts. Based upon the answers provided, the main program writes a second program that is tailored specifically to the user's needs. This affords a user the opportunity to design or truly create a custom program. *Flex-File* is such an application program.

Immediately upon receiving a program, I generally load the disk and run it. I tend to leave the instruction manual for assistance during periods of confusion, or as required reading during my next bout with the flu. However, *Flex-File* documentation, consisting of 36, 8½ x 11 pages of operational instructions, is actually a series of mini-lessons to be used with the sample inventory file provided on the disk.

Features

Filed data is handled via a keyed random access method. This allows for any and, if desired, all data fields to be designated as key fields.

The record size is limited to 250 characters; however, the number of records you can generate is limited only by the amount of storage space available on your disk system. Defined field lengths may be varied from record to record. Using a standard mailing list format, 1,000 records can be handled. Almost three times as much data can be handled if you are fortunate enough to own an 8050 drive.

All operations are menu driven. General commands such as add, delete, change, transfer, key (change), and exit perform as expected. An interesting feature incorporated in the file maintenance routine is the ability to scroll forward or backwards through the file.

Two available commands not normally found on many data base programs are BROWSE and USER. Exercising the BROWSE command provides a quick review or comparison of any two fields of data, i.e., the primary keyed field and the selected browse field. This function allows a cursory review of selected data fields without commanding total record recall. From a programmer's (or hacker's) standpoint, the program is very friendly. Menu space has been provided for a user's routine. There are no protection securities which prevent modifications. In fact, user modifications are encouraged. A special section of the documentation provides important information to allow program changes if desired. Additionally, a complete list of program variables is provided.

The author has provided two very powerful operations that are normally optional modules with other data bases. A full function math/calculating capability and a field-selectable, integrated mailing list routine enhance the flexibility of the system. Any column of numeric data may be mathematically processed with other columns. As a result, you can print averages, sub-totals, totals, ratios, etc. In fact, you can manipulate numeric data with +, -, x, /, % and log/trig functions.

Another handy feature of *Flex-File* is its random access to sequential file conversion ability. This allows the user to transfer in or out formatted data to other programs. For example, *Flex-File* compiled data may be transferred to a word processor for form letter generation, or inventory data can generate re-order forms using your word processor.

Considering the tremendous flexibility of *Flex-File*, the documentation could have been expanded to include other examples of interfacing with word processors, program listings, or suggestions for various professional applications. This is really a small criticism of such a powerful program though.

Michael Riley has developed an excellent, professional data base program that contains many features – too many to cover in a short review. Suffice it to say that you can spend much more for a high quality data base, but it's doubtful you will find the flexibility of an application program such as *Flex-File.*

Flex-File
AB Computers
252 Bethlehem Pike
Colmar, PA 18915
$80.00

©

Reviews:

Moonbase Io And Space Ace For Atari

Tom R. Halfhill
Features Editor

Moonbase Io is an arcade-style space game – actually a combination of three games – inspired by the recent flights of Voyagers 1 and 2 to Jupiter. The "Io" (pronounced "eye-oh") in the game's title refers to one of the four major moons of the solar system's largest planet. Since Jupiter was discovered to be a huge ball of inhospitable liquids and gases, unsuitable for landings, this game uses the moons Io, Europa, and Ganymede as moonbases for your spacecraft.

This choice of bases, however, has upset the local bug-eyed populace. (Can you blame them? They were probably never even consulted for the environmental impact statement.) Determined to send you back where you came from, or destroy you in the process, the aliens launch swarms of ships to battle your probe.

Audio Effects

Moonbase Io is a one-player game available on disk or cassette and requires 16K RAM and joystick. It also requires fast coordination and is suitable for all but the youngest children.

One extraordinary feature of this game is its use of the Atari's capability to synchronize the screen with a soundtrack on cassette tape. Both the cassette and disk versions include a soundtrack tape that is snapped into the Atari Program Recorder when the game starts. The PLAY button is left on, and the program starts and stops the tape at appropriate moments. The first time you play *Moonbase Io*, you get a long briefing on your mission from your superiors on Earth. Meanwhile, the screen displays a control room, a busy robot, and a window through which moving stars are visible. After receiving all your instructions (including a warning that the aliens are suspected to be unfriendly), you hear a countdown and blast-off, complete with sound effects and Cape Canaveral background chatter.

Because this initial briefing takes several minutes, there's a shortened version on Side B of the cassette which starts at the countdown for subsequent games. At various points during the game, during breaks between the three levels, the tape comes on again with additional messages. And if you make it past all the levels, you even get a congratulatory speech from the President of the Earth Federation.

This is the first time I've seen a soundtrack with a game program, although they are often used with Atari educational software. It's a good concept, and a laudable attempt to take advantage of all the machine's capabilities. However, some players, caught up in the intensity of *Moonbase Io*, may be frustrated by the procedure necessary to restart everything after a sudden explosion prematurely ends the game.

Besides the soundtrack, programmer John Konopa has used other Atari features to good advantage, too. The machine language program is fast and incorporates player/missile graphics, redefined characters, fine scrolling both vertically and horizontally, and well-executed sound effects from the Atari's four voices.

Moonbase Io is really three games in one, with two distinct phases. To secure each of the three moonbases, the player must pass a docking phase – impeded by flocks of aliens – and, if successful, transport to the surface of each moon for a phase which involves defending the base against more formations of fast-moving enemies. In addition, there are seven skill levels, from Novice to Galactic Wizard, with variations in scoring and availability of reserve ships. Some "hidden features" are also promised, although I never managed to survive long enough to experience them. Overall, *Moonbase Io* is a challenging arcade-style space game.

Space Ace

Like *Moonbase Io*, *Space Ace* is another fast-action, arcade-style, space warfare game for one player. It comes on tape or disk, requires 16K, a joystick, and the reaction instincts of a pro hockey goalie. It's suitable for all but very young children and is aimed at the video game addicts among us.

Space Ace starts you off at the bottom of the screen with a spaceship maneuverable in all eight directions with the joystick. The movement is smooth – player/missile graphics is obviously used here – and pressing the trigger button fires up to two shots at a time.

Programmer Greg Young makes good use of fine vertical scrolling to move a star field down the screen, creating the illusion of forward travel. This travel is not unobstructed, however, since your path is blocked by increasing numbers of asteroids (redefined characters). Blasting a path through the rocks isn't too hard, at least at the outset, but a the SOFTWARE connection

FOR THE MOST EXCITING VIDEO GAMES AT THE LOWEST PRICES

We have one of the largest selections of software available for your home computer at the lowest prices. You will find all of the top games and office management software in our catalog at from 20% to 30% below retail. We also have special offers for user groups.

TRY THESE GET-ACQUAINTED SPECIALS

FOR YOUR ATARI®

	Retail	Our Price
RASTER BLASTER	$29.95	**$20.00**
DR. GOODCODE'S CAVERN	$29.95	**$21.00**
APPLE PANIC	$29.95	**$18.00**
MATCH RACER (Disk or Cassette)	$29.95	**$18.00**
TRACK ATTACK	$29.95	**$18.00**
STAR BLAZER	$31.95	**$21.00**
DAVID'S MIDNIGHT MAGIC	$34.95	**$22.00**
PATHFINDER	$34.95	**$23.00**
GHOST HUNTER	$29.95	**$21.00**

ANY 3 FOR JUST $50.00

CARTRIDGE SPECIALS FOR YOUR ATARI 400/800

	Retail	Our Price
PAC MAN	$44.95	**$34.95**
EMBARGO	$49.95	**$34.95**
FIREBIRD	$49.95	**$34.95**

FOR YOUR APPLE®

	Retail	Our Price
FIREBIRD	$29.95	**$17.50**
RASTER BLASTER	$29.95	**$20.00**
NEPTUNE	$29.95	**$20.00**
LAZER SILK	$29.95	**$20.00**
STAR BLAZER	$31.95	**$21.00**
CHOP LIFTER	$34.95	**$23.00**
ZENITH	$34.95	**$23.00**

ANY 3 FOR JUST $50.00

DISKETTE SPECIAL

FREE PLASTIC LIBRARY CASE WITH PURCHASE OF EVERY BOX OF 10

$24.95

Personally labeled for THE SOFTWARE CONNECTION by one of the most respected producers of magnetic media. Each diskette is single-sided and certified double density at 40 tracks. To insure extended media life, each diskette is manufactured with a reinforced hub-hole.

10 Boxes or more: $22.50/box

Product #1540BK-Soft Sector

WE CARRY COMPLETE LINES FROM THE FOLLOWING COMPANIES:

ADVENTURE INTERNATIONAL • ARCADE PLUS • ARTSCI • AUTOMATED SIMULATIONS • ATARI • AVALON HILL GAME COMPANY • BRODERBUND SOFTWARE • BUDGECO • CALIFORNIA PACIFIC • COMPUTER MAGIC LTD • CONTINENTAL SOFTWARE • DATAMOST • DATASOFT • DELTA SOFTWARE • EDU WARE • GEBELLI SOFTWARE • HAYDEN SOFTWARE • HIGHLANDS COMPUTER SERVICES • HOWARD SOFTWARE • INNOVATIVE DESIGN • IUS • K-BYTE • LJK • MICRO LAB • MICRO PRO INTERNATIONAL • MICROSOFT • MUSE • ON-LINE SYSTEMS • QUALITY SOFTWARE • SIRIUS SOFTWARE • SOFTAPE • SPECTRUM COMPUTERS • STRATEGIC SIMULATIONS • SYNERGISTIC • UNITED SOFTWARE OF AMERICA • VERSA COMPUTING • VOYAGER SOFTWARE

MAIL ORDERS: For fast delivery, send certified check, money orders, or Visa or MasterCard number and expiration date, for total purchase price plus 1% or $2 minimum for postage and handling. Add $5 for shipment outside the continental U.S. California Residents add 6% sales tax.

COD: and Chargecard orders may call **(916) 925-2666.** Subject to stock on hand. Prices subject to change.

Catalog free with any order or send $2 postage and handling and your computer type to

the SOFTWARE connection
5133 Vista Del Oro Way Fair Oaks, CA 95628

number of other twists are thrown in as well.

First, there are the "space vortexes." These are whirling player/missile shapes which look something like runaway blades from an old-style push lawnmower. The vortexes drop down the screen at you – they seem to know where your ship is – and cannot be destroyed, only evaded. Dodging them can be hazardous, though, since in your haste you often collide with stray asteroids.

Then there are the "Silurian space bombs," small objects which traverse the top of the screen until crossing above your ship. These things definitely know where you are. With a terrifying shriek, they plunge downward faster than you can see. Unless you dodge at just the right moment, or get off a lucky shot, you've had it.

As if these hazards weren't enough, occasionally a "space-mine field" stretching all the way across the screen descends upon your craft. You either blast a path through the mines or get trapped against the bottom of your TV.

And finally, at odd intervals, a "master Silurian warship" crosses the top of the screen. This warship unleashes a hail of missiles with deadly accuracy and can only be destroyed by hitting its central "atomic drive core" – not a large target.

Despite all this activity, *Space Ace* is very fast and smooth and makes extensive use of the Atari graphics and sound effects. There are multiple difficulty levels, and you can choose to receive a bonus ship every 1,000 points. I recommend this option for survival, especially since the scrolling asteroid clusters seem to speed up at 1,000-point intervals. Also, the "Training" level is fairly easy and suitable for children.

There is no pause between the destruction of one of your ships and its replacement with another. This happens so quickly that sometimes the same asteroid cluster which knocked you off the first time gets your new ship as well. However, it is possible to freeze all action on the screen at any time by pressing any key – if you can remember to do it in the heat of combat.

Fans of arcade-style games should find *Space Ace* both well-executed and challenging.

Space Ace
London Software
374 Wildwood Avenue
Piedmont, CA 94611
$29.95 Disk/Tape

Moonbase Io
Program Design, Inc.
11 Idar Court
Greenwich, CT 06830
$29.95 Disk/Tape

©

Review:

VIC Invaders

Harvey B. Herman
Associate Editor

With three VIC versions of the venerable computer game "invaders," how can an invader aficionado (and there are still lots of them around) make the proper choice? It seemed to me that it would be helpful to have a summary of the characteristics of each of these three programs. The prospective owner then would have more of a chance to make an informed decision.

I do have a problem. The invaders seem to get me before any of the advanced features come into play. Luckily, a friend of my son is a super player and was able to give the programs a good test.

Here is his summary of the relevant features of each program:

	VICVADERS	ALIEN BLITZ	VIC AVENGER
Medium	tape	tape	cartridge
Colors	various	blue/white	various
Skill Levels	1	10	1
Number of Aliens	40	40	50
Points per Alien	5	10-30	10-30
Mystery Ship Points	10-80	100	50-300
Base/Ship Size	large	small	large
Bases	3	3	3
Points for Bonus	1000	1000	1500
Continuous Fire	no	no	no
Instructions	no	no/sample game	yes/sample game
Adjustable Display	no	no	yes
Graphics	good	good	good
Price	$9.95	$24.95	$29.95
Vendor	Skyles Electric	Tensor Technology	Commodore

©

This month we conclude our overview of one of Commodore's new computers with game programming examples and the latest news on the software and peripherals soon to be available for the Commodore 64.

Sprite Graphics And Sound Synthesis On The Commodore 64

Tom R. Halfhill
Features Editor

Sometimes those shiny new shoes you buy wind up feeling like concrete blocks after a few hours' wear. Or those fancy stereo headphones turn into a vise by the end of one album.

My first few hours with a prototype Commodore 64, however, led me to believe it will be a comfortable computer. The manual is coherent, the keyboard is friendly, the full-screen editing is fun. Now that manufacturers are not merely selling computers to experienced users, but are also pushing them as home appliances to the wary masses, nothing less will do anymore. Someone with no previous experience on Commodores (or new to computing entirely) will quickly feel at home on the 64.

Not that the Commodore 64 is easy to master, or that its *User's Guide* is the ultimate reference book. Quite the contrary. My first real contact with the 64 at Commodore International's headquarters near Philadelphia reveals it to be a computer of formidable complexity – and flexibility – that will keep magazine writers and aspiring book authors busy for a long time explaining its inner workings. But while mastery may be elusive, familiarity is not. It's quite possible, at a first sitting, to be manipulating sprites and shaping envelopes.

Color Facilities

Sprites? Envelopes? Relax. The Commodore 64's advanced features take a little explaining, but are easy to grasp.

Let's start at the beginning. First, the 64 should be available in small quantities at Commodore dealers by the time you're reading this; Commodore predicts production of 15,000 to 20,000 units a month by late 1982. The $595 computer comes in the same compact plastic case and has the same keyboard as the $299 VIC-20. The 64 plugs into any color (or black-and-white) television via its built-in RF modulator, and upon power-up tells you that 38,911 bytes of Random Access Memory (RAM) are available (52K for pure machine language programming).

The screen is light blue surrounded by a darker blue border, and characters appear as dark blue. The characters are a bit hard to see against the background, and most people change them to white by holding down the control key (marked "CTRL") and typing "1". You can change the characters to any of 16 colors this way with the CTRL and numeric keys. And by inserting color control numbers (given in the manual) into two memory locations with the BASIC command POKE, you can also change the screen border and background to any of these 16 hues. It's the cheapest way of getting a green-screen you'll ever see.

The manual, which doubles as an elementary self-teaching guide for BASIC programming, steps the first-time user through these kinds of basic concepts with readable explanations and easy-to-follow examples. Early on, it shows how to load and save programs with disks and cassettes, so people who are not interested in programming are spared wading through swamps of murky text.

The computer has other features in line with this philosophy, such as a two-key "warm start" which clears everything except the program in RAM, and supposedly makes the computer uncrashable. (I say "supposedly" because I managed to rise to the challenge by inadvertently – and irretrievably – crashing the 64 within an hour, mystifying the Commodore people present. They suggested it might be a bug in the prototype, but not production, models. Oh well, nobody's perfect.)

Anyway, for the programmers among us, rest assured the 64 will be a hacker's machine too. Impressive screen displays can be created with 16 simultaneous colors and the high-resolution 320- by 200-dot graphics mode. The screen editing ranks with the best, and nearly all BASIC keywords and commands can be abbreviated merely by shifting the second letter as on other Commodore products.

Sprite Graphics

But the real power of the Commodore 64 lies in its more esoteric features: sprite animation and sound synthesis.

"Sprites" also are known as "MOBs" (for Movable Object Blocks) and "player-missiles" (in Atari parlance). Basically, a sprite is a graphics block corresponding to a block of memory which the programmer can sculpt into any shape and move around on the screen. This movement can be extremely smooth and fast, and is independent of other sprites or of objects drawn on the screen the usual ways.

Sprites can be made to pass in front of or behind other sprites and screen shapes, simulating three dimensions. The computer also detects collisions between sprites and other objects, so programmers can trigger explosions or other effects (sprites are usually used in games). Until now, this very powerful feature was available only on Atari and Texas Instruments computers.

Sprites are relatively easy to program. Within an hour after sitting down with the manual's 11-page chapter on sprites, an experienced BASIC programmer can be drawing and manipulating them without much trouble. And yes, BASIC can move at least a few sprites fast enough to make home-grown arcade-style games practical in BASIC.

Different computers implement sprites in different ways. The Commodore's sprites are blocks 24 pixels (screen dots) wide by 21 pixels tall. They can be any of 16 colors, and up to three colors can be combined in one sprite. Up to eight sprites can appear at once on a TV's horizontal scan line; much larger numbers are possible with a "raster-scan interrupt," an advanced technique the manual avoids. The sprites can be doubled or quadrupled in both dimensions with a POKE statement.

POKEs, in fact, are required for just about everything involving sprites on the 64. BASIC has no special keywords for these functions. However, Commodore is promising a plug-in cartridge soon which will add extended commands for sprites, graphics, and sound.

In the meantime, start POKEing numbers of memory locations into your brain until you can rattle them off like a baseball fan quoting batting averages. One very important location is the starting address of the special chip which controls sprites on the Commodore, 53248. Many other addresses are calculated from this location, so one approach is to store 53248 in a variable and handle everything else as "offsets."

To draw a sprite on the screen, it must first be "drawn" in memory. This is done by storing numbers corresponding to the image in DATA statements, and then POKEing them into memory with a FOR-NEXT loop (just like storing a machine language routine from BASIC). To determine these image numbers, you mark off a 24 by 21 grid on graph paper (to represent the 24- by 21-pixel sprite) and color the squares to draw your image. We won't cover the details here, but a colored square is an "on" bit, a blank square an "off" bit. The binary numbers that result are converted to decimal numbers and stored in the DATA statements.

A few more things are required to display and move sprites on the 64. A memory location offset from 53248 is POKEd to turn on the sprite, a pointer is set to tell the computer where to find the image data, and two more offsets from 53248 are POKEd to change the sprite's vertical and horizontal coordinates. These coordinates, by the way, extend beyond the visible screen. These steps are necessary for each sprite.

Here's a short example program, adapted from the *Commodore 64 User's Guide:*

```
 10 SPRITEBASE=53248
 20 POKE SPRITEBASE+21,4:REM Enable sprite #2.
 30 POKE 2042,13 (Set sprite pointer to where sprite
    data should be read from; the address varies for
    each sprite.)
 40 FOR X=0 TO 62:READ SHAPE:POKE 832+X,
    SHAPE:NEXT (This loop READs the 63 DATA
    bytes needed to draw a sprite.)
 50 FOR LOOP=0 TO 200
 60 POKE SPRITEBASE+4,LOOP (Update horizontal
    coordinates of sprite's screen location.)
 70 POKE SPRITEBASE+5,LOOP (Update vertical
    coordinates.)
 80 NEXT LOOP
 90 GOTO 50
    This data draws the sprite:
100 DATA 0,127,0,1,255,192,3,255,224,3,231,224
110 DATA 7,217,240,7,223,240,7,217,240,3,231,224
120 DATA 3,255,224,3,255,224,2,255,160,1,127,64
130 DATA 1,62,64,0,15,128,0,156,128,0,73,0,0,73,0
140 DATA 0,62,0,0,62,0,0,62,0,0,28,0
```

This program creates a hot-air balloon which drifts diagonally down the screen from the upper-left corner, disappears at the bottom, and then starts again from the top. As you see, long variable names are possible, though only the first two letters are significant; it is not necessary to protect areas of memory to display sprites, unlike with other computers; by changing the pointer at line 30 with a quick POKE, it is possible to instantly redraw sprites with alternate shapes previously stored in memory (to fit a sprite's shape to the direction it's traveling, for example); smooth, rapid movement in any direction is possible with simple POKEs into the memory locations which store the horizontal and vertical coordinates.

You'll notice the program is not really very complicated. Remember, though, we moved just one, single-colored sprite with a simple loop; manipulating many multi-colored sprites with joysticks, "firing" shots (made of similar sprites), checking for collisions, and synchronizing sound effects would add considerably to the program's complexity – and slow it down. Fast-action games involving more than a couple of sprites probably will need some help from machine language subroutines. Still, the point is that the basics of sprite animation are quite accessible from BASIC.

Advanced Sound Synthesis

The same is true of the Commodore 64's amazing sound capabilities. When it comes to sound, no other personal computer on the market can hold a diode to the 64.

First, the 64 replaces the common, simple tone generator with a true three-voice sound synthesizer of musical instrument quality. Furthermore, the programmer has control over a great number of parameters: volume, frequency, waveforms, attack/decay, sustain/release, and filtering. This allows you to custom-design each note's "envelope," the shape of its sound wave, and its tone color.

As with the sprites, however, this requires various POKEs, at least until the extended-command cartridge becomes available. Here's an example, borrowed from the *User's Guide*, of how to create one note (a middle C):

```
10 POKE 54296,15 (Loudest volume setting.)
20 POKE 54277,190 (A number from 0 to 255 sets the
   attack/decay slope to define how fast a note rises to
   and falls from its peak volume.)
30 POKE 54278,248 (The opposite of attack/decay, this
   number sets the sustain/release slope to prolong the
   note at a certain volume before releasing it.)
40 POKE 54273,17:POKE 54272,37 (This creates a
   middle C. Two POKEs, for both high and low
   frequency, are required for each note because of the
   16-bit frequency resolution.)
50 POKE 54276,17 (Choose one of four waveforms; in
   this case, "triangle.")
60 FOR DUR = 1 TO 250:NEXT (A timing loop sets the
   note's duration; here, a quarter note.)
70 POKE 54276,0:POKE 54277,0:POKE 54278,0 (Turn
   off the waveform control, attack/decay, and sustain/
   release settings.)
```

Whew! Can you imagine programming a symphony?

Actually, it's not so bad. Most of these parameters are set at the beginning of the program and left alone. Values for notes can be stored in DATA statements and summoned easily and quickly with a READ loop. And, as with sprite programming, important memory locations can be stored in variables to save typing.

Still, it may relieve you to know that Commodore plans to sell a plug-in piano-type keyboard for the 64. If Commodore doesn't, someone else will. With the appropriate software, this could make the 64 worth its cost as a musical instrument alone, aside from its other capabilities.

In an appendix of the *User's Guide* (which, incidentally, is crammed with invaluable charts, tables, and pin-maps for the 64) is a listing for a sound demo program that hints at the possibilities. The program converts a row of keyboard keys into a simple organ. Pressing the SPACE bar switches from solo to polyphonic sound. The four programmable function keys select among four octaves and, when shifted, among four waveforms. When the computer is plugged into a decent sound system (easily done), you'll swear you're in the same room with either an organ, electric piano, or harpsichord. It's that good.

And, needless to say, the same capabilities to create beautiful music also give you unprecedented control over sound effects. A gunshot sound can be built like this:

```
10 VOICE1 = 54296: WAVEFORM = 54276:
   ATTACKDECAY = 54277: HIFREQ = 54273:
   LOFREQ = 54272: NOISEWAVE = 129
20 FOR LOUDNESS = 15 TO 0 STEP -1
30 POKE VOICE1,LOUDNESS
40 POKE WAVEFORM,NOISEWAVE
50 POKE ATTACKDECAY,15
60 POKE HIFREQ,40:POKE LOFREQ,200
70 NEXT
80 POKE WAVEFORM,0:POKE ATTACKDECAY,0
```

Even echo effects are possible by juggling these settings.

Hardware And Software Support

Finally, for those who are interested in buying packaged software as well as (or instead of) writing their own, Commodore is promising a flood of support with the computer's introduction, or soon after. In a first for Commodore, not all of it is being produced in-house. Some hardware goodies are under development, too. All of it is supposed to be for sale by January 1983. Here's a rundown:

- We've already mentioned that most VIC-20 peripherals will work with the 64 with little or no modification. The VIC 1540 disk drive will require a ROM chip change. Already in production is a 1541 drive to replace the 1540 to be compatible with both computers.
- Besides the $109 plug-in VICmodem, another direct-connect, plug-in modem will be made for the 64. This will feature auto-dial and auto-answer and, like the VICmodem, will come with terminal software on tape and a subscription to CompuServe. It will cost less than $200. Two more sophisticated terminal programs will be available on cartridge and disk.
- A PET Emulator will be targeted especially toward schools. Commodore thinks a big selling point of the 64, besides its price, will be the claim that its emulator can run 90 percent of the programs now used by schools on their popular PETs.
- Programmers will be interested in the languages nearing completion for the 64: an assembler for machine language programming, LOGO, and even a BASIC compiler.
- A word processor modeled after *WordPro* is in prototype stages. To accommodate the 64's 40-column screen, the version I saw allowed horizontal scrolling of more than 100 columns. The price will be much less than *WordPro*. A less powerful, and less expensive, word processor for home use will be sold on cassette.
- A whole family of business software is on the way, all with the prefix "Easy" – *EasyCalc*, *EasyGraph*, and so on. Commodore claims these will be improvements on the current mass of electronic ledger-type programs.

● The plug-in CP/M cartridge promised at recent trade shows and a networking cartridge aimed at schools are also supposed to be well on their way from the drawing boards to the marketplace.

● A new joystick-type controller is in prototype stages and offers unusual possibilities. Sorry, but I had to swear secrecy on this one before they'd even show it to me. It's not a track-ball. But I can say it will work on the VIC-20, too, and probably even on the Ataris.

● Then, of course, there are the games. We've seen some interesting ones, but Commodore hints that even more fantastic stuff using sprites in simulated 3-D is under development. Someone also is working on a music composition utility.

All in all, the Commodore 64 should have some solid hardware and software support behind it much sooner than most new computers, since Commodore seems to be regarding the support as more than just an afterthought. This is a welcome change from the way things were done in the past (not just by Commodore), as even one Commodore official admits. "I think it's safe to say that Commodore has learned from everyone's past mistakes," he says, "and is in a position to capitalize on the lessons." ©

For Upgrade and 4.0 Commodore BASIC, this is a fast, effective sort. It will operate on DATA statements or arrays and rather than exchanging strings, it manipulates their pointers. Records can be of any length. It sorts 200 records in 11 seconds, 800 records in about three minutes. That's speedy. Just type in the BASIC program and it will locate and protect the machine language and adjust it to your ROM version. The program contains some example sorts to get you started.

PET Pointer Sort

David Lummis
Stoney Creek, Ontario

Most previously published sorting subroutines have been written in BASIC. This meant that if you had a large list to sort it would take quite a while to finish.

One day, after I had waited 15 minutes for the computer to finish sorting a list of some 300 names, I decided that I'd had enough.

A friend of mine had suggested writing a machine language program to do the job much faster. This program is the result.

A Bubble With A Twist

I have used a bubble sort with a twist. Instead of switching the data (strings) itself (since not all strings are the same length), I simply switch the pointers that point to where the string is sitting in memory.

In order for you to understand this better, it might be best to explain how variable arrays are stored in memory.

When you DIMension a variable in BASIC, the computer sets aside some room in memory for a table. This table includes all the characteristics of the variable dimensioned. These would include the variable name, whether it's a numeric or string variable, the size of the array, and a few other items.

In order to examine this table we must first find out where it is located in memory. Locations $2C and $2D in hexadecimal (44 and 45 decimal) hold the address where the tables for *all* dimensioned variables start. Locations $2E and $2F hold the address where the last table ends.

As an example, let's assume that location $2C contains the value #$90 in hex, and that location $2D contains the value #$04 in hex. This means that the first table would start at memory location $0490. Here is a typical table created by the statement DIM A$(2): [*type SYS 4 then, in the monitor, type M 0490 0498 to see the hex dump*].

```
0490 41 80 10 00 01 00 03 00
0498 00 00 03 FB 3F 02 F7 3F
```

Figure 1.

The first seven numbers in the table represent what is called the header of the table. The first two characters, 41 and 80, are the name of the variable for that table. In this case 41 hex is the ASCII value of the letter A, the first letter of our variable name. Since we had no second letter in the variable name, the computer placed an 80 where the second letter would normally have gone.

If we had said DIM AA$(2) then the header would have looked like this:

```
0490 41 C1 10 00 01 00 03 00
```

Instead of placing the ASCII value of our second A in the second spot in the header, it placed the ASCII value of a shifted A in its place. This shifted character allows the computer to distinguish between a string variable and a numeric variable. The tables we'll be looking at will deal only with string variables.

The third and fourth bytes of the header are the size of the table, stored in low/high order.

The fifth byte is the number of dimensions in the array. In other words, it would be an 01 in the case of a one dimensional array such as A$(2), an 02 in the case of a two dimensional array such as A$(3,6), and so on.

The sixth and seventh bytes are the number of elements in the array, stored in high/low order.

Following the header there are three bytes reserved for each element in the array. The first of the three bytes is the length of the string. The two bytes following the length is a pointer, stored in low/high order, that points to where the string is stored in memory.

Now let's look back at Figure 1, which can be interpreted as follows:

Bytes	Explanation
1-2	The variable name is A$.
3-4	The length of the table is $0010 bytes long.
5	Indicates that it is a one dimensional array.
6-7	There are $0003 elements in the table.
8-10	Byte 8 (a zero) shows there is no string.
11-13	These show that the second element in the table is a string of three characters starting at $3FFB in memory.
14-16	These show that the third element in the table is a string of two characters starting at $3FF7.

Now let's take a look at the strings themselves as they are stored in memory.

Since A$(0) was a null string, we'll start with A$(1). A$(1) is a string three characters long starting at $3FFB.

```
3FFB 41 42 43 9A 04 40 40 40
```

The first three characters make up the string ABC. The two bytes following each string are a pointer that points back to the table where the length of the string is. Note: only BASIC 4.0 has this pointer following each string.

For the Upgrade ROMs, the string A$(1) would look like this:

```
3FFB 41 42 43 53 45 40 40 40
```

Again, the first three bytes make up the string ABC, which is what A$(1) is equal to. The difference in the Upgrade ROMs is that there is no pointer after each string. The strings are simply stored one after another with nothing in between them. The 53 45 following the string ABC (in Upgrade BASIC) is simply another string.

The Sorting Method

The program compares two strings at a time as in a bubble sort. If the first string is greater than the second one, then the computer switches the pointers that point to where each string is stored in memory. It also has to switch the lengths of each string that it switches. If the program is RUN on a computer with 4.0 ROMs then the pointers after each string, that point back to the table where the length of the string is, will also be switched. When the program returns to BASIC, the array will be sorted in ascending order.

The program is a BASIC loader that places the machine language routine at the top of memory. Line 75 checks to see if it is running on a computer with Upgrade ROMs and, if it is, makes the appropriate corrections to the machine language program. The computer then protects the machine language from being written over.

The variable S is the starting location of the machine language program and is the SYS address which will sort a list. Before sorting a list, however, you must POKE the first two letters of the variable name into memory locations 679 and 680. POKE the ASCII value of the first letter into 679 and the ASCII value of the second *plus* 128. If there is no second letter in the name, simply POKE 680, 128.

For example, to POKE the variable name NA$ into memory you would enter:

```
POKE 679,ASC("N"): POKE 680,ASC("A") + 128
```

For the name N$ it would be:

```
POKE 679,ASC("N") : POKE 680,128
```

The second part of the program is an example of how the sort works in a program. It starts off by having you enter ten strings which the computer then sorts. The key line is line 540, which POKEs the variable name into memory before executing SYS S.

Using DATA Statements

Lines 600-680 show how strings read from DATA statements can be sorted. When data is READ into a string, the pointer in the array table points to the spot in the DATA statement where the string is located. In order to sort strings READ in from DATA statements, the strings must first be placed into upper memory. This is done by adding a null string (two quotes with nothing between them) to the string. Line 630 in the program demonstrates how this is done. If this is not done before sorting your list, then the computer may crash or the strings will contain unexpected characters.

Lines 60-410 are the important lines if you wish to include the sort in a program. They should be executed at the very beginning of a program, before any strings are given a value, in order to get the sort stored at the top of memory right away.

For the computer to sort a complete list of strings, there must not be any empty (null) strings among the ones to be sorted. When the computer sees an empty string within a group of strings containing characters, it assumes that it has reached the end of the list to be sorted and does not sort anything past this point. However, there may be as many null strings as you want *before* or *after* the strings you want sorted.

```
30 REM   THIS ROUTINE SORTS THE STRING
31 REM ARRAY OF YOUR CHOICE IN
32 REM ASCENDING ORDER (A-Z).
38 REM
39 REM   TO ACTIVATE THE ROUTINE IN
40 REM A PROGRAM, TYPE   'SYS S'
41 REM WHERE 'S' IS THE DECIMAL
42 REM STARTING ADDRESS OF THE ROUTINE
43 REM
44 REM   BEFORE ACTIVATING THE ROUTINE
45 REM BE SURE TO POKE THE VARIABLE
46 REM NAME INTO MEMORY LOCATIONS
47 REM 679 AND 680 !
48 REM POKE 679,X WHERE X IS THE ASCII
49 REM VALUE OF THE FIRST LETTER.
50 REM POKE 680,Y WHERE Y IS THE ASCII
51 REM VALUE OF THE SECOND LETTER+128
52 REM   **IMPORTANT***IF THERE IS NO
53 REM SECOND LETTER IN THE NAME THEN
54 REM POKE 680,128.
55 REM
60 REM CALCULATE STARTING(SYS) ADDRESS
63 S=PEEK(52)+PEEK(53)*256-324
64 PRINT"STARTING (SYS) ADDRESS IS"S
70 FORI=0TO324:READA:POKES+I,A:Q=Q+A:NEXT
71 IFQ<>41326THENPRINT"ERROR IN DATA STATEMEN
   TS":STOP
75 IFPEEK(50003)=1THENFORJ=S+229TOS+282:POKEJ
   ,234:NEXTJ
80 REM *PROTECT PROGRAM FROM STRINGS
81 M=INT((S)/256):N=S-M*256:POKE52,N:POKE53,M
   :POKE48,N:POKE49,M
100 DATA 165, 45, 197, 47, 208, 7, 165, 44, 19
   7, 46, 208
110 DATA 1, 96, 165, 44, 133, 1, 165, 45, 133,
   2
120 DATA 160, 0, 177, 1, 205, 167, 2, 208, 8,
```

Skyles Electric Works

Epson-PET/CBM Graphic ROM Pack

For PET/CBM Owners Who Want:

Complete Program Listing Printouts
Complete Screen Graphic Printouts
Graphic Printouts From Programs

on your Epson Printer

Order the Skyles Electric Works EPSON-PET GRAPHIC ROM Package. The ROM when installed in an EPSON MX80, MX80FT, or MX100 with Graftrax Plus printer will reproduce most of the PET/CBM graphics characters. Most importantly when using the accompanying high speed machine language program, the Epson-PET Graphic ROM pack gives a complete program listing with all screen controls shown (cursor, home, clear, etc.). This high speed machine language program for program listing, screen image printout (screen dumps) and BASIC program controlled printing (i.e. PRINT) automatically translates the PET-ASCII characters to the EPSON-GRAPHIC ROM characters. A BASIC sample program and PRINT subroutine, that may be incorporated into any existing BASIC program, completes this "complete solution" package.

EPSON MODEL	ROM MODEL
MX70	Not Available
MX80 (serial no. to 359999)	EPG80 (EPG82, 3 ROM Version)
MX80 (serial no. after 360000)	EPG81 (EPG83, 3 ROM Version)
MX80FT	EPG8F
MX80 Graftrax	Not Available
MX80 Graftrax Plus	EPG8G+
MX80FT Graftrax Plus	EPG8G+
MX100	Not Available
MX100 Graftrax Plus	EPG10G+

The Epson-PET Graphics ROM Pack has been designed to furnish you with PET/CBM graphics printing in the easiest way possible. This is done by furnishing a high speed machine language program that is "hidden" at the top of your PET/CBM memory.

The machine language program serves 3 major functions.

1: Translates PET-ASCII code to ASCII code for program listing.

2: Translates screen code to ASCII code for screen image printouts.

3: Translates PET-ASCII code strings to ASCII strings for normal program printout. This feature may also be used for making ASCII files for your disk or tape recorder.

INSTALLATION: **Installs into your Epson printer**

PRICE: **Epson-PET Graphics ROM Pack EPG80, EPG81, EPG8F.....$75.00**

Please specify your Epson printer model type and and serial number when ordering.

For all PET/CBM's BASIC 2.0/Revision 3, or BASIC 4.0

AVAILABILITY: Immediately from your LOCAL DEALER

or

VISA, MASTERCHARGE ORDERS CALL (800) 227-9998 (except California residents)
CALIFORNIA ORDERS PLEASE CALL (415) 965-1735

Skyles Electric Works

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

```
    200
130 DATA 177, 1, 205, 168, 2, 240, 31, 160, 2,
     24
140 DATA 177, 1, 101, 1, 72 , 200, 177, 1, 101

150 DATA 2, 133, 2, 104, 133, 1, 165, 2, 197, ~
    47, 144, 215, 165, 1, 197
160 DATA 46, 144, 209, 96, 160, 5, 177, 1, 141
    , 160
170 DATA 2, 200, 177, 1, 141, 161, 2, 169, 1, ~
    141
180 DATA 162, 2, 169, 0, 141, 163, 2, 24, 165,
     1
190 DATA 105, 7, 133, 187, 165, 2, 105, 0, 133
    , 188
200 DATA 165, 187, 133, 177, 165, 188, 133, 17
    8, 24, 165
210 DATA 177, 105, 3, 133, 187, 165, 178, 105,
     0, 133
220 DATA 188, 160, 0, 177, 177, 208, 46, 24, 1
    73, 162
230 DATA 2, 105, 1, 141, 162, 2, 173, 163, 2, ~
    105
240 DATA 0, 141, 163, 2, 173, 163, 2, 205, 160
    , 2
250 DATA 144, 204, 173, 162, 2, 205, 161, 2, 1
    44, 196
260 DATA 173, 164, 2, 208, 1, 96, 169, 0, 141,
     164
270 DATA 2, 240, 160, 141, 165, 2, 177, 187, 2
    40, 236
280 DATA 141, 166, 2, 200, 177, 177, 133, 185,
     177, 187
290 DATA 133, 189, 200, 177, 177, 133, 186, 17
    7, 187, 133
300 DATA 190, 160, 0, 177, 185, 209, 189, 144,
     174, 240
310 DATA 2, 176, 11, 200, 204, 165, 2, 240, 16
    4, 204
320 DATA 166, 2, 208, 235, 160, 0, 24, 165, 18
    5, 109
330 DATA 165, 2, 133, 214, 165, 186, 105, 0, 1
    33, 215
340 DATA 24, 165, 189, 109, 166, 2, 133, 218, ~
    165, 190
350 DATA 105, 0, 133, 219, 177, 214, 72, 200, ~
    177, 214
360 DATA 72, 136, 177, 218, 145, 214, 200, 177
    , 218, 145
370 DATA 214, 104, 145, 218, 136, 104, 145, 21
    8, 160, 0
380 DATA 173, 166, 2, 145, 177, 173, 165, 2, 1
    45, 187
390 DATA 200, 165, 189, 145, 177, 165, 185, 14
    5, 187, 200
400 DATA 165, 190, 145, 177, 165, 186, 145, 18
    7, 169, 1
410 DATA 141, 164, 2, 169, 0, 240, 156, 95, 25
    5, 95

500 PRINT"{CLEAR}":FORI=1TO10
510 PRINT"NOW INPUTTING INTO VARIABLE A$("I")"
    ;:INPUTA$(I):NEXT
520 PRINT"{CLEAR}NOW STARTING SORT.
525 TI$="000000"
535 REM POKE VARIABLE NAME AND SORT LIST
540 POKE679,ASC("A"):POKE680,128:SYS S
550 PRINT"{02 DOWN}SORT FINISHED IN"TI"JIFFIES
    .{DOWN}
560 FORI=1TO10:PRINTA$(I):NEXT
570 PRINT"{02 DOWN}PRESS {REV}RETURN{OFF} TO C
    ONTINUE
580 GETA$:IFA$<>CHR$(13)THEN580
600 REM ***SORTING DATA READ IN FROM
601 REM *DATA STATEMENTS IS SLIGHTLY
602 REM *DIFFERENT FROM INPUTTING
603 REM *THE DATA.
604 DATAZ,D,S,F,H,R,C,H,U,U
605 PRINT"{CLEAR}NOW READING IN DATA.
610 FORI=1TO10:READNA$(I):NEXT
620 REM ***THIS IS THE DIFFERENCE***
630 FORI=1TO10:NA$(I)=NA$(I)+"":NEXT
640 PRINT"{03 DOWN}NOW STARTING SORT.
645 TI$="000000"
655 REM POKE VARIABLE NAME AND SORT LIST
660 POKE679,ASC("N"):POKE680,ASC("A")+128:SYS ~
    S
670 PRINT"{02 DOWN}SORT FINISHED IN"TI"JIFFIES
    .{DOWN}
680 FORI=1TO10:PRINTNA$(I):NEXT
```

©

THE BUSINESS MANAGER...

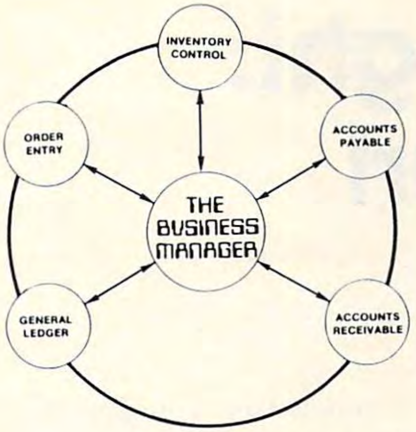
FULLY INTERACTIVE ACCOUNTING SYSTEM. NO DISKETTE CHANGING! FINALLY ALL THE FEATURES YOU'VE BEEN WAITING FOR – IN ONE COMPLETE INTERACTIVE SYSTEM.

FULL SOFTWARE SYSTEM

ONLY $995.00

QUESTAR INTERNATIONAL INC

MANUAL ONLY – $40.00
REFUNDABLE UPON
SYSTEM PURCHASE.



HARDWARE REQUIREMENTS:

- Commodore Model 8032 or SuperPET or 8096 Micro-computer
- Commodore Model 8050 One Megabyte Dual Disk Drive
- Commodore Model 4022 (or Larger) Printer
- Commodore PET — IEEE cable
- Commodore IEEE — IEEE cable

DISKSHARE 9000

WORKS WITH ALL COMBINATIONS PET/CBM 100% HARDWARE. NO SOFTWARE REQUIRED! UP TO 8 USERS SHARE DISK AND/OR PRINTER. CONNECTS IN MINUTES – NO TOOLS REQUIRED. INCREASES PRODUCTIVITY – REDUCES COSTS. IDEAL FOR EDUCATION AND BUSINESS. CABLE LENGTHS UP TO 50 FEET.

ONLY $749.00 (U.S.)
8 USER SYSTEM ONLY $1149

DEALER
INQUIRIES
WELCOME.

QUESTAR
INTERNATIONAL
INC.
Suite 102 7270 Woodbine Avenue
Markham Ontario Canada L3R 1A4
Tel (416) 475 8044

Multi User System

Insight: Atari

Bill Wilkinson
Optimized Systems Software
Cupertino, CA

In addition to a continuation of the game development that I started last month, this month's column will delve into the argument of what makes BASIC run, including a chip that makes Atari BASIC run better. But first ...

FMS And Burst I/O, Yet Again

Well, July's column was supposed to fix the mistakes I made in the May column. And then, lo and behold, I blew it again in July. On page 172 of issue 26 of **COMPUTE!** there is a listing of changes to be made to FMS to help burst I/O work properly in update mode. The assembly language listing and the BUG changes were correct. Unfortunately, the POKEs from BASIC had one typo (my fault). The last POKE read

```
POKE 2773,13   ... WRONG!
```

should be

```
POKE 2773,31   ... RIGHT!
```

Speed And BASIC

Personally, I have never been sure that it is necessary for an interpreted language (e.g., BASIC) to be fast. Typically, I choose to use an interpreter for ease of use and speed of debugging, for writing quickie little programs, and for creating utilities that can run at any speed at any time.

But an increasing number of people are trying to use BASIC for writing serious software, including games, utilities, and business applications. Now I maintain that the speed of BASIC is irrelevant when it is being used for utilities (who cares how fast a disk fixer-upper runs?) or business applications (the program is usually waiting for keyboard, printer, or disk I/O anyway). But for writing games and a certain category of other programs (e.g., sorts), speed is important. But then why use BASIC? Because it's the easiest language to use? Because it can be made fast enough? Because it's the only language the author knows?

Actually, those (and many others) are all valid reasons to choose BASIC, as long as the author doesn't expect more than BASIC is capable of delivering. So what is BASIC capable of delivering? A lot of adequacy. After all, look at some of the very successful games that are written in BASIC (*Crush, Crumble, and Chomp* is the first one that comes to my mind). Or look at some games that should never have been written in BASIC and yet were (a lot come to mind, but I will refrain from naming any).

Certain other authors writing in another magazine have claimed that Atari BASIC is the slowest language ever created. My first impulse was to say, "Who cares? It is the easiest to use, and that's more important." But I simply couldn't take that statement lying down, as it were. After all, if Atari BASIC is such a snail, how come all these programs seem to work just fine?

So I armed myself with five different BASIC interpreters: Applesoft, Atari BASIC, Atari Microsoft BASIC, BASIC A+, and Cromemco's 32K Structured BASIC. Now OSS produced three of these five BASICs, so it might seem that I am prejudiced. Well...maybe a lot, but not too much. Some comments follow on what I decided to try to do.

I wanted to use a benchmark program that would, to some degree, show the fundamental speed of each BASIC. But I also wanted to see what impact such things as constants, variable names, and multi-statement lines would have. Luckily, at just about this same time, I happened upon a benchmark (as yet unpublished) which showed Atari faster than Applesoft in a very simple program. "Oh ho!" says I. "How can this be? Atari is the slowest machine ever, say certain voices."

Anyway, I began experimenting with a small benchmark program, allowing various changes so that I could see the impact on speed. The most fundamental program was simply:

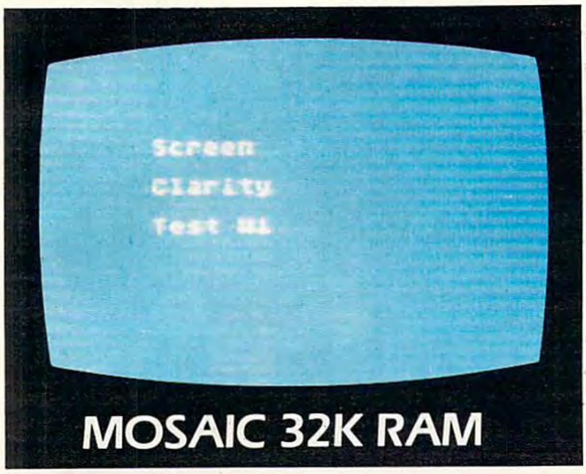
```
10 < start a timer >
20 A = 0 : B = 12345.6
30 A = A + 1.234567
40 IF A < B THEN 30
50 < print time used >
```

Obviously, the intent of this program is to cause a loop to execute 10,000 times. But what can be changed that will significantly affect the execution time without materially altering the program? Below I show all the versions of lines 20 and 30 that I tested. (Line 40 is not shown, but it followed line 20 in the naming of variables and otherwise remained unchanged.) The table also shows the times for the various languages, rounded to three significant figures.

In addition to the timings shown in Table 1, I also tried adding several variables to the programs. Adding 18 variables (in lines 11 and 12) added about five or six seconds to the Microsoft BASICs, about 1.5 seconds for Atari BASIC and BASIC

THE MOSAIC RAM SYSTEMS FOR ATARI*

CLEARLY THE BEST



MOSAIC 32K RAM

THE SCREEN CLARITY TEST

THE COMPETITION

YOU CAN SEE THE MOSAIC DIFFERENCE

WHAT THE EXPERTS HAD TO SAY

A.N.A.L.O.G. 400/800 MAGAZINE said in a 32K RAM board comparison test: "The Mosaic 32K RAM showed no sign of interference and gave the best screen clarity" and "Mosaic uses what we feel are the highest quality components which may improve reliability".

INTERFACE AGE said after replacing the Atari memory board with a Mosaic 32K RAM: "Once in place there is no noticeable change in screen clarity" and "in view of its excellent performance it should be a serious choice for those Atari owners intending to expand their memory."

Each Mosaic RAM board gives you more than just the best screen clarity but also the best in reliability, flexibility, and compatibility. The Mosaic RAM systems offer you the best in every way — these features prove it.

- Works in both the Atari 400*& 800*computers.
- 4 year guarantee.
- Complete instructions.
- Test cycled 24 hours for reliability.
- Gold edge connectors for better reliability.
- Quick no-solder installation.
- Full flexible memory configuration.
- Can be used with 8K, 16K and future products.
- Designed to take advantage of Atari 800's superior bus structure.
- Designed for inter-board communication in Atari 800.
- Allows Atari 800 to have 2 slots open for future expansion.
- Always the best components used for superior screen clarity and reliability.
- Low power design for safety and reliability.
- Available companion board ($5) to allow running The Mosaic RAM systems independent of other boards.

THE MOSAIC RAM SYSTEMS FOR THE SERIOUS ATARI OWNER.

THE MOSAIC 32K RAM

For the serious Atari owner. This is the 32K RAM board you've been hearing about. It has every feature you could want from a RAM board and more features than any other board offers. Each Mosaic 32K RAM comes with complete instructions so in a few minutes you will have expanded your Atari 800 to 48K RAM. The Mosaic 32K RAM works as well in the Atari 400, but we suggest the NEW Mosaic Expander.

THE MOSAIC EXPANDER

This is the most effective way you can expand to 32K RAM for your Atari 400 computer. And at almost the 16K price! The revolutionary Mosaic Expander is a memory board with 16K RAM in place and open slots to add 16K more from the Atari 16K board that comes with your Atari 400. Each board comes with complete instructions so in a few minutes you will have expanded to 32K RAM.

For your nearest Mosaic dealer call toll free
800-547-2807

*Trademark of Atari, Inc.

MOSAIC
ELECTRONICS

P.O. Box 708 Oregon City, Oregon 97045
503/655-9574 Toll Free 800-547-2807

the PROGRAM STORE

Great Autumn Offerings

PROBE ONE

From Synergistic Software

High-res action game. You're the commander of the Terran space ship PROBE ONE during the war between the Terran Confederation and the Drelgan Hegemony. Only you can save the human race from extinction by the Drelgans...if you can find the matter transmitter and bring it to Terra. You'll need a quick mind and fast reflexes to fight off the guard robots as you search. Multiple skill levels. BASIC required; paddles or joystick.

40K Tape, ~~$34.95~~ *Save 15%!*

Now Thru Oct. 1 You Pay Just $29.71

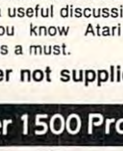
CAVERNS OF MARS

By Greg Christensen from APX

An award-winning arcade game that will challenge your skill for many, many plays. You command the fighter Helicon VII on a mission to invade the Martian high command. The headquarters are located deep in the caverns, so you must descend through radar, rockets, laser gates, and space mines to reach your destination. Once there, you activate the fusion bomb and try to get out before it detonates. Good luck! Joystick required.

16K Tape, $34.95 *-(WHILE THEY LAST!!*

16K Disk, $37.95



DE RE ATARI

From APX

Translated from Latin, the title of this book is "All About Atari" and it means what it says! Used in combination with Atari's Technical Reference Manual, advanced programmers will be able to learn to exploit the many hardware and operating system features that make the Atari 400/800 so tremendously versatile. Includes a useful discussion of the new GTIA chip. Once you know Atari BASIC and assembler, this book is a must.

Loose leaf (binder not supplied), $19.95

From DataSoft Inc.

Games, Graphics, Word Processing Programs

SHOOTING ARCADE

From DataSoft

Step right up, ladies and gentlemen, and try your luck! It's your favorite amusement park game right in your own home! Aim at the elephants, ducks, bunnies and more. Every time you hit the target you score. But aim carefully—don't waste bullets. Fire at a frowning face and your bullets turn to blanks; shoot the smiling face and win BIG! If you make it to the 10th round, there's a ***surprise.*** Requires joystick.

16K Disk, ~~$29.95~~ *Save 15%!*

Now Thru Oct. 1 You Pay Just $25.46

CLOWNS AND BALLOONS

From DataSoft by Frank Cohen

A huge bunch of balloons is stuck at the top of the circus tent. How do you get them down? By bouncing a clown on a trampoline, high enough to burst them, of course! You must break the balloons in order, one row at a time—if you miss any, the whole row fills up again. Aim your clowns where you want them by moving the trampoline to just the right position. 1 or 2 players; multi-skill levels; joystick or paddle required.

16K Tape or Disk, $29.95

CANYON CLIMBER

From DataSoft by Tim Ferris

You're at the bottom of the Grand Canyon, trying to scale the world's toughest wall. If that's not enough of a problem, there are three challenges to face along the way: angry sheep, threatening Indians and attacking birds. Action-packed arcade game you'll really enjoy. Requires joystick.

16K Tape or Disk, $29.95

SPELL WIZARD

From DataSoft

SPELL WIZARD is a proof-reading package, compatible with Atari's word processor and Text Wizard. It has a dictionary of 20,000 words; you can add up to 5,000 more. Insert your letter, term paper or the latest chapter of your novel, and SPELL WIZARD will proofread it for you. Your copy will be perfect.

32K Disk, $79.95

MICROPAINTER

From DataSoft by Bob Bishop

Create your own color graphics with this "electronic coloring book" for all ages and abilities. The special features include: draw mode, diamond fill, microscope mode for "close-up" viewing, inversing, rubberband line draw, and joystick-controlled easel for selecting pattern, colors and intensities.

48K Disk, ~~$34.95~~ *Save 15%!*

Now Thru Oct. 1 You Pay Just $21.21

PACIFIC COAST HIGHWAY

From DataSoft by Ron Rosen

Fast-action game with good graphics for 1 or 2 players. You're a tortoise, trying to cross the Highway without getting hit by the rush hour traffic. Once across, you're at the shore, and must leap from boats to rafts on the Pacific Ocean. Be careful 'cause you can't swim! Requires joystick.

16K Tape or Disk, $29.95

TEXT WIZARD 2.0

From DataSoft

This is the newest version of the Atari word processing package TextWizard, so popular with writers, students annd small business people. Like the original, you can learn and use TextWizard 2.0 in about an hour; write copy, then edit; underline, insert or replace words or paragraphs (in one place or **throughout** text); print in 1 or 2-column format; get multiple copies, print in boldface, elongated or condensed lettering with proportional spacing. It even warns you about errors ***before*** you make them!

TEXTWIZARD 2.0 now has file compatibility with File Manager 800 so you can change your text with information taken from File Manager files; address your form letters from File Manager's mailing lists, etc. It's also compatible with: EPSON MX-80 with Graftrax Plus ROMS; NEC 8023-A; Okidata and Pro-Writer. Instruction manual with 3-ring binder.

32K Disk $99.95

Over 1500 Programs for TRS-80, ATARI 400/800, APPLE & IBM.

For Information Call
202-363-9797

Visit our other stores:
Seven Corners Center, Falls Church, VA
W. Bell Plaza, 6600 Security Blvd., Baltimore MD
Coming Soon to Columbus, OH and Greensboro, NC

the PROGRAM STORE™

To Order Call Toll-Free
VISA **800-424-2738** MasterCard

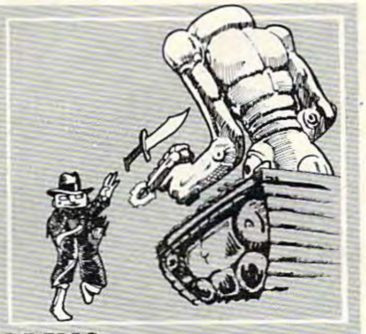
MAIL ORDERS: Send check or M.O. for total purchase price, plus $1.00 postage & handling. D.C., MD. & VA.: add sales tax. Charge cards: include all embossed information on the card.

4200 Wisconsin Avenue, NW, Dept. U209 Box 9609 Washington, D.C. 20016

For Your ATARI™ 400/800

the PROGRAM STORE

From Synapse Software
Arcade-Type Action Games, Application Software



SHAMUS

From Synapse Software

It's the 21st century, and you're the SHAMUS, looking for your arch-enemy, the Shadow, to destroy him. You're armed with Ion-Shivs, the most powerful weapons in the galaxy. Can you handle the Shadow's henchmen: Robo-Droids, Whirling Drones and Snap-Jumpers, all armed and evil? Can you find the Shadow in his lair of 4 levels with 32 rooms each . . . every one of them dangerous? Intensive arcade action; requires joysticks.

16K Tape or Disk, ~~$29.95~~ Save 15%!

Now Thru Oct. 1 You Pay Just $25.46

PROTECTOR

From Synapse Software

The planet's inhabitants are endangered by a malevolent alien that beams them to his ship and transports them to an active volcano. You must pick them up one-by-one with your rescue/attack ship and transport them to another city while dodging lasers and rough terrain. After you save as many as possible, the volcano erupts. You must then move each person to a volcano-proof vault in the mountains! Great graphics and sound in this arcade game. Requires joystick.

32K Tape or Disk, $29.95

FILE MANAGER 800

From Synapse Software

An extremely powerful and versatile database manager for use in both professional and personal applications. You define the format of the records to be filed and FILE MANAGER 800 gives you full control over sorting, searches, and retrieval.

You can store up to 1000 records on each disk and the program will allow up to four drives. Access to any record takes less than 1.5 seconds, and most commands can be entered with self-prompting single key strokes. Works with Text Wizard 2.0 to make editing easier.

40K Disk, $99.95

FORT APOCALYPSE

From Synapse Software

The Warlords of Kraltha have constructed a prison deep inside Earth—the terrifying FORT APOCALYPSE. All those who have tried to rescue their slave/captives have vanished without a trace. ***Now it's your turn!*** Can you descend thru the Kralthian disruptor fields and penetrate the vast underground Vaults of Draconis? Can your Rocket-Copter fight off the Wormlings, Servo-Tank Interceptors and Robo-Choppers? Multi-player game with arcade action; requires joysticks.

32K Tape or Disk, ~~$29.95~~ Save 15%!

Now Thru Oct. 1 You Pay Just $25.46

SLIME

From Synapse by Mike Hales

Large drops of "Slime" fall from the sky onto your ships. If ***one drop*** hits your ship, it will sink. Use the triangular diverters to aim Slime into buckets on the sides of the screen. If Slime falls into the ocean, the level of the ocean rises. When the ocean reaches the top of the screen, the game is over, and your score is calculated. One-player game with excellent graphics.

16K Tape or Disk, $29.95 Requires joystick.

From Synapse Software

There's trouble in the barnyard, and this machine-language, arcade game will challenge even the most experienced arcade player. You try to help Ma Hen save the eggs and chickens from the wily fox. The action gets faster and faster as eggs turn into chicks, feathers fly, chickens squawk, and all bedlam breaks loose. You'll really have to think fast to outwit this fox.

Requires paddles.

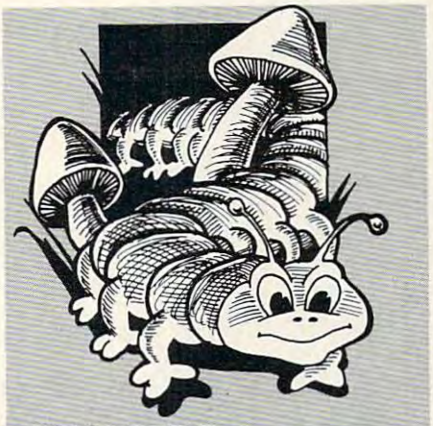
16K Tape, $29.95

NAUTILUS

From Synapse by Mike Potter

Two-player interaction combat game: the underwater cities and their destroyers against the helicopters and submarines. The helicopters bomb the ships; the subs fire torpedoes at the cities and surface to fire at the destroyers. The destroyers fire at the helicopters and drop depth charges on the subs. They can also lower men into the ocean to rebuild cities hit by torpedoes. Top and bottom halves of screen scroll separately for maximum field of play. Great graphics. Requires joystick.

16K Tape or Disk, $29.95



CENTIPEDE

From Atari

Very fast arcade game for the nimble-fingered. The green centipede is streaking across the screen, trying to get from the top to the bottom row. If you're fast enough, you can shoot it—but don't hit it in the ***wrong spot.*** Gain points by killing the spiders and mushrooms, too. Requires joystick.

ROM Cartridge, ~~$42.95~~ Save 15%!

Now Thru Oct. 1 You Pay Just $36.51

FROGGER

From On Line Systems

Based on arcade game Frogger. Dodge traffic and jump the moving logs in the stream to get home. Fantastic graphics. Requires joystick.

16K Tape or 32K Disk, $34.95

TEMPLE OF APSHAI

From Automated Simulations

The first of the DujonQuest series, and still one of the most popular. In exploring over 200 rooms in the magical labyrinth, you will encounter more than 30 kinds of fearsome monsters guarding over 70 treasures. Some of the treasures will help you in your quest, but you must still watch out for the many monsters and traps that spring out from the walls and shadows.

32K Tape or Disk, $39.95

THE PROGRAM STORE • Dept. U209 • Box 9609 • 4200 Wisconsin Avenue, NW • Washington, D.C. 20016

Item Tape/Disk/Book Price

____

____

____

____

Postage $1.00

Total ______________

□ CHECK □ VISA

□ MASTERCARD

Name______________________

Address____________________

City__________ State__________ Zip______

Card #______________________ Exp________

Table 1. The Speed Matrix

Lines 20 and 30		Cromemco 32K BASIC	Atari BASIC	Atari uSoft	BASIC A+ (Atari)	Applesoft
20 A = 0 : B = 12345.6 30 A = A + 1.234567		37.0	72.6 (63.6)	270.	62.9 (59.3)	275.
20 A = 0 : B = 12345.6 30 A = A + 1.23456789		37.0	72.6 (63.6)	710.	62.9 (59.3)	350.
20 A = 0:B = 12345.6:C = 1.234567 30 A = A + C		37.0	73.1 (64.1)	56.3	63.4 (59.8)	50.8
20 LONGVARIABLEA = 0 : LONGVARIABLEB = 12345.6 30 LONGVARIABLEA = LONGVARIABLEA + 1.234567	**	37.0	72.6 (63.6)	320.	62.9 (59.3)	can't do
20 LONGVARIABLEA = 0: LONGVARIABLEB = 12345.6 30 LONGVARIABLEA = LONGVARIABLEA + 1.23456789	**	37.0	72.6 (63.6)	752.	62.9 (59.3)	can't do
20 LONGVARIABLEA = 0 : LONGVARIABLEB = 12345.6 : C = 1.234567 30 LONGVARIABLEA = LONGVARIABLEA + C	**	37.0	73.1 (64.1)	106.	63.4 (59.8)	can't do

** These tests made using double precision variables in Cromemco BASIC and Atari Microsoft BASIC. Single precision times were shorter, but not significantly so.

() Times shown in parentheses are explained in the text.

A+, and nothing at all to Cromemco BASIC.

Also, I tried the effects of combining lines 30 and 40 into a single line. For example:

```
30 A = A + C : IF A<B THEN 30
```

The time savings were all in the area of one second, not surprisingly, so I have not detailed them here.

But, look at the surprises! Let's look at the "foreigner," Cromemco 32K BASIC, first. *Nothing* seems to make a difference to it! Actually, I knew that this would happen before I ran the tests. Of all the BASICs shown, Cromemco's is the most like a compiler. I simply included it to give you an idea of what a truly properly structured interpreter can accomplish, but we must be fair and admit that the language is 26K bytes in its smallest usable incarnation.

For you Atari BASIC and BASIC A+ programmers, the happiest surprise is perhaps simply finding out that these languages do as well as they do. Also, note that the various program changes have only a small effect on the running times. So you don't have to be too careful about how you write your programs. (But it is still true that putting subroutines and FOR/NEXT loops at the beginning of a program will make a noticeable speed difference. Don't feel too bad: all Microsoft BASICs have this same quirk.)

And now to the Microsoft BASICs. Obviously, you pay a penalty for using constants in a loop. Using double precision constants (1.23456789 in our examples) costs so much that you should try to avoid them. Watch for long variable names: 41 seconds to go from a one-character name to LONGVARIABLEA ? Ouch! (Actually, I also tried three-character names and found the penalty there to be over seven seconds.) And there is a penalty for having lots of variable names in use. Hmmm... we need to-use variable names instead of constants, because constants are so slow; but using lots of variable names costs time also, so...

How about the other side of the Microsoft coin? What can we do that will show off the Microsoft BASIC speed? Two answers: use integer variables and do some transcendental function calculations. It's reasonably obvious why integer variables help: integer arithmetic is guaranteed to take less time than floating point. But why the transcendentals, if we just showed that the speeds are similar? Simple. I cheated. I used only addition, where the Atari BASIC floating point package shows up pretty good. But oh boy! Did we blow it when it comes to multiply! When using SIN, COS, etc., Atari Microsoft BASIC is three to six times faster

ATARI SAYS ITS FIRST WORD

WITH A VOICE BOX BY THE ALIEN GROUP!

THE ALIEN GROUP has emerged from the underground, daring to offer a full-featured speech synthesis system that is flexible, low in cost, and needs no-accessory devices. No Interface, Cables, Speaker, Amp, or External Power Required! **VOICE BOX** has been designed and programmed by Atari users to become the integral voice of a *400* or *800* computer. Simply plugged into the serial port, **VOICE BOX** automatically routes all speech into the speaker of your television monitor. With the menu-driven operating system supplied, you'll be creating original, intelligible speech within moments after loading disk or cassette. No lengthy or obscure instructions to wade through.

The system includes a dictionary which translates typed text into **VOICE BOX's** phonetic language. The dictionary can be expanded to include as many as *5,000* words of your own custom vocabulary. Unlimited speech can be produced by straightforward phonetic definiton at any time, even if the dictionary should be full.

The speech routines can be called from other programs for any purpose you can conceive. Here are a few suggested uses:

VOICE BOX
Speech Synthesizer

SOUND EFFECTS

- Access 64 phonemes at any of 4 pitch levels to add filtered, contoured sound to the Atari audio repertoire.

GAMES

- Program aliens to hiss threats, moan when destroyed.
- Devise weird, non-human tongues for dungeon dwellers.
- Insert cryptic spoken clues in maze games.

COMPUTER OPERATION

- Code verbal prompts and error messages that command attention and leave the current display intact.

EDUCATION

- Gain an introduction to the principles of phonetics.
- Learn touch typing through spoken feedback from the keyboard.

In addition, the *Random Sentence Generator* included in the operating system, which prints and speaks endlessly startling, amusing, even poetic combinations of words supplied by the user, helps teach school children to identify parts of speech and recognize a variety of sentence structures.

A minimum of 16K RAM is required by the operating system. Either disk or cassette includes both 16K and 32K versions. Try **VOICE BOX** for up to 10 days, and if it isn't the finest value you've ever seen in a computer peripheral, the most challenging and provocative addition you've ever made to your system, return it in its original condition for a full refund.

When ordering specify disk- or cassette-based operating system, and enclose check or money-order for $169, or state your **VISA** or **MASTERCARD** number. Send mail orders to: **THE ALIEN GROUP**
27 West 23rd Street
Dept. PT-1
New York, New York 10010

or telephone orders from
10 AM to 6 PM New York time
(212) 924-5546

ALSO AVAILABLE AT LEADING COMPUTER STORES THROUGHOUT THE WORLD.

Atari is a registered trademark of Warner Communications.

than Atari BASIC. Until now. But before I explain that "until," let me make a few points.

Microsoft BASIC is definitely capable of more speed than Atari BASIC, but *only* if you are *very* careful and use lots of programming tricks. If you are an advanced programmer, this won't bother you. But I still believe, as I did over three years ago when we designed Atari BASIC, that for most people (and especially for beginners and hackers like me) the ease of use that is the hallmark of Atari BASIC makes it a real standout. But of course I'm not the perfect, objective judge. So try all of the BASICs, if your budget can afford it, and judge for yourself.

Fast, Faster, Fastest

This section will explain that "until now" that I wrote in the next to last paragraph. As I said, we (OSS and predecessors) blew it when it came to implementing the multiply algorithm, and as a result the transcendental routines take long enough for you to go out and get a cup of coffee. *But...*

Newell Industries (alias Wes Newell) of 3340 Nottingham, Plano, Texas (75074) has introduced the *Fastchip*. Actually, the *Fastchip* is a ROM which replaces the OS Floating Point ROM in an Atari 400 or 800. Major portions of the 2K bytes of ROM have been changed, resulting in several speed and/or accuracy improvements. The biggest changes were to the multiplication (ta da!) routine and floating-point to integer conversion (which is used *all* the time, by GOTO, POKE, SETCOLOR, XIO, OPEN, and many, many other statements and functions).

I have said that I will not normally review software, but I think the *Fastchip* deserves an exception to this rule on two points: it *can* be considered hardware, and it is a must for anyone contemplating heavy math usage with an Atari. Just as an example, note the times in parentheses in Table 1. These times are those recorded with a *Fastchip* installed. And this in a benchmark which does *not* make heavy use of *Fastchip's* best features!

Newell Industries has done some fairly complete timings of the various routines, so I won't belabor that point here. I will, however, include my own small benchmark program, just to give you an idea of the improvements available.

As you will note, I have included the Microsoft timings, also. Quite frankly, comparing Microsoft with Atari BASIC in this benchmark is almost as ludicrous as the reverse comparisons in Table 1. Which perhaps says a lot about how worthwhile benchmark programs *really* are.

Anyway, note that using the *Fastchip* brings the Atari BASIC timings within striking range of the Microsoft timings. A *most* respectable performance when you consider that the Atari BASIC routines use six byte floating point while Microsoft uses a four byte floating point. Incidentally, the BASIC A+ timings were all only a small fraction of a second faster than the Atari BASIC times here, so I omitted them.

Table 2. Transcendental Timings

line 30	Atari Microsoft	Atari BASIC	Atari BASIC with Fastchip
30 J=ABS(I)	1.15	1.53	1.48
30 J=SIN(I)	6.85	25.3	10.9
30 J=EXP(I)	6.75	33.7	9.93
30 J=I^I	12.4	74.0	20.8

```
10 <start timer>
20 FOR I=0 TO 6.3 STEP 0.02
30 J= <a function of I...see table>
40 NEXT I
50 <print elapsed time>
```

(program used with Table 2)

Enough hard work. On with the games!

BOING ... Part 2

Last month, we started with a simple program to bounce a ball around in a box. We noted some problems having to do with bouncing fast balls against a wall when the "clock" is slow: either the ball hits the wall "invisibly" or the bounce has to look strange. This month, we will extend that program into a real game and present an alternative method of moving the ball.

If you did type in last month's program, you might try changing it so that you assign XMOVE and YMOVE instead of having the program pick random directions. I would suggest that you try values of 0, 0.5, 1.0, and 2.0 in various combinations. If you choose XMOVE = 1 and YMOVE = 0.5, you will accomplish roughly what this month's program will use. Note, though, that the ball appears to jerk across the screen in strange directions. If you slow down the movement loop (put a delay in it), you will see that the ball really does go in as straight a line as it can (given the coarseness of the display we chose, Graphics 3). The jerkiness is simply an optical illusion, as far as I can tell, due to your eye expecting a certain movement and then being fooled.

The solution? Really, with finite pixel positioning, there is none. But you can greatly improve the situation by using a higher resolution graphics mode while retaining a relatively large ball: the

Experience the Magazine of the Future . . .

MAGATARI™

The Programmer's Institute's magnetic magazines will entertain, educate, and challenge you. Each issue features ready-to-load programs ranging from games, adventures, home applications, and utilities to personal finance, educational, and our "unique" teaching programs. Our magazines include fully listable programs, a newsletter containing descriptions and instructions for all programs, and notes on programming techniques used.

Subscription price for cassette is $50 per year, $30 per ½ year and $10 for a trial issue.
For diskette $75 per year, $45 per ½ year and $15 for a trial issue. . . . Don't miss another issue!

"Received my first copy . . . it's great! Please rush to me one of each back issue, so I'll have a complete set." **R.G., Chicago, IL**

"Not only are the games fun and the applications useful, but the quality of the programs is excellent." **S.P., Midwest Computer Software Sales**

"(Your magazine) is everything I expected and more!" **P.L., Durand, MI**

the programmer's program

The Programmer's Institute has designed a series of programs which examine the innovative use of BASIC commands. By exploring the fundamentals of computer programming, **The Programmer's Program** provides a valuable supplement to the BASIC manual.

Each technique is explained in depth, then graphicaliy displayed, enabling you to gain a better understanding of programming concepts. Such methods as **multiple loops**, **stacks**, **queues**, **graphics**, **game programming**, **sorts**, **trees**, **boolean logic**, **and memory savers** are explored, making programming tasks easier and more efficient.

The Programmer's Program is ideal for anyone who has ever been frustrated by the manuals! **($50 cassette, $55 diskette).**

FREE OFFER!

Order both The Programmer's Program and 1 year's subscription, and the total cost is only $75 cassette, $100 diskette. You will receive our **Haunted House Adventure** (14.95) ABSOLUTELY FREE with any $75 order. Offer expires September 30, 1982.

FREE CATALOG UPON REQUEST
Dealer Inquiries Invited
ALL SOFTWARE REQUIRES 16K

PROGRAMMER'S INSTITUTE
a division of FUTUREHOUSE
P.O. BOX 3191, DEPT. 1-C
CHAPEL HILL, NC 27514
MasterCard/VISA Welcome
10 AM - 9 PM, Mon - Sat
See your local dealer or order direct
1-919-489-2198
Add $2 postage and handling to all orders.

Beat the clock! Outsmart your friends!

WORDRACE

The high-speed dictionary game

At last, an educational game that's really fun to play! You start each turn with a word, 6 definitions, and a counter set to 600 points. As time passes the points tick away. The sooner you pick the **correct** definition, the more points you get, but the sooner you guess wrong, the more points you **lose**.

3 levels of play on one disk:
• Beginner (ages 9-14) • Regular • Challenge
2000 words and definitions
$24.95

ATARI: 32K — disk — BASIC
APPLE: 48K — disk — APPLESOFT

YOU CAN PLAY 3 NEW GAMES WITH YOUR* WORDRACE *DISKETTE:

The next disk in the **WORDRACE** System. Use it along with your **WORDRACE** disk to play:

- **CLAIM TO FAME** (600 famous people in history)
- **SPORTS DERBY** (600 pieces of sports trivia)
- Plus more vocabulary words

$19.95
Requires **WORDRACE** disk

Admit it: you've cursed out your computer.
Every programmer does it eventually. Ever wonder how it would reply?

Try **ABUSE**
The insult program.
• Funny • Unpredictable • Interactive
• Guaranteed to call you something you've never been called before!
$19.95

ATARI: 40K — disk — BASIC
APPLE: 48K — disk — APPLESOFT

Dealer inquiries welcome

Available at your computer store or direct from DON'T ASK
Include $2.00 shipping for each program (Calif. residents add 6% tax)

ATARI is a trademark of ATARI INC.

APPLE and APPLESOFT are trademarks of APPLE COMPUTER INC.

DON'T ASK
COMPUTER SOFTWARE

2265 WESTWOOD BLVD. B-150
LOS ANGELES, CALIF. 90064
(213) 397-8811

ORIGINAL SOFTWARE FOR THE ATARI 400/800 AND THE APPLE II/II+ FROM DON'T ASK

jumps in the higher resolution mode are small in comparison to the ball and so are not perceived as readily. With an Atari, the easiest way to accomplish this is with Player/Missile Graphics; but I will not delve into that in this series of articles since the subject has been covered so thoroughly and well elsewhere. Rather, the intent of these articles is simply to give beginners to graphics and/or assembly language a start in converting ideas from paper to BASIC to assembler.

This month, though, there simply isn't room or time to show and explain both the BASIC program and its assembly language counterpart. So the assembly language version will wait for next month, but I promise that it will be as closely related to this month's BASIC as last month's pair of programs were interrelated.

By the way, for those of you who simply want to play the game, just type it in as carefully as possible. Then simply RUN it for a two player Table Tennis-like game, using joysticks (not paddles – and, by the way, you must hold the joysticks turned 90 degrees left from normal position). For a one player game (not exciting, but a good demo), hold down the START key as you hit the RETURN key after typing RUN. And thus we start a skeletal explanation of how this program works.

What Makes BOING Ping?

First, note that YP(x) and SCORE(x) are simply the Y (vertical) paddle position of player "x" and a count of that same player's misses (x is 0 or 1, only). SINGLE is a flag set by examining the console switches which creates either a two player or one player game. LASTWIN is a -1 or +1 flag which indicates who scored the last point (we initialize it randomly).

At line 2000, the real work begins. In Graphics mode 3, we draw top and bottom boundaries and left and right paddles and print the current score. If this is a single person game, we overlay the right paddle with another wall. Also, in line 2060, we initialize each player's paddle position to 10, smack in the middle of each side. The ball is also initialized somewhere in the middle of the screen and given a starting shove.

At lines 2600 and 2700, we use my trick for reading the left and right joystick positions (this is the reason for turning the paddles), and we skip moving the paddle if the joystick is centered (and we never move the right paddle in a SINGLEs game). The method of moving a paddle is sheer simplicity: since each paddle is three units high, we erase the pixel on one end and create a new one on the other end. Presto, the paddle is moved. Oh, yes, we update YP(x).

Then, at line 3000, we start moving the ball. This is pretty much like last month, except that the XMOVE is always plus or minus one while the YMOVE is -1, -0.5, 0, +0.5, or +1. Note that if the ball won't hit something on its next move, it is because it will miss a paddle, so someone (HITP) will lose a point.

But if the ball is hit by a paddle, its YMOVEment is not determined by simple reflection. Rather, if the ball hits the center of a paddle, it is reflected straight across the playing field (with YMOVE=0). If it hits directly on either side of center, it returns at a slight angle (YMOVE = -0.5 or +0.5). But if it just barely hits the edge of the paddle, it rebounds at a satisfactorily nasty angle (YMOVE= -1 or +1). All this is done in line 3080.

Finally, the "LOSE" and "SCORE" routines are fairly simple. We force the ball to continue its flight for two more steps and then make a nasty noise and a simple but flashy display. We award a hit point as appropriate and figure out who LASTWIN should be.

This is *not* a sophisticated game. It is *not* intended to awe you with the power and flexibility of the Atari computer. It is intended to be a simple enough game that most of you will be able to follow its logic. And it certainly is intended to be easily translated to assembly language. But that's next month.

```
1000 REM *** STARTUP THE GAME ****
1010 DIM YP (1),SCORE(1):SCORE(0)=0:SCORE(1)=0
1020 SINGLE=(PEEK(53279)<>7)
1100 LASTWIN=1:IF RND (0)>=0.5 THEN LASTWIN=-LA
   STWIN
2000 REM *** PREPARE FOR A SERVE ***
2010 GRAPHICS 3: COLOR 2: PLOT 0,0:DRAWTO 39,0
2020 PLOT 0,19:DRAWTO 39,19
2030 PRINT :PRINT SCORE(1),,SCORE(0):PRINT "
        SCORE";
2035 IF SCORE(0)>20 OR SCORE(1)>20 THEN END
2040 COLOR 3 :PLOT 0,9:DRAWTO 0,11:PLOT 39,9:DR
   AWTO 39,11
2050 IF SINGLE THEN COLOR 2:PLOT 39,0:DRAWTO 39
   ,19
2060 YP(0)=10:YP(1)=10:REM VERTICAL POSITION
2070 IF SINGLE THEN LASTWIN=1
2100 REM SET UP BALL
2110 XMOVE=LASTWIN:YMOVE=INT(3*RND(0))--1:Y=INT
   (12*RND(0))+4
2120 YNEW=Y:X=19-5*XMOVE:XNEW=X
2500 REM *** MAIN PLAYING LOOP ***
2510 REM
2520 REM 1. CHECK AND MOVE PADDLES
2530 REM 2. SHOW NEW BALL POSITION
2540 REM 3. CHECK FOR COLLISIONS, ETC.
2550 REM
2590 REM *** FIRST CHECK AND MOVE PADDLES
2600 V0=PTRIG(0)-PTRIG(1):IF NOT V0 THEN 2700
2610 VP0=YP(0)-V0:IF VP0<2 OR VP0>17 THEN 2700
2620 COLOR 0:PLOT 0,YP(0)+V0:COLOR 3:PLOT 0,VP0
   -V0:YP(0)=VP0
2700 V1=PTRIG(2)-PTRIG(3):IF SINGLE OR V1=0 THE
   N 3000
2710 VP1=YP(1)-V1:IF VP1<2 OR VP1>17 THEN 3000
2720 COLOR 0:PLOT 39,YP(1)+V1:COLOR 3:PLOT 39,V
```

You've invested a lot of time and money into your computer . . .
It's time that investment paid off!

THE COLOR ACCOUNTANT

The Programmer's Institute introduces **THE COLOR ACCOUNTANT,** the only complete personal financial package specifically designed for the Atari 400/800 computer. This unique package includes:

1. Complete Checkbook Maintenance
2. Chart of Accounts Maintenance
3. Income/Expense Accounts
4. Net Worth Statement
5. Payments/Appointments Calendar
6. Color Graph Design Package (graphs any files)
7. Stock Market Analysis
8. Home Budget Analysis
9. Schedule 1040 (Long Form)
10. Decision Maker
11. Mailing List

After the initial setup, **THE COLOR ACCOUNTANT** requires less than an hour of data input each month.

The checkbook maintenance program is the key to the entire package. Once your checkbook is balanced, the checkbook summary file will automatically update the home budget analysis, net worth, and income/expense statements. You can then graph any file, record bills and appointments, make decisions, print a mailing list, analyze various accounts or stocks, and even calculate taxes.

All programs are menu-driven and allow add/change/delete. Each file and statement can be listed to screen or printer, and saved to cassette or diskette.

THE COLOR ACCOUNTANT also comes with 40 pages of documentation that leads you step-by-step through the entire package. The Atari 400/800 requires 24K for cassette and 32K diskette for this package.

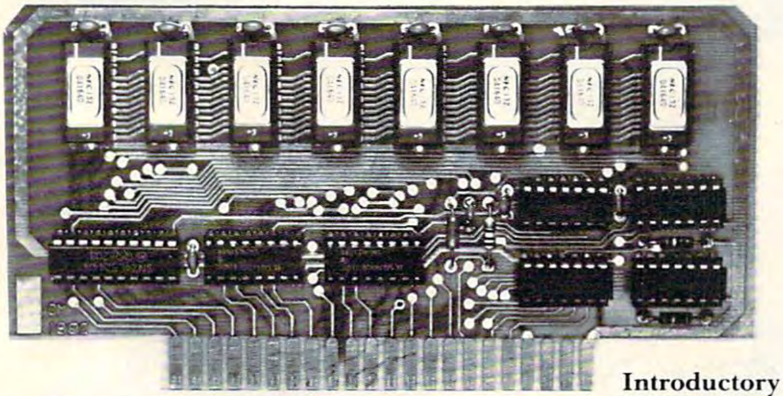
($75 cassette, $80 diskette).

FREE OFFER!

Order before Sept. 30, 1982 and you will receive **ABSOLUTELY FREE** Itemized Deductions (Schedule A) *and* Income Averaging (Schedule G).

PROGRAMMER'S INSTITUTE
a division of FUTUREHOUSE
P.O. BOX 3191, DEPT. 1-C
CHAPEL HILL, NC 27514
MasterCard/Visa Welcome
10 AM - 9 PM, Mon - Sat
See your local dealer or order direct
1-919-489-2198
Add $2 postage and handling to all orders.

Tara
Computer Products



PUT AN END TO YOUR MEMORY EXPANSION PROBLEMS!

With the only logical choice for 48K memory expansion of the Atari 400.

Upgrade with the new Tara Computer 48K RAM card for the 400.

Features:

- Easy installation.
- Cooler, less power hungry operation compared to standard 16K or 32K products.
- Uses state-of-the-art 64K Dynamic RAMs.
- Extends 400 useable memory to 48K.
- Allows higher performance 800 software to be run on your 400.
- Quality construction with gold edge connector.
- Allows for disk operation.

Increase the performance of your personal system efficiently and economically with the new Tara Computer 48K RAM card. Available direct from Tara Computer or from select dealers.

Introductory Offer $199⁰⁰ US PLUS SHIPPING

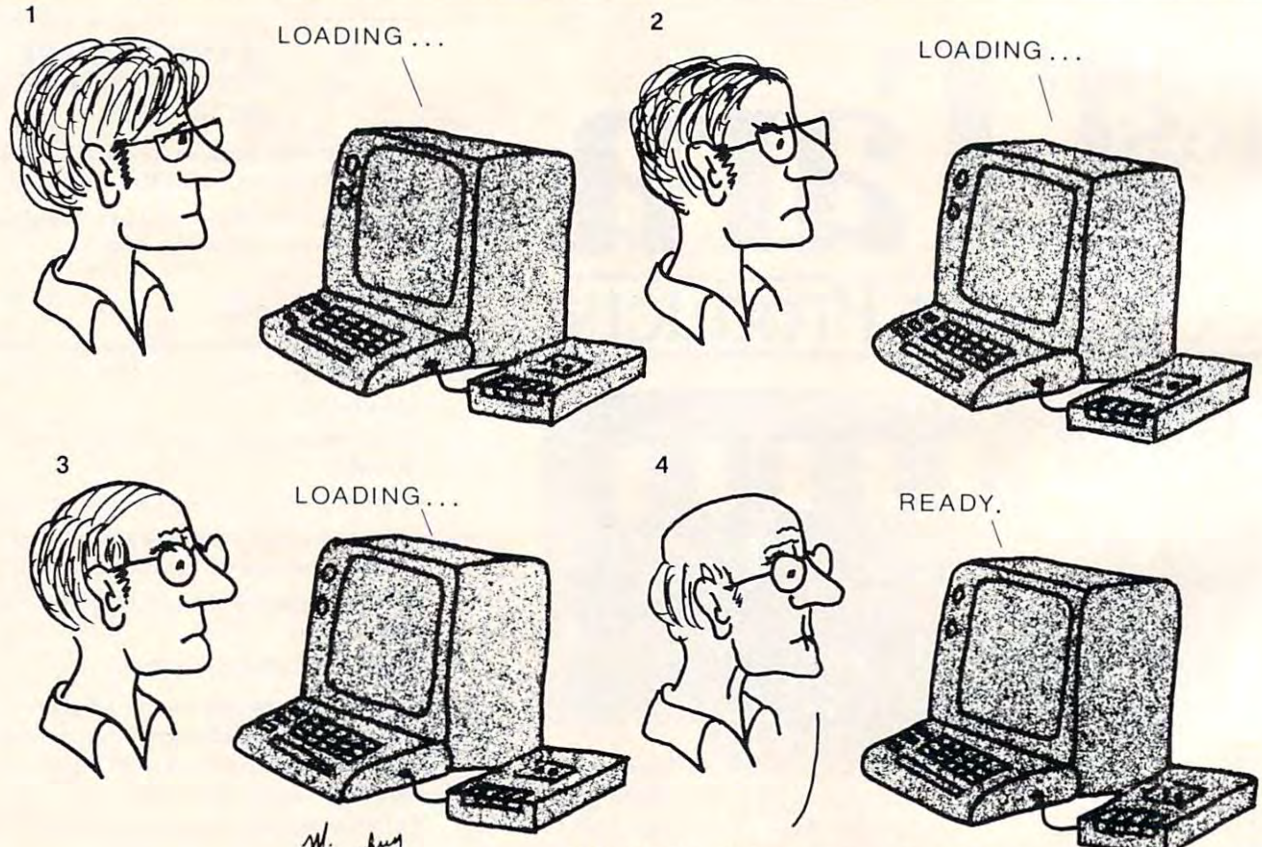
Tara Computer Products
3648 Southwestern Blvd.
Orchard Park, NY 14127
(716)662-7219

Tara Computer Products
2 Robert Speck Pkwy., Suite 1500
Mississauga, Ontario
Canada L47 1H8
(416)273-6820

(Dealers' Inquiries Welcome).

```
    P1-V1:YP(1)=VP1
3ØØØ REM *** BALL CONTROL ***
3Ø1Ø COLOR Ø:PLOT X,Y
3Ø2Ø COLOR 1:PLOT XNEW,YNEW
3Ø3Ø X=XNEW:Y=YNEW
3Ø4Ø XNEW=XNEW+XMOVE:YNEW=YNEW+YMOVE
3Ø5Ø IF XNEW<38 AND XNEW>1 THEN 32ØØ
3Ø6Ø HITP=(XNEW>2Ø):XHIT=39*HITP
3Ø7Ø IF SINGLE THEN IF HITP THEN 31ØØ
3Ø8Ø YMSAVE=YMOVE:YNEW=INT(YNEW):YMOVE=(YNEW-YP
    (HITP))/2
3Ø9Ø IF ABS(YMOVE)>1 THEN GOTO 4ØØØ
31ØØ XMOVE=-XMOVE
32ØØ IF YNEW=1 OR YNEW=18 THEN YMOVE=-YMOVE
329Ø GOTO 26ØØ
4ØØØ REM *** THE 'LOSE' ROUTINE
4Ø1Ø COLOR Ø:PLOT X,Y
4Ø2Ø COLOR 1:PLOT XNEW,YNEW
4Ø3Ø FOR I=1 TO 1Ø:NEXT I
4Ø4Ø COLOR Ø:PLOT XNEW,YNEW
4Ø5Ø COLOR 2:PLOT XNEW+XMOVE,YNEW+YMSAVE
413Ø SOUNDØ,132,12,12:POKE 2Ø,Ø
414Ø SETCOLOR 1,Ø,PEEK(2Ø)*4:IF PEEK (2Ø)<32 TH
   EN 414Ø
415Ø SOUND Ø,Ø,Ø,Ø
42ØØ REM *** SCORE IT ***
421Ø SCORE(HITP)=SCORE(HITP)+1
422Ø LASTWIN=1:IF HITP THEN LASTWIN=-LASLTWIN
499Ø GOTO 2ØØØ
```

©



SOMEDAY I'M GOING TO REPLACE
THIS CASSETTE WITH A DISC DRIVE !

PET Autoload

Lou Cargile and Richard Mansfield

"Autoload" for 80-column PETs allows you to easily LOAD and RUN any program from disk drive zero. You put the directory on screen and then move the cursor to the line which lists the target program. Hit the backslash key "\ " and the program boots into the computer and takes control. The cursor need not be in the first position on a line; it can be anywhere.

The routine is turned on (or off) by a SYS 634. The only potential problem would be inadvertently LOADing a program which overwrites the second cassette buffer. While Autoload is armed, all output to the screen jumps through the code at $0287. If this is overwritten, there will be trouble. Otherwise, this routine might be handy to place, for example, at the beginning of a disk of game programs. It would facilitate easy selections.

Autoload makes use of the vector at $EB, $EC, which can intercept characters on the way to the screen. This is an aspect of the newer Commodore machines which perhaps deserves further exploration. The disassembly (Program 2) illustrates how all characters typed are first examined by the routine starting at $0287. At $027A, the contents of $EB and $EC are toggled to alternatively point to $0287 or back to the normal $E20C. The routine in BASIC ROM which handles output to the screen passes through $E209 which is an indirect JMP down to $00EB. Normally, it bounces right back to $E20C because this vector at $EB is loaded with $E20C during initialization.

With the output to the screen flowing through $0287, a quick test of $D9 reveals whether or not the backslash key is being pressed. If it is, the keyboard buffer is loaded with the characters which will cause an automatic LOAD/RUN. Since LOAD itself returns BASIC to a warm start condition, the RUN must be pushed into the keyboard buffer. It would not survive LOAD if printed to the screen on the same line as LOAD.

In addition to this application, the concept of pre-screen interception could solve a variety of other programming problems. For example, you could assign certain keys to transmit special characters to a printer. Or a LISTing could be sent to the printer, spelling out cursor characters, in the fashion of **COMPUTE!**'s listing conventions. In effect, this technique gives you real control over the keyboard.

Program 1: Autoload

```
100 DIMCODE(87)
110 FORN=634TO721:BYTE=N-634
120 READCODE(BYTE):POKEN,CODE(BYTE)
130 NEXT
140 IFPEEK(235)=12ANDPEEK(236)=226THEN
  SYS634
150 DIRECTORY
200 DATA165,235,73,139,133,235,165,236
210 DATA73,224,133,236,96,165,217,41
220 DATA127,201,92,208,64,169,13,141
230 DATA111,2,141,115,2,169,147,141
240 DATA112,2,169,82,141,113,2,169
250 DATA213,141,114,2,169,5,133,158
260 DATA165,217,201,92,208,4,169,1
270 DATA133,158,169,1,133,198,169,68
280 DATA32,210,255,169,204,32,210,255
290 DATA169,32,32,210,255,169,24,133
300 DATA198,169,58,133,217,76,12,226
```

Program 2: Autoload Disassembly

```
027A A5 EB       LDA $EB
027C 49 8B       EOR #$8B
027E 85 EB       STA $EB
0280 A5 EC       LDA $EC
0282 49 E0       EOR #$E0
0284 85 EC       STA $EC
0286 60          RTS
0287 A5 D9       LDA $D9
0289 29 7F       AND #$7F
028B C9 5C       CMP #$5C
028D D0 36       BNE $02C5
028F A9 0D       LDA #$0D
0291 8D 6F 02    STA $026F
0294 8D 73 02    STA $0273
0297 A9 93       LDA #$93
0299 8D 70 02    STA $0270
029C A9 52       LDA #$52
029E 8D 71 02    STA $0271
02A1 A9 D5       LDA #$D5
02A3 8D 72 02    STA $0272
02A6 A9 05       LDA #$05
02A8 85 9E       STA $9E
02AA A9 01       LDA #$01
02AC 85 C6       STA $C6
02AE A9 44       LDA #$44
02B0 20 D2 FF    JSR $FFD2
02B3 A9 CC       LDA #$CC
02B5 20 D2 FF    JSR $FFD2
02B8 A9 20       LDA #$20
02BA 20 D2 FF    JSR $FFD2
02BD A9 18       LDA #$18
02BF 85 C6       STA $C6
02C1 A9 3A       LDA #$3A
02C3 85 D9       STA $D9
02C5 4C 0C E2    JMP $E20C
```

©

There are times when you'll want to process other kinds of disk files besides Text files. The technique is illustrated with a useful cross-reference program which shows where and how often variables are used in a BASIC program.

Process Any Apple Disk File

Keith Falkner
Venice, FL

Apple's Disk Operating System recognizes four types of files: Applesoft, Integer, Binary, and Text. When the DOS command CATALOG is entered, the names of all files on the disk are displayed, and the type of each file is indicated by a letter A, I, B, or T. A-files are of course Applesoft programs and are stored by the DOS command: SAVE programname. Similarly, I-files are programs in Integer BASIC. B-files are merely copies of memory onto disk, although they are often machine-language programs or subprograms. T-files are the only genuine data files, and these have invariably been written by programs.

Apple DOS contains a very sensible restriction: a program may OPEN only a Text file. Investigation verifies that all the other types of files usually contain many null bytes, that is, bytes with no bits on, or in Applesoft, CHR$(0). Unless a program explicitly writes CHR$(0), a Text file will never contain a null byte. So when data is being read from an open file into memory, Apple DOS tests each byte transferred. If a null byte is found, Apple DOS assumes that the program has read beyond the end of data in the Text file and issues the error message END OF DATA IN #####, where ##### is a line-number.

Many Good Things

This restriction really is a great nuisance, because there are many good things we could do if only a program could process the other types of files. For instance, a program could print a program listing neatly, produce a cross-reference report, or devise some documentation from the REM statements in an Applesoft or Integer BASIC program.

In fact, a program can circumvent the restriction and OPEN any type of file, by patching DOS as follows:

POKE 42948,234: POKE 42949,169: POKE 42950,0 : REM DOS 3.2 OR 3.3 IN 48K. This changes the instruction at $A7C4 from EOR $B5C2 into NOP and LDA #0, and thus circumvents the test for type of file.

Those POKEs are effective until DOS is rebooted (via PR#6 for example). If the DOS command INIT is issued after the POKEs, the disk so initialized will contain the patch permanently, thus the version of DOS on it will never be able to issue the error message FILE TYPE MISMATCH for the OPEN command.

Figure 1. The Cross-Reference Program's Variables

```
A    200 280   330 340 420 430
A$   330 490   520 530 540 550 620 630
A$(  100 360   370 380 390
B    210 220   260 350 360 370 380 390
B$   510 530   550 630 640
B$(  100 510
C    310 400   420 440 450 460 470 480 580
C$   510 550   630 670
C(   100 140   150 160 340
C1   310 340   400 450 470
C2   150 230   310 400 410 480 610
C9   340 440   450 470 500 510
J    140 150   220 250 260 360 370 380 680
K    220 250   260 350 370 380 390 610 620 630
L    300 310
L$   220 240   310
M$   360 370   390 400 480 490
P$   180
Q    200 280   300 330 520 690
Q$   330 520   690
QQ$  120 540
S$   120 310   640 660 670
X    220 260
X$   230 240   250 260 600 630 640
X$(  100 250   260
Y    640 650   660
Z    580 650
Z$   580
```

Figure 2. The Cross-Reference Program's Line Numbers

```
210  500  520
250  250
280  210
330  500  530
390  350  370
400  360
410  340
450  410
490  470
500  420  430  440  460  480
510  500
520  540  550
560  280
630  610
650  630
670  650
690  200  280  300  330  520
700  190
```

Any program using the above technique to OPEN a non-Text file must be prepared to detect and process the expected null bytes; therefore, the commands GET and INPUT will not work. Fortunately there is a convenient routine in DOS, and here is how to use it:

```
PRINT CHR$(4) "READ" filename
CALL 42636 : Q=PEEK (46531)
```

The above line corresponds to a GET statement and delivers in Q the ASCII value of one byte. If the byte is a null byte, then Q will simply be zero, and that can be processed as easily as any other value. DOS will not issue the error message END OF DATA, unless the program reads a byte past the last sector containing the disk file.

The final tip is therefore how to detect end-of-data when processing a file other than a Text file. Actually that is the easiest part, and is simple to deal with when the file has just been opened. Both A-files and I-files start with a two-byte counter which indicates how many bytes of data are in the program, i.e., how many bytes to process. B-files contain this same counter, but before it is another two-byte counter which tells where the image of memory is to be loaded. As usual with 6502 software, all these two-byte counters have the less significant byte first and the more significant last.

Illustrating these techniques is an adaptation of a program which first appeared in **COMPUTE!**, May/June 1980. The program prints a cross-reference list of either the variables or line-numbers in an Applesoft program. The program is very handy and was admirably explained by its esteemed author.

```
100 TEXT : HOME : NORMAL : DIM A$(15),B$(3),X$
    (500),C(255)
110 PRINT "CROSS-REF    JIM BUTTERFIELD": PRIN
    T
120 QQ$ = CHR$ (34):S$ = "      ":B$(1) = Q$:B$
    (3) = CHR$ (58)
130 INPUT "VARIABLES OR LINES? ";Z$:C2 = 5: IF
     ASC (Z$) = 76 THEN C2 =6
140 FOR J = 1 TO 255:C(J) = 4: NEXT : FOR J = ~
    48 TO 57:C(J) = 6: NEXT
150 IFC2=5THENFORJ=65TO90:C(J)=5:NEXT:C(36)=7:
    C(37)=7:C(40)=8
160 C(34) = 1:C(178) = 2:C(131) = 3
170 PRINT : INPUT "PROGRAM NAME: ";P$
180 IF P$ < "A" THEN PRINT CHR$ (4)"CATALOG": ~
    GOTO 170
190 GOSUB 700: PRINT CHR$ (4)"OPEN"P$: PRINT C
    HR$ (4)"READ"P$
200 GOSUB 690:A = Q: GOSUB 690: PRINT 256 * Q ~
    + A" BYTES"
210 IF B = 0 GOTO 280
220 PRINT L$;:K = X: FOR J = B TO 1 STEP - 1: ~
    PRINT " ";A$(J);:X$ = A$(J)
230 IF C2 = 6 AND LEN (X$) < 5 THEN X$  = " " ~
    + X$: GOTO 230
240 X$ = X$ + L$
250 IF X$(K) > = X$ THEN X$(K + J) = X$(K):K =
     K - 1: GOTO 250
260 X$(K + J) = X$: NEXT J:X = X + B: PRINT :B
     = 0
270 REM : GET NEXT LINE, TEST END
280 GOSUB 690:A = Q: GOSUB 690: IF A + Q = 0 G
    OTO 560
290 REM GET LINE NUMBER
300 GOSUB 690:L = Q: GOSUB 690:L = Q * 256 + L

310 C = C2:C1 = - 1:L$ = RIGHT$ (S$ + STR$ (L)
    ,6)
320 REM GET BASIC STUFF
330 GOSUB 690:A = Q:A$ = Q$
340 C9 = C(A): IF C9 > C1 GOTO 410
350 K = 0: IF B = 0 GOTO 390
360 FOR J = 1 TO B: IF A$(J) = M$ GOTO 400
370 IF A$(J) < M$ THEN NEXT J:K = B: GOTO 390
380 FOR K = B TO J STEP - 1:A$(K + 1) = A$(K):
     NEXT K
390 B = B + 1:A$(K + 1) = M$
400 C = C2:C1 = - 1:M$ = ""
410 IF C2 = 5 GOTO 450
420 IF A = 171 OR A = 172 OR A = 176 OR A = 19
    6 THEN C = 6: GOTO 500
430 IF A = 44 OR A = 32 GOTO 500
440 IF C9 < > 6 THEN C = 9: GOTO 500
450 IF C9 = C THEN C = - 1:C1 = 4
460 IF C > 6 GOTO 500
470 IF C < 0 AND C9 > C1 AND C9 > 6 THEN C1 = ~
    C9: GOTO 490
480 IF C2 = 5 THEN IF LEN (M$) > 2 OR C > 0 GO
    TO 500
490 M$ = M$ + A$
500 ON C9 + 1 GOTO 210,510,510,510: GOTO 330
510 B$ = B$(C9):C$ = ""
520 GOSUB 690:A$ = Q$: IF Q = 0 GOTO 210
530 IF A$ = B$ GOTO 330
540 IF A$ < > QQ$ GOTO 520
550 A$ = B$:B$ = C$:C$ = A$: GOTO 520
560 PRINT : PRINT CHR$ (4)"CLOSE"
570 INPUT "PRINTER? ";Z$
580 C= 3:Z = 6: IF ASC (Z$) = 89 THEN C = 4:Z ~
    = 12: PRINT CHR$ (4)"PR#1"
590 PRINT : PRINT "CROSS REFERENCE - PROGRAM "
    ;P$
600 X$ = "": FOR J = 1 TO X:A$ = X$(J)
610 IF C2 = 6 THEN K = 6: GOTO 630
620 FOR K = 1 TO LEN (A$): IF MID$ (A$,K,1) < ~
    > " " THEN NEXT K: STOP
630 B$ = LEFT$ (A$,K - 1):C$ = MID$ (A$,K + 1)
    : IF X$ = B$ GOTO 650
640 PRINT :Y = 0:X$ = B$: PRINT X$; MID$ (S$,1
    ,5 - LEN (X$));
650 Y = Y + 1: IF Y < Z GOTO 670
660 Y = 1: PRINT : PRINT S$;
670 PRINT LEFT$ (S$,6 - LEN (C$));C$;
680 NEXT J: PRINT : PRINT CHR$ (4)"PR#0"; END
690 CALL 42636:Q = PEEK (46531):Q$ = CHR$ (Q):
     RETURN
700 POKE 42948,234: POKE 42949,169: POKE 42950
    ,0: RETURN
```

©

This extension of the TRS-80 "Energy Monitor" program requires Extended BASIC and 16K.

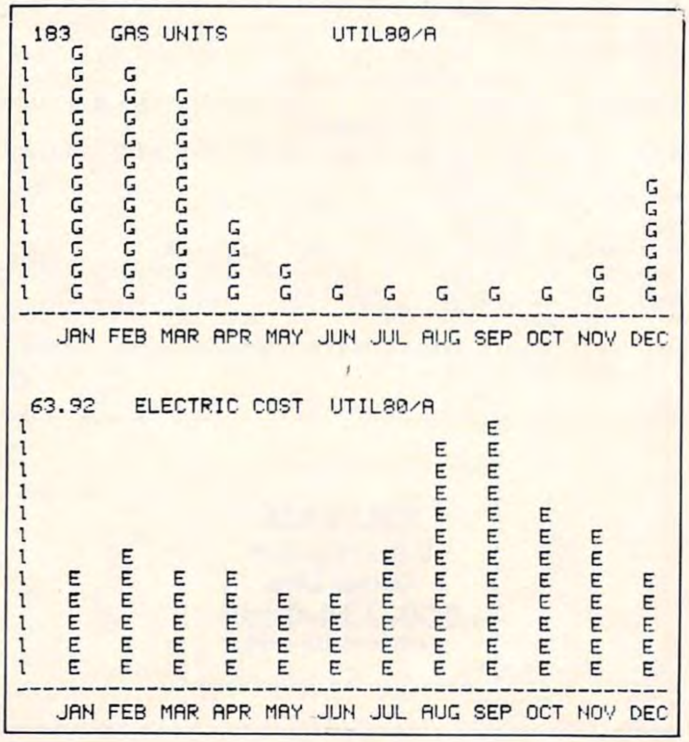
TRS-80 Color Computer Energy Monitor Graphics

Linton S. Chastain
Greensboro, NC

For those of you who have added "Energy Monitor" (**COMPUTE!**, August 1982) to your software repertoire, I have written a corollary program called "Energy Monitor Graphics." This new program will give you a visual display of your annual energy consumption or cost. It can be either displayed to your TV screen or printed out. This new program uses the data files created by the "Energy Monitor" program to generate the graphics.

The program was written on a TRS-80 Color Computer with Extended BASIC with disk drive. For those of you who do not have disk yet, you will have to change lines 70, 140, 160, and 180. The following changes will permit you to enter the data files from a cassette recorder:

```
70 CLS:PRINT"1-READ OLD MASTER FILE":
   PRINT"FROM CASSETTE
140 OPEN"I",#-1,T$:PRINT"READING FILE: ";T$:
    INPUT#-1,N
160 FORJ = 1 TO N:INPUT#
    -1,D$,A(J),B(J),C(J),D(J),E(J),
    F(J),G(J):PRINTJ:NEXTJ
100 CLOSE
```



```
 183   GAS UNITS         UTIL8Ø/A
l   G
l   G   G
l   G   G   G
l   G   G   G
l   G   G   G
l   G   G   G
l   G   G   G                                   G
l   G   G   G                                   G
l   G   G   G   G                               G
l   G   G   G   G                               G
l   G   G   G   G   G                       G   G
l   G   G   G   G   G   G   G   G   G   G   G   G
--------------------------------------------------
   JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC

 63.92   ELECTRIC COST  UTIL8Ø/A
l                                   E
l                               E   E
l                               E   E
l                               E   E
l                               E   E   E
l                               E   E   E   E
l       E                   E   E   E   E   E
l   E   E   E   E           E   E   E   E   E   E
l   E   E   E   E   E   E   E   E   E   E   E   E
l   E   E   E   E   E   E   E   E   E   E   E   E
l   E   E   E   E   E   E   E   E   E   E   E   E
l   E   E   E   E   E   E   E   E   E   E   E   E
--------------------------------------------------
   JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
```

This program's displays will help you see the effects of the seasons on your energy consumption and cost. It may help you organize your strategy to conserve energy.

```
30 CLEAR2ØØ
40 MR=2Ø:N=Ø
50 DIMD$(MR),A(MR),B(MR),C(MR),D(MR),E(MR),F(
   MR),G(MR),K(MR),T(MR)
6Ø R=Ø
7Ø CLS:PRINT"1-READ OLD MASTER FILE FROM DISK
   "
8Ø PRINT"2-DISPLAY GRAPH"
9Ø PRINT"3-RETURN TO COMMAND LIST"
1ØØ INPUT"ENTER COMMAND BY NUMBER";R:IFR<1 OR ~
    R>3 THEN6Ø
11Ø ON R GOSUB 12Ø,2ØØ:GOTO6Ø
12Ø R$*"READING":PRINT
13Ø INPUT"NAME OF FILE ";T$
14Ø OPEN"I",#1,T$:PRINT"READING FILE: ";T$:INP
    UT#1,N
15Ø PRINT"READING RECORDS # "
16Ø FORJ=1 TO N:INPUT#1,D$(J),A(J),B(J),C(J),D
    (J),E(J),F(J),G(J):PRINTJ:NEXTJ
17Ø PRINT:PRINTN;" DATA RECORDS READ"
18Ø CLOSE#1
19Ø PRINT"PRESS <ENTER> TO RETURN.":GOSUB112Ø:
    RETURN
2ØØ CLS:PRINT"COMMAND LIST # 2":PRINT
21Ø PRINT "1-DISPLAY WATER COST"
215 PRINT "2-DISPLAY WATER UNITS"
22Ø PRINT "3-DISPLAY GAS COST"
225 PRINT "4-DISPLAY GAS UNITS"
23Ø PRINT "5-DISPLAY ELECTRIC COST"
235 PRINT "6-DISPLAY ELECTRIC UNITS"
24Ø PRINT "7-RETURN TO COMMAND LIST #1"
25Ø INPUT"ENTER COMMAND BY NUMBER";R:IFR<1 OR ~
    R>7 THEN2ØØ
26Ø ON R GOSUB 34Ø,35Ø,36Ø,37Ø,38Ø,39Ø:GOTO27Ø
265 GOTO6Ø
27Ø CLS:PRINT"1-DISPLAY TO CRT"
28Ø PRINT"2-DISPLAY TO PRINTER"
29Ø PRINT"3-RETURN TO COMMAND LIST # 1"
3ØØ INPUT"ENTER COMMAND BY NUMBER";R:IFR<1 OR ~
    R>3 THEN 27Ø
33Ø ON R GOSUB 9ØØ,12ØØ,335:GOTO6Ø
335 GOTO6Ø
34Ø C$="WATER COST":FORJ=1 TO N:K(J)=A(J):NEXT
    J:RETURN
35Ø C$="WATER UNITS":FORJ=1 TO N:K(J)=B(J):NEX
    TJ:RETURN
36Ø C$="GAS COST":FORJ=1 TO N:K(J)=C(J):NEXTJ:
    RETURN
37Ø C$="GAS UNITS":FORJ=1 TO N:K(J)=D(J):NEXTJ
    :RETURN
38Ø C$="ELECTRIC COST":FORJ=1 TO N:K(J)=E(J):N
    EXTJ:RETURN
39Ø C$="ELECTRIC UNITS":FORJ=1 TO N:K(J)=F(J):
    NEXTJ:RETURN
9ØØ CLS:P=Ø:A$=CHR$(133):B$=CHR$(131):FORI=68 ~
    TO 388 STEP32:PRINT@I,A$;:NEXTI
9Ø1 RESTORE
91Ø FORI=388 TO 412:PRINT@I,B$;:NEXTI
94Ø FORI=421 TO 443 STEP2:READE$:PRINT@I,E$;:N
    EXTI
95Ø DATA"J","F","M","A","M","J","J","A","S","O
    ","N","D"
96Ø FORJ=1 TO N
97Ø IFK(J)-P=>Ø THEN P=K(J) ELSE P=P
```

```
980 NEXTJ
1020 PRINT@32,P;" ";C$
1030 FORJ=1 TO N
1040 IFP=0 AND K(J)=0 THEN 1110 ELSE 1050
1050 IFK(J)/P*10<1 THEN KK=1 ELSE KK=FIX(K(J)/P
    *10)
1060 T(J)=INT(36+((11-KK)*32))
1070 FORI=(T(J)+J*2) TO 388 STEP32
1080 IFT(J)=356 AND K(J)=0 THEN 1100 ELSE 1090
1090 PRINT@I,A$
1100 NEXTI:NEXTJ
1110 PRINT@453,"PRESS <ENTER> TO RETURN";GOSUB1
    120
1120 Z$="":R$*INKEY$:IFR$=Z$ THEN 1120
1130 RETURN
1200 CLS:PRINT"PRINTING GRAPHICS":P=0:L=0
1210 FORJ=1 TO N
1220 IFK(J)-P=>0 THEN P=K(J) ELSE P=P
1230 NEXTJ
1240 PRINT#-2,P;" ";C$;TAB(24)T$
1250 A$=LEFT$(C$,1)
1260 'FORL=1 TO 12
1270 IFP=0 THEN GOSUB 1380 ELSE 1280
1275 GOTO1350
1280 GOSUB1400
1340 'NEXTL
1350 PRINT#-2,STRING$(80,45)
1360 PRINT#-2,TAB(4)"JAN";TAB(8)"FEB";TAB(12)"M
    AR";TAB(16)"APR";TAB(20)"MAY";
1361 TAB(24)"JUN";TAB(28)"JUL";TAB(32)"AUG";TAB
    (36)"SEP";TAB(40)"OCT";
1362 TAB(44)"NOV";TAB(48)"DEC"
1370 RETURN
1380 FORL=1 TO 12
1385 PRINT#-2,CHR$(108)
1386 NEXTL
1390 RETURN
1400 FORL=1 TO 12
1405 PRINT#-2,CHR$(108
1410 FORJ=1 TO N
1420 GOSUB1460
1430 NEXTJ
1450 NEXTL
1455 RETURN
1460 IFINT((K(J)/P)*12)<=12-LTHEN PRINT#-2,TAB(
    4*J)"";ELSE PRINT#-2,TAB(4*J)A$;
1470 IFJ=>N THEN 1480 ELSE 1490
1480 PRINT#-2,CHR$(13);
1490 RETURN
```

©

A digitizer allows you to draw in free form on an "electric tablet" and communicate the results to your computer. It's similar to using a light pen, but more versatile.

$20 VIC Digitizer

Jeff Knapp
Charleston, WV

Have you ever looked at the commercially available digitizer tablets available for the TRS-80, PET, Apple, and other microcomputers and wished they were available in an affordable form for the VIC? Well, here's an inexpensive version that you can assemble in a few hours. If you don't have the parts in your junk box, it will probably cost less than $20 to buy all the parts new.

Take a look at Figure 1. This drawing shows the digitizer in its finished form. It consists of a base plate that becomes the stage for the materials to be digitized or a reference grid for input to the computer; a small box mounted on that contains the X-axis potentiometer (POTX); an arm extending from that pot that has the Y-axis pot (POTY) at its other end; and a second arm attached to the shaft of the Y-axis pot that has a hole drilled in the free end. This last arm becomes our cursor.

Construction is straightforward and simple. First, determine the desired size of your base plate. I recommend that it be at least 12" deep and 16" wide in order to allow for reasonable tolerances in the finished grid effective area. Choice of materials is up to you. I used Plexiglas plastic sheeting for mine. In the center of one long edge, mount one half of a mini-box or some other suitable container. It should be narrow enough that it doesn't take up too much space, and shallow enough that it is barely taller than the mounted potentiometer. About 2"x3"x1" will do for a 12"x16"' base plate.

Next, construct the arms. They should not be too long, since the cursor should be able to be placed over every intended coordinate on the grid. Mine were 7" long each. You can fudge a lot on the measurements, as long as everything is secure and the arms reach every point on the grid you will later place on the stage. You may find that it will be necessary to counterweight the first arm because of the weight of the second arm. All the construction can be as fancy or as simple as you please, as long as the arms cover the grid area adequately.

Now for the electronics. Figure 2 shows the schematic diagram for the digitizer. As you can see, it's very simple. One caution, though: try to use potentiometers that have permanently mounted metal shafts. Some pots are made to accept interchangeable shafts, and these have tolerances too loose for our purposes. If the shaft wobbles in the pot, chances are that your arms will sag because the weight of the arms pulls the shaft to one side. Use 250K pots, linear taper, and a DP9S 9-pin socket for termination. If you can't find the socket at the local electronics supply shop, try a hobby store. They're often used in radio controlled models. The cable linking the digitizer to the VIC can be just about anything. I used ribbon cable, about three feet long.

Mount one pot in the center of the other half of the mini-box, and provide a hole in the side for the cable to exit and enter. Mount the other pot in one end of the first arm, using the hardware provided with the pot. Mount the other pot in one end of the first arm, using the hardware provided with the pot. Mount the free end of that arm to the pot in the box. Use any method you wish, even press-fit, but be sure that it is secure and that the arm does not rotate without rotating the shaft as well. Mount the other arm to the pot at the end of the first arm in a similar manner. If you have drilled a hole in the free end of the second arm, that becomes the cursor. If not, mark the arm with cross-hairs of some sort.

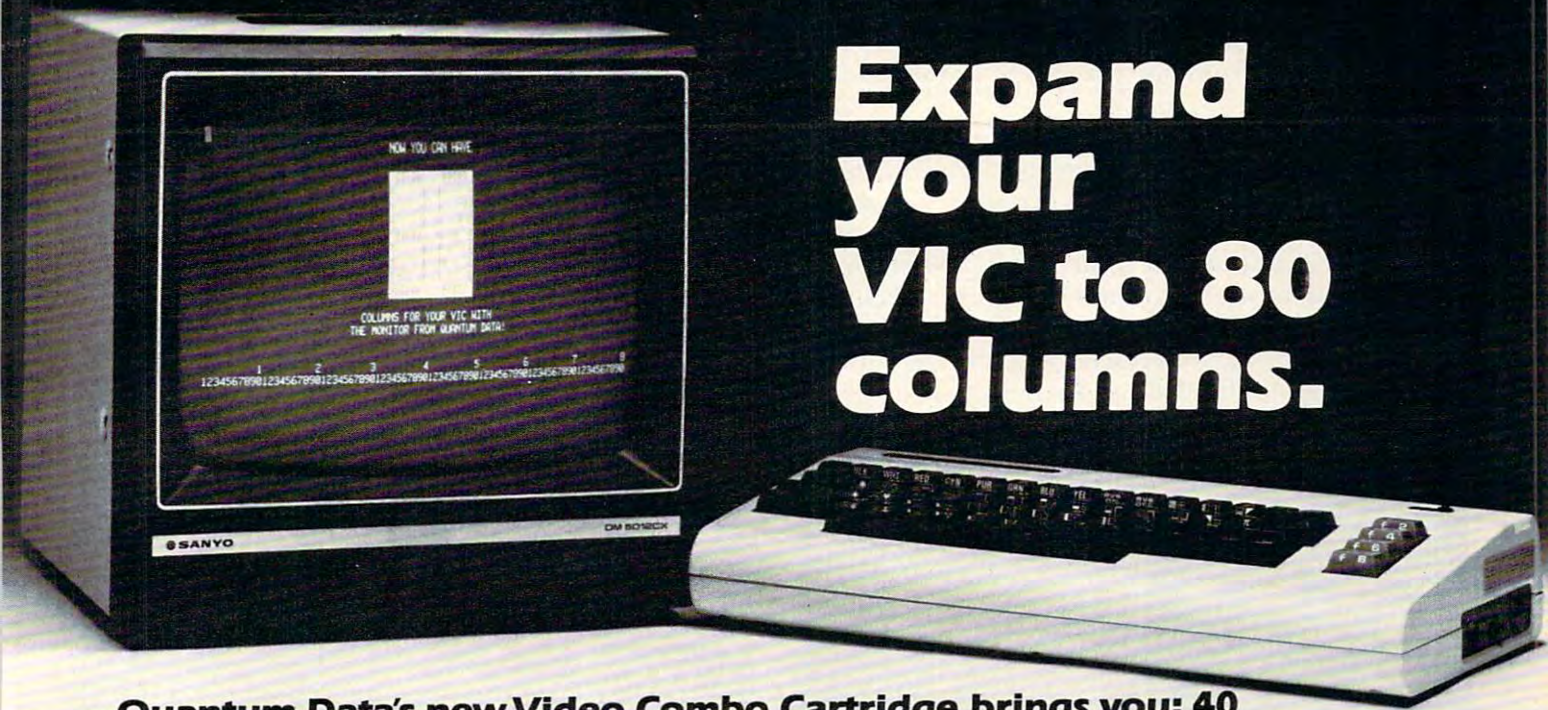
The Secret Is The Game Port

How does such a simple device work? Well, we have to depend on the VIC and some software for a lot of the job. The secret is that the VIC has two A-D (analog-to-digital) converters available at the game port. Although these are usually available as accessories for most microcomputers, the VIC has not one but *two* A-D converters built-in! Each generates a number between 0 and 255 based on the resistance of the two pots in our digitizer, or game paddles, or photocells, or any other variable resistance. The scale involved is one count for every 1K of resistance. Full scale would require a 255K pot, but ask your parts jobber for one and see the funny looks he gives you. 250K is as close as we can get. Plug the digitizer socket into the game port on the side of the VIC. Turn the VIC on and type in the following program:

```
10 PRINT"X=";PEEK(36872),"Y=";PEEK(36873)
20 GOTO 10
```

Run the program, and you will see X and Y values of the digitizer scroll up the screen. As you move the arms, watch the values change. In order to calibrate your digitzer, place a grid of your own construction, or some graph paper, on the base.

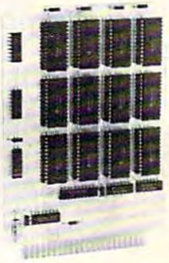
22-40-80 HIKE!



Expand your VIC to 80 columns.

Quantum Data's new Video Combo Cartridge brings you: 40 or 80 column display, plus 16K RAM and PROM socket. $299^00

With the Video Combo Cartridge from Quantum Data you can now have 40 or 80 column display, 16K RAM and PROM all in one cartridge. It comes set for 40 column Display compatible with the VIC video modulator and your home T.V. Then, when you are ready to upgrade to 80 columns and a video monitor, just make a simple, no-cost change inside the cartridge. Instructions are provided. Also provided is a socket for a PROM, 16K of memory and AC adaptor. If you don't need memory, then 80 columns can be yours for only $199.50. A listing of the driver software is provided at no charge. A programmed PROM containing this software is also available for $19.95.

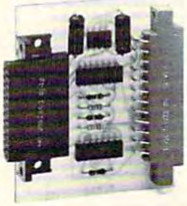


QDI expandor:

- Expands Basic user memory up to 24K in 8K steps
- PROMS may be mixed with RAM in 8K blocks
- 8K can be assigned to machine language area
- Plugs directly into VIC expansion port
- Low power, no additional power supply required
- Professional Quality, full buffering on all signals
- Small size: 6 x 4.5 inches.

16K expandor........ **$149.95**
24K expandor....... **$199.95**

VISA and MasterCard accepted.



QDI Printor RS-232 interface:

- Provides RS-232 voltage conversion for VIC serial port
- Allows use of a wide variety of RS-232 peripherals including printers, modems and voice synthesizers
- Low power CMOS circuitry requires no external power supply
- Small size: 2½ x 3 inches

Printor.............. **$49.95**

QDI Minimother:

- Adds 3 slots to the memory expansion port
- Removable card guides allow either boards or cartridges
- Requires no additional power supply
- Fused to protect VIC power supply from overload
- Simple plug-in installation

Minimother.......... **$69.95**

QUANTUM DATA, INC.
(714) 966-6553
DEALER HOT LINE (714) 754-1945
3001 Redhill Bldg. 4, Suite 105, Costa Mesa, CA 92626

Adjust the positions of the pots in the arms so that when the cursor is pointing to the upper left coordinate of the grid, the X and Y values are 0. If the numbers on the screen get larger rather than smaller, then reverse the outside pin connection on the affected pot from one end of the pot to the other.

What can you do with it? That's up to your imagination, but how about putting a map of the U.S. on the grid and creating a program that asks a student to identify states by placing the cursor over them? You might put the alphabet at the bottom of the grid and let your program recognize the characters by the position of the cursor.

In the digitizer described, data is being continuously sent to the VIC, and it is up to your program to decide what is valid data and what is garbage. In order to help make the system more "intelligent," you might add a push-button to the tablet and connect it to the game port or the User port; then make your program wait for the button to be pushed before accepting data from the digitizer.

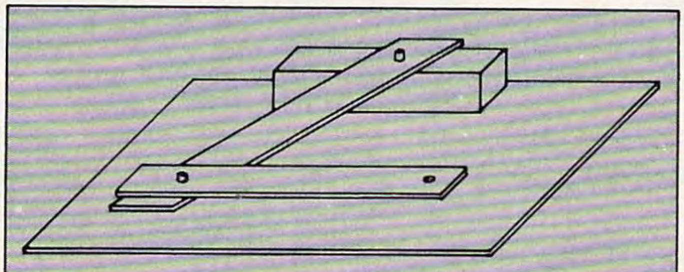


Figure 1.

References:

6560/6561 Video Interface Chip, Commodore International, 950 Rittenhouse Road, Norristown, PA, 19403

Personal Computing on the VIC, Commodore International, 950 Rittenhouse Road, Norristown, PA, 19403

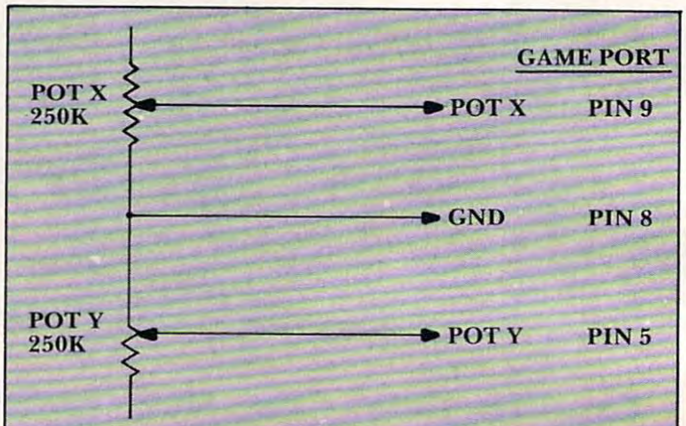
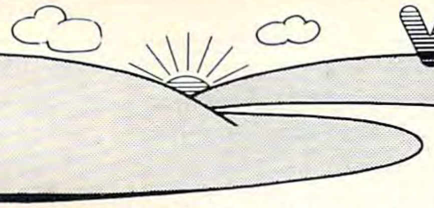


Figure 2.

©



VIC-20 SOFTWARE from MIDWEST MICRO

UN-WORD PROCESSOR 2 . . . $14.95

An easy-to-use word processor. Works with **any size VIC** — standard or expanded memory (3K, 8K, 16K) — and VIC-1515 or RS-232 printer*. Enter text . . . Edit . . . List . . . Save to tape or disk and append paragraphs for a final document. Allows control codes within the text for changing printer modes (e.g. **BOLD**). Menu selection of single or double space, form feed, print width, and number of copies to print.

TERMINAL-40 . . . $29.95

NEW, improved TERMINAL-40 program displays 40 uppercase characters per line (3x6 dot matrix) for easier reading. Enables VIC to emulate a standard terminal. Add a BIZCOMP or VIC modem (or RS-232 modem with interface*) and access SOURCE, TELENET, or any of the free Bulletin Boards around the country (list included). 300 baud; full or half duplex; supports control codes; screen dump to printer. Requires VIC-20 and 8K memory expansion.

TERMINAL-22 ($14.95) Same as TERM-40 except 22 character lines and full duplex only. Runs on standard 5K VIC.

GRAFIX MENAGERIE ($11.95). Demonstrate what your $300 miracle can do! Two-program set unleashes VIC's graphics. **SHOWOFF** contains Color Kaleidoscope, Arcade Critters, Custom Fonts, Electronics Schematic, and Music Notation. **PLOTTING** uses dot-plot and line-plot routines to make equations perform computer video-art on your screen. Change equation values and create your own interesting patterns. Plot routines may be easily included in your own programs.

BANNER/HEADLINER ($14.95). Two-program set makes GIANT headlines and banners on your printer. **HEADLINER** prints large characters across the page in three sizes. **BANNER** turns the characters sideway:, printing continuously down the paper roll. Up to three lines of text, nearly unlimited in length. (How about a ten-foot long "WELCOME HOME"?) For VIC-1515 or RS-232 printers.*

TICKERTAPE ($14.95). Interrupt-driven! Watch your message glide smoothly across the screen in color. Adds motion and interest to any message display. Position on any line, even mix with normal printing. Two built-in character sets: standard and BOLD (or use custom sets from our **LIBRARY VOL. 1**). Message capacity: @ 2K bytes.

LIBRARY VOL. 1 ($12.95). Add style to displays with six full sets of custom character fonts; UPPERCASE, lowercase, numerals, punctuation, etc. Simple to fancy styles. Upper and lower case stored separately; load upper alone to save space . . . load both for a full set. May be used with **TICKERTAPE.**

All programs on high quality digital cassette tape

VIC-PICS .... digitized pictures! $18.95

Nineteen fascinating high-resolution pictures to display on your VIC screen. Created by digitizing video camera images. Includes portraits, models, scenery, and much more. Over 16K points analyzed in each picture. Three styles: hi-contrast, dithered, and colorized. Compatible with both color and B/W sets.

GRAFIX DESIGNER ($14.95). Two-program set helps you design custom graphics characters. **GEN/EDIT** displays an enlarged 8x8 square; move the cursor around in it and turn dots on or off to form a character (holds 100). Erase, edit or recall at random. Load **DATAMAKER** when finished designing. Characters automatically become numbered data statements. Save them on tape just like a program. Instructions included for appending to any new or old program.

Build libraries of graphics . . . throw away the graph paper!

RS-232 INTERFACE ($49.95). Get more OUT of your VIC. Plug-in interface communicates with most standard serial printers and modems. Simply plug into User Port; needs no external power. Bi-directional operation. 90 day warranty. Full instructions for use. Includes M/L handshake "wedge."

VIC-20 is a trademark of Commodore Business Machines.

Dealer inquiries invited.

*RS-232 printers require an interface. See ours above.

MIDWEST MICRO Associates PO Box 6148 K.C. Mo. 64110

Include $1.25 for postage and handling.
Missouri residents add 4.6% sales tax.

NEW TOUCH-N-LIGHT PEN for the VIC 20

Imagine this . . .

1. draw pictures
2. play games
3. answer questions

and much, much more.

All without touching the keyboard. Just point the Touch-N-Light Pen at the screen and watch the VIC come alive. No skill necessary.

Game & Educational Programs available

Suggested list price $75.00
Introductory Offer—NOW ONLY

$59.95 (until Oct. 1)

N.Y. STATE RESIDENTS ADD TAX

Send Check or Money Order to:

Sunshine Peripherals
1229 East 28th Street
Brooklyn, N. Y. 11210

Dealer Inquiries Invited

VIC is a registered trademark of Commodore Business Machines

World's Most Cost Effective Development System

- Type your programs directly from your Commodore VIC-20 keyboard into the built-in 4 kilobyte ROM emulator
- Jumper direct to target ROM socket
- Test programs in circuit
- Burn them direct to EPROM with built-in EPROM programmer and power supply
- Burns & runs EPROMS for the Commodore VIC-20, too
- Software on EPROM included
- Comprehensive 25 page manual
- Fits EXP PORT or SUPER EXP Leaves user PORT free

PROMQUEEN CARTRIDGE COMPLETE ONLY $169.50

manual available separately for $20.00 credited on cartridge purchase
Jumper Cable, 2 ft: $18.50 with cartridge; 2732A EPROMS $12.50 each with cartridge

Gloucester Computer Bus Co. 6 Brooks Rd., Gloucester, MA 01930
617-283-7719

Shipping charges for all items extra, Mass. residents add 5% tax.
VISA AND MASTERCARD ACCEPTED

There are many things you can do with disk files to clean up some problem with their contents. But what do you do when you have a problem right now, and need to take action on the spot?

On-The-Spot Commodore Disk Fixes

Jim Butterfield
Associate Editor

Dual Personality

Remember that Commodore machines have intelligent disks. When you have a file open for reading or writing, there are two systems that are keeping track of it: your computer and your disk. Normally they will have the same information and be working on the same file; but if they ever get out of step, watch out!

Let's trace a simple activity and see how the problem might arise. You have this program which reads file A and writes file B. When you say RUN on your computer, the two files are opened. Your computer knows about both of them, and so does the disk.

Everything goes smoothly for a while, and then the program stops for some reason. It might be a syntax error or some other problem. Now, the files are still open. The computer and the disk both know they are open, but the computer program is stopped, and the files won't close without help from you. You should spot this: glance at the disk and you'll see that the drive light is still on. You can fix up the open files. But first, let's detour a little.

The Problem

You can abandon file A, the one you are reading, without serious harm. You may not have gotten all the information from the file, but you can always do another RUN – the file won't be harmed.

The file you are writing, file B, is a different matter. The data you have sent to the file is *not* all on the disk yet. Some of it is being held in a buffer within the disk, and that buffer will be written only when it's full or when the file is closed. If you take the diskette out of the drive now, the information won't be there. You must CLOSE the file.

Compounding The Problem

All you need to do is to say CLOSE 1, CLOSE 2, or whatever the logical file numbers are, and the write file will be properly closed. If you have BASIC version 4.0, you may say DCLOSE. But sometimes we don't think.

When we get an error like Syntax or Subscript or whatever, our first instinct is to look at the bad line. So we say LIST and see a line such as CHOSE 1 instead of CLOSE 1. Still no problem. But when we decide to fix it...

Most programmers know that the moment they change a line of a BASIC program, the variables are scrapped. It's not hard to see that the identity of a live file is going to be scrapped, too. So if we correct that bad line – which is the natural and instinctive thing to do – the computer has lost all its file information.

Now we have a bigger problem. The disk still knows it's got a couple of open files, but BASIC has forgotten about it. The drive light is still on, but we can't CLOSE the files in BASIC any more. The job gets tougher.

Even though the computer has lost track of what files are live on disk, there's a way to get those files closed. Here's how it's done: when you close the Command Channel to disk (secondary address 15), the disk closes all files. So to close those files, we type: OPEN 1,8,15:CLOSE 1. We haven't done anything on the Command Channel, but the act of closing it causes the disk to wrap everything up.

From Bad To Worse

But suppose you didn't fix it. What then? If you take your disk out, or start a new file-handling program, or shut the power off, your write file can never be complete. The information in the buffer never got written, remember? Your write file is no good, and the next time you take a CATALOG you'll see two odd things. First, the file length for the bad file will be shown as zero. Second, there will be an asterisk by the file type, e.g., *SEQ.

The natural thing to do when you see this type of bad file is to SCRATCH it. A word of advice: don't. SCRATCHing an asterisked file may create a poisoned disk. You are planting a bad seed on your disk which may wreck an important file or program two or three months from now.

If you want to get rid of the file, perform a COLLECT (sometimes called a disk Verify or Validate). That will do the job, and do it correctly.

All the advantages of Mupet, PLUS...

MUPET II

saves money and time too!

Mupet II offers all the advantages of Mupet, plus:

- Can protect against incorrect use of harmful disk commands.
- No more printer interface - background printing supported on IEEE, RS 232C and parellel ports included in controller.
- Programmable 32K RAM controller provides user defineable operating systems.
- Up to 16 computers may be connected to central printer and disk drives.

Integrate intra-office terminals for common data access allowing independent analysis for such functions as:

- Accounting
- Sales forecasts, budgets and orders
- Production scheduling
- Word processing

The only product available capable of serving Daisy Chained Commodore Computers up to 100 feet.

CMD

CANADIAN MICRO
DISTRIBUTORS LTD.

500 Steele Ave.,
Milton, Ontario L9T 3P7
416-878-7277

MUPET II
a small price for
huge capabilities, from:

$995.00

Looking Back

If you have doubts about any of your disks, you can always check them out with program Disk Checker (**COMPUTE!**, April 1982, #23). Start reading at the subheading Disk Test. The information you need starts there. If you invoke the program's Option 1, File Check, all files will be closely checked for any traces of "poison."

A series of seemingly natural actions on your part can lead to wrecking a file – or worse, endangering a disk. In case of program troubles, learn how to close your files and avoid trouble. ©

VIC Sticks

Jim Butterfield
Associate Editor

VIC joysticks have already been described by David Malmberg (*Home and Educational COMPUTING!*, Number 1). There's also a more formal description in the *Programmer's Reference Guide*. Let's pick up on joysticks one more time.

Collision With Keyboard

Try this. Hold the joystick over to the right and, while you're holding it, press the VIC number keys. You'll see that the odd numbers appear correctly on the screen, but the even digits are either missing or butchered. As soon as you release the joystick, the keyboard action returns to normal.

What's going on? In the interests of economy, Commodore has made one of the keyboard lines do "double duty" and test part of the joystick as well as perform its normal keyboard-checking functions. This is a two-way interference. We've seen that the joystick can interfere with the keyboard; in addition, the keyboard-servicing routines can make it impossible to check part of the joystick. The routines which read the keyboard are a special type called "interrupt" programs; this makes them hard to control.

Once you know the question, the answer isn't hard. To check the joystick completely, we must shut off part of the keyboard. If we need to keep the whole keyboard live, we must turn it back on again after checking the joystick.

We may shut down part of the keyboard with POKE 37154,127 and restore it with POKE 37154,255. We need to do this only to check the Right position of the joystick, which is done by looking at (PEEK(37152) AND 128).

Solving The Collision

What are our options? First, if we have a program that doesn't need the joystick's Right position, we can ignore the whole question.

If we have a program that doesn't need the keyboard, we can start with POKE 37154,127 as our first statement and restore the full keyboard only when the program ends. It won't matter that the keyboard is partly disabled during the program run. If the user/player stops the program rather than allowing it to end normally, however, he'll find his keyboard is acting badly. This isn't serious: the RUN/STOP-RESTORE key combination will fix everything up.

If we want to keep the keyboard live during play, each check of the Right position must include the whole set of three: disconnect part of the keyboard, check Right, reconnect keyboard. It will cost us a little more running time, but it's neater. It's not perfect, however: some keys will tend to hiccup if held down.

Machine language programs can solve everything, of course. They won't have a speed problem, so the keyboard can be quickly disabled and re-established. And the "hiccup" will go away if the interrupt is disabled during joystick checking; the interrupt routine won't jump in and be fooled during this check. Even in BASIC, however, you can do a competent job.

Difficult Diagonals

Joysticks are often inexact. You may think you are pushing Up, but you are slightly off true and the joystick might record both Up and Left.

The computer detects this, but your program must make a decision. If your program doesn't want diagonals, you must decide which of the two directions – say, Up and Left – is intended. It's easy to get the wrong one.

Directions are picked up as follows: UP is PEEK(37151) AND 4; DOWN is PEEK(37151) AND 8; LEFT is PEEK(37151) AND 16. The "fire" button is detected with PEEK(37151) AND 32, and RIGHT is checked as above, doing a partial keyboard disable and then working with PEEK(37152) AND 128. Each of these values becomes zero when the appropriate direction/button is activated.

You might write your program to check UP, then DOWN, then LEFT, etc., and to go to the appropriate action when you find an active position. If so, you'll miss the diagonals: UP/LEFT will exit on the UP condition and never check the LEFT, for example. This might be good for your particular game, but think of the human interface: the player might believe that he is pressing LEFT; the joystick is signalling LEFT and UP; and your program is reading only UP.

There's no absolute answer to this kind of question. Depending on the application, you can make certain choices. If you have on the screen a missile which is flying to the right, for example, you might choose to ignore all RIGHT/LEFT signals from the joystick and honor only UP/DOWN. Another approach is to design your game so as to use diagonals.

It's possible to write programs which "debounce" the joystick – that is, it must be returned to the center or rest position before a new signal will be accepted from it. This gives the effect of an impulse type of stick – action takes place only when the stick is moved.

A Simple Joystick Algorithm

One of the annoying things about joystick testing is that the input is logically inverted: the appropriate input is zero when activated, rather than zero when off. Although the information is the same either way, our minds don't like it. It seems more sensible to us to have bits turned on when the joystick is pushed; this allows us to extract combinations of bits with the logical AND function. A simple conversion statement which allows this is:

X = (NOT PEEK(37151))AND 60-((PEEK(37152)AND 128) = 0)

Don't forget to POKE 37154 with 127 before doing this test, or the Right position won't be detected properly; and remember to POKE 37154 back to 255 after the test.

After executing the above statement, variable X will contain complete information about the joystick. If nothing is active, X will be zero. If we want to check a change in the joystick status, we can see if the value of X has changed since last time.

We may now detect the various control positions with the appropriate AND statements:

Fire Button	**– X AND 32**
Left	**– X AND 16**
Down	**– X AND 8**
Up	**– X AND 4**
Right	**– X AND 1**

In each case, the result of the AND will be zero if this position is not active. Combinations can be used: for example, if we are interested in only UP and DOWN at this moment, we could check X AND 12.

When coding this, use parentheses liberally around the AND statements. For example, to test for Left, code: IF (X AND 16)<>0 THEN ... It won't work properly otherwise.

For motion, we can extract the Left/Right and Up/Down components with coding such as:

H = SGN(X AND 1) – SGN(X AND 16)
V = SGN(X AND 8) – SGN(X AND 4)

This produces value for H and V as follows: 0 for no motion in this direction; + 1 or -1 for motion in the appropriate direction.

Putting It All Together

The following simple program gathers together the joystick techniques we have discussed. It's a simple sketching program.

```
100 REM JOYSTICK PROGRAM
110 PRINT CHR$(147);CHR$(142) : REM
   CLEAR SCREEN
120 DATA 5,28,30,31,144,156,158,159
130 DIM C(7) : REM COLOURS
140 FOR J=0 TO 7:READ C(J):NEXT J
150 S=1:PRINT CHR$(C(S));
200 REM TEST JOYSTICK
210 POKE 37154,127
220 X=(NOT PEEK(37151))AND 60-((PEE
   K(37152)AND 128)=0)
230 POKE 37154,255 : REM  RESTORE K
   EYBOARD
240 IF (X AND 32)=0 GOTO 300 : REM
   NO BUTTON
250 IF B>0 GOTO 200 : REM DEBOUNCE
   BUTTON
260 B=1:S=S+1:IF S>7 THEN S=0
270 PRINT CHR$(C(S)); : REM CHANGE
   COLOUR
280 GOTO 200
300 B=0
310 H=SGN(X AND 16) - SGN(X AND 1)
320 V=SGN(X AND 4) - SGN(X AND 8)
330 PRINT CHR$(209);CHR$(17);CHR$(1
   7);CHR$(29);
340 FOR J=0 TO H+1:PRINT CHR$(157);
   :NEXT J
350 FOR J=0 TO V+1:PRINT CHR$(145);
   :NEXT J
360 GOTO 200
```

A few comments on the above coding. The Fire button is used to change color on the screen; the program debounces the button (using variable B) so that holding down the button does not cause a continuous color change.

Lines 310 and 320 compute reverse values of V and H compared to the algorithms given previously. In this case, we're computing an inverse activity – how many places to back the cursor up for a given position.

Lines 330 to 350 are rather "gimmicky"; we force the cursor right and down, and then count our way back to the desired position using cursor-left and cursor-up characters. The intent here is to illustrate the use of the V and H directional values. You may find other ways to achieve the same objective when you write your own programs.

The program prints the "ball" character, CHR$(209); you can switch to another character by making the appropriate change in line 330.

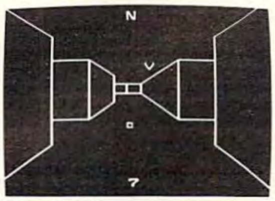
The joystick can indeed be interfaced with your program; all you need is to learn a few rules. You must set your own objectives as to how the joystick best interfaces with the user in your application. Once you have learned the mechanics, it's not hard to make everything work. ©

VIC-20®
COMMODORE

VIC-20®
COMMODORE

VICTORY
SOFTWARE, INC.®

WE SELL FUN!™



TREASURES OF THE BAT CAVE **$14.95**
Battle the vampire bats as you search their cave for gold bullion. Fast, real time action will keep you playing for hours. Of course, you are in a different cave every time you play.

COSMIC DEBRIS **$14.95**
This highly addictive arcade type game will keep you battling the aliens for days.

GRAVE ROBBERS **$13.95**
Introducing the first GRAPHIC ADVENTURE ever available on the VIC-20! With realistic audio-visual effects, you explore an old deserted graveyard and actually see the perils that lie beyond.

NIGHT RIDER **$12.95**
High speed night time driving simulator.

STREET SWEEPERS **$14.95**
Gobble up all of the dots in the maze before the ensuing nemesis gets you. The maze is different every time, and if you succeed in getting all the dots, you get progressively harder mazes to complete as your skills increase. Does this sound like Pac Man? It isn't! Highly recommended and extremely addicting.

Send for free catalog. All programs fit in the standard VIC memory, and can be controlled from the keyboard. All programs on cassette tape.

Ordering—Please add $1.50 postage and handling per order.
PA residents please add 6% sales tax.
Mastercard or VISA Charges—Please send your account number and card expiration date.

master charge

VISA

VICTORY SOFTWARE INC.
2027-A S. J. Russell Circle, Elkins Park, PA 19117
(215) 576-5625

VIC-20®
COMMODORE

VIC-20®
COMMODORE

STAR ATTRACTIONS

By Wunderware.
Presenting these games and many more for your VIC-20.

★**THE MAD PAINTER** This game is a little unique and a lot of fun. You control a paint brush, moving it around a colorful maze. Your job is to paint the entire maze. This is not as easy as it sounds, because in the maze with you are two voracious Bristle Biters (they love paint brushes). Occasionally you will receive a visit from an Invisible Stomper who leaves footprints in your fresh paint. Requires joystick. **$9.95 cassette, $12.95 disk**

★**GALAXY INVASION** Deeper and deeper you go into the hostile alien galaxy. Gain points by maneuvering your ship to rescue men as they drift by. Deep space fuel stations so you can continue your trek, asteroids and space mines which you must avoid at all cost, are all a part of this fun and exciting one player game. Joystick required. **$9.95 cassette, $12.95 disk**

★**SNAILBAIT!** Don't let the name fool you! This game is fast arcade action all the way! Your job is to protect a flower bed from an onslaught of snails. You are armed with a spray gun full of insecticide. As they zip across the field on their way to chomp down your flowers, the snails leave eggs which you had better destroy too! For one player, requires joystick. **$10.95 cassette, $13.95 disk**

• Send 50¢ (refundable with order) for catalog and free program listing.
• Price includes Postage & Handling. • Catalog is included with order.
• Foreign orders & COD's: Please add $3.00. • Prices are subject to change without notice. • Send check or money order to:

wunderware
P.O. BOX 1287, JACKSONVILLE, OR 97530 (503)899-7549

VIC-20 is a registered trademark of Commodore Business Machines.

VICWARE
by MicroSpec Ltd.

Colorful, Exciting Software for the Commodore VIC Computer!

Utility applications make your VIC earn its living while the games allow you and your family to enjoy it!

Moon Lander	9.95	Home Finance	12.95
Reversal 1	9.95	*Reversal 2	13.95
Bricks	9.95	Wrap	9.95
Blackjack	9.95	Capture	9.95
Tennis	9.95	Target Moon	9.95
Slots	9.95	Math Drill	9.95
Bombs Away	9.95	Alien Raiders	9.95
*Biorhythm	9.95	*Home Inv.	16.95
*Data Manager	19.95	*Portfolio Mgr.	19.95
Starfighter	13.95	*Hangman	12.95
*Auto Expense	19.95	Spelling 2-6	9.95
Typing Drill	12.95	(Specify Grade)	

*Requires Memory Expansion

Write for FREE Catalog

VISA

2905 Ports O'Call Court
Plano, Texas 75075
(214) 867-1333

MasterCard

VISA and MASTERCARD Accepted

VIC-20
Assembler

EDITOR Used to create and update the assembly language source-program library, it syntax-checks each statement as it is entered, allowing only valid mnemonics and addressing modes. Menu options are Create, Print, Load, Delete, Insert, Change, and Save. Automatic tabbing, full use of cursor/insert/delete keys. Truly "human engineered" software.

ASSEMBLER Inputs source-program segments created with The Editor. It lists the original source as well as the generated machine language in hexadecimal or decimal, and optionally produces machine language load segments for The Loader. Printer output is supported.

LOADER Inputs and links the load segments from the library created by The Assembler. The machine language program is POKEd into memory and BASIC's top-of-memory pointer may be automatically reset to a value specified in the source program. Machine language and BASIC may be combined in the same program through use of The Loader.

SIGNIFICANT FEATURES

Programs of many hundred statements may be assembled on even the 5K VIC-20. Long programs are assembled in segments, each segment capable of containing up to 100 statements on the 5K VIC-20, and several hundred statements on expanded systems. Any number of labels may be used and labels are variable up to 75 characters in length. Out-of-memory and symbol table overflow errors never occur.

The BYT instruction may define literal and hex strings of any length as well as decimal data and address constants in 6502 format.

Operand fields may include complex algebraic expressions utilizing the + - * / ↑ (integer and fractional powers supported), & (logical AND), ! (logical OR) operators, and any number of terms in combinations of decimal, hexadecimal, program counter, symbolic address and literal formats. Negative numbers generate two's complement values.

This is a quality piece of software, combining maximum capability with ease of use.

$24.95 + $1.00 postage & handling. VISA/MC accepted with $1.00 service charge and inclusion of acct. # and expiration date. Send payment to French Silk, P.O.Box 207, Cannon Falls MN, 55009.

VIC-20 is a registered trademark of Commodore Business Machines.

French Silk smooth ware

In this, the conclusion of a three-part series, several demonstration programs teach the concepts (and show off) the new Atari GTIA graphics chip. The GTIA costs nothing if your machine is still under warranty. If you have an older Atari, your nearest authorized center should be stocking it by now and will install it for about $60 according to Atari. If you just bought your computer. it's in there.

Atari Video Graphics And The New GTIA: *Part 3*

Craig Chamberlain
Birmingham, MI

Welcome back to our discussion of Atari playfield graphics and the exciting new GTIA chip. In Parts 1 and 2 I presented definitions of various terms related to graphics, explained the normal graphics modes, and then introduced the three new modes provided by the GTIA. Specifically, these new modes are:

MODE	DESCRIPTION
9	16 shades of one color
10	8 indirected colors
11	16 colors (one luminance)

Here are several programs in Atari BASIC to demonstrate how these new modes might be put to use. But first, let's tie up a few loose ends from the previous articles.

We used a standard method to show bit designations in the first parts of this article. If you are not familiar with this convention, here's how it works. Any given memory location or hardware address consists of one byte made up of eight binary units called bits. These bits are numbered zero to seven and are frequently shown as D0, D1, D7, etc. Individually, each bit can have two values, zero or one, but from the viewpoint of a byte, they take on quite different values known as "powers of two." For example, D3 means "two to the power of three," which also means "the number two used as a factor three times." Two times two times two is eight, so if we wanted to turn on only bit three in a given hardware register, we would POKE it with an eight. If we want to turn several bits on, we must add all the proper values together.

BIT	VALUE
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Mode 11 can be invoked by turning on bits six and seven of GPRIOR, location 623 (decimal). Thus we would POKE 623 with 64 + 128, which is 192. This brief explanation should help you deal with the memory locations and hardware registers described in the previous articles. Now for a review of the primary graphics statements of Atari BASIC and some special notes about the GTIA.

Graphics Statements

GRAPHICS aexp

This statement is the same as OPEN #6, 12 + 16,aexp, "S:", and tells the screen handler to open the screen to one of 12 modes. The number "aexp", which means "arithmetic expression," can range from zero to 11. Characteristics of these modes are explained in chapter nine of the *Atari BASIC Reference Manual.*

Some modes allow split screen configurations, which means that a text window appears at the bottom of the screen. Of course, mode zero does not allow a text window because mode zero is the text mode, although you can experiment with POKE 703,4. Modes one through eight do support text windows, and the only way to get a full screen (no text window) in one of these modes is to add 16 to aexp in the Graphics statement. When using a full screen mode, Atari BASIC forces a mode zero if it has to print normal text. It is impossible to use these full screens in the immediate programming mode because the "READY" prompt forces the mode zero screen.

Due to technical reasons explained last month, modes 9, 10, and 11 do not normally allow text windows. You can fool the operating system into giving you one of these modes with a text window by asking for mode 8 and then doing a couple of POKE statements, like this:

MODE	POKES
9	POKE 87,9: POKE 623,64
10	POKE 87,10: POKE 623,128
11	POKE 87,11: POKE 623,192

Location 87, known as DINDEX, tells the operating system the current mode and is used in the computation of row and column addresses for plotting, so any number from nine to eleven will give the same results. Unfortunately, the text win-

JANUARY 15, 1982.

FERNANDO HERRERA

becomes the *first* person ever to win the

ATARI® STAR AWARD

His program was voted number *one!*

Now...

He proudly announces

FIRST STAR SOFTWARE

"WHEN BEING FIRST COUNTS... WE'RE NUMBER ONE"

A new software company providing innovative, quality games for the star player, retailer and distributor.

FIRST STAR SOFTWARE
22 E.41ST STREET
NEW YORK, N.Y. 10017
(212) 532-4666

Retailer and distributor inquiries welcome.

ATARI® is a registered trademark of ATARI, Inc.

ASTRO CHASE
-ultimate graphics-
available Oct. '82
THERE IS NO ESCAPE!

dow obtained by this method looks weird. The only way to get a real text window is to use a display list interrupt, discussed later.

If you add 32 to aexp, the screen will not be cleared when the new mode is requested.

Finally, the CTIA and GTIA support five other modes which the operating system does not recognize. They are the eight by ten matrix character version of mode zero, the multi-color text character modes, and the single scan line versions of modes six and seven, for 160 by 192 plotting in one or three colors. The only way to access these modes is to write a custom display list, which has been discussed in previous **COMPUTE!** articles.

COLOR aexp

This specifies the playfield that will be used for PLOT and DRAWTO statements, until changed by another COLOR statement. It does not in any way change any of the color/luminance registers for the various playfields! The range of aexp depends on the number of different playfields available in the current graphics mode. This still holds true for the new GTIA modes. For example, a COLOR 2 in mode 9 means that future plotted points will be rather dark, whereas bright lines will be drawn after a COLOR 12.

In mode 11, COLOR 5 chooses a purple color, as indicated by the chart in part one of this article. For all modes, COLOR 0 (zero) is the background or "erasing" color. Normally, the operating system wants you to specify the playfield each time you write to the screen, but Atari BASIC automatically tells the operating system which playfield you have chosen every time you use PLOT or DRAWTO. Incidentally, the data part of the COLOR statement is stored in memory location 200 (decimal), but I would not recommend using that.

One other note. To be technically accurate, COLOR 1 corresponds to playfield zero, COLOR 2 means playfield one, and so on.

POSITION aexp1,aexp2

This statement moves the graphics cursor to the location on the screen designated by the two numbers, according to the Cartesian coordinate system. No range checking is done.

PLOT aexp1,aexp2

This is the same as POSITION aexp1,aexp2: PUT #6,color where "color" is the playfield type chosen by the most recent COLOR statement. You will get an error number 141 if you try to PLOT outside the bounds of the screen. All three new modes have resolution of 80 by 192.

DRAWTO aexp1, aexp2

Essentially, this is the same as PLOT except that a line is drawn from the most recently plotted point to the new point indicated by aexp1 and aexp2. You can also do this with an XIO 17,#6,0,0,"S:". See page 54 of the *Atari BASIC Reference Manual* to see how XIO 18 can be used to fill areas with a playfield.

LOCATE aexp1,aexp2,avar

I don't know why, but nobody seems to know about this statement. It could be considered the reverse of PLOT. Instead of putting a playfield point at a certain location on the screen, this statement returns to you, in the arithmetic variable "avar", the playfield number of the point at location aexp1,aexp2. This playfield number will be the same as the value of COLOR that was in effect when the point was plotted. LOCATE is actually quite handy. It is very useful in games where collisions occur between differently colored players, but it has many other applications, too. LOCATE is the same as POSITION aexp1,aexp2: GET #6,avar.

SETCOLOR aexp1,aexp2,aexp3

This is the statement which changes the color and luminance of a playfield register. The number aexp1 designates which playfield register is being changed, and is related to the number in the COLOR statement in the following way:

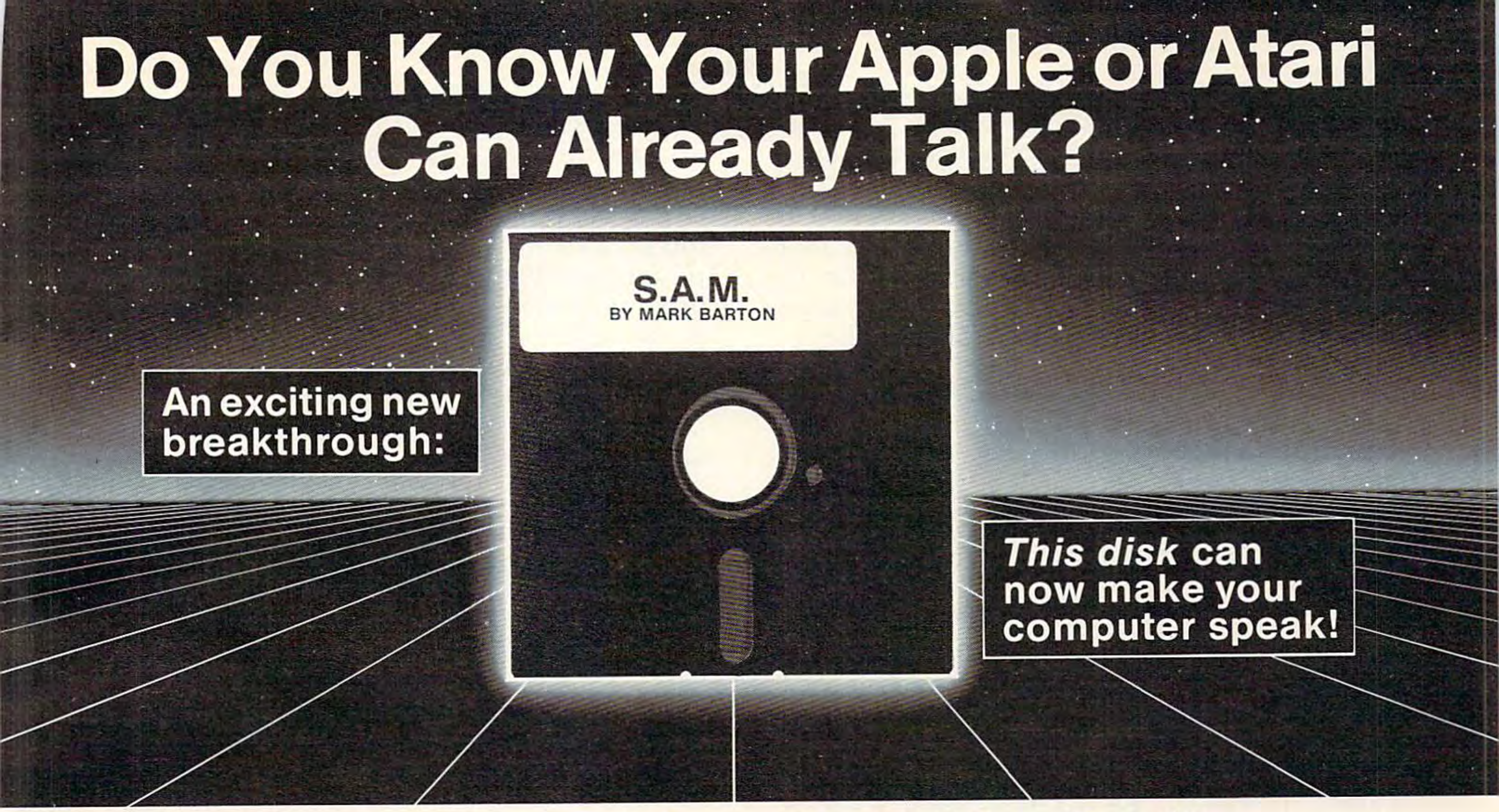
COLOR	SETCOLOR (playfield number)
1	0
2	1 (also used for luminance in modes zero and eight)
3	2
–	3 (used only in four color text modes one and two)
0	4 (background, or border in modes zero and eight)

The value for aexp2 is a number from zero to 15 which specifies one of 16 colors. See the chart in part one of this article, or on page 50 of the *Atari BASIC Reference Manual*, to find which numbers go with which colors. The luminance is chosen by aexp3, which can range from zero to 15, but only eight true luminances can be selected. A value of zero here gives the same luminance as one, two the same as three, and so on. The larger the number, the greater the luminance.

Remember that modes 9 and 11 do not use color indirection or the playfield registers, so SETCOLOR has little use in these modes. It can be used to set the background color/luminance in these two modes, but that's about it.

Now for mode 10. This mode uses the player/missile color/luminance registers, which cannot be accessed using SETCOLOR. An equivalent POKE statement must be used instead. The location to POKE is similar to the aexp1 of SETCOLOR. The shadows of the playfield registers run from locations 708 (decimal) to 712. The value to POKE contains the color and luminance information and is a combination of aexp2 and aexp3. This value is

Do You Know Your Apple or Atari Can Already Talk?


S.A.M.
BY MARK BARTON
An exciting new breakthrough:
This disk can now make your computer speak!

It's the **Software Automatic Mouth — S.A.M.**
The brand new, all-software, high quality speech synthesizer
from **DON'T ASK.**

S.A.M. gives you:
Unlimited vocabulary
Full inflection at your control
Effortless access from BASIC
Separate pitch and speed control
Thorough, instructive owner's manual
Easy-to-learn phoneme spelling system
And automatic English-to-speech conversion
Elaborate internal pronunciation rules for natural-sounding speech

S.A.M. for the APPLE II/II+ includes:
8 bit digital-to-analog converter and audio amplifier on a board (amplifies the sound of all your APPLE games).

- **S.A.M.** on disk
- Complete documentation
- English-phoneme dictionary

Only $124.95

APPLE is a trademark of APPLE COMPUTER, INC.

S.A.M for the ATARI 400/800 includes:

- **S.A.M.** on disk
- Complete documentation
- English-phoneme dictionary

Only $59.95

Note: On the ATARI, screen blanks during vocal output.

ATARI is a trademark of ATARI INC.

Hear S.A.M. at your favorite computer store today.
For more information, contact:

DON'T ASK
Computer Software
2265 Westwood Blvd., Ste. B-150
Los Angeles, California 90064
(213) 397-8811

- For a copy of the owner's manual, specify computer and send $2.00 to **DON'T ASK.**
- You can order **S.A.M.** directly from **DON'T ASK.** Add $2.00 for shipping and handling to your check or money order (or order C.O.D.).

Dealer inquiries welcome

Talk is cheap

the sum of 16 times the color number, plus the luminance. In effect, SETCOLOR X,Y,Z will do the same as POKE 708 + X, 16*Y + Z. If you want to change the player/missile color/luminance registers, which run from locations 704 to 707, use the same procedure of multiplying the color by 16 and then adding the luminance. Refer to part one of this article for a chart that tells which COLOR numbers match with which registers.

Some Lively Demos

Now comes the good part, where the action is! If your Atari computer has a GTIA in it, here are some programs to show off the talents of this remarkable chip.

How to put 16 colors on the screen? It could be done in one line:

```
GRAPHICS 11: FOR K=0 TO 79: COLOR K: PLOT
K,0: DRAWTO K,191: NEXT K: FOR K=0 TO 0 STEP 0:
NEXT K
```

The endless loop is necessary to prevent Atari BASIC from printing a "READY" prompt which would force mode zero. To make the vertical color bands wider, change the COLOR K to COLOR K/5. To see 16 shades, change the GRAPHICS 11 to GRAPHICS 9.

A fancier way of showing 16 shades is found in Program 1. After drawing the shades, the background color is rotated through all 16 colors.

Program 3 randomly draws lines in 16 colors. You can make these colors appear darker or more pastel by changing the luminance of the background. Please note that mode 11 is the only mode in which the background is set by the operating system to a luminance of six. All other modes have backgrounds of color/luminance zero (black).

Program 2 demonstrates the color indirection capabilities of mode 10. Location 20 is the lowest counter of the realtime clock, so it is always changing. Continuously PEEKing this location and POKEing the value into a color register gives a nice "throbbing" color spectrum effect.

How about a doodling program that lets you draw in 16 colors using the joystick? Program 4 does this in only three lines of Atari BASIC code! Press the joystick trigger to change colors.

Program 5 is a really beautiful color kaleidoscope generator, considering it is only four lines long. It's not something you will spend hours watching, but it can produce some nice pictures. Try changing the K = I + J in the second line to K = I for a different picture. Or you can reverse the direction of the main loops, as in FOR I = 31 TO 1 STEP -1. If you change the J loop (note that it starts at zero, FOR J = 31 TO 0 STEP -1), you will also want to change the H loop (FOR H = 1 TO 3 STEP 1).

To show 256 colors on the screen all at once, use Program 6. This program does not show the colors. Rather, it produces a single line which you can ENTER from disk or cassette. This single line performs all the magic. What is also neat about this program is that when you ENTER the line in, the program already in memory is untouched. If you examine Program 6, you will see that it writes a line to the chosen device, but the line has no line number in front of it. When you ENTER this line, it is the same as typing it in the immediate mode. When Program 6 asks for a device specification, respond with C: for cassette or D:filename for disk.

I included the assembly source code and Atari BASIC installation routine for a display list interrupt service routine (Program 8) that creates a text window on modes 9, 10, or 11. An interrupt is requested at the last mode line of the graphics mode portion of the screen. The service routine takes the value of GPRIOR, sets the GTIA mode select bits to zero, and stores the result in PRIOR, the hardware register. PRIOR gets reset to the value of GPRIOR as part of the vertical blank service routine. The routine also stores a zero into the background hardware register. This was necessary to fix a conflict in mode 11. Setting the luminance in 712 also changes the border around the text window. But this "fix" created another problem in mode 10. For mode 10, change the fourteenth DATA element, which normally should be a zero, to be the same as the number POKEd into 704.

The service routine is written using relocatable code, so you can put the routine anyplace in memory simply by changing the assignment of ADDRES in the second line. It is currently set to start at the beginning of page six. The first three lines of Program 7 actually install the routine. The fourth line just draws a picture for purposes of demonstration. Notice the luminance change of the colors when 712 is POKEd.

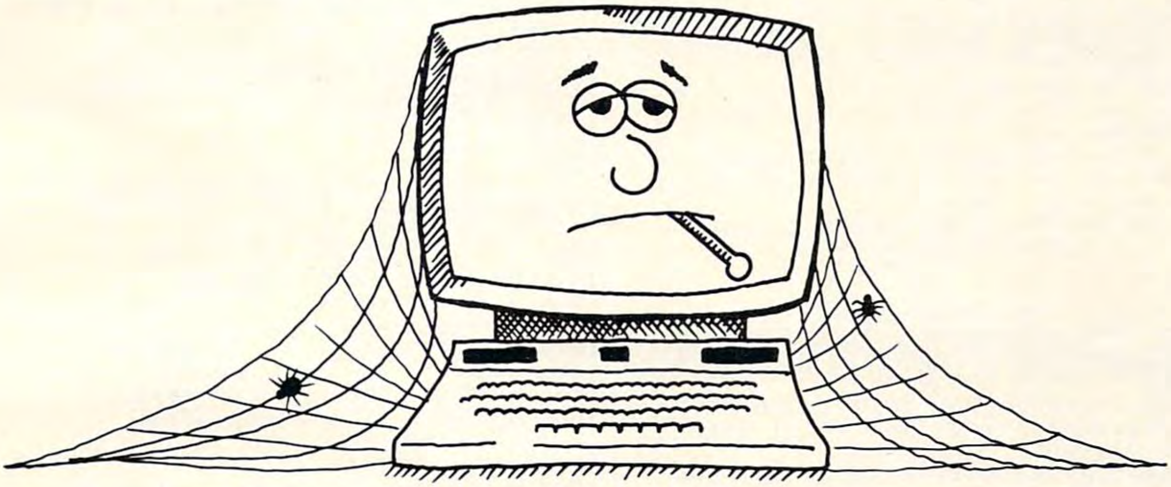
My routine shares the problem of many display list interrupt service routines; keyboard clicks can affect the display. Obviously this routine is suitable only for programs that do not accept keyboard input (use the joystick or PEEK the hardware keycode register 764 directly) or use serial I/O (the vertical blank routine is abbreviated and PRIOR does not get reset).

Program 1.

```
10 GRAPHICS 9:FOR K=1 TO 10 STEP 2:FOR J
   =0 TO 15:COLOR J:PLOT 0,K*16+J+1:DRAW
   TO 79,K*16+J+1
20 PLOT 0,K*16-J:DRAWTO 79,K*16-J:NEXT J
   :NEXT K
30 FOR K=1 TO 255 STEP 16:POKE 712,K:FOR
    J=1 TO 500:NEXT J:NEXT K:GOTO 30
```

QUICKSOFT™

CURES THE #1 TERMINAL ILLNESS:



S.D.S.

(SLOW DELIVERY SYNDROME)

QUICKSOFT'S cure doesn't come in a bottle, but it does come in handy when your computer needs software delivered promptly.

24 HOUR SHIPPING: QUICKSOFT is the software-house with a difference: **SPEED!!** When you place a phone order with **QUICKSOFT we guarantee that it will be shipped within twenty-four hours.***

WIDE SELECTION: QUICKSOFT carries thousands of software disks, tapes and cartridges for every purpose. We have business systems to cure those overflowing files, educational disks to relieve the worry of a D in chemistry, and adventure games to aid a case of the blues.

QUICKSOFT HAS THEM ALL, and right now we're having specials on these items from our "most wanted" list:

Wordrace (D)	**$19.95**
Bug Attack (T,D)	**23.95**
Canyon Climbers (T,D)	**23.95**
Protector (T,D)	**23.95**
Shooting Arcade (T,D)	**23.95**
Tumble Bugs (D)	**23.95**
Pathfinder (D)	**27.95**
Ulysses and the Golden Fleece (D)	**27.95**
Pac-man (cart)	**35.95**
Text Wizard (D)	**79.95**

Prices effective through October 16, 1982.

FREE DELIVERY: QUICKSOFT offers free delivery on *all* orders of $50.00 or more. We not only ship within 24 hours but we also pay the freight! (For orders of less than $50 add $1.50 for shipping and handling.)

CALL TOLL FREE ANYTIME. QUICKSOFT'S toll free number is available anytime of the day or night. For placing an order or for requesting our **FREE CATALOG** simply dial:

1-800-547-8009

IN OREGON CALL 1-342-1298

Please specify for Apple or Atari computer on your order.

***GUARANTEE: QUICKSOFT** guarantees shipment within 24 hours on each phone order. If the item ordered is not readily available and cannot be shipped on time, you will receive an additional 5% OFF the regular **QUICKSOFT** price.

VISA-MASTERCHARGE and C.O.D. Welcome

QUICKSOFT™

Department 3, P.O. Box 10854, Eugene, OR 97440

Program 2.

```
10 GRAPHICS 10:FOR K=705 TO 712:POKE K,1
   2:NEXT K:FOR K=0 TO 79:COLOR (K+4)/10
   :PLOT K,0:DRAWTO K,191:NEXT K
20 FOR K=704 TO 712:FOR J=1 TO 300:POKE
   K,PEEK(20):NEXT J:NEXT K:GOTO 20
```

Program 3.

```
10 GRAPHICS 11:FOR K=1 TO 124:COLOR K:DR
   AWTO RND(1)*79,RND(1)*191:NEXT K:GOTO
    10
```

Program 4.

```
10 GRAPHICS 11:DIM SX(15),SY(15):FOR K=5
    TO 14:READ X,Y:SX(K)=X:SY(K)=Y:NEXT
   K:X=40:Y=96:COLOR 1
20 PLOT X,Y:X=X+SX(STICK(0)):X=X+(X<0)-(
   X>79):Y=Y+SY(STICK(0)):Y=Y+(Y<0)-(Y>1
   91):IF STRIG(0) THEN 20
30 C=C+1-15*(C=15):COLOR C:GOTO 20:DATA
   1,1,1,-1,1,0,0,0,-1,1,-1,-1,-1,0,0,0,
   0,1,0,-1
```

Program 5.

```
10 GRAPHICS 10:FOR I=705 TO 712:POKE I,P
   EEK(53770):NEXT I:FOR I=1 TO 31 STEP
   1:C=C+1-9*(C=8)
20 POKE 704+C,PEEK(20):FOR J=0 TO 31 STE
   P 1:COLOR INT(RND(1)*15)+1:K=I+J:J3=J
   *3:K3=K*3:J8=J+8:J71=71-J
30 PLOT K+7,J3:DRAWTO K+7,191-J3:PLOT 72
   -K,J3:DRAWTO 72-K,191-J3:FOR H=3 TO 1
    STEP -1
40 PLOT J8,191+H-K3:DRAWTO J71,191+H-K3:
   PLOT J8,K3-H:DRAWTO J71,K3-H:NEXT H:N
   EXT J:NEXT I:POKE 77,0:GOTO 10
```

Program 6.

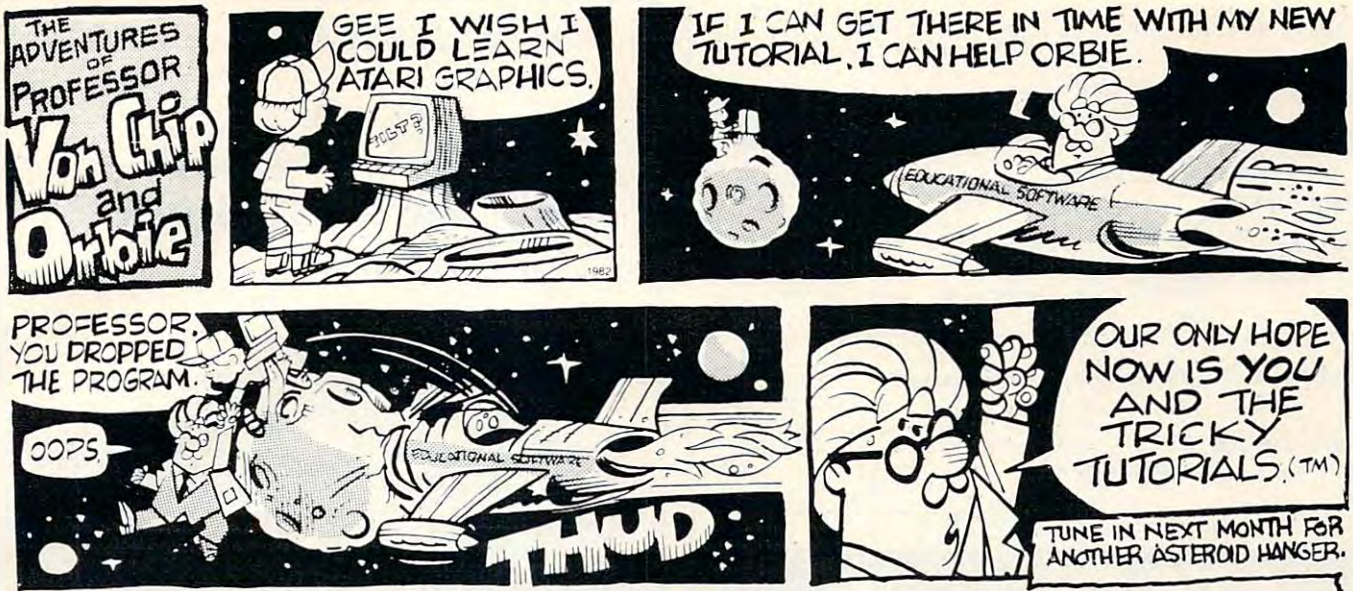
```
100 IF PEEK(87) THEN GRAPHICS 0
105 ? CHR$(125):? "GTIA DEMONSTRATION":?
     "by Tom Giese  4/15/82":?
110 ? "This program creates an ATASCII f
    ile"
120 ? "for ATARI BASIC.  The file consis
    ts"
130 ? "of one line which will produce tw
    o"
140 ? "hundred fifty six colors on your"
150 ? "screen if you have a GTIA install
    ed.":?
170 DIM S$(120):? "Please enter device s
    pecification."
180 INPUT S$:IF S$="" THEN 180
190 ? :TRAP 260:OPEN #1,8,0,S$
200 ? #1;"GR.9:F.K=0TO79:C.K/5:PL.K,0:DR
    .K,191:N.K:K=USR(ADR(";
210 PUT #1,34:FOR K=1 TO 15:READ P:PUT #
    1,P:NEXT K:PUT #1,34
220 DATA 173,11,212,10,229,20,41,240,141
    ,26,208,208,243,240,241
230 ? #1;"))":CLOSE #1:? "File has been
    written."
245 POSITION 2,19:? "ENTER ";CHR$(34);S$
    ;CHR$(34)
250 POSITION 2,15:? "Now press the RETUR
    N key if"
255 ? "you want to ENTER the file.":NEW
260 STATUS #1,P:? "I/O ERROR ";P:END
```

Program 7.

```
10 POKE 54286,0:GRAPHICS 8:POKE 87,11:PO
   KE 623,192:POKE PEEK(560)+256*PEEK(56
   1)+166,143
20 ADDRES=1536:POKE 54286,64:FOR K=0 TO
   18:READ P:POKE ADDRES+K,P:NEXT K:P=IN
   T(ADDRES/256):POKE 513,P
30 POKE 512,ADDRES-256*P:POKE 54286,192:
   DATA 72,173,111,2,41,63,141,10,212,14
   1,27,208,169,0,141,26,208,104,64
40 FOR K=0 TO 159:COLOR K/10:PLOT 0,K:DR
   AWTO 79,K:NEXT K:POKE 712,6:STOP
```

Program 8.

```
0000        10          .PAGE
            11 ;
            12 ;necessary operating system and
hardware equates
            13 ;
026F        14 GPRIOR   =    $026F
;GTIA priority control (shadow)
D01A        15 COLBK    =    $D01A
;background color register
D01B        16 PRIOR    =    $D01B
;GTIA priority control (hardware)
D40A        17 WSYNC    =    $D40A
;horizontal blank synchronization
            18 ;
            19 ;
0000        20          *=   $0600
            21 ;
            22 ;this service routine for the
display list interrupt
            23 ;can be placed anywhere in RAM,
and was placed on page six
            24 ;only for purposes of
demonstration
            25 ;
            26 ;begin interrupt service routine
code
            27 ;
            28 ;save contents of accumulator
            29 ;
0600 48     30          PHA
            31 ;
            32 ;get the multicolor player, fifth
player, and priority bits
            33 ;
0601 AD6F02 34          LDA  GPRIOR
            35 ;
            36 ;force the GTIA mode select bits
to zero but save the other bits
0604 293F   37          AND  #$3F
            38 ;
            39 ;wait until next scan line for a
nice clean change
            40 ;
0606 8D0AD4 41          STA  WSYNC
            42 ;
            43 ;change hardware register until
VBLANK
            44 ;
0609 8D1BD0 45          STA  PRIOR
            46 ;
            47 ;reset COLOR4 to zero (for modes
9 and 11)
060C A900   48          LDA  #$00
060E 8D1AD0 49          STA  COLBK
            50 ;
```


THE ADVENTURES OF PROFESSOR Von Chip and Orbie
GEE I WISH I COULD LEARN ATARI GRAPHICS.
1982
IF I CAN GET THERE IN TIME WITH MY NEW TUTORIAL, I CAN HELP ORBIE.
EDUCATIONAL SOFTWARE
PROFESSOR, YOU DROPPED THE PROGRAM.
OOPS.
EDUCATIONAL SOFTWARE
THUD
OUR ONLY HOPE NOW IS YOU AND THE TRICKY TUTORIALS.(TM)
TUNE IN NEXT MONTH FOR ANOTHER ASTEROID HANGER.

TRICKY TUTORIALS (tm)

There are many things that the ATARI computers can do either better, or easier than other small computers. The following series of programs is designed for anyone who is at least familiar with BASIC programming. What each tutorial offers is similar to an extensive magazine article with all discussion in as simple language as possible, plus you get MANY examples already typed in and running. The instruction manuals range from 10 to 50 pages, and some tutorials fill up a complete tape or disk. There is little overlap in what is taught, so anyone wanting to know all they can should buy them all (my banker thanks you). ATARI buys these from us to use in training their own people! Rave reviews have been published in ANTIC, ANALOG, CREATIVE COMPUTING, and even INFOWORLD. You trust INFOWORLD, don't you?

TT #1: DISPLAY LISTS—This program teaches you how to alter the program in the ATARI that controls the format of the screen. Normally, when you say "Graphics 8", the machine responds with a large Graphics 8 area at the top of the screen and a small text area at the bottom. Now, you will be able to mix various Graphics modes on the screen at the same time. The program does all of the difficult things (like counting scan lines). You will quickly be able to use the subroutines included in your own programs.
16K Tape or 24K Disk. **$19.95**

TT #2: HORIZONTAL/VERTICAL SCROLLING—The information you put on the screen, either GRAPHICS or TEXT, can be moved up, down, sideways, or diagonally. We provide the basic methods and leave the rest up to your skill and imagination. Includes 18 examples to get you started, with several using a small machine language subroutine for smoothness.
16K Tape or 24K Disk. **$19.95**

TT #3: PAGE FLIPPING—Now you don't have to redraw the screen every time you change the picture or text. You will learn how to have the computer draw the next screen you want to see while you are still looking at the previous screen, then flip to it instantly. You won't see it being drawn, so a complicated picture can seem to just appear. Depending on your memory size and which graphics or text modes you are using, you can instantly look at up to 50 pages. The basic method takes only 9 lines and the usefulness is infinite.
16K Tape or 24K Disk. **$19.95**

TT #4: BASICS OF ANIMATION—This program shows you how to animate simple shapes (with some sound) using the PRINT and PLOT commands, and it also has a nice little PLAYER/MISSILE GRAPHICS game you can learn from. The P/M example is explained and will get you started on this complicated subject (more fully explained in TT #5). This would be an excellent way to start making your programs come alive on the screen with movement! Recommended for beginning users.
16K Tape or 24K Disk. **$19.95**

TT #5: PLAYER/MISSILE GRAPHICS—Learn to write your own games and other animated applications! The tutorial begins with many small examples that compliment the 50 page manual, then gradually builds up to a complete game where everything you need to know is fully explained. Also included are two machine language utilities that you can use to animate Players with from BASIC. Next we include two of the best editors currently available; one for editing playfield shapes (backgrounds); and one to edit your players, and all in glorious Technicolor!! Everything except the two editors run in 16K Tape or 32K Disk. **$29.95**

TT #6: SOUND AND MUSIC—Unless you have spent many years experimenting with the four voice channels, you will learn a lot from this one! Learn to play standard notes, chords, and whole songs using some simple "tricks". One of the nicest parts are the examples of special sound effects that you can refer to whenever you need a sound for a program or to impress a friend. This program will be of interest to all ages and levels of experience!
16K Tape or 24K Disk. **$19.95**

SPECIAL DISCOUNT

Order the first six tutorials in a 3-ring binder for $99.95, a $30.00 savings!

TT #7: DOS UTILITIES—We at Educational Software have been shocked by some of the prices others are charging to offer you small utilities to help in the use of your Disk Drive. We now offer you all of the following plus explanation as to how each was written, and how to use them: A UNIQUE MENU PROGRAM, AN AUTORUN.SYS BUILDER, DISK INSPECTOR (LOOK AT SECTORS), DISK JACKET PRINTER, AUTOMATIC FORMATTER, RECORD SAVE AND LOAD UTILITY.
32K Disk Only. **$29.95**

MASTER MEMORY MAP (tm)

This book is the most valuable source of information for your ATARI you can buy. It starts out by explaining how to PEEK and POKE values into memory, so that even new computer owners can use many of these "Tricks". Then you are given 32 pages of the memory locations that are the most useful, along with hints on how to use many of the locations. Finally, it includes hints on problems you may be having with the computer and discusses the new Graphics modes 9 to 11. Even ATARI buys this book from us! **$6.95**

CONTACT YOUR LOCAL DEALER or ORDER BELOW

We have other fine programs for ATARI computers.
Write for a catalog.
Send us your programs to sell too!

USER SUBMITTED PROGRAMS

MINI-DATABASE/DIALER—stores and edits up to 8 lines of information such as names & addresses, phone numbers, messages, inventories, or anything you want. It has the usual sort, search, and print options, but it also has an unusual feature: If your file includes phone numbers and your phone company allows touch-tone phone signals, the program will DIAL THE PHONE NUMBER FOR YOU! 16K Tape or 24K Disk. **$24.95**

THE GRAPHICS MACHINE—allows the ATARI to act like more expensive graphics computers using simple commands like line, box, circle, polygon, fill, and savescreen to get a high resolution picture you can save on disk in only five seconds! Many more features! 48K Disk Only. **$19.95**

BOB'S BUSINESS—14 small business type programs accessed from a common menu. 16K Tape or 32K disk. **$14.95**

MINI WORDPROCESSOR—A simple text editor to write, save, and print several pages at a time. 32K Tape or Disk. **$19.95**

KID'S #1—Includes a MATH QUIZ, a children's TREASURE HUNT, and a DIALOGUE program. 16K Tape or 24K Disk. 3 for . . . **$14.95**

KID'S #2—SPELLING BEE, WORD SCRAMBLE, and TOUCH. 16K Tape or 24K Disk. 3 Educational games for . . . **$14.95**

PLAYER PIANO—Turns your keyboard into a mini-piano and more. 24K Tape or 32K Disk. **$14.95**

GRAPHIC SYMBOL LABELS—for your keyboard to remind you of the built-in Graphics symbols. 2 complete sets for . . . **$2.95**

OUR NEWEST PROGRAMS

DOG DAZE—Two cute little doggies race for the fire hydrants, shoot their bones, and just have a lot of fun! A fast action program for all ages. 8K Tape or 16K Disk, in machine language. **$16.95**

*** OUR BEST GAME ***

SPACE GAMES—Our family is being attacked by ALIENS, and only you can save us. A comic book manual will guide you through three games that test your ability in space skills. Includes ALIENS, SURVIVE, and ROBOT ATTACK, and is for all ages. The first two games require 16K for Tape. The last game and all Disk users need 32K. **$24.95**

MATHS FOR FUN—Another ENGLISH import teaching basic math skills. Very colorful and enjoyable to use. For ages 5 to 16. 16K Tape or 24K Disk. **$19.95**

MARATHON—This is a unique math quiz for one or two players. You are in a race to move your runner across the screen first! There are four levels of play with five modes of operation for each. The game uses joysticks for all input, so play is easy for young children. This wonderful learning tool is imported from ENGLAND for your learning pleasure. Your kids will never even notice they are playing an EDUCATIONAL program. 16K Tape or 24K Disk. **$19.95**

GRADE BOOK—This nice record keeper will maintain a file of 35 students' test scores along with comments. Up to 7 subjects are allowed. 24K Tape or 32K Disk. A printer is optional. **$24.95**

To order COD, VISA or MasterCard call — **(408) 476-4901**

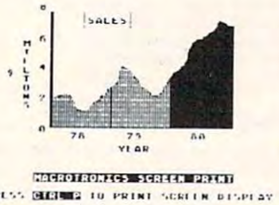
By mail include $3.00 postage (USA only) or $1.50 Memory Map only—
California residents add 6.5% TAX.—Specify Tape or disk.

EDUCATIONAL SOFTWARE inc.

5425 JIGGER DR.
SOQUEL,. CA 95073

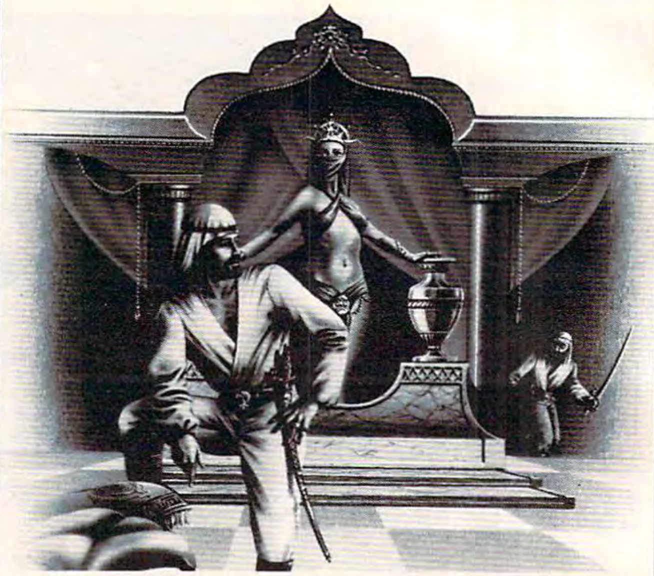
```
                51 ;restore accumulator
                52 ;
0611 68         53          PLA
                54 ;
                55 ;return from the display list
interrupt
                56 ;
0612 40         57          RTI
                58 ;
                59 ;end of interrupt service routine
                60 ;

GTIADLI 4/30/82 by Craig Chamberlain

=026F GPRIOR         =D01A COLBK        =D01B
PRIOR       =D40A WSYNC
```

©



MACROTRONICS SCREEN PRINT

PRESS CTRL P TO PRINT SCREEN DISPLAY



Type in these graphics demonstrations and see the startling displays made possible with the new GTIA chip.

Atari GTIA: An Illustrated Overview

Louis and Helen Markoya
Shelton, CN

Have you ever seen computer-generated graphic displays that are truly 3-D? The ones of landscapes or the ones with molecular structures? Have you ever wished you could generate similar graphics on your own machine? If so, and you own a 400 or 800, you're halfway there. The new GTIA chip allows for three more high resolution graphic modes.

These three modes are far from ordinary. They are called from BASIC by typing Graphics 9, 10 or 11. They all offer the same screen resolution, 80 Horizontal x 192 Vertical, but different color selection.

Graphics 9 offers *16* shades of any of the 16 colors, thus raising the machine's color capabilities to *256*. Graphics 10 offers the programmer a choice of nine of any of the 128 colors normally offered by the Atari, and Graphics 11 gives the ability to present 16 different colors in any of the eight luminances (shades).

For those of us who had the machine before this new addition, the authorized Atari service center nearest you should now be stocking this part. If your machine is under warranty, replacing the chip is free. If not, the chip can be bought for a reasonable fee from your dealer. If you are so inclined, you can install the chip yourself, but you have to tear down your unit to the CPU Board. This board is under the aluminum housing that covers most of the mother board of either the 400 or 800. The disassembly is not difficult if you take your time and are cautious. The CPU Board is easy to identify by the large chips on it. The 6502, ANTIC and CTIA are the large 40-pin chips on this board. GTIA replaces CTIA. One word of warning. These chips are not placed as pictured in the *Hardware Manual.* The easiest way to identify CTIA is that it has the same manufacturers' stamp on it as the GTIA. Authorized service centers will make the swap for you for an additional fee. Either way the cosı is well worth it.

GTIA is Atari's new television Interface Chip. It is completely compatible with the hardware and software previously available. The only problem arises when software relying on the GTIA Modes is run on a computer without this chip. Something will go to the screen, but not the desired effect.

GTIA is controlled for the most part by ANTIC, a microprocessor dedicated to the screen display. The GTIA processes digital commands from ANTIC or the 6502 (in the case of an interrupt) into the signal that goes to the television. GTIA also handles the tasks of Color, Player/Missile Graphics, and Collision Detection.

GTIA adds powerful capabilities in Graphics Modes 9, 10, and 11. All modes are extensions of Graphics Mode 8 + 16, ANTIC Mode 15. The display list remains the same, and the new modes are selected by the Priority Register. This Operating System Shadow Register, called PRIOR, is located at decimal 623, Hex 26F. Bits 6 and 7 control the GTIA modes. When both are zero, GTIA works exactly the same as CTIA. When only bit 6 is set, Graphics 9 is called; when only bit 7 is set, Graphics 10 is called, and when both bits 6 and 7 are set, Graphics 11 is called.

Graphics 9

Setting bit 6 of PRIOR produces Graphics 9, giving 16 luminances of one color. ANTIC provides the pixel data, and the background register, 712, is used to select your color (POKE 712, Color * 16 or SETCOLOR 4,Color,0). Each screen byte is broken in half for screen formatting. A display block is four pixels across by one pixel down. Each four bits represents 16 color choices. The number you choose (0-15) in your color statement equates the luminance value you wish to use. Here's a simple BASIC program used to demonstrate this:

```
10 GRAPHICS 9:REM GRAPHICS MODE 9 (16 SH
   ADES OF ONE COLOR)
20 SETCOLOR 4,6,0:REM SET BACKGROUND REG
   ISTER TO COLOR DESIRED (PURPLE)
30 FOR I=0 TO 15:REM SET UP VARIABLE FOR
    BOTH COLOR (SHADE) AND POSITION
40 COLOR I:REM SHADE OF COLOR
50 PLOT I,0:REM PLOT FROM UPPER LEFT COR
   NER
60 DRAWTO I,191:REM DRAWTO LOWER LEFT CO
   RNER
70 NEXT I:REM NEXT SHADE AND NEXT LINE
80 GOTO 80:REM HOLD SCREEN
```

The wide choice of luminances or shades available here will be particularly useful for shading objects to give the impression of bas-relief or the third dimension. With some background in perspective and lighting, a person could create scenes with a great illusion of depth, realistic or contrived.

Graphics 10

Graphics 10 is called when bit 7 of PRIOR is set to one and bit 6 to zero. This mode utilizes all nine of the Atari's Color Registers found at decimal 704-712 (hex 2C0 through 2C8). Any nine of the 128 colors normally available to your computer could be used in this mode by simply POKEing the desired color (remember, 16 * Color + luminance) into each register or POKEing the desired color into the Player/Missile registers (704 through 707), using SETCOLOR statements for the playfield and background registers.

Color 0 represents the background and is located at decimal register 704, Colors (for color statements) 1-8 follow in order from 705-712. The big advantage to Mode 10 is that any of the colors you choose can be changed independently of the others. For example, once a scene is created, you could change the color of the sky from dark to light blue very easily (FOR I = 128 TO 144: POKE 704,I: NEXT I). This will rotate the background color smoothly through its eight shades. You may wish to add a loop to delay the color change. Playfield or Player/Missile Colors could be changed at any time. Also, special effects and animation could be achieved by rotating the values in all these registers.

The following program draws a border around the screen in eight colors (first register is used for background) and then rotates the colors to give a special effect:

```
10 GRAPHICS 10
20 POKE 704,96:REM SETS BACKGROUND (COLO
   R 0, COLPMO) TO DARK PURPLE
30 POKE 705,22:REM SETS COLOR 1, COLPM1
   TO YELLOW
40 POKE 706,38:REM SETS COLOR 2, COLPM2
   TO YELLOW ORANGE
50 POKE 707,54:REM SETS COLOR 3, COLPM3
   TO ORANGE
60 POKE 708,70:REM SETS COLOR 4, COLPFO
   TO RED
70 POKE 709,86:REM SETS COLOR 5, COLPF1
   TO PURPLE
80 POKE 710,104:REM SETS COLOR 6, COLPF2
    TO BLUE
90 POKE 711,120:REM SETS COLOR 7, COLPF3
    TO BLUE GREEN
100 POKE 712,180:REM SETS COLOR 8, COLPF
    4 TO GREEN
110 FOR I=1 TO 64:REM SETS UP VARIABLE F
    OR COLOR AND POSITION
120 C=C*(C<8)+1:COLOR C:REM CHANGES COLO
    R VALUE
130 PLOT I,I:REM START AT LEFT HAND CORN
    ER
140 DRAWTO I,191-I:REM DRAWTO BOTTOM LEF
    T CORNER
150 DRAWTO 79-I,191-I:REM DR. BOTTOM RIG
    HT CORNER
160 DRAWTO 79-I,I:REM DR. TOP RIGHT CORN
    ER
170 DRAWTO I,I:REM DR. TOP LEFT TO COMPL
    ETE BORDER
180 NEXT I
190 Z=PEEK(712):REM SETS Z EQUAL TO THE
    VALUE IN THE LAST REGISTER
200 POKE 712,PEEK(711):REM ROTATES VALUE
    S FROM 711 TO 712
210 POKE 711,PEEK(710):REM ROTATES VALUE
    S FROM 710 TO 711
220 POKE 710,PEEK(709):REM ROTATES VALUE
    S FROM 709 TO 710
230 POKE 709,PEEK(708):REM ROTATES VALUE
    S FROM 708 TO 709
240 POKE 708,PEEK(707):REM ROTATES VALUE
    S FORM 707 TO 708
250 POKE 707,PEEK(706):REM ROTATES VALUE
    S FROM 706 TO 707
260 POKE 706,PEEK(705):REM ROTATES VALUE
    S FROM 705 TO 706
270 POKE 705,Z:REM ROTATES VALUES FROM 7
    12 TO 705
280 FOR I=0 TO 15:NEXT I:REM SLOW DOWN R
    OTATION
290 GOTO 190:REM START AGAIN
```

This program rotates the border colors to give a theater marquee effect. To display even more of this mode's capabilities, add the following lines:

```
185 A=96:REM SETS A VARIABLE FOR THE BAC
    KGROUND COLOR
272 A=A+1:POKE 704,A:REM CHANGES BACKGRO
    UND COLOR
275 IF A=255 THEN A=1:REM ALLOWS ONLY GO
    OD COLOR VALUES
```

These additional lines will rotate the background color through all its possibilities while the border is rotating.

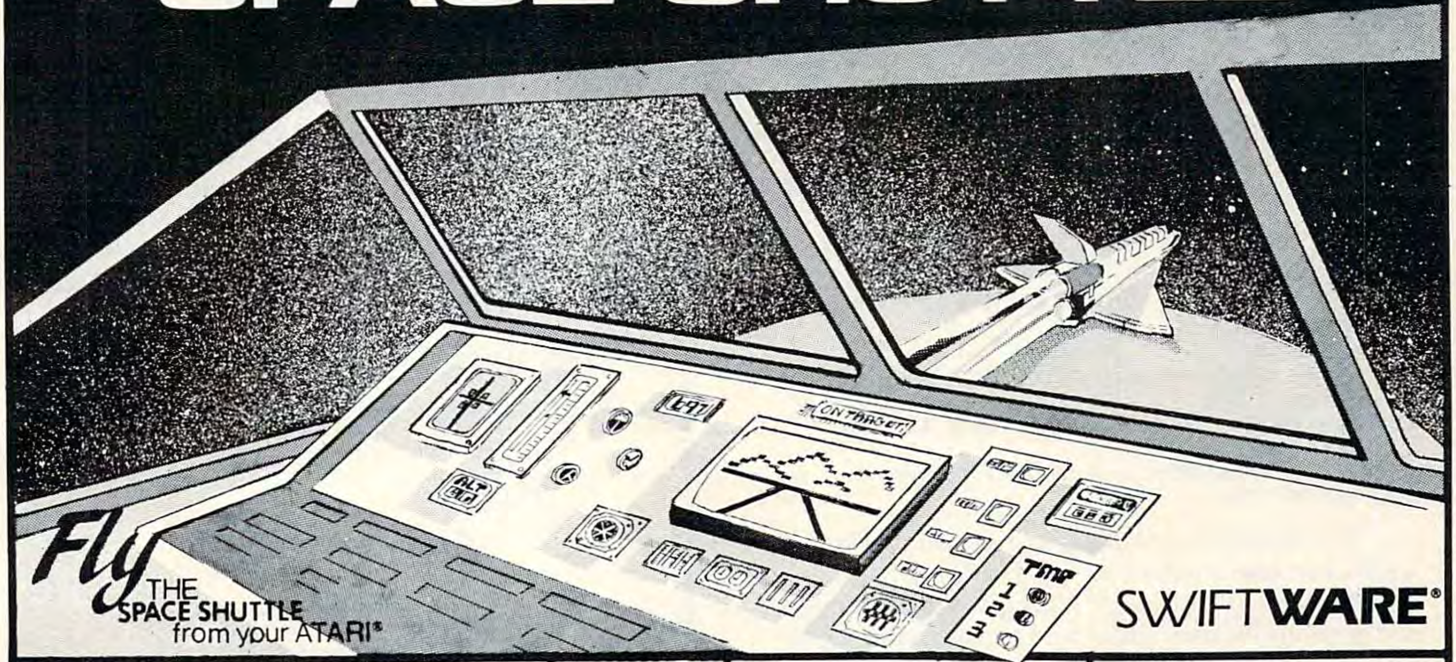
Graphics 11

Mode 11 operates similarly to Mode 9. The difference is that only one luminance or shade is used and a choice of all 16 colors is given. Bits 6 and 7 are set to one for this mode. Again, the background register is used for the colors, with ANTIC supplying the data. COLOR 0-15 relates exactly to the COLOR segment in the SETCOLOR command. To initiate this mode you must choose the luminance or shade you want. The color would be set by your COLOR statement (SE. 4,0,0-15 Lum choice). The background is always COLOR 0 (black). This mode allows fine color blending to produce rainbow effects and therefore a wider color choice for picture making.

The following demonstration program draws a cross in 16 colors, again using a 1 x 1 Display Block, and then draws an ellipse in 16 colors around the center of the cross. This program shows the versatility of color use in Mode 11. No longer are we restricted to horizontal screen architecture for extra color with Display List Interrupts.

```
5 REM GRAPHICS 11 DEMONSTRATION PAGE 5
10 A=1:R=26:REM SETS VARIABLES
20 DIM X(360),Y(360):REM ALLOW STORAGE S
   PACE FOR X AND Y COOORDINATES
30 GRAPHICS 11:SETCOLOR 4,0,12:DEG :REM
   SETS GR. MODE, LUM OF COLORS AND DEGR
   EE MODE FOR ELIPSE
40 FOR I=0 TO 15:REM COLOR AND POSITION
   VARIABLE
```

SPACE SHUTTLE



GAMES

SPACE SHUTTLE ™ by Paul Kindl

Join the crew of the *Space Shuttle* as they prepare to take the next step into the world of space travel. Take control of the world's first reusable spaceship, the *Space Shuttle*, and in an accurate full graphics simulation, place yourself in the cockpit. Pilot the *Space Shuttle* through take-off with booster stage separations, orbit, descent down the glide path and landing to touchdown complete with a chase plane and scrolling runway visible through the cockpit windscreen. You assume command throughout all phases of the mission aided by complete instrumentation.

Price: $29.95 (32K Diskette)

HAUNTED HILL ™ by George Richardson

Fight bats and ghosts in the dark of a cemetery. This exciting, all machine language game has arcade quality graphics and speed. Requires Joystick.

Price: $29.95 (16K Diskette/$24.95 16K Cassette)

TRIVIA TREK ™ by Jerry White

A multiple game of trivia for one or two players. Comes complete with 500 trivia questions in fifty categories and over two thousand multiple choice answers. A program for creating your own trivia questions and answers is also included. Play it for fun, test your knowledge or entertain friends. Fun for the whole family and great at parties. Joystick optional.

Price: $29.95 (32K Diskette)

FUN 'N' GAMES ™ by Jerry White

Wordgames, Possible and *Leapfrog* giving you hours of fun, challenge and entertainment. *Wordgames*, two games in one, contains *Guessit* - a deductive alphabetic reasoning game for one or two players and *Wordjumble* - a multiple word descrambling puzzle with play-on-word hints and mystery answers. Instructions show how you can substitute your own words. Use *Possible* to help descramble word jumble puzzles or to create your own. All letter/number combinations or permutations of input are printed to screen or optional printer. *Leapfrog* is a Chinese-Checker type jumping game in which you try to position two sets of animated jumping frogs in a minimum number of moves. Disk version of *Guessit* works with a Votrax Type 'N' Talk. A real crowd pleaser. Joystick optional.

Price: $17.95 (16K Cassette/$19.95 24K Diskette)

DATABASE MANAGEMENT

FILE-IT C ™ by Jerry White

A cassette only start-up database management system, designed to work in a 16K computer, that files and manages personal information and data. This friendly, menu driven package creates, sorts, stores and manipulates information. *File-It C* easily configures and handles data such as appointment calendars, address or telephone records, credit and charge card records, stock investments, medical and prescription information, hobby, coupon and other types of collection inventories. Because of its flexibility, *File-It C* is limited only by the user's imagination and creativity. Supports optional *Atari 825* or *Epson MX-80* printers. Prints one or two across mailing labels and neat copies of all *File-It C* data files. Fast and easy to use, *File-It C* holds over 300 records in 40K.

Price: $19.95 (16K Cassette)

FILE-IT 2 ™ by Jerry White

A deluxe package which contains all of the features and programs in *File-It C* plus five additional especially designed file handling and financial management programs. Six user defined fields are created with up to 5 subfields beneath each main field. The *Financial Entry* and *Financial Report* programs create a powerful personal financial management system recording transaction dates, expense and income entries. Three additional user designated fields give bottom line accounting capability. Two additional *File-It* programs provide random access updating and user controlled record selection. Full field and sub-field sorting is available. Data-files are stored on separate data disks permitting large scale data management. Subfiles may be created, merged and sorted by any field. A *Bar Graph* program generates a screen display of financial data including values and percentages in bar graph form which may optionally be printed. Supports up to four disk drives as well as the *128K Axlon Ramdisk*. Extensive documentation, attractively bound, provides clear user instructions along with a tutorial on computer filing. The *"Alternative"* to more costly data-base management systems. 80 column printer is optional.

Price: $49.95 (24K Diskette)
plus $3.50 for shipping and handling

FAMILY FINANCIER ™ by Paul Thompson

Family Financier, a menu driven cassette based personal finance package, provides a fast and easy means to balance checkbooks, record checks and generate account statements and monthly totals keyed to budget items and/or payees. Stores all data on cassette. Printer optional.

Price: $24.95 (16K Cassette)

DISK MAINTENANCE

SWIFTY TACH-MASTER ™ by Tony Dobre

An accurate *(disk speed diagnostic)* utility program designed specifically for *ATARI 810* Disk Drives. Provides easy-to-read visual indication of the speed of any drive connected to your system. Using machine language, *Tach-Master* displays five RPM readings per second with a working tachometer accurate to ¼ RPM. Allows adjustment of your drive(s) to factory specs easily without special equipment. Comes complete with easy to follow user's manual.

Price: $24.95 (16K Cassette/$29.95 16K Diskette)

DISKETTE INVENTORY SYSTEM ™ by Jerry White

Use this system to gain control of your expanding disk/program inventory. Quickly get locations of single or multiple copies of your programs and all your valuable files. An invaluable tool, this system is easy and convenient to use and to update. Printer optional

Price: $24.95 (24K Diskette)

ACCESSORIES

DISK SENTRY ™

An intellegent digital accessory for your *Atari 810 Disk Drive*, lets you selectively write data to both sides of single sided and write protected disks. *Disk Sentry* cannot harm your drive disks. Installs and removes easily; no soldering required. *Disk Sentry's Led* signals system status, preventing accidental erasure of data. *Disk Sentry* is a convenient push button write-protect override which can pay for itself with your first box of disks.

Price: $39.95 each

VINYL DUST COVERS

Custom sewn upholstery grade vinyl dust covers for *Atari 800* and *400* computers and the *810* and *825* peripherals. Protecting your equipment when not in use, these crisp black covers go well with any decor. Specify model(s) when ordering.

Price: $10.95 each

TUTORIAL

JERRY WHITE'S MUSIC LESSONS ™

INCLUDING Player Piano and Singalong

This two sided diskette or 2 cassette music and sound tutorial by *Jerry White* contains 13 separate programs. It is designed to provide demonstrations, in *Atari Basic*, of the *Atari Home Computer* system's sound capabilities both for music and sound effects. *Music Lessons* comes with extensive, bound documentation complete with program listings. *Player Piano* turns your computer into a twenty note mini-piano that includes a full, graphically accurate, screen display of the piano keyboard. The *Singalong* programs teach and play songs in four part harmony while displaying lyrics on the screen.

Price: $29.95 (16K Cassette/$29.95 32K Diskette)

TELECOMMUNICATIONS

DATALINK ™ by Tony Dobre

Top rated national magazine reviews, purchased by *NASA*, this ultra-sophisticated menu driven multioption smart terminal Communications package is a *must* for all modem users. Features full performance uploading and downloading in full-duplex or simplex modes and supports *ASCII* and *ATASCII* transmission. Automatically off-loads line data to your disk drive upon completion of a transmission with options available to screen and/or printer dump the same data. May be used as a remote terminal compatible with all the commercial services and bulletin boards such as the *Source* and *Compuserve*, etc. Saves significant connect time charges when using commercial services. Requires *Atari 850 Interface* or equivalent and*Atari 830 Modem* or any other 300 Baud modem. Printer optional.

Price: $39.95 (24K Diskette)

ALSO AVAILABLE

PROGRAMMING AIDS PACKAGE ™

SWIFTY UTILITIES ™

MAILIST ™ by Leo Cerruti

— AVAILABLE AT SELECT COMPUTER STORES —

Send check or money order plus $2.50 shipping and handling to

(516) 549-9141
Call to Place Your Order
N.Y. Residents add 7¼% sales tax

SEND $1.00 FOR CATALOG
(refunded on your first order)

SWIFTY SOFTWARE, INC.
64 BROAD HOLLOW ROAD
MELVILLE, New York 11747

DEALER INQUIRES INVITED

VISA MasterCard

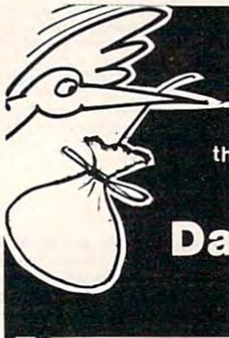
© 1981, 1982 SWIFTY SOFTWARE, INC.
ATARI® is a registered trademark of Atari Inc., a Warner Communications. Co.

```
50 COLOR I
60 PLOT 31+I,0:DRAWTO 31+I,191
70 PLOT 0,86+I:DRAWTO 79,86+I:REM DRAWS
   CROSS
80 NEXT I
90 FOR I=0 TO 360 STEP 2
100 X(I)=R*COS(I)+34
110 Y(I)=R*SIN(I)+95
120 NEXT I:REM SETS X AND Y VALUES FOR P
    LOTTING ELIPSE
130 FOR I=0 TO 360 STEP 2:REM CALLS ABOV
    E VALUES
140 COLOR A
150 PLOT X(I)+A,Y(I)+A:REM PLOT EACH COL
    ORS' ELIPSE
160 NEXT I
170 A=A+1:REM NEXT COLOR AND NEXT ELIPSE
     POSITION
180 IF A=16 THEN 200:REM END IF ALL COLO
    RS ARE USED
190 GOTO 130:REM DRAW NEXT ELIPSE
200 GOTO 200
```

The final demonstration program draws a landscape and a simple molecular structure floating high above it. This display truly gives the impression of depth and shows what can be done using light and shadow in Graphics Mode 9:

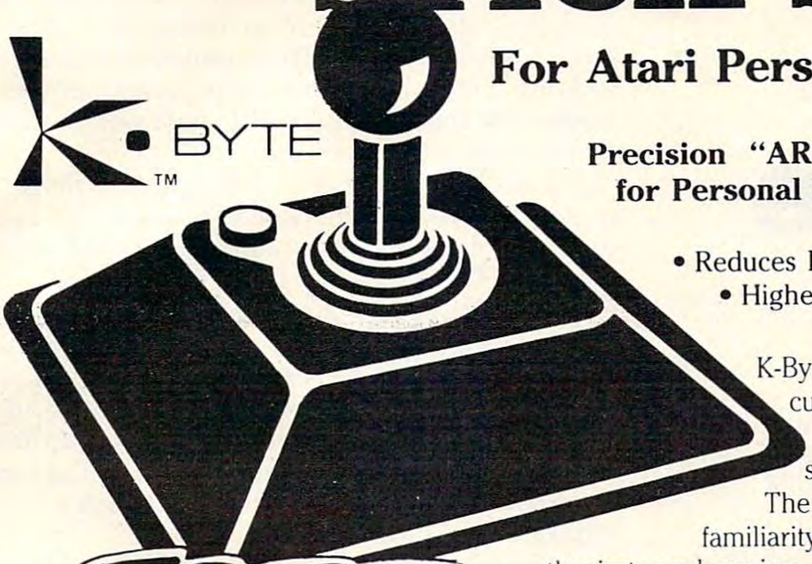
```
10 R=16:X=0:C=15
20 GRAPHICS 9:SETCOLOR 4,13,0
30 FOR I=130 TO 191
40 COLOR C
50 PLOT 0,I:DRAWTO 79,I
60 X=X+1:IF X=4 THEN X=0:C=C-1
70 NEXT I
80 FOR I=0 TO 79 STEP 8
90 COLOR 3:PLOT 59,130:DRAWTO I,191
100 NEXT I
110 COLOR 1:FOR I=0 TO 7:PLOT 2,164:DRAW
    TO 21,158+I:NEXT I
120 COLOR 15:FOR I=0 TO 3:PLOT 21,140:DR
    AWTO 21+I,164-I*2:NEXT I
130 COLOR 4:FOR I=0 TO 4:PLOT 20,140:DRA
    WTO 17+I,160+I:NEXT I
140 FOR Z=1 TO 15
150 FOR I=0 TO 360 STEP 6
160 X=0.25*R*COS(I)+35
170 Y=R*SIN(I)+50
180 COLOR Z
190 PLOT X,Y
200 PLOT X+10,Y+17
210 PLOT X+30,Y-20
220 PLOT X-2,Y+12
230 PLOT X+21,Y+70
240 NEXT I
250 R=R-1
260 NEXT Z
270 FOR I=2 TO 4:COLOR I:PLOT 46,72:DRAW
    TO 51+I,106
280 PLOT 43,62:DRAWTO 39,50+I
290 PLOT 47,62:DRAWTO 60+I,35
300 NEXT I
310 GOTO 310
```

These demos are only an introductory hint of the truly spectacular effects possible via GTIA. You could add more color to Modes 9 and 11 by using Players and Missiles or create dramatic effects by switching between these modes (or with Graphics 8) by POKEing PRIOR with the desired value.



STICK STAND

K•BYTE™



For Atari Personal Computers

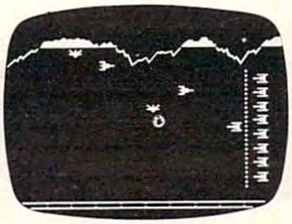
Precision "ARCADE" action now assured for Personal Home-Computer games!

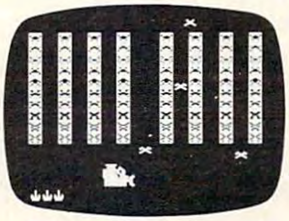
- Reduces Hand & Wrist Fatigue
- Higher Scores Thru Better Control

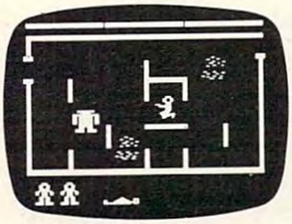
K-Byte's new, innovative "Stick Stand" secures the Joystick to reduce wrist fatigue and frees one hand for a more skillful operation of the fire button. The comfortable, Fastball Knob extends familiarity and ease-of-play in Arcades, to enthusiasts and novices of Personal Home-Computer Games.

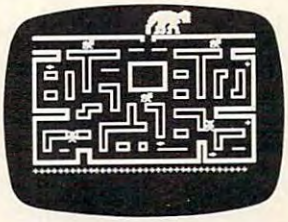
Cat No. 4170 ***DEALERS — CALL FOR PRICING***

FREE W/PURCHASE OF ANY K-BYTE PROGRAM









K-BYTE K-STAR PATROL™

Confrontation between Star Ships and Alien Attack Ships test wits and physical skills as intergalactic leeches invade the territory!

The lead Star Ship must destroy the Alien Attack Ships as they materialize, while eliminating Intergalactic Leeches that invade the territory. The Star Ship must also replenish its Force Field Energy, periodically, by diving between mountains into the Hydrogen Laden Lakes.

K-Star Patrol offers ten sectors of play, each subsequent sector requires a greater degree of hand/mind coordination, as the Alien Forces rise and attack at increased speed and frequency, and the Low-level Avoidance System goes into effect.

Cat No. 4167 Atari 8K, ROM ~~$49.95~~

44^{95}*

K-BYTE K-RAZY KRITTERS™

Excitement . . . Fast Action . . . Challenge . . . and more are ensured for enthusiasts with Command Ships battling Alien Kritters and Patrols with supermissiles!

The Command Ship is challenged to avoid or use "Supermissiles" to destroy the eight columns of Alien Kritters beaming down on the Star Base. Regular missiles will eliminate the Free-Falling Kritters descending at various speeds and frequencies. If the Command Ship is hit, the tidy Sanitation Crew will remove it from the screen.

K-Razy Kritters offers ten levels of play, each representing a greater degree of difficulty and making mental and physical dexterity a necessity.

Cat No. 4166 Atari 8K, ROM ~~$49.95~~

44^{95}*

K-BYTE K-RAZY SHOOT-OUT™

Space Commanders . . . Alien Forces . . . Laser Weaponry . . . and more . . . to challenge the mental and physical dexterity of any enthusiast!

The object of the game is to advance the Space Commander into the Alien Control Sectors, eliminate the Alien Forces with Laser Weaponry and escape from the sector.

The reward for escape is advancement into the next sector, seven sectors in all, each representing a greater degree of difficulty. No two games or game levels are alike as the Alien Control Sectors are created at random with millions of combinations of barriers and escape routes.

Cat No. 3521 Atari 8K, ROM ~~$49.95~~

44^{95}*

K-BYTE K-RAZY ANTIKS™

Invading, Enemy Ants . . . Anteaters . . . Torrential Rains . . . and Enemy Eggs will challenge the skills of any computer game enthusiast!

The player must guide the White Ant safely through the maze of tunnels in the Anthill while avoiding the Enemy Ants, the Anteater and the unpredictable Rain Floods. He must also deposit and protect White Eggs to perpetuate the specie and eat the Enemy Eggs to prevent them from hatching.

K-Razy Ant ks offers the player a choice of six mazes and 99 levels of difficulty. Each subsequent level requires a greater degree of physical and mental skill.

Cat No. 4168 Atari 8K, ROM ~~$49.95~~

44^{95}*

*Atari is the registered Trademark of Atari, Inc.

****SPECIAL INTRODUCTORY PRICES***

HW ELECTRONICS

WHEN IN SOUTHERN CALIFORNIA, VISIT OUR RETAIL STORES

19511 Business Center Dr.
Northridge, CA 91324

2301 Artesia Blvd.
Redondo Beach, CA 90277

444 S. Indian Ave.
Palm Springs, CA 92262

HOW TO ORDER

Write or phone. Pay by check, M/C, VISA, or COD (add $1.50 for COD). CREDIT CARD CUSTOMERS — please include your daytime phone number.

Outside Calif. (800) 423-5387
Inside Calif. (213) 886-9200

Offer expires Sep. 30, 1982

Mention this ad and WE PAY SHIPPING! (UPS ground only). Offer does not apply to Stick Stand (pls. include 2^{00} shipping).

19511 Business Ctr. Dr.
Dept. G9
Northridge, CA 91324

A Monthly Column

Machine Language

Jim Butterfield
Associate Editor

If I May Interrupt

The 6502 goes about its job, executing instructions at lightning speed. As each instruction is completed, the processor checks: should this process be interrupted?

There are two kinds of interrupt, called IRQ and NMI. They have different features and uses, but they share common characteristics. They may only take effect when the current machine language instruction has been completed. At that time, the address of the following machine language instruction is pushed to the stack together with the Status Register. Then the machine gets an interrupt address stored high in memory, and starts to execute instructions from that address. At a later time, when the interrupt job has been serviced, an RTI instruction will cause the previously stored information to be reclaimed from the stack and the interrupted program to continue.

Interrupt is high priority. It tends to be used where fast response is vital. Don't throw it away on some unimportant job which is not time-sensitive; save these big guns for a real time crunch. I tend to recommend the following priorities: if you can, use straight coding; if you need to, use a timer or two; if you must, use interrupt.

Because an interrupt stops the work in progress to handle a special rush job, users often tend to think of it as instantaneous. Not quite. Don't forget that there's a variable wait to complete the instruction under way (up to seven cycles) in addition to the fixed delay of seven cycles while the interrupt does its bookkeeping work. The effect of the variable wait is "timing jitter" – occasionally important even though the time involved is small.

The Big Two

IRQ – Interrupt Request – is the less powerful of the two interrupts, but it's usually easier for the programmer to handle. It may be locked out with an SEI instruction (Set Interrupt Disable) to prevent interruption from striking at an embarrassing moment; the lockout is released with CLI (Clear Interrupt Disable). Using SEI/CLI adds to the possible timing jitter by a substantial amount, of course. When an interrupt takes place, an SEI-type lockout automatically takes effect, so that another IRQ interrupt will have no effect until RTI releases the lockout. This is handy for the programmer – he knows that the code in his IRQ type system will be free from further interrupts.

NMI – Non-Maskable Interrupt – is more powerful and less controllable. It cannot be locked out. As a result, the programmer has to be much more careful in sensitive areas: for example, changing the interrupt vector itself can be a ticklish job since the coding cannot prevent the NMI from striking in mid-change with potentially disastrous results. To add to the complexity: an NMI could cause an interrupt, and while it is being handled, another NMI could interrupt again. Careful coding is needed to avoid data corruption if such a multiple-level interrupt is anticipated.

There's another fundamental difference between IRQ and NMI. IRQ is level-sensitive: when the IRQ pin on the 6502 chip receives a low level, interrupt is being requested. NMI, on the other hand, is edge-sensitive: when the NMI pin on the 6502 chip goes from high level to low, a "latch" is triggered within the chip that will signal that NMI needs attention. Think of it this way: if I held the IRQ pin low permanently, the computer would be continuously interrupting. It would go into interrupt, do the job, and upon completing with RTI, the interrupt would take place again since IRQ is still low. In contrast, if I pulled NMI low permanently, I would have only one interrupt – the one that was triggered when the signal went low. A new "edge" would be needed to trigger NMI again.

IRQ Latches

This gives us two seemingly conflicting requirements for the interrupt signal at the IRQ pin. First, it must remain active until the interrupt takes place; too brief an IRQ signal might be missed entirely. Next, it must be turned off before the interrupt coding completes its activity, or RTI will just cause a new interrupt. This seems difficult – not too fast and not too slow – but, in fact, we accomplish the job very easily with the help of extra chips.

Most of the interface chips (the best known to 6502 users are the 6520 PIA and the 6522 VIA) contain latches that may be set by the external interrupting circuits, and reset by the 6502. For example, if a timer counts down to zero and signals an interrupt, this will be latched and signalled to the 6502. When the 6502 gets around to servicing the interrupt, it can switch off the latch.

This system of latches allows many interrupts to be received and forwarded to the processor

CFi computer solutions

201 WEST 92 STREET • NEW YORK • NY 10025 • 212 • 724 • 1711

OFFICIALLY APPROVED BY COMMODORE

PROFESSIONAL TAX PREPARATION SYSTEM

WILL PROCESS THE FOLLOWING SCHEDULES AND FORMS:

1040
LETTER SCHEDULES A, B, C, D, E, F, G, SE
NUMBERED FORMS: 3903, 2106, 2441, 3468, 5695, 2210, and 1040-ES

STATE TAXES FOR: PA, CA, NY, NJ, MA, IA, and FL property.

Husband, Wife, and Joint forms prepared simultaneously. Prints directly onto tax forms.

ASERT

Aid for Search and Retrieval of Text

To the usual Data Manager record format ASERT adds an area of FREETEXT, for all those "odd" bits of information that don't quite fit anywhere; and 180 SEARCHWORD fields, a unique system of coding with powerful search capabilities. Now used by Real Estate Agencies, Employment Agencies, Schools, Travel Services and Libraries across the U.S. and in Europe, ASERT makes the filing cabinet obsolete!

11,000 CHARACTER "VIRTUAL" RECORD LENGTH
INTERFACE WITH POPULAR WORD PROCESSORS
COMPILED FOR SPEED

OTHER CFI SOFTWARE

Personal Tax Calculator • Emergency Control Program • VIC Animation Tutorial

AVAILABLE FROM YOUR COMMODORE DEALER AND CFI.

chip. The computer can then interrogate the interface chip and find out what caused the interrupt. There might even be two events calling for service at about the same time. The computer can decide to service one of them, turn that particular latch off, and do the job. The moment it gives RTI the other event (whose latch is still locked in) will re-interrupt and be serviced. It works out remarkably elegantly.

The interface chips may have external ports or built-in devices such as timers and shift registers which are allowed to cause interrupts. Each of these may be logically connected to or disconnected from the interrupt line. It seems complex at first; but a little practice will show the system to be straightforward and logical.

Registers

You may recall that only the instruction address (Program Counter) and Status Register (sometimes called the PSW) are saved on the stack during an interrupt. If you plan to use the A, X or Y registers during your interrupt processing, you must save them by pushing them to the stack. Just before giving RTI, bring them back. Your interrupt must be truly "invisible" to the code that was interrupted.

It's quite easy to implement interrupt. You must be especially careful; debugging is much more difficult for this type of code.

Try to keep your interrupt code short, and let the "background" program pick up and do most of the work. The briefer the interrupt program, the more often you'll be able to service interrupts; that will often yield a more powerful system.

Be very careful that a long interrupt doesn't disturb a critical timing process in background code. More than one thermal printer has had its head "smoked" by a sluggish interrupt that didn't know that the background program was waiting to turn the heat off. ©

This is a short and very useful addition to VIC. Some PET/CBM computers allow you to stop a LISTing from scrolling and then start it up again. This "Pause" feature adds that capability to VIC (any memory size). Type in the first line very carefully – it's got to be exactly as printed.

VIC Pause

Doug Ferguson
Elida, OH

For VIC owners who have not bought a printer yet, studying a BASIC program by using the LIST command can be tedious. The screen displays only about 20 lines at a time if you hit the STOP key. And then the only way to re-start at the point at which you stopped is to retype LIST – again and again. Even the CONTROL key is not much help; the lines still move by too fast for more than a superficial look. What is needed on the VIC-20 is a PAUSE key.

For months I have been trying a method for the PET from G. H. Watson's "Linelist" program (in **COMPUTE!**, September 1981, #16), but without total success. At least the experience taught me a lot about disassembling the BASIC ROM. Finally, with the help of Eric Brandon's advice (**COMPUTE!**, May 1981, #12, pg. 126) I discovered an alternate method which patches into the hardware interrupt vector. This vector in the VIC is located at 788 and 789 and contains the low-byte/high-byte jump vector $EABF which the VIC visits every 60th of a second to check the STOP key, to update the clock, and to do other chores.

The routine which I have "persuaded" the VIC to jump to every 60th of a second is very similar to Mr. Watson's: to pause a LIST, hold down the SHIFT key. The SHIFT key is the ideal PAUSE key because it can be "locked" with the SHIFT LOCK key. This allows totally hands-free operation so that you can study (or copy in longhand) your BASIC program a few lines at a time.

I have written the machine language loader without DATA statements in case you want to add the PAUSE feature to a program which may already contain conflicting DATA statements. I preferred the cassette buffer as a house for my 26-byte masterpiece because this area stays stable regardless of memory expansion of the VIC.

I used location 888 as a start point because I thought it easier to recall than other traditional cassette buffer addresses. Please note that the first line is a very tight fit; type no space after the line number, and consider the final quotation mark optional.

```
63997 A$="12Ø169133141Ø2ØØØ3169ØØ3141Ø21ØØ3Ø88Ø9
    6Ø32159255169ØØ1Ø44141ØØ22Ø8246Ø76191
    234"
63998 P=1:FOR X=888 TO X+25: POKE X,VAL(MID$(A$,
    P,3)): P=P+3: NEXT
63999 SYS 888
```

When you have finished (carefully) typing the loader, execute with RUN 63997. (With an 8K expander, the VIC may give you an ILLEGAL QUANTITY error. Assuming no errors in your typing of the program, try again several times. The unexpanded VIC does not have this trouble with string variable manipulation.) After a successful RUN, feel free to delete lines 63997 - 63999.

Now it's time to test your new PAUSE key. Try it by freezing the blinking cursor (on or off) with the SHIFT key. Next, press RUN/STOP and hit RESTORE to clear the screen. Note that the SHIFT key does not operate as before. The RUN/STOP-RESTORE technique returns the hardware interrupt vector to its normal contents. If you wish to "freeze" a LIST again, merely SYS 888 to reactivate the PAUSE feature.

Be Sure To Turn It Off Before SAVEing

It is essential to deactivate by the RUN/STOP-RESTORE method when you are not using the PAUSE feature. For one reason, any I/O operations (such as a SAVE of whatever you have been debugging) require that the hardware interrupt vector be returned to normal. (Also remember that a SAVE uses the cassette buffer where the PAUSE routine lives; you may expect to lose the PAUSE feature after a SAVE or a LOAD.)

Another reason to deactivate the PAUSE is that shifted characters such as uppercase and graphics are strange to type, in that they appear *after* you let up on the SHIFT key. Even worse is the loss of repeat keys such as cursor-up and -left and the disappearance of the cursor when it moves rapidly right or down.

I bet you're wondering if there are any other drawbacks to my idea. Well, yes. When you do SYS 888 to activate the PAUSE key and you do try to freeze a LIST, you will occasionally get a kind of "ripple" on the screen which causes a line to be, more or less, repeated twice. I believe that this is a tolerable annoyance, especially if one remembers that the upper half of these "paired" lines is always correct.

By the way, the PAUSE will also work when you RUN a program. However, if you only want a PAUSE feature for this reason, there are much better ways. Lines 132-136 of Amihai Glazer's "Amortize" program (**COMPUTE!**, May 1982, #24) are just perfect in the main routine of any BASIC program in which you want a PAUSE.

The assembly listing that follows was written on Eric Brandon's excellent assembler program (**COMPUTE!**, June 1981, #13). It required memory expansion and a few TAB changes.

```
HIV        = $0314
INTRPT     = $EABF
SCNKEY     = $FF9F
PAUSE      = $0385

1                               *        =     888
2                               HIV      =     788
3                               INTRPT   =     $EABF
4                               SCNKEY   =     $FF9F
5     0378  78                           SEI
6     0379  A9    85                     LDA   #$85
7     037B  8D    14   03                STA   HIV
8     037E  A9    03                     LDA   #$03
9     0380  8D    15   03                STA   HIV+
10    0383  58                           CLI
11    0384  60                           RTS
12    0385  20    9F   FF       PAUSE    JSR   SCNKEY
13    0388  A9    01                     LDA   #1
14    038A  2C    8D   02                BIT   653
15    038D  D0    F6                     BNE   PAUSE
16    038F  4C    BF   EA                JMP   INTRPT
```

Here are a few notes on the assembly listing. Lines 5-11 change the hardware interrupt vector (HIV) to $0385 (PAUSE); the SEI and CLI are necessary to make the computer wait until both low and high bytes are changed. I used a BIT test because location 653 also determines the CONTROL key, which is often used to slow down a LIST. The whole thing ends with JMP INTRPT to send the VIC where it had intended to go. Lines 12-15 are just sort of a detour.

This has been my first real foray into machine code, and I would appreciate any helpful comments. ©

What is ASCII? How and why is this special code used to communicate between one computer and another? Some computers make slight modifications to the code which must be taken into account when sending messages to a different model. The Atari uses 155 to stand for a carriage return, for example, where most computers use 13. To see what the code number 77 looks like to your machine, you can type ?ASC(77). Whatever character shows up on screen is the meaning of that number. Of course some are special characters. ?ASC(13) will generally perform a carriage return. For a complete table of standard ASCII codes, see **COMPUTE!**, *July 1982, p. 140.*

Telecommunications:

All About ASCII

Michael E. Day
Chief Engineer, Edge Technology
West Linn, OR

ASCII is an acronym for American Standard Code for Information Interchange. More specifically, it is a definition of a code that is used in most computers to store and transmit information.

The computer works with information in *bits* or "on/off conditions" of its memory cells. This means that in order for it to work with numbers and letters it must work with them as a group of on/off bits. The ASCII code defines the representation as seven bits, while most computer systems deal with numbers and letters as an eight-bit code sequence, with the eighth bit being used internally by the computer for its own purposes. The seven bits can be formed into 128 possible combinations, with each combination representing a single character or letter. These 128 characters are broken down into 52 upper and lowercase letters (A-Z & a-z), ten numbers (0-9), 33 special characters (including the space), and 33 special "control" characters. ASCII is a non-shifted code; that is, any of the seven-bit code combinations represent a unique character.

BAUDOT Code

BAUDOT, on the other hand, is an example of a shifted code. BAUDOT code uses only five bits to represent information. As a result, only 32 possible combinations are available. In order to use this method, some decisions had to be made as to what information would be allowed to be represented.

The first thing thrown out is the lowercase letter, as information can be maintained in uppercase only without destroying its meaning. Next the special control functions are reduced to the bare minimum needed – carriage return, line feed, and space. The 26 uppercase letters and the three control functions use up 29 of the available codes. We cannot fit the ten numbers in the three remaining codes, so we have to go about it in a different way.

The numbers and some special characters are provided for by setting aside two of the remaining codes as shift functions. A shift code changes the definition of all following codes, until another shift function directs a return to the original code representation. By using shift functions, it is possible to almost double the number of characters that can be represented by the same 32 codes. The final remaining original code is not used and is referred to as a blank or null character.

The ASCII code standard... permits most computers to talk to one another.

The null character is used in the BAUDOT code as a non-operational code (it is ignored if it is encountered). The reason for this requirement is the way the machines function. When data is transmitted, it is sent in serial form, or a bit at a time, to the receiving equipment. The old teletype machines which sent and received this code used current flowing through a wire (or not flowing) to represent the on or off bit condition. The problem with this was that, if the wire was broken (which happened quite often), it was the same as sending a continuous stream of off bits, which of course represents one of the possible codes. By ignoring this particular code, at least the machine wouldn't spit out reams of paper if the wire did get broken.

The shifting codes were called the letters shift and the figures shift, which corresponded to the functions they performed. That is, when the letters shift was sent, it meant that all the codes that followed would represent the letters A-Z. The figures shift meant that all the codes following would represent numbers and special characters. The individual codes were assigned to the various letters in a way to ensure a minimum number of errors occurring in transmission, so that when an error did occur it would be noticeable. The letters shift code was chosen to be all on bits; this was to help both in transmission error problems as well as easing editing for transmission.

Erasing And Editing

The all-bits-on code has a special function. Information used to be stored as holes punched in a paper tape. If a mistake was made during creating this tape, then a new one would have to be made. As with most writing, however, the mistakes were often caught as they were made. With the letters shift code being made up of all on bits, all that was needed to wipe out the bad code was to back up to that section of tape and type the letters shift, which punched out all holes on the tape, changing any code on the tape to the letters code. The disadvantage to this was, of course, the possible inadvertent shift to letters case which required a following figures shift (if that was the case that you were in).

When the ASCII code was developed, the paper tape method was still in use, so mistakes were taken care of with a special function code called a *rubout*. The rubout code works the same way as the null code: when it is encountered it is ignored, except that it has the additional editing function of punching all holes in the tape to wipe out the existing code.

Smarter Editing

When computers came along, and it was possible to add some intelligence to the editing process, the rubout was used to tell the computer to back up and erase the last character that was typed. Since we were still using a printer though (one that did not know how to backspace at that), it was still difficult to deal with the information as it was written. Several editing methods were tried to overcome this problem. One of the most common methods was to simply print the character again as it was deleted. This tended to make the result rather confusing, but it did provide a way for the computer to indicate that it had performed its function.

As backspacing printers became available, another method of editing came into being: when the rubout key was depressed, the printer would back up and strike over the previous character with a backslash or some other character. It would then wait for the next character to be typed. If it was another rubout, it would back up and strike over the previous character. If the next character typed was not a rubout, then the printer would move the paper up a line and strike the character on the line below the original character. Another editing method sometimes used because it's easy on a computer was simply to ignore the entire line just typed and start over again. The operator indicated this method by doing a carriage return.

When video terminals came along, the editors took on a slightly different approach to the problem. The video terminal was capable of truly removing the wrong character from the text as it appeared on the screen. Since the rubout key was already being used for the printers, the backspace was used to back up and erase the previous character. Since the error was truly erased from the screen, there was no need to go to the next line, so the backspace simply remained in that character position after it was done. The usual sequence involved was to backspace, space to erase the character that was there, and then backspace again to return to the erased character's position.

Now the printer tends to be relegated to the side to be used only when the text is ready to be printed. The exception is in portable or very low budget operations where the printer must serve as both the editing device and the printing device.

The video terminal has grown from a "glass teletype" to what is now called *full-screen editing*. Full-screen editing is an extension of the backup and erase function, except it is enormously more powerful. Anything that appears on the screen can be directly altered by moving the current position indicator (the cursor) to the position you wish to modify and simply typing over it, deleting it, or inserting new information. The full-screen editor has the advantage of being quite easy to use since what you see is what you get.

Taking things a step further, we end up with a full-blown word processing system where not only can you change the data as it appears on the screen, but you can also move it around, add in whole new sections from storage, format the way it is to be printed, and even have it check your spelling.

Who knows? Maybe next it will start writing things for us, and we can just sit back and enjoy the show.

The Origin Of ASCII

As you can see, the ASCII code was developed as a result of the interaction between the needs of the equipment and the needs of the operator. This is an important aspect of equipment design and is often the dividing line between good and bad machinery. It should be remembered, however, that the coding system used is just that: a coding system. It can be changed, modified, compressed, or expanded, and many other things can be done to it following standard coding theories. The ASCII code standard is simply there because it is the common element that permits most computers to talk to one another. This of course doesn't mean that all computers use the ASCII code standard.

A very early code system used on punched cards was a ten-bit code, later expanded to 12 bits, called the Hollerith code. An outgrowth of the Hollerith code was the BCD (Binary Coded Decimal) code, which was a simplification of the

Hollerith code.

The BCD code uses six bits to represent 64 possible code combinations. Later on, due to the needs of the computer systems, this code was expanded to eight bits and called EBCDIC, for Expanded Binary Coded Decimal Interchange Code.

By this time the code had become quite a mess, what with all the expansions and extensions. The ASCII code was then developed by the American National Standards Institute (ANSI) to simplify code use – to make the code easier for machines to use, as well as easier for humans who had to use it. The main improvement was to place all letters in sequential order, so that it became easier for the computer to sort out information.

There are many other codes in use in computers and other equipment, each having been developed to satisfy a particular need – Selectric code to mechanically control the Selectric typewriter; Addressograph code to control Addressograph machines; seven segment code to drive seven segment readouts; Gray code to provide positional information of rotational devices; and a slew of error correcting and detecting codes designed to provide a more error-free information flow. ©

There's a bit of a problem with the VIC INPUT command. These suggestions will keep you out of difficulties when you are "prompting" questions and expecting to INPUT an answer.

A VIC Bug

Jim Law
Toronto

For Commodore PET users, one of the most talked about commands is the INPUT statement. Specifically, how do you prevent a user from stopping the program by hitting the RETURN key on an empty line? When Commodore designed the VIC-20 they decided to do something about this persistent problem. They decided that if a user presses RETURN and nothing else, the value(s) of the input variable(s) should remain unchanged and the program should continue.

```
10 J=77
20 INPUT "GIVE ME A VALUE";J
30 PRINT J
```

When the above program is RUN on a PET and the user hits only RETURN when asked "give me a value," the program stops and line 30 is never executed. If RUN on a VIC-20, the program will continue to line 30 and the unchanged value, 77, of the variable J will be printed. (If you want to break out of a VIC INPUT statement, press RUN/STOP and RESTORE simultaneously.) But there's a bug in this.

The INPUT Problem

In 40 column PETs of any vintage is an operating system feature called the line-wrap table. This allows the PET to link two screen rows together to get an 80 column line. In the VIC the line-wrap table is able to link up to four 22 column rows together to make one 88 column line. Without it, the VIC's full-screen editor would be unable to handle lines longer than 22 characters.

If a VIC INPUT statement places the input cursor anywhere other than on the first screen row of a multiple-linked line, the INPUT statement will erroneously return the entire multiple-linked line to the input buffer for parsing.

What this means is that if you have some rows which are linked together to make a 44, 66, or 88 column line and the input cursor is not placed on the first screen row of one of these multiple lines, the INPUT statement will "crash" or return a wrong value. If your prompt is on an un-linked line, you'll have no problem.

Suppose we have a very simple program to ask for a person's name and then print it on the screen:

```
10 PRINT "[CLEAR]";
20 INPUT "TYPE YOUR NAME AND PRESS
   RETURN";N$
30 PRINT N$
```

On a PET this works great, and if Fred is using the program he gets back "Fred" on the screen. On the VIC, he gets back "type your name and press return? Fred". The prompt in the INPUT statement in line 20 ("type your...") is longer than 22 characters and is printed over the end of the first row, creating a 44 column line. When Fred pressed RETURN, the entire 44 characters were returned to the input buffer, leading and trailing blanks dropped and the result assigned to the variable N$. This works even if Fred did nothing other than press RETURN.

The fun is just beginning.

```
10 PRINT "[CLEAR]";
20 INPUT "HOW OLD IS YOUR GRANDMOTHER"
   ;AG
30 PRINT AG
```

No matter what Fred types in response to this prying question, all he gets back is: "?redo from start". If he types "83"[RETURN], then: "how old is your grandmother? 83" gets moved to the input buffer; leading and trailing blanks are removed; and the computer attempts to convert "how old..." into a number for variable AG. Forget it, Fred. Won't work.

If your prompt is shorter than one line, everything gets sorted out correctly and the statement works as expected. Usually. Long INPUT prompts are not the only way to get linked screen rows:

```
10 PRINT "[CLEAR]LINE #1          "
20 INPUT "[HOME][DOWN] FROM LINE #2";X$
30 PRINT X$
```

Be sure to put a bunch of spaces followed by a quote mark after "line #1" on line 10. This sets the trap.

If line 10 were not there, this program would work fine, as the value of x$ would be assigned from a non-linked line. But line 10 creates a 44 character line at the top of the screen. The INPUT statement then types over the second row of the screen, which is also the second screen row in the 44 character line created in line 10. The value returned to x$ and printed in line 30 could be: "line #1   from line #2? hello".

The One Sure Cure

The only sure ways to make sure the rows from which you will input are not linked are: 1) clear the screen – this clears the line-wrap table to all unlinked rows; and 2) scroll off the bottom of the screen – any new rows placed at the bottom of the screen are always single. In addition, if you have any long prompts, put them in separate PRINT statements before your INPUT and do not use a semicolon or a comma at the end of the PRINT.

```
10 INPUT "[CLEAR] WHAT'S YOUR NAME";N$
20 PRINT "THE INPUT IS ON A SEPARATE LINE"
   : INPUT X$
25 :
30 REM NEXT LINE LEAVES THE CURSOR AT
   THE START OF A SCREEN LINE
40 INPUT "ABCDEFGHIJKLMNOPQRSTUV";X$
50 INPUT "[CLEAR]ABCDEFGHIJKLMNOPQRSTU
   [RIGHT]ABCDEF";X$
```

Line 10 should work unconditionally. Line 20 will work if the row from which the actual input is done is not linked. Lines 40 and 50 work because the character which sends the cursor to the second row is not a printable character, and so does not link the two rows together. Lines 40 and 50 will not work if the rows were already linked. ©

Have you ever wanted to use text and graphics modes on screen at the same time? *Among the techniques in this article is a method which allows you to display both character sets (uppercase/graphics and upper/lowercase) simultaneously. This IRQ-driven routine uses streamlined, carefully calculated timing delays to synchronize the 6502 with the video scan lines on the CRT. This technique, called a* kernal *on some machines, permits previously unheard-of displays. If you want to play around with the routine a little, try POKEing various values into location 684. POKE 684,64 will change the third display line into a "hybrid line" with all shifted characters made of half of each character (shift-Q would be the top of the Q and the bottom of the "ball" character). It's designed for the 8032, with suggestions for modifying it for 40-column screens.*

Three PET Innovations

Timothy Stryker
Samurai Software
Pompano Beach, FL

Have you ever written a program in which you needed to mimic the flashing cursor that appears on the screen when an INPUT statement is in progress? Sometimes it's useful to be able to GET user keystrokes, but at the same time to display a flashing cursor to the user so that he thinks he's in INPUT mode. Here are a couple of little routines that can come in handy for this:

```
            CRSON  EQU *           TURN CURSOR ON
A0  00             LDY #0          SET BASIC CURSOR-ON FLAG
84  A7             STY $A7
C8                 INY             START FLASHING
                                   IMMEDIATELY
84  A8             STY $A8
60                 RTS
            CRSOFF EQU *           TURN CURSOR OFF
78                 SEI             MASK INTERRUPTS
A5  AA             LDA $AA         CHECK CURSOR-RVS-FLD
                                   FLAG
F0  0D             BEQ ALROFF      IF OFF, OK, PROCEED
A9  01             LDA #1          SET BLINK-COUNTER TO 1
85  A8             STA $A8
58                 CLI             UNMASK INTERRUPTS
A5  A8      WAIT   LDA $A8         WAIT FOR BLINK-COUNTER
C9  01             CMP #1            TO CHANGE
F0  FA             BEQ WAIT
D0  EE             BNE CRSOFF      GO MAKE SURE IT'S OK NOW
A9  01      ALROFF LDA #1          SAFE TO KILL IT FOR GOOD
85  A7             STA $A7
58                 CLI             ALL SET
60                 RTS
```

Most people are able to figure out how to turn the thing on (usually via the BASIC command "POKE 167,0"), but the problem is that if you try to move the cursor around while in this state, little renditions of it are left behind as you go. This can be distinctly annoying, so you then experiment around with things like POKEing 167 with a one just before moving the cursor again, which also doesn't work. Eventually either you go crazy, or you write a routine like CRSOFF above, which is guaranteed to turn the cursor completely off so that you can move it someplace else and turn it on again.

The two above routines are completely relocatable, so you can stick them anyplace you like: in the tape buffers, in between BASIC lines – even on the screen. Here are the decimal POKEs from BASIC to put them in the second cassette buffer:

```
100 FOR I = 900 TO 931: READ P : POKE I,P : NE
    XT
110 DATA 160,0,132,167,200,132,168,96
120 DATA 120,165,170,240,13,169,1,133,168,88,1
    65,168
130 DATA 201,1,240,250,208,238,168,1,133,167,8
    8,96
```

Having done this, just SYS to 900 to turn the cursor on and SYS to 908 to turn it back off. Remember not to try to PRINT anything with the cursor turned on this way, or you'll find that it leaves the same little residues as before. Turn it off, do your PRINT, and then turn it back on again. Incidentally, the above will work only under the Upgrade and V4.0 ROMs. Change all the $A7's to $0224's, the $A8's to $0225's, and the $AA to a $0227 to make this work on an Original ROM machine (don't forget to change the addressing modes of the opcodes and the branch offsets too).

Hidden PRINTs

Another little item I've found useful doesn't involve machine code at all. Have you ever put debugging PRINT statements into a program, removed them once you had the thing debugged, and then later wished you had them back in again? This can be particularly likely if your program does any modem communications or that sort of thing, where the program might be working fine, but the line is getting garbled for some reason and you need to find out why. How about this: leave the debugging PRINT statements in, but make them conditional on the value of the PEEK of 152 (516 on the Original ROMs). The PEEK of 152 is a 1 if the SHIFT key is being held down, and a 0 otherwise. This way, if you want to see the debugging PRINTs, you can just hold down the SHIFT key; otherwise, the program operates normally. The reason for using the SHIFT key rather than some other key is that this way no extraneous characters will get stuck in the keyboard input buffer.

The last little item I have here is neither as

CBM Maintenance
In-House

ALL UNDER ONE ROOF

DOWNEY, CA
213-923-9361/714-778-5455

HOME OF

"VIC VILLE"

DES **Data Equipment Supply Corp.**
8315 Firestone Blvd., Downey, CA 90241

*— ONE STOP **VIC 20** CENTER —*

SOFTWAREHARDWARE**EXPANSIONS**PERIPHERALS**ACCESSORIES**

VIC 20 - VIC 1515 PRINTERS - VIC 1540 DISK DRIVES

GAMES: From our professional programmers *(Robert Winter, Ralph Orton, Dan Haste, Robert Burnett, Doug Weick, Doug Cornish)*

Astro-Miners **$17.00**
Hi-res graphics and sound space game. Requires 3k or 8k expansion.

Panzer Attack **$14.00**
Enemy tanks are attacking and you must destroy them in the combat zone. Hi-res graphics. Standard VIC.

Pedestrian Polo **$14.00**
Drive your car thru the streets. Based upon Death Race. Standard VIC.

Yahtzee **$12.00**
Solitaire version of this famous dice game. Standard VIC.

Lunar Command **$16.00**
Rescue 3 stranded astronauts from Lunar surface. Hi-res asteroid belt. Many levels. Standard VIC.

3D Labyrinth **$12.00**
Very good 3D maze game. Standard VIC.

VIC Finance **$20.00**
Buy and sell stocks on the VIC market. Good strategy game. 32k VIC game.

Air Strike **$11.00**
Fly the new super bomber V-20 on a mission. Standard VIC.

Attack on Silo III **$12.00**
You are the commander of Silo III. Defend your country. Standard VIC.

Baseball Strategy **$12.00**
The excitement of baseball as a video strategic game. Standard Vic.

Galactic Blaster **$12.00**
The fate of the Earth is in your hands. Hi-res graphics. Standard VIC.

Vic Poker **$14.00**
Play poker against the VIC. Hi-res graphics and sound. Standard VIC.

Bonzo by (c) Kavan **$20.00**
Based upon arcade game. Climb the ladders, avoid the aliens.

Boss by (c) Kavan **$39.95**
Video chess, 10 levels, 2 clocks, choose colors. Hi-res graphics. 8k expansion required.

Backgammon by (c) Kavan **$29.95**
Hi-res playing board. You against the VIC. Requires 16k expansion.

Frogger by (c) Kavan **$14.00**
Eat the flies and avoid the car. Standard VIC.

commodore

Space Phreeks **$20.00**
by Interesting Software
Pilot the space vessel "Infinity" and fight the "Space Phreeks". Machine language, hi-res graphics and sound. Arcade quality, Standard VIC.

Radar Tank & Weed Hopper **$12.00**
by Interesting Software
Two player tank game. Hi-res graphics.
Catch fast moving bugs. Hi-res graphics. Standard VIC.

Night Crawler **Coming Soon**
by Interesting Software
Fast action arcade game. Hi-res graphics.

Dunes Day 1991 **Coming Soon**
by Interesting Software
Fast action arcade game. Hi-res graphics.

Syber Attack **Coming Soon**
by Interesting Software
Fast action arcade game. Hi-res graphics.

The Black Castle **Coming Soon**
Adventure, travel the countryside, fight demons, buy goods, storm the castle. Requires Super Expander.

A Maze Ing **$12.00**
Travel through the maze. Game of skill and tense action. Standard VIC.

Gobbler **$11.00**
Sounds easy? You have 25 seconds to get him and the time gets shorter at each higher level. Standard VIC.

Hang U **$12.00**
Traditional Hangman plays against the VIC's 250 word dictionary or another person. Standard VIC.

Coggle **$11.00**
Computerized version of Boggle. Standard VIC.

Gold Brick **$14.00**
Many levels of play, sound, and color.

Simple Inventory Control **$39.95**
LIFO system works with all VICs. Complete documentation.

Mailing List **$25.00**
Keep mailing list, print reports, labels. 8k expansion or 16k expansion required.

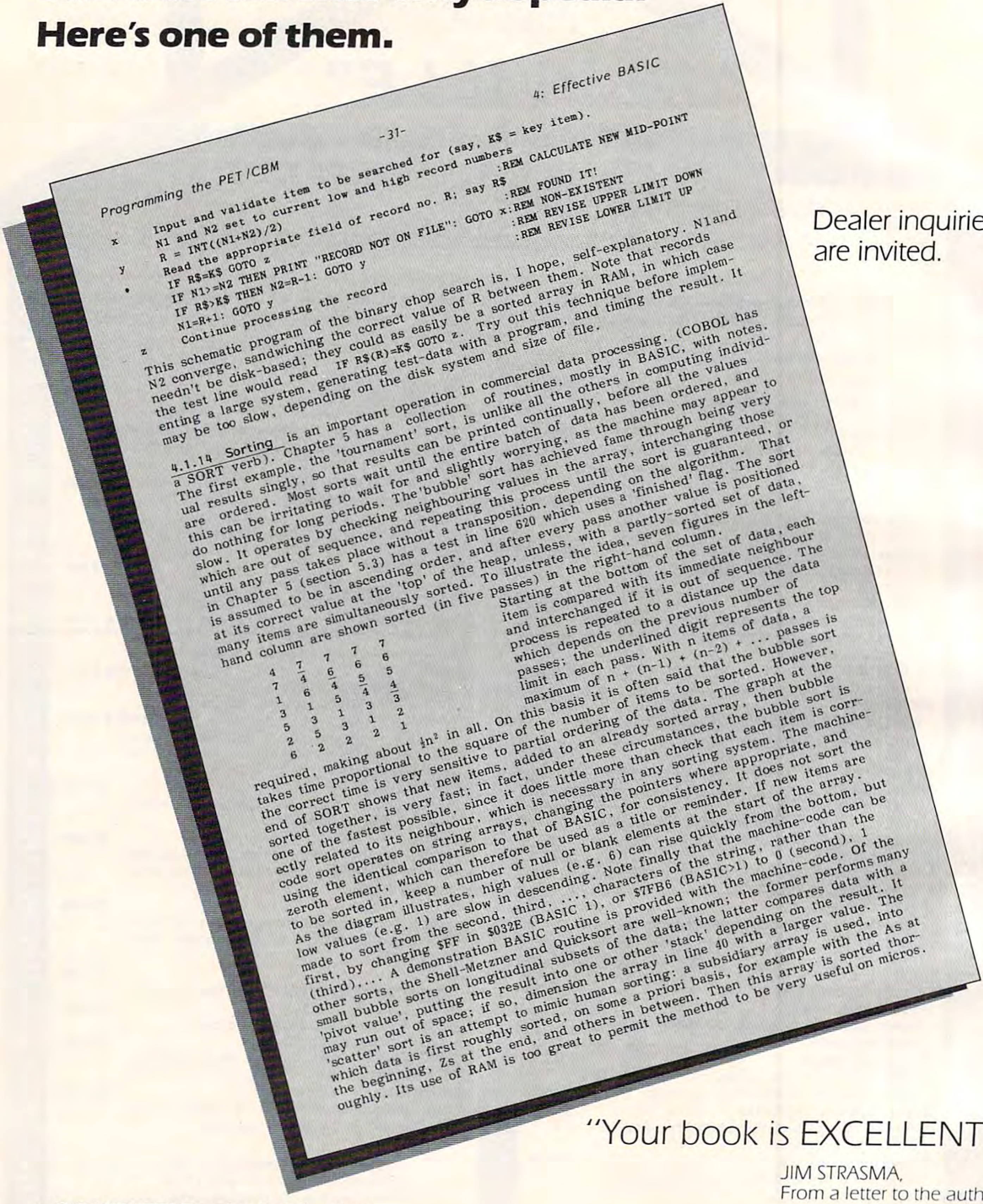
Add $3.00 for shipping & handling
Complete descriptive catalog $2.00
Dealer inquiries are most welcome.

VIC 20 - VIC 1515 PRINTERS - VIC 1540 DISK DRIVES

(213) 923-9361
(714) 778-5455

Built on a firm foundation!

There are over 500 reasons to own this reference encyclopedia. Here's one of them.



Programming the PET/CBM -31- *4: Effective BASIC*

```
      Input and validate item to be searched for (say, K$ = key item).
x     N1 and N2 set to current low and high record numbers
y     R = INT((N1+N2)/2)                          :REM CALCULATE NEW MID-POINT
      Read the appropriate field of record no. R; say R$
      IF R$=K$ GOTO z                             :REM FOUND IT!
•     IF N1>=N2 THEN PRINT "RECORD NOT ON FILE": GOTO x:REM NON-EXISTENT
      IF R$>K$ THEN N2=R-1: GOTO y                :REM REVISE UPPER LIMIT DOWN
      N1=R+1: GOTO y                              :REM REVISE LOWER LIMIT UP
z     Continue processing the record
```

This schematic program of the binary chop search is, I hope, self-explanatory. N1and N2 converge, sandwiching the correct value of R between them. Note that records needn't be disk-based; they could as easily be a sorted array in RAM, in which case the test line would read IF R$(R)=K$ GOTO z. Try out this technique before implementing a large system, generating test-data with a program, and timing the result. It may be too slow, depending on the disk system and size of file.

4.1.14 Sorting is an important operation in commercial data processing. (COBOL has a SORT verb). Chapter 5 has a collection of routines, mostly in BASIC, with notes. The first example, the 'tournament' sort, is unlike all the others in computing individual results singly, so that results can be printed continually, before all the values are ordered. Most sorts wait until the entire batch of data has been ordered, and this can be irritating to wait for and slightly worrying, as the machine may appear to do nothing for long periods. The 'bubble' sort has achieved fame through being very slow. It operates by checking neighbouring values in the array, interchanging those which are out of sequence, and repeating this process until the sort is guaranteed, or until any pass takes place without a transposition, depending on the algorithm. That in Chapter 5 (section 5.3) has a test in line 620 which uses a 'finished' flag. The sort is assumed to be in ascending order, and after every pass another value is positioned at its correct value at the 'top' of the heap, unless, with a partly-sorted set of data, many items are simultaneously sorted. To illustrate the idea, seven figures in the left-hand column are shown sorted (in five passes) in the right-hand column.

	7	7	7	7
				6
4	7	6	6	5
7	4	4	5	4
1	6	5	4	3
3	1	1	3	2
5	3	3	1	1
2	5	2	2	
6	2			

Starting at the bottom of the set of data, each item is compared with its immediate neighbour and interchanged if it is out of sequence. The process is repeated to a distance up the data which depends on the previous number of passes; the underlined digit represents the top limit in each pass. With n items of data, a maximum of n + (n-1) + (n-2) + ... passes is required, making about $\frac{1}{2}n^2$ in all. On this basis it is often said that the bubble sort takes time proportional to the square of the number of items to be sorted. However, the correct time is very sensitive to partial ordering of the data. The graph at the end of SORT shows that new items, added to an already sorted array, then bubble sorted together, is very fast; in fact, under these circumstances, the bubble sort is one of the fastest possible, since it does little more than check that each item is correctly related to its neighbour, which is necessary in any sorting system. The machine-code sort operates on string arrays, changing the pointers where appropriate, and using the identical comparison to that of BASIC, for consistency. It does not sort the zeroth element, which can therefore be used as a title or reminder. If new items are to be sorted in, keep a number of null or blank elements at the start of the array. As the diagram illustrates, high values (e.g. 6) can rise quickly from the bottom, but low values (e.g. 1) are slow in descending. Note finally that the machine-code can be made to sort from the second, third, ..., characters of the string, rather than the first, by changing $FF in $032E (BASIC 1), or $7FB6 (BASIC>1) to 0 (second), 1 (third),... A demonstration BASIC routine is provided with the machine-code. Of the other sorts, the Shell-Metzner and Quicksort are well-known; the former performs many small bubble sorts on longitudinal subsets of the data; the latter compares data with a 'pivot value', putting the result into one or other 'stack' depending on the result. It may run out of space; if so, dimension the array in line 40 with a larger value. The 'scatter' sort is an attempt to mimic human sorting: a subsidiary array is used, into which data is first roughly sorted, on some a priori basis, for example with the As at the beginning, Zs at the end, and others in between. Then this array is sorted thoroughly. Its use of RAM is too great to permit the method to be very useful on micros.

Dealer inquiries
are invited.

"Your book is EXCELLENT!"

JIM STRASMA,
From a letter to the author.

A **COMPUTE! Books** Publication
In conjunction with Level Limited.

Programming The PET/CBM

by Raeto Collin West

The book described by Jim Butterfield as

"...unquestionably the most comprehensive and accurate reference I have seen to date..."

The Reference Encyclopedia for Commodore 2000, 3000, 4000, and 8000 series computers and peripherals.

Here's just a sample of reviewer and reader reaction:

From reviewers:

Educational Computing Review by Stephen Potts
"Of all the books I have read on the PET this book *Programming the PET/CBM* by Raeto West must rank as one of the most comprehensive and readable accounts on the PET that I have ever had the pleasure to see...

"If you wish to get more from your PET than arcade games and simple teaching programs then this book is a must for your bookshelf. It does not matter whether you run on BASIC 1, BASIC 2, or BASIC 4 since all routines are supplied with addresses and changes to make them run on any machines wherever possible...

"...this book, with its lucid explanations of the PET, its useful routines and programming hints, is an essential purchase."

IPUG Magazine Review (British PET User Group) by Ron Geere
"This publication represents over a year's intensive research ... and the resulting product is a valuable work of reference. A tremendous amount of useful information has been packed in this 500+ page work at which I was so over-awed that I did not know how to start this review at first...

"This book is a must for every CBM/PET user."

From readers:

"...a book the average to advanced user cannot afford *not* to possess..."

"My copy of your *'Programming the PET/CBM* has been in daily use for nearly a month and I am finding it totally addictive, suffering severe withdrawal symptoms whenever I try half-heartedly to move on to other reading matter. It is without doubt the best book on its subject available today..."

"I have recently acquired a copy of your book *Programming the PET/CBM* and must congratulate you on its concept and on packing in so much detail. It's so very much better than anything I have had up to now that it'll be my constant reference manual."

"I have received my copy of *Programming the PET/CBM* by Raeto West and I have recommended it to several of my students. This book is so valuable that I cannot now afford to be without it."

Published exclusively in North America by **COMPUTE! Books**. The book is an astonishing reference manual of useful information. Contents include this and much more:

1 Introduction and overview: Plan of the book, sources of information, features and chronology of CBM hardware.
2 BASIC and how it works: Storage of BASIC and its variables; tokens, pointers, syntax; optimising BASIC.
3 Program and system design: Capabilities of the equipment; charts, algorithms, space, timing.
4 Effective programming in BASIC: Seventeen examples, including subroutines, dates, DATA, INPUT, rounding.
5 Alphabetic reference to BASIC keywords: Full descriptions, with examples, of all keywords, with methods for adding additional commands not present in CBM BASIC, e.g. AUTO, DEL, OLD, POP, PRINT USING, SORT, VARPTR.
6 Disk drives: Descriptions of operation and workings of disk drives, with BASIC and machine-code examples; bugs.
7 Alphabetic reference to disk BASIC commands: BASIC 4 disk commands with examples and notes.
8 Other peripherals and hardware: Tape storage and handling; printers; modem; keyboard; EPROMs; reset switches.
9 Graphics and sound: Tables of CBM characters; CRT chip; animation, bar plots, 80 by 50 etc.; user-port sound.
10 The transition to machine-code: Introductory concepts; a BASIC monitor; use of MLM, Supermon, Extramon; easy examples.
11 More 6502 machine-code: 6502 hardware features; eighteen common problems in programming; debugging.
12 Alphabetic reference to 6502 opcodes: Examples, notes, and explanations on each opcode from ADC to TYA.
13 Using ROM routines: IRQ, NMI, RESET; the Kernel; examples – modifying LIST; ordinary and relocating loaders.
14 Effective 6502 programming: Assemblers; CHRGET and wedges; PIAs, VIA, IEEE; common mistakes.
15 Index to BASIC ROMs and RAM: Memory map of RAM and ROM, detailing and comparing BASICs 1, 2, and 4.
16 Mathematical programming: Precision; equations; statistics; simulation; finance; calculus; machine-code.
17 Programming in business and education: Examples, applications and pitfalls in business and education.
Appendices: 6502 reference charts; Supermon listings; ASCII; glossary

Plus many programs, diagrams and charts. Paperback, 504 pages. ISBN 0 942386 04 3. **$24.95.**

To Order
Programming The PET/CBM
Call
TOLL FREE 800-334-0868
In NC Call 919-275-9809
Or send coupon to
COMPUTE! Books, P.O. Box 5406, Greensboro, NC 27403

In England, order from Level Limited, P.O. Box 438, Hampstead, London, NW3 1BH. Price in England is £14.90, including P & P.

Please send ________ copy (copies) of **Programming The PET/CBM** at $24.95 each. (In the US and Canada, add $3.00 shipping and handling. Outside North America add $9.00 for air mail delivery, $3.00 for surface delivery).

All orders must be prepaid in US funds (money order, check, or charge).

□ Payment Enclosed
Please charge my □ VISA □ MasterCard □ Am. Express
Account No. ________ Expires ____ / ____
Name ________
Address ________
City ________ State ________ Zip ________
Country ________
Allow 4-6 weeks for delivery. Foreign surface delivery allow 2-4 months.

simple nor as clean as either of the above, but it can be very useful in certain circumstances. Suppose you plan to make a display using the lowercase character set, but you find that you also need certain characters from the graphics set. You can have only one set or the other enabled at one time, so what do you do? You can (a) change out the character-generator ROM; (b) change your mind about what you really wanted to do to begin with; or (c) give up computers and start a farm. Well, consider this: the PET IRQ's are exactly synchronized with the vertical retrace interval of the CRT. You could make use of this fact to write a little routine which grabs the IRQ's, enables one character set for a while, and then enables the other character set and allows the PET to go about its business. In this way, the top N lines of the CRT can be made to display characters from a different set from what the rest of the screen displays.

Phantom Windows

I have worked out the numbers on this for the 8032. The screen is updated 60 times a second, and each update takes exactly 12 ½ milliseconds. There are 250 scan lines, so each line takes exactly 50 microseconds. Of this, ten microseconds are consumed in the horizontal retrace, so 40 microseconds of character output are provided for. This works out to exactly 500 nanoseconds per character-width of beam scan (if these numbers sound awfully exact, it's because they are – this is one *stable* display).

The difference between 12½ milliseconds and 1/60th of a second is a little less than 4.2 milliseconds. The PET apparently takes its IRQ in the dead center of the vertical retrace, so you have to blow away a couple of milliseconds right off the bat before the beam even turns on again at the top of the screen. From that point on, each additional 50-microsecond interval that you wait before switching the character set back gives you one additional display scan line using the other set. Here are the kinds of routines you need:

Twin-Screen Disassembly

```
ADDR  OBJECT        ASSEMBLY SOURCE

027A                         ORG  634          START OF TAPE-1 BUFFER
027A                DUALON   EQU  *            TURN ON DUAL CHAR MODE
027A  A5 90                  LDA  $90          SAVE CURRENT IRQ VEC
027C  8D CA 02               STA  HIVEC          IN HOLDING REGISTER
027F  A5 91                  LDA  $91
0281  8D CB 02               STA  HIVEC+1
0284  78                     SEI               MASK INTERRUPTS
0285  A9 95                  LDA  #<DUAIRQ     POINT IRQ VEC AT DUAIRQ
0287  85 90                  STA  $90
0289  A9 02                  LDA  #>DUAIRQ
028B  85 91                  STA  $91
0289D 58                     CLI               UNMASK INTERRUPTS
028E  60                     RTS
028F                DUAOFF   EQU  *            TURN OFF DUAL CHAR MODE
028F  78                     SEI               MASK INTERRUPTS
0290  20 BF 02               JSR  RSIRQV       RESTORE ORIG IRQ VEC
0293  58                     CLI               UNMASK INTERRUPTS
0294  60                     RTS
0295                DUAIRQ   EQU  *            DUAL CHAR MODE IRQ RTN
0295  A5 97                  LDA  $97          STOP KEY PRESSED?
0297  C9 03                  CMP  #3
0299  D0 06                  BNE  NOSTOP       NO, CONTINUE
029B  20 BF 02               JSR  RSIRQV       YES, RESTORE ORIG VEC
029E  6C 90 00               JMP  ($90)        GO ABOUT BUSINESS
02A1  A9 0E         NOSTOP   LDA  #14          USE LOWER CASE CHAR SET
02A3  8D 4C E8               STA  59468
02A6  A0 07                  LDY  #7           DELAY 36 USEC TO GET
02A8  88            IDELAY   DEY                IN SCAN-LINE SYNC
02A9  D0 FD                  BNE  IDELAY
02AB  A2 47                  LDX  #41+30       THREE FULL LINES
02AD  A0 08         NXTSCN   LDY  #8           DELAY 50 USEC PER SCAN
02AF  EA                     NOP
02B0  EA                     NOP
02B1  88            SDELAY   DEY
02B2  D0 FD                  BNE  SDELAY
02B4  CA                     DEX               DELAYED LONG ENOUGH?
```

continued on page 171

```
02B5  D0 F6                BNE  NXTSCN      NO, DELAY ANOTHER SCAN
02B7  A9 0C                LDA  #12         YES, USE GRAPHICS NOW
02B9  8D 4C E8             STA  59468
02BC  6C CA 02             JMP  (HIVEC)     DO REGULAR IRQ THING
02BF              RSIRQV   EQU  *           RESTORE IRQ VEC ROUTINE
02BF  AD CA 02             LDA  HIVEC       DO IT TO IT
02C2  85 90                STA  $90
02C4  AD CB 02             LDA  HIVEC+1
02C7  85 91                STA  $91
02C9  60                   RTS
02CA              HIVEC    EQU  *           HOLDING REG FOR IRQ VEC
```

Unfortunately, it is impossible for a facility like this to be made position-independent on the 6502. In fact, because of the critical nature of the timing loops, you must take care if you do relocate it to ensure that none of these loops cross memory page boundaries; otherwise, they will take longer than they are supposed to and the routines will not work properly.

As given here, a SYS to 634 will cause the top three print lines of an 8032 display to appear in the lowercase character set, while the rest of the screen appears in the graphics set. The argument to the LDX at $02AB determines the number of video scan lines of delay used: setting this to (decimal) 42 will give you exactly one scan line of lowercase if the display is in the uncompressed mode, 43 will give you two scans, etc. Use 41 as your delay base in the uncompressed mode, and add ten scan lines per print line of delay. In the compressed display mode, the delay base is 67, and you should add only eight scan lines per print line.

If you would rather have graphics in the upper part of the screen and lowercase toward the bottom, simply interchange the 14 at $02A2 with the 12 at $02B8. The STOP-key check from $0295 to $029E is there just so that if you are calling this thing from BASIC or RPL, you don't have to worry about the display staying in this weird mode if you abort execution via the STOP key. Normally, your program should SYS to DUAOFF at 655 to restore the display to normal when exiting.

Naturally, the above madness cannot be used verbatim on a 40-column machine, but with a little effort you should be able to coerce it into working. Keep in mind that Upgrade machines, and some (but not all) 4.0 40-column machines stick a four into location $97 when the STOP key is hit, instead of a three.

Program 1. LOADER For Dual Screen Routine

```
634 DATA 165, 144, 141, 202, 2, 165
640 DATA 145, 141, 203, 2, 120, 169
646 DATA 149, 133, 144, 169, 2, 133
652 DATA 145, 88, 96, 120, 32, 191
658 DATA 2, 88, 96, 165, 151, 201
664 DATA 3, 208, 6, 32, 191, 2
670 DATA 108, 144, 0, 169, 14, 141
676 DATA 76, 232, 160, 7, 136, 208
682 DATA 253, 162, 71, 160, 8, 234
688 DATA 234, 136, 208, 253, 202, 208
694 DATA 246, 169, 12, 141, 76, 232
700 DATA 108, 202, 2, 173, 202, 2
706 DATA 133, 144, 173, 203, 2, 133
712 DATA 145, 96
800 FOR ADRES=634TO713:READ DATTA:X=X+DATTA:PO
    KE ADRES,DATTA:NEXT ADRES
801 IFX<>10264THENPRINT"ERROR IN DATA STATEMEN
    TS"
```

GTIA Demonstration

Jerry White
Levittown, NY

This GTIADEMO program* won't work properly on a machine that has the CTIA chip. However, if you have the GTIA, this demo will provide a colorful and animated display you won't soon forget.

At first glance, you may be reluctant to enter such a large program just for a demo. For this reason, I have separated the three independent modules with REMarks. The first two modules do not require very much typing at all. I suggest you enter each module separately, then LIST them onto disk or cassette. When you RUN the first module, you just might find the incentive to continue typing.

The third module is, in my opinion, the most impressive to watch. Admittedly, it does take about a minute and a half before the animation begins. I'm quite sure you will find the effort required to enter this program, and the brief delay, to be justly rewarded.

Module three of the demo program uses Graphics 10. In this mode, we have nine hues and luminances. In the demo, a small assembler subroutine is used to shift values in the color registers. This provides a hypnotic, animation effect. To end the program, simply press any key.

**Software adapted by permission of Atari, Inc.*

```
100 GOSUB 770:REM IDENTIFICATION DISPLAY
     (VERSION DATE 4/2/82 JERRY WHITE)
110 REM GTIADEMO MODULE ONE
120 JW=15:GRAPHICS 9:SETCOLOR 4,JW,0
130 FOR Y=55 TO 0 STEP -10:FOR X=0 TO 24
    :C=X:IF X>11 THEN C=24-X
140 C=C+3:Z=Y+(X):D=INT(SQR(144-(X-12)*(
    X-12)))/2:COLOR 15-C:PLOT Z,Y+7-D:DR
    AWTO Z,Y+7+D:COLOR C:DRAWTO Z,180-Y+
    D
150 NEXT X:NEXT Y
160 JW=JW-1:IF JW=0 THEN 250
170 SETCOLOR 4,JW,0:READ P
180 FOR ME=8 TO 0 STEP -0.5:SOUND 0,P,10
    ,ME:SOUND 1,P-1,10,ME:SOUND 2,P+1,10
    ,ME:SOUND 3,P+2,10,ME:NEXT ME
190 POKE 540,30
200 IF PEEK(540)<>0 THEN 200
210 GOTO 160
220 REM DATA FOR SOUND ROUTINE
230 DATA 243,230,217,204,193,182,173,162
    ,153,144,136,128,121,60
240 REM GTIADEMO MODULE TWO
250 GRAPHICS 11:DIM ML$(21):FOR ME=1 TO
    21:READ IT:ML$(ME,ME)=CHR$(IT):NEXT
    ME
260 REM DATA FOR MACHINE LANGUAGE SHIFT
    COLOR ROUTINE
270 DATA 104,162,0,172,193,2,189,194,2,1
    57,193,2,232,224,8,144,245,140,200,2
    ,96
280 FOR I=1 TO 8:READ A:POKE 704+I,A+224
    :NEXT I
290 REM DATA FOR COLOR REGISTER POKES
300 DATA 2,4,6,8,6,4,2,2
310 FOR I=0 TO 38:COLOR Q:X=I:Y=I*2:PLOT
     X,Y
320 DRAWTO 79-X,Y:PLOT X,Y+1:DRAWTO 79-X
    ,Y+1:DRAWTO 79-X,190-Y
330 DRAWTO X,190-Y:PLOT 79-X,190-Y+1:DRA
    WTO X,190-Y+1:DRAWTO X,Y
340 Q=Q+1:IF Q>8 THEN Q=1
350 NEXT I:JW=0:ME=JW
360 X=USR(ADR(ML$)):SOUND 0,JW,2,2:SOUND
     1,16-JW+10,12,2:SOUND 2,JW,0,2
370 JW=JW+1:IF JW>15 THEN JW=0:ME=ME+1
380 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,
    0,0,0
390 IF ME>12 THEN 420
400 SETCOLOR 1,JW,6:GOTO 360
410 REM GTIADEMO MODULE THREE
420 GRAPHICS 10:JW=-2:FOR ME=705 TO 712:
    JW=JW+18:POKE ME,JW:NEXT ME

430 NUM=1:FOR Y=0 TO 191:COLOR NUM:PLOT
    0,Y:DRAWTO 79,191-Y:NUM=NUM+0.4167:I
    F NUM>8 THEN NUM=1
440 NUM=NUM+1:NEXT Y
450 FOR X=79 TO 0 STEP -1:COLOR NUM:PLOT
     X,0:DRAWTO 79-X,191:NUM=NUM+1:IF NU
    M>8 THEN NUM=1
460 NEXT X
470 JW=-2:FOR ME=705 TO 712:JW=JW+18:POK
    E ME,JW:NEXT ME
480 FOR ME=0 TO 359 STEP 2:NUM=8:READ X,
    Y:COLOR 0:PLOT X,Y:IF ME<181 THEN 52
    0
490 FOR JW=1 TO 45:LOCATE X,Y+JW,IT:IF I
    T=0 THEN POP :GOTO 520
500 COLOR NUM:PLOT X,Y+JW:NUM=NUM-1:IF N
    UM<1 THEN NUM=8
510 NEXT JW
520 NEXT ME:KULR=10
530 JW=USR(ADR(ML$)):SOUND 0,0,0,0:SOUND
     1,0,0,0:SOUND 2,0,0,0:SOUND 3,0,0,0
540 IF PEEK(764)<>255 THEN 580
550 P=P+5:IF P<249 THEN SOUND 0,P,10,2:S
    OUND 1,P+2,10,2:SOUND 2,P+4,10,2:SOU
    ND 3,P+6,10,2:GOTO 530
560 P=0:KULR=KULR+1:IF KULR>15 THEN KULR
    =1
570 SETCOLOR 1,KULR,6:GOTO 530
580 GRAPHICS 0:POKE 764,255:? :? "END OF
     GTIA DEMONSTRATION":? :END
590 DATA 60,96,60,97,60,98,60,99,60,99,6
    0,100,60,101,60,101,60,102,60,103
600 DATA 59,103,59,104,59,105,58,105,58,
    106,58,106,57,107,57,108,57,108,56,1
    09
610 DATA 56,109,55,110,55,110,54,111,54,
    111,53,112,53,112,52,113,52,113,51,1
    13
620 DATA 50,114,50,114,49,114,49,115,48,
    115,47,115,47,116,46,116,45,116,45,1
    16
```

```
630 DATA 44,116,43,116,43,116,42,116,41,
    116,40,116,40,116,39,116,38,116,38,1
    16
640 DATA 37,116,36,116,36,116,35,116,34,
    116,34,115,33,115,32,115,32,114,31,1
    14
650 DATA 31,114,30,113,29,113,29,113,28,
    112,28,112,27,111,27,111,26,110,26,1
    10
660 DATA 25,109,25,109,24,108,24,108,24,
    107,23,106,23,106,23,105,22,105,22,1
    04
670 DATA 22,103,21,103,21,102,21,101,21,
    101,21,100,21,99,21,99,21,98,21,97
680 DATA 20,96,21,96,21,95,21,94,21,94,2
    1,93,21,92,21,92,21,91,21,90
690 DATA 22,90,22,89,22,88,23,88,23,87,2
    3,87,24,86,24,85,24,85,25,84
700 DATA 25,84,26,83,26,83,27,82,27,82,2
    8,81,28,81,29,80,29,80,30,80
710 DATA 31,79,31,79,32,79,32,78,33,78,3
    4,78,34,77,35,77,36,77,36,77
720 DATA 37,77,38,77,38,77,39,77,40,77,4
    0,76,41,77,42,77,43,77,43,77
730 DATA 44,77,45,77,45,77,46,77,47,77,4
    7,78,48,78,49,78,49,79,50,79
740 DATA 50,79,51,80,52,80,52,80,53,81,5
    3,81,54,82,54,82,55,83,55,83
750 DATA 56,84,56,84,57,85,57,85,57,86,5
    8,87,58,87,58,88,59,88,59,89
760 DATA 59,90,60,90,60,91,60,92,60,92,6
    0,93,60,94,60,94,60,95,60,96
770 GRAPHICS 17:POSITION 6,6:? #6;"GTIA 
    DEMO"
780 ? #6:? #6:? #6;"{3 SPACES}BY JERRY W
    HITE":POKE 764,255:POKE 710,154:POKE
     708,14:POKE 711,74:FOR JW=1 TO 100:
    NEXT JW
790 ? #6:? #6:? #6;"{4 SPACES}press any
    key":? #6:? #6:? #6;"{6 SPACES}to be
    gin":BLINK=0
800 BLINK=BLINK+1:IF BLINK>20 THEN POKE
    709,0:BLINK=-20:POKE 53279,0
810 IF BLINK=0 THEN POKE 709,234:POKE 53
    279,0
820 IF PEEK(764)=255 AND PEEK(53279)>6 T
    HEN 800
830 POKE 764,255:RETURN
```

©

For Upgrade and 4.0 BASIC PET/CBM's of any memory size, this program can be entered and used whether or not you know machine language. It's a handy utility which instantly deletes any line ranges from within a BASIC program.

PET Machine Language Delete

Roger Burrows
Ottawa, Ontario

If you never make mistakes or revise programs, you won't need this tool. However, if you're like me you'll find this fast line delete program very useful. With it, you can delete a range of lines (or a single line) by a few keystrokes. And it won't take more than a second.

The "BASIC loader" program here will enter the program into the computer wherever you wish. If you RUN it as presented, it loads into the upper memory of an 8K PET. To "protect" it from being overwritten by a BASIC program, you should then type: POKE 53,31. If you have 16K of memory, substitute the following in line 20: FOR I = 16208 TO 16381 (etc...) and then use POKE 53,63 to protect. Finally, for 32K machines, substitute in line 20: FOR I = 32592 TO 32765 and use POKE 53,127.

The program is *relocatable* because the machine language code has no internal JMPs or JSRs or other self-references to particular addresses within its boundaries. By changing the POKE loop in line 20, you can send it wherever you want to within RAM memory. Program 1 is for Upgrade BASIC PET/CBM's. Substitute the lines in Program 2 if your computer uses 4.0 BASIC. Also note the checksum change in line 10 for 4.0 BASIC.

When you want it to delete lines from a BASIC program, type SYS 8016 (or SYS 16208 or 32592, whatever your starting address is in line 20) plus the start and end line numbers. For example:

SYS 8016,120,180

would delete lines 120 through 180. To delete a single line, you can use SYS 8016,120.

Program 1. Upgrade Version

```
10 CK = 21339: REM CHANGE TO 22051 FOR 4.0
     BASIC
20 FOR I = 8016 TO 8189: READ X: Y=Y+X: PO
     KEI,X: NEXTI
100 DATA 32,118,0,240,109,32,112,0,240,104,
     176,103,32,115,200
110 DATA 165,17,133,178,133,180,165,18,133,
     179,133,181,32,118,0
120 DATA 240,18,32,112,0,240,13,176,76,32,1
     15,200,165,17,133
130 DATA 180,165,18,133,181,165,178,133,17,
     165,179,133,18,32,44
140 DATA 197,165,92,133,182,165,93,133,183,
     165,180,133,17,165,181
150 DATA 133,18,32,44,197,144,12,160,0,177,
     92,133,184,200,177
160 DATA 92,24,144,6,165,92,133,184,165,93,
     133,185,197,183,144
170 DATA 8,208,10,165,182,197,184,144,4,96,
     76,3,206,162,1
180 DATA 160,0,177,184,145,182,208,7,232,22
     4,3,176,13,144,2
190 DATA 162,0,200,208,238,230,185,230,183,
     208,232,152,101,182,133
200 DATA 42,165,183,105,0,133,43,160,171,32
     ,86,241,160,174,32
210 DATA 86,241,32,66,196,76,57,196,0,0,0,0
     ,0,0,0
220 IF CK <> Y THEN PRINT "ERROR IN DATA ST
     ATEMENTS"
```

Program 2. Substitutions For 4.0 BASIC

```
100 DATA 32,118,0,240,109,32,112,0,240,104,
     176,103,32,246,184
120 DATA 240,18,32,112,0,240,13,176,76,32,2
     46,184,165,17,133
130 DATA 180,165,18,133,181,165,178,133,17,
     165,179,133,18,32,163
140 DATA 181,165,92,133,182,165,93,133,183,
     165,180,133,17,165,181
150 DATA 133,18,32,163,181,144,12,160,0,177
     ,92,133,184,200,177
170 DATA 8,208,10,165,182,197,184,144,4,96,
     76,0,191,162,1
200 DATA 42,165,183,105,0,133,43,160,171,32
     ,133,241,160,174,32
210 DATA 133,241,32,182,180,76,173,180,0,0,
     0,0,0,0,0
```

©

To thoroughly document your computer's BASIC or operating system (or any significant machine language program), you need to create a commented map of the routines. "Resource" is a collection of BASIC programs which, working together, help you to produce annotated disassemblies.

Last month **COMPUTE!** *published explanatory text and the first program. "Resource" now concludes with the rest of the programs and some example results. The author used "Resource" to aid in generating annotated cross reference lists for the OSI version of Microsoft's BASIC.*

Resource: Part II

Mapping Machine Language Code

T. R. Berger, Coon Rapids, MN

The tables which accompany these final programs comprising "Resource" are selections from annotated cross reference lists for OSI-Microsoft 8K disk BASIC from 0S65D V3.2 NMHZ disks. The tables were produced by using "Resource" and the annotations derive from both Jim Butterfield's memory maps (**COMPUTE! II**, June/July 1980) and my maps of OS65D (**COMPUTE!**, January-March 1981).

All addresses within the example tables are in hex and the first address on any line is the called address. Thereafter, the addresses refer to the place where the *calling* code resides. In addition, many of the addresses have preceding letters. These letters mean different things in different tables. In a JMP or JSR table, an M means the calling code is a JUMP instruction.

An S means the calling code is a JUMP TO SUBROUTINE instruction. In the MEMORY table, the letter is always the first letter of the calling opcode. For example,

1DF3 STA $0100,Y

is referenced in the table beside 0100 as S1DF3. The Zpage table has no leading letters. This table was produced by an early version of "Resource," before the extra information was added.

Table 1. Keyword Action Addresses

Word	Token	Address						
END	80	082A	TO	9D		TAN	B9	1FF2
FOR	81	0748	FN	9E		ATN	BA	2056
NEXT	82	0C4B	SPC	9F		PEEK	BB	1688
DATA	83	08F9	THEN	A0		LEN	BC	15F6
INPUT	84	0B2C	NOT	A1	1E88	STR$	BD	12E9
DIM	85	0F24	STEP	A2		VAL	BE	1627
READ	86	0B58	+	A3	16D9	ASC	BF	1605
LET	87	09A6	–	A4	16C2	CHR$	C0	1566
GOTO	88	08A6	*	A5	18F4	LEFT$	C1	157A
RUN	89	087E	/	A6	1A0D	RIGHT$	C2	15A6
IF	8A	0929	^	A7	1E4F	MID$	C3	15B1
RESTORE	8B	080A	AND	A8	0E8C	NF	C4	ERROR
GOSUB	8C	0889	OR	A9	0E89	SN	C5	ERROR
RETURN	8D	08D3	>	AA		RG	C6	ERROR
REM	8E	093C	=	AB		OD	C7	ERROR
STOP	8F	0828	<	AC		FC	C8	ERROR
ON	90	094C	SGN	AD	1B34	OV	C9	ERROR
NULL	91	086D	INT	AE	1BC7	OM	CA	ERROR
WAIT	92	169C	ABS	AF	1B53	US	CB	ERROR
EXIT	93	223C	USR	B0	22F7	BS	CC	ERROR
DISK	94	2253	FRE	B1	1204	DD	CD	ERROR
DEF	95	1235	POS	B2	1225	/O	CE	ERROR
POKE	96	1693	SQR	B3	1E45	ID	CF	ERROR
PRINT	97	21AA	RND	B4	1F66	TM	D0	ERROR
CONT	98	0853	LOG	B5	18B3	LS	D1	ERROR
LIST	99	06B9	EXP	B6	1EC1	ST	D2	ERROR
CLEAR	9A	067C	COS	B7	1FA2	CN	D3	ERROR
NEW	9B	0662	SIN	B8	1FA9	UF	D4	ERROR
TAB	9C							

Table 2. Memory Table

```
.      Zpage
0001   L173A
0002   L1733
0003   L172C
0004   L1725
0016   S05EF L05F2 S0612
0017   L0512
0018   S062B
00A0   B0E8B
00A2   B0909
00FF   S1CF6 S1D67 S1D70 S1DB4 S1DBE L1DD1 S1E
      0F
.
.      Stack
0100   S1DF3 S1E14
0101   L03A6 L107B S1086 S1DEE
0102   L03B1 C03C2 L0FC9 L1077 S1081 S1E05
0103   L03B6 C03BB S1E01
0104   S1E0A
0109   L0C75 S0C8B
010F   L0C90
0110   L0C95
0111   L0C9F
0112   L0C9A
01DE   L0E79
01DF   L0E7E
.
.      Start of keyword address table
0200   L07F9
0201   L07F5
.
.      Start of operator hierarchy and address
      table
0266   C0D20 C0D48 L0D64
0267   L0D53
0268   L0D4F
.
.      Table of BASIC keywords (Start $0284)
0283   L061D
0284   S05E0 L0622 L0736 L073E
.
.      Error messages
0364   L0456
0365   L045C
.
.      BIT hiding code
07A9   B057C
08A2   B10CF
0EA2   B08E3
1410   B19BE
.
.      Constants
1E21   A1D91
1E22   A1D8A
1E23   A1D83
1E24   A1D7C
.
.      Operand pointing to IO flags
21D5   S2104
21DA   S210A
.
```

```
.      Stack pointer
226F   S211F
.
.      Table index for OS buffer write routine
228A   S217F
.
.      Buffer read write data for OS
22C8   L22E2
22C9   L22D6
22CA   L22DC
.
.      USR pointer to OS and disk
22F2   S22D9
22F3   S22DF
.
.      OS Input flag
2321   S20F5 L2101 S21D6 S2201 L2215
.
.      OS Output flag
2322   S20F8 L2107 S215D S21DB L21FE S2208
.
.      OS Passed char. (Control C)
2325   L0819 S0823
.
.      OS Disk sector number
265E   S22AC
.
.      BIT hiding code
28A9   B0E0F
.
.      OS Default IO flag
2AC6   L20F2
.
.      BIT hiding code
2CA9   B0E12
.
.      OS Read buffer pointer
2CE5   S2142
.
.      OS End of buffer on read
2CED   S2113
.
.      Transient GET and PUT pointer
2E7A   L22A6
.
.      OS Swapped value ($E1,$E2) Start pointer
     for buffer read
305A   S2116
305B   S2119
.
.      Pointer to SOURCE File header
3178   S2126 L2273
.
.      Number of tracks in SOURCE File
317D   S2136
.
.      BIT hiding code
3FA9   B0AEB
A4A2   B1AC4
```

Table 3. Zpage Table

```
. ;Index for Zpage, Jump vectors for BASIC
00  05AD 05D9 0609 0627 1357
01  135B 170D 173D 1832 1838 1848 184C 184E
   1850
02  135F 1736 182E 1834 185A 185C 185E
03  172F 182A 1830 1866 1868 186A
04  1728 1826 182C 1872 1874 1876
.
```

(Table 3. continued)

```
.  ;Search Character
0A  1320 1BDF 1E70 1EE2
0A  090C 0914 0916 0976 0998 0B95 0B9D 0E97
   0EB2 130D
.
.  ;Scan between quotes FLAG
0B  130F 1324
0B  05B3 0607 060D 0910 0912 0918 091E 0BA2
   0E9D 0EA9
.
.  ;POINTER: Input Buffer, # of subscripts
0C  0EAB 0EB0 0EB4 1028 1091 10DE 1108 112F
   1175 11AD
0C  049C 04F7 050F 05D1 05E9 061A 0E8E 0E95
   0E9B 0EA7
.
.  ;Default DIM FLAG
0D  0F33 1059 109E 10D7 1113 116E
```

```
.
.  ;Type: FF=string 00=numeric
0E  09B5 0A4D 0B91 0CBF 0D10 0D36 0DB4 0E31
   0ED3 0F44
0E  0F63 105E 1097 1204 121A 1368 1601
.
.  ;
0F  09B2 0BBC 0E36 0F46 0F71 105B 109A
.
.  ;FLAG: DATA scan; LIST quote; memory
10  05AB 05B9 0600 1372 1396 139F
.
.  ;Subscript FLAG; FNx FLAG
11  06A8 074A 0F6B 0F81 0F8C 1240 126A
.
.  ;0=Input; $40=GET; $98=READ
12  0B0D 0B5E 0B80 0C12
.
```

Table 4. JMP and JSR Table

```
.    Jump vector for commands
0003  S047A
.
.     Jump vector for evaluation
006F  M0D84
.
.     Jump vector for functions
00A1  S0E83
.
.    CHRGET subroutine:  get BAISC character
00C0  S1615 S1C05 S1C12 S1C35 S2163 M220B
     S2259
00C0  S0484 S06D0 S079F S07E0 M07FD S0960
     S09A0 S0AC1 S0B8E
00C0  S0CB3 S0D00 S0DB6 M0E1B S0E4D S0F48
     S0F53 S0F7B S103D
.
.     Subentry: get previous character
00C6  S1652 S16A3 M2160 S21B0 S21EB
00C6  S0CAC S0CE7 S0F28 S0F30 S0F37 S108A
     S12C5 S15B5 M1624
00C6  S06C7 S0798 S089D S092C S0941 S0A32
     S0B7B S0BC1 S0BDD
```

```
.
.     Search stack for FOR and GOSUB activity
03A1  S074F S08D9 S0C58
.
.     Open space in memory
03CF  S0504 S0FF1
03D6  S14A2
.
.     Test stack depth
0412  S075D S088B S0CDD
.
.     Check available memory
041F  S03CF S10EC S1142
.
.     Send error message then:
044C  M1194
044E  M1A87
044E  M085B M08E6 M0CCA M0E20 M10D2 M1232
     M1352 M14D4 M1821
.
.     Warm start BASIC
```

continued on p. 178

```
.PASS 2
.       RESOURCE 2 ** SOURCE AND EQUATE
        FILE BUILDER **
.
.               TWO BUFFERS REQUIRED
.
.
.
.
.
.
10 REM *** RESOURCE 2 - SOURCE AND EQUATE
   FILE BUILDER ***
20 REM T.R. BERGER 11/80
.
.  ** REMOVE COMMA AND SEMICOLON **
50 POKE2972,13:POKE2976,13
.
60 PRINT:PRINT"RESOURCE ** STEP 2 - BUILD
   SOURCE & EQUATE FILE **"
70 PRINT:PRINT
80 INPUT"SCRATCH FILE";SF$
90 INPUT"RESOURCE FILE";OF$
100 PRINT:INPUT"SYMBOL FILE";FS$
110 INPUT"EQUATE FILE";EF$
.
140 SP$="      "
.
.  ** COUNT SYMBOLS **
.
180 DISK OPEN,6,FS$
.
.  * SYMBOL COUNTER *
200 SN=-1
210 INPUT #6,IN$
220 IF IN$="XIT" THEN250
230 SN=SN+1
240 GOTO210
250 DISK CLOSE, 6
.
.  ** LOAD SYMBOLS **
.
260 DISK OPEN,6,FS$
.
.  * DIMENSION STRING AND MARKER ARRAYS *
280 DIM SS$(SN), SS(SN)
300 FOR I=0 TO SN
310 INPUT #6, SS$(I)
320 NEXT I
330 DISK CLOSE,6
.
.  ** MAIN PROGRAM **
.
.  * LINE NUMBERS AND INCREMENT *
380 CL=10000: IN=10
390 DISK OPEN,6,SF$
400 DISK OPEN,7,OF$
.
.  * LOOP BACK HERE *
410 INPUT #6,IN$
420 IFIN$="XIT"THEN640
.
.  * GET ADDRESS OF LINE *
440 A1$=LEFT$(IN$,4)
.
.  ** BINARY SEARCH FOR SYMBOL **
.
```

```
   * SEARCH *
470 L=0: R=SN
480 M=INT((L+R)/2)
.
.  * EXIT HERE IF NOT FOUND *
490 IF L>R THEN OU$=SP$+MID$(IN$,5): GOTO
    580
.
.  * EXIT HERE IF FOUND *
500 IF A1$=SS$(M) THEN 550
510 IF A1$>SS$(M) THEN L=M+1: GOTO480
520 R=M-1: GOTO480
.
.  * END OF SEARCH *
.
.  * CREATE SYMBOL AND MARK ADDRESS USED
   *
550 SS(M)=1: OU$="HH"+IN$
.
.  * CREATE RESOURCE LINE *
580 OU$=STR$(CL)+" "+OU$
.
.  * INCREMENT LINE NUMBER *
600 CL=CL+IN
.
.  * PRINT LINE *
610 PRINT #7,OU$
620 PRINTOU$
630 GOTO410
.  * LOOP BACK FROM HERE *
.
.  * CLOSE FILES *
640 PRINT #7,IN$
650 DISK CLOSE,7
660 DISK CLOSE,6
.
.  * END OF MAIN PROGRAM *
.
.  * WRITE TWO BYTE EQUATES *
710 DISK OPEN,7,EF$
.
.  * FIRST LINE NUMBER *
720 CL=5000
.
.  * TITLE *
730 PRINT #7,STR$(CL)+" ;EQUATES"
740 CL=CL+IN
.
.  * COUNTER FOR EQUATES *
750 K=0
.
.  * PRINT EQUATES *
760 FORI=0TOSN
.
.  * SKIP SYMBOLS WHICH ARE LABELS *
770 IF SS(I)=1 THEN 830
780 A1$=STR$(CL)+" HH"+SS$(I)+" = $"+SS$(
    I)
790 PRINT #7, A1$
800 PRINT A1$
.
.  * NEXT LINE NUMBER *
810 CL=CL+IN
.
.  * INCREMENT EQUATES COUNT *
820 K=K+1
830 NEXTI
```

```
840 PRINT #7,"XIT"
850 DISK CLOSE,7
.
.  * FINISHED WITH EQUATES *
870 PRINT:PRINT
880 PRINT"CODE SOURCE FILE REGENERATED":
    PRINT
890 PRINT TAB(10) "RESOURCE FILE:  "OF$
900 PRINTTAB(10) "EQUATES FILE: ";EF$
910 PRINT:PRINTTAB(10)"SCRATCH FILE: "SF$
920 PRINTTAB(10) "SYMBOL FILE:  "FS$
930 PRINTTAB(9) SN+1" SYMBOLS"
940 PRINTTAB(9) K" EQUATES"
950 PRINT:PRINT"PASS 2 COMPLETE"
960 PRINT:PRINT:END
```

```
.PASS 3
.
.     RESOURCE 3 ** CROSS REFERENCE FILES
      **
.
.           TWO BUFFERS REQUIRED
.
.
.
.
.
100 REM *** RESOURCE 3 - CROSS REFERENCE
    BUILDER ***
110 REM *** T.R. BERGER 11/80
.
.  * DELETE COMMA AND SEMICOLON *
120 POKE2972,13: POKE2976,13
.
130 PRINT:PRINT"RESOURCE ** STEP 3 -
    CROSS REFERENCE GENERATOR"
140 PRINT:PRINT
150 PRINTTAB(20)"TYPES OF REFERENCES"
160 PRINT:PRINT"B"TAB(10)"BRANCH"
170 PRINT"J"TAB(10)"JSR AND JMP"
180 PRINT"M"TAB(10)"MEMORY"
190 PRINT"Z"TAB(10)"ZPAGE"
.
200 PRINT:INPUT"YOUR CHOICE (J/B/M/Z)";
    CR$
210 IFCR$<>"B"ANDCR$<>"J"ANDCR$<>"M"AND
    CR$<>"Z"THEN200
.
220 PRINT:INPUT"SCRATCH FILE";SF$
230 INPUT"REFERENCE FILE";RF$
240 PRINT:INPUT"NUMBER OF REFERENCES";NR
.
.  * DIMENSION ARRAYS *
250 DIM SS$(NR), SA$(NR), V(NR)
.
.  * SET SYMBOL NUMBER AND TYPE *
260 T=1: SN=-1
270 IF CR$="M" THEN T=2
280 IF CR$="Z" THEN T=3
.
.  * SYMBOL PLUCKER *
290 S=13:NL=4
300 IF CR$="Z" THEN S=15: NL=2
.
.  ** MAIN PROGRAM **
.
310 DISK OPEN,6,SF$
320 DISK OPEN,7,RF$
.
.  * LOOP BACK HERE *
330 INPUT #6,IN$
340 IF IN$="XIT" THEN600
.
.  * TOO SHORT, NO SYMBOL *
350 IF LEN(IN$)<16 THEN330
.
.  * CHECK FOR NO SYMBOL *
360 IF MID$(IN$,11,2)<>"HH" THEN330
.
.  * DISPLAY LINE WITH SYMBOL *
370 PRINT IN$
.
.  * DETERMINE SYMBOL TYPE *
380 ON T GOSUB740 ,790,860
.
.  * CHECK FOR RELEVANT SYMBOL *
390 IF FL=0 THEN330
.
.  * GET ADDRESS OF LINE *
400 A1$=M$+LEFT$(IN$,4)
.
.  * GET SYMBOL *
410 A2$=MID$(IN$,S,NL)
.
.  * SEARCH SYMBOL TABLE *
.  * BINARY SEARCH *
.
420 L=0: R=SN
.
.  * SYMBOL NOT FOUND, INSERT IT *
430 IF L>R THEN480
440 M=INT((L+R)/2)
.
.  * SYMBOL IN TABLE *
450 IF A2$=SS$(V(M)) THEN540
460 IF A2$>SS$(V(M)) THEN L=M+1: GOTO430
470 R=M-1: GOTO430
.
.  * ADD A SYMBOL *
480 SN=SN+1: SS$(SN)=A2$
.
.  * POINT TO ITS PROPER POSITION IN
   ORDERING *
490 IF L=SN THEN530
500 FOR I=SN-1 TO L STEP -1
510 V(I+1)=V(I)
520 NEXT I
530 V(L)=SN: M=L
.
.  * ADD A CROSS REFERENCE *
540 SA$(V(M))=SA$(V(M))+" "+A1$
.
.  *  CHECK IF CROSS REF LINE TOO LONG *
550 IF LEN(SA$(V(M)))<50 THEN330
.
.  * PRINT CROSS REF LINE *
560 PRINT #7, SS$(V(M))+" "+SA$(V(M))
570 PRINT SS$(V(M))+" "+SA$(V(M))
580 SA$(V(M))=""
590 GOTO330
.  * LOOP BACK FROM HERE *
.
.  * CLOSE SCRATCH FILE *
```

```
600 DISK CLOSE,6
.
.  * PRINT REMAINING CROSS REF LINES *
610 FOR I=0 TO SN
620 IF SA$(V(I))="" THEN650
630 PRINT #7, SS$(V(I))+" "+SA$(V(I))
640 PRINT SS$(V(I))+" "+SA$(V(I))
650 NEXT I
660 PRINT #7,"XIT"
670 DISK CLOSE,7
.
.  * END OF MAIN PROGRAM *
.
680 PRINT:PRINT
690 PRINTTAB(10)CR$" REFERENCES COMPLETE"
700 PRINTTAB(10)"SYMBOLS FOUND:  "SN+1
710 PRINTTAB(10)"REFERENCE FILE:  "RF$
730 PRINT:PRINT:END
.
. ** SUBROUTINES **
.
.  * BRANCH AND J (T=1) *
740 IF MID$(IN$,6,1)<>CR$ THEN FL=0: GOTO7
    80
.
.  * SIFT OUT BIT INSTRUCTIONS *
750 IF MID$(IN$,6,3)="BIT" THEN FL=0: GOTO
    780
.
.  * LABEL FOR TYPE *
760 M$=MID$(IN$,7,1)
770 FL=1
780 RETURN
.
. * MEMORY (T=2) *
790 M $=MID$(IN$,6,1)
800 A2$=MID$(IN$,13,1)
810 IF M $="J"ORA2$="Z" THEN FL=0: GOTO850
.
.  * SIFT OUT BRANCHES *
820 IF M $="B"ANDMID$(IN$,6,3)<>"BIT" THEN
    FL=0: GOTO850
.
.  * LABEL TYPE *

840 FL=1
850 RETURN
.
.  * Z PAGE REFERENCES (T=3) *
860 IF MID$(IN$,13,1)<>CR$ THEN FL=0: GOTO
    900
870 M$=" "
.
.  * LABEL FOR INDEXING *
880 IF LEN(IN$)>16 THEN M$=RIGHT$(IN$,1)
890 FL=1
900 RETURN
```

```
.PASS 4
.
.     RESOURCE 4 ** Z PAGE EQUATE FILE **
.
.          ONE BUFFER REQUIRED
.
.
.
.
.
100 REM *** RESOURCE 4 - ZPAGE EQUATES ***
110 REM T.R. BERGER 11/80
120 PRINT:PRINT
.
130 PRINT"RESOURCE ** STEP 4 - ZPAGE
    EQUATE FILE **"
.
140 PRINT:PRINT:INPUT"ZPAGE CROSS
    REFERENCE FILE";ZF$
150 INPUT"ZPAGE EQUATE FILE";ZE$
.
160 PRINT:INPUT"NUMBER OF SYMBOLS";NS
.
.  * LINE NUMBER AND INCREMENT *
170 FL=1000: IN=10
.
.  * DIMENSION ARRAYS *
180 DIM SS$(NS), V(NS)
.
.  * SYMBOL COUNTER *
190 SN=-1
.
.  ** LOAD SYMBOLS **
.
200 DISK OPEN,6,ZF$
210 PRINT:PRINT"LOADING SYMBOLS"
.
.  * LOOP BACK HERE *
220 INPUT #6,IN$
230 IF IN$="XIT" THEN380
.
.  * JUST THE ZPAGE REFERENCE *
240 IN$=LEFT$(IN$,2)
.
.  * PUT SYMBOLS IN ORDER *
.
.  * SEARCH FOR SYMBOL *
.  * BINARY SEARCH *
.
250 L=0:R=SN
.
.  * GO ADD NEW SYMBOL *
260 IF L>R THEN310
270 M=INT((L+R)/2)
.
.  * HAVE THIS ONE, GET ANOTHER *
280 IF IN$=SS$(V(M)) THEN220
290 IF IN$>SS$(V(M)) THEN L=M+1: GOTO260
300 R=M-1: GOTO260
.
.  * ADD SYMBOL TO LIST *
310 SN=SN+1: SS$(SN)=IN$
.
.  * POINT TO ITS PROPER POSITION IN
   ORDERING *
320 IF L=SN THEN360
330 FOR I=SN-1 TO L STEP -1
340 V(I+1)=V(I)
350 NEXT I
360 V(L)=SN
.
370 GOTO220
.  * LOOP BACK FROM HERE *
.
380 DISK CLOSE,6
```

```
.
.  * SYMBOLS ALL LOADED *
.
.  * PRINT EQUATES *
.
390 DISK OPEN,6,ZE$
.
.  * TITLE *
400 PRINT #6,STR$(FL)+" ;ZPAGE EQUATES"
.
.  * PRINT EQUATES NOW *
410 FOR I=0 TO SN
420 FL=FL+IN
430 IN$=STR$(FL)+"  HHZZ"+SS$(V(I))+" = $
    "+SS$(V(I))
440 PRINT #6,IN$
450 PRINT IN$
460 NEXT I
.
470 PRINT #6,"XIT"
.
.  * BUFFER 6 REQUIRES A PUT *
480 DISK PUT
490 DISK CLOSE,6
500 PRINT:PRINT
.
.  * OUTPUT DATA *
510 PRINTTAB(9) SN+1" SYMBOLS"
520 PRINTTAB(10) "ZPAGE CROSS REF FILE:
    "ZF$
530 PRINTTAB(10) "ZPAGE EQUATE FILE:
    "ZE$
540 PRINT:PRINT:END
```

```
.ONE PASS RESOURCE
.
.     RESOURCE S ** ONE PASS PROGRAM **
.
.        TWO BUFFERS REQUIRED
.
.
.
.
.
100 REM *** RESOURCE S ***
110 REM T.R. BERGER 2/81
120 PRINT
130 PRINTTAB(10)"RESOURCE - SINGLE PASS"
140 PRINT
.
.  ** REMOVE COMMA AND SEMICOLON **
150 POKE2972,13:POKE2976,13
.
160 INPUT"SOURCE FILE";SF$
170 INPUT"RESOURCE FILE";RF$
180 INPUT"EQUATE FILE";EF$
190 INPUT"CROSS REF FILE";CF$
200 INPUT"SCRATCH FILE";JF$
.
210 PRINT:INPUT"NUMBER OF SYMBOLS";NS
220 INPUT"NUMBER OF ZPAGE SYMBOLS";NZ
.
.  ** DIMENSION SYMBOL AND POINTER ARRAYS
   **
230 DIM SS$(NS),SB$(NS),SJ$(NS),SM$(NS),V(
    NS),SS(NS)
240 DIM ZS$(NZ),ZA$(NZ),U(NZ)
.
.  ** SYMBOL COUNTERS **
250 SN=-1:ZN=-1:SP$="          "
.
. ** FIRST PASS **
.
260 DISK OPEN,6,SF$
270 DISK OPEN,7,JF$
.
.  ** LOOP BACK HERE **
280 INPUT #6,IN$
.
290 IF IN$="XIT" THEN790
300 IF LEN(IN$)<15 THEN280
.
.
. ** ADJUST SOURCE, PICK UP SYMBOLS **
.
.  A1$=XXXX  ADDRESS
.  A2$=OPCODE +
.  A3$=OPERAND (SYMBOL)
.  A4$=ADDR MODE
.  OU$=A1$+A2$+A3$+A4$
.  IN$=INPUT FROM OSI DISASSEMBLER
.
.
.
310 A3$="":A4$=""
.
.  ** GET ADDRESS **
320 A1$=LEFT$(IN$,4)
.
.  ** DO ERRORS **
330 IFMID$(IN$,13,1)="?"THENA2$=" .BYTE $
    "+MID$(IN$,6,2):GOTO760
.
.  ** DO REFORMATTING **
.
.  ** ELIMINATE END SPACES **
340 IN$=MID$(IN$,12): L=LEN(IN$)
350 IF MID$(IN$,L,1)=" " THEN L=L-1: GOTO
    350
360 IN$=LEFT$(IN$,L)
.
.  ** DO IMPLIED, ACCUMULATOR, IMMEDIATE
   ADDRESSSING **
370 IF L<7ORMID$(IN$,6,1)="#" THEN A2$=IN
    $: GOTO760
.
.  ** ADJUST OPERAND POSITION **
380 IFMID$(IN$,6,1)="$"THENK=7:A2$=LEFT$(
    IN$,5)+" HH":GOTO400
390 K=8:A2$=LEFT$(IN$,6)+"HH"
.
.  ** ZPAGE CHECK **
400 M=K+2
.
.  ** DO ZPAGE OPERANDS **
410 IFM>LTHENA3$=RIGHT$(IN$,2):A2$=A2$+"Z
    Z":GOTO500
420 IF MID$(IN$,M,1)>"/"THEN440
430 A3$=MID$(IN$,K,2):A2$=A2$+"ZZ":A4$=
    MID$(IN$,M): GOTO500
.
.  ** TWO BYTE OPERAND CHECK **
440 M=K+4
```

```
.
.  ** DO TWO BYTE OPERANDS **
450 IFM>LTHENA3$=RIGHT$(IN$,4):GOTO470
460 A3$=MID$(IN$,K,4): A4$=MID$(IN$,M)
.
.  ** SEARCH FOR SYMBOL **
470 GOSUB1670
.
.  ** SYMBOL NOT FOUND, INSERT IT **
480 IF L>R THEN660
.
.  ** SYMBOL FOUND, ADD CROSS REF **
490 GOTO720
.
.  ** SEARCH FOR ZPAGE REF **
500 L=0: R=ZN
.
.  ** SYMBOL NOT FOUND, INSERT IT **
510 IF L>R THEN560
520 M=INT((L+R)/2)
.
.  ** SYMBOL FOUND, ADD CROSS REF **
530 IF A3$=ZS$(U(M)) THEN620
540 IF A3$>ZS$(U(M)) THEN L=M+1: GOTO510
550 R=M-1: GOTO510
.
.  ** ADD SYMBOL **
560 ZN=ZN+1: ZS$(ZN)=A3$
.
.  ** POINT TO PROPER POSITION IN ORDERING
   **
570 IF L=ZN THEN610
580 FOR I=ZN-1 TO L STEP-1
590 U(I+1)=U(I)
600 NEXT I
610 U(L)=ZN: M=L
.
.  ** GET ADDRESSING MODE **
620 A5$=" "
630 IF A4$<>"" THEN A5$=RIGHT$(IN$,1)
.
.  ** ADD CROSS REF TO STRING **
640 ZA$(U(M))=ZA$(U(M))+" "+A5$+A1$
650 GOTO760
.
.  ** ADD SYMBOL **
660 SN=SN+1:SS$(SN)=A3$
.
.  ** POINT TO PROPER POSITION IN ORDERING
   **
670 IF L=SN THEN710
680 FOR I=SN-1 TO L STEP-1
690 V(I+1)=V(I)
700 NEXT I
710 V(L)=SN: M=L
.
.  ** FIND CORRECT CROSS REF TABLE **
720 A5$=MID$(A2$,2,1): A0=1
730 IF A5$="B"ANDMID$(A2$,2,3)<>"BIT" THEN
    A0=2
740 IF A5$="J" THEN A0=3
.
.  ** ADD CROSS REF TO TABLE **
750 ON A0 GOSUB1640 ,1650 ,1660
.
.  ** GENERATE LINE FOR SCRATCH FILE **
760 OU$=A1$+A2$+A3$+A4$
```

```
770 PRINT #7,OU$: PRINT OU$
780 GOTO280
.  ** LOOP BACK HERE **
.
.  ** CLOSE SOURCE AND SCRATCH FILES **
790 PRINT #7, IN$
810 DISK CLOSE,6
820 DISK CLOSE,7
.
.  ** END FIRST PASS **
.
.  ** PASS 2, WRITE CROSS REF FILES **
.
830 DISK OPEN,7,CF$
840 PRINT #7,".  CROSS REFERENCES"
850 PRINT #7,"."
860 PRINT #7,".  ZPAGE"
870 PRINT #7,"."
.
.  ** DO ZPAGE REFS **
880 FOR I=0 TO ZN
890 A0$=ZA$(U(I)): ZA$(U(I))="": A2$=ZS$(
    U(I))
.
.  ** BREAK UP LONG LINES, PRINT FILE **
900 GOSUB1730
910 NEXT I
920 PRINT #7,".":PRINT #7,"."
930 PRINT #7,".  JMP & JSR"
940 PRINT #7,"."
.
.  ** DO JMP & JSR REFS **
950 FOR I=0 TO SN
960 A0$=SJ$(V(I)): SJ$(V(I))="": A2$=SS$(
    V(I))
.
.  ** BREAK UP LONG LINES, PRINT FILE **
970 GOSUB1730
980 NEXT I
990 PRINT #7,".": PRINT #7,"."
1000 PRINT #7,".  MEMORY": PRINT #7,"."
.
.  ** DO MEMORY REFS **
1010 FOR I=0 TO SN
1020 A0$=SM$(V(I)): SM$(V(I))="": A2$=SS$
     (V(I))
.
.  ** BREAK UP LONG LINES, PRINT FILE **
1030 GOSUB1730
1040 NEXT I
1050 PRINT #7,".": PRINT #7,"."
1060 PRINT #7,".  BRANCH": PRINT #7,"."
.
.  ** DO BRANCH REFS **
1070 FOR I=0 TO SN
1080 A0$=SB$(V(I)): SB$(V(I))="": A2$=SS$
     (V(I))
.
.  ** BREAK UP LONG LINES, PRINT FILE **
1090 GOSUB1730
1100 NEXT I
1110 PRINT #7,"XIT"
1120 DISK CLOSE,7
.
.  ** END REF FILES **
.
.  ** GENERATE RESOURCE FILE **
```

```
.
1130 DISK OPEN,6,JF$
1140 DISK OPEN,7,RF$
.
.  ** LINE NUMBER AND INCREMENT **
1150 CL=10000: IN=10
.
.  ** LOOP BACK HERE **
1160 INPUT #6,IN$
1170 IF IN$="XIT" THEN1260
.
.  ** GET ADDRESS OF LINE **
1180 A3$=LEFT$(IN$,4)
.
.  ** SEARCH FOR SYMBOL **
1190 GOSUB1670
.
.  ** SYMBOL FOUND, MARK IT, ENTER LABEL
   **
1200 IF L<=R THEN SS(M)=1: OU$="HH"+IN$:
     GOTO1220
.
.  ** SYMBOL NOT FOUND, DELETE ADDRESS **
1210 OU$=SP$+MID$(IN$,5)
.
.  ** ADD LINE NUMBER AND OUTPUT **
1220 OU$=STR$(CL)+" "+OU$
1230 CL=CL+IN
1240 PRINT #7,OU$: PRINT OU$
1250 GOTO1160
.  ** LOOP BACK FROM HERE **
.
.  ** CLOSE SCRATCH AND RESOURCE FILES **
1260 PRINT #7,IN$
1280 DISK CLOSE,6
1290 DISK CLOSE,7
.
.  ** RESOURCE FILE DONE **
.
.  ** DO EQUATE FILES **
.
1300 DISK OPEN,7,EF$
.
.  ** LINE NUMBER **
1310 CL=1000
1320 PRINT #7,STR$(CL)+"  ;EQUATE FILE"
1330 CL=CL+IN: PRINT #7,STR$(CL)+" ;"
1340 CL=CL+IN: PRINT #7,STR$(CL)+"  ;ZPAGE
     "
1350 CL=CL+IN: PRINT #7,STR$(CL)+" ;"
.
.  ** DO ZPAGE EQUATES **
1360 FOR I=0 TO ZN
1370 CL=CL+IN
1380 PRINT #7,STR$(CL)"  HHZZ"ZS$(U(I))"
     = $"ZS$(U(I))
1390 PRINT STR$(CL)"  HHZZ"ZS$(U(I))" = $"
     ZS$(U(I))
1400 NEXT I
.
1410 CL=CL+IN
1420 PRINT #7, STR$(CL)+" ;"
1430 CL=CL+IN: PRINT #7,STR$(CL)+" ;"
1440 CL=CL+IN: PRINT #7,STR$(CL)+"  ;TWO
     BYTE"
1450 CL=CL+IN: PRINT #7,STR$(CL)+" ;"
.
.  ** DO TWO BYTE EQUATES **
```

```
1460 FOR I=0 TO SN
1470 IF SS(I)=1 THEN1510
1480 CL=CL+IN
1490 PRINT #7,STR$(CL)"  HH"SS$(V(I))" =
     $"SS$(V(I))
1500 PRINT STR$(CL)"  HH"SS$(V(I))" = $"S
     S$(V(I))
1510 NEXT I
.
1520 PRINT #7,"XIT"
1530 PRINT #7,"E": PRINT #7,"E"
1540 DISK CLOSE,7
.
.  ** END OF EQUATES **
.
.  ** FINAL DATA **
.
1550 PRINT:PRINTTAB(10)"RESOURCE COMPLETE
     "
1560 PRINTTAB(7)SN+1" SYMBOLS"
1570 PRINTTAB(7)ZN+1" ZPAGE LOCATIONS"
1580 PRINTTAB(8)"SOURCE FILE:      ";SF$
1590 PRINTTAB(8)"SCRATCH FILE:     ";JF$
1600 PRINTTAB(8)"EQUATE FILE:      ";EF$
1610 PRINTTAB(8)"RESOURCE FILE:    ";RF$
1620 PRINTTAB(8)"CROSS REF FILE:   ";CF$
1630 PRINT:PRINT:END
.
.  ** END OF PROGRAM **
.  ** SUBROUTINES **
.
.  ** MEMORY CROSS REFS **
1640 SM$(V(M))=SM$(V(M))+" "+A5$+A1$:
     RETURN
.
.  ** BRANCH CROSS REFS **
1650 SB$(V(M))=SB$(V(M))+" "+MID$(A2$,3,1)
     +A1$: RETURN
.
.  ** JMP & JSR CROSS REFS **
1660 SJ$(V(M))=SJ$(V(M))+" "+MID$(A2$,3,1)
     +A1$: RETURN
.
. ** SEARCH FOR SYMBOL **
1670 L=0: R=SN
.
.  ** SYMBOL NOT FOUND **
1680 IF L>R THEN RETURN
1690 M=INT((L+R)/2)
.
.  ** SYMBOL FOUND **
1700 IF A3$=SS$(V(M)) THEN RETURN
1710 IF A3$>SS$(V(M)) THEN L=M+1: GOTO1680
1720 R=M-1: GOTO1680
.
.  ** BREAK UP LONG LINES, PRINT CROSS REF
   FILE **
1730 L=LEN(A0$)
1740 IF L=0 THEN RETURN
1750 IF L<49 THEN A1$=A0$: A0$="": GOTO
     1770
1760 A1$=LEFT$(A0$,48): A0$=MID$(A0$,49)
1770 PRINT #7,A2$" "A1$: PRINT A2$" "A1$
1780 GOTO1730
```

©

"Message Board" allows you to put up to 20 pages of messages on a Graphics 2 screen and display them repeatedly for as long as you like. It makes a fine message board for any kind of meeting and can even be a title maker for home video recorders.

Atari Message Board

Dennis J. Harkins
Hatfield, PA

We use an expanded version of this program to generate messages for a cable TV channel here, and we leave our Atari 800 running for hours, flipping through page after page of program schedules and community announcements. We also make titles and credits for our student video productions with the program, since the Atari has a nice "clean" video output through the DIN socket on the side.

String With Dynamic Keyboard

The program stores the information in DATA statements through Atari's "dynamic" keyboard technique (see Bruce Frumker's article in **COMPUTE!**, August 1981, #15, for a good example of this). This keeps the program short and simple and makes it easy to save messages on either disk or cassette.

The program first asks whether you'd like to write messages or run your previously stored messages. If you choose to write messages, it then asks for a color combination for your message and background. You then enter your message line by line. Hitting RETURN leaves a blank line. After the 10th (or 20th) line is entered, you can see it displayed and have the option to make any changes. After you are satisfied with the "page," you are asked if you want to write another page. When you are through writing messages, you can then either RUN them or save the program to disk or tape to RUN later.

Choosing colors and luminance values can be time consuming at first, so try the following values when asked for color information, if you don't have any favorite color combinations:

1, 14, 7, 8	(Gold on Blue)
7, 8, 1, 14	(Blue on Gold)
2, 14, 4, 6	(Gold on Red)
14, 12, 11, 6	(Gold on Green)
7, 2, 3, 6	(Blue on Red)
7, 12, 0, 6	(Blue on Gray)

You will learn a lot about the Atari's colors by playing a bit with the combinations. We have a little color selector program that lets you preview the colors, but I'll save that for another time. Remember to avoid using commas in your message, since commas in a DATA statement tell the computer to end that particular "read." You may notice that the "window" is still at the bottom of each page: it's just the same color and luminance as the background. We use this for our logo and the date.

The loop in line 2620 controls the amount of time each page is displayed, and you will probably want to change this to suit your messages.

I'm sure you can think of some refinements (I've already added dozens of them). How about messages in four colors? It's a simple matter to add the extra SETCOLOR information and then type, in lowercase and inverse characters, some of the lines of your message. Experiment a bit. You can come up with some pretty fancy looking messages with a little work.

Program Details

Line 2010 dimensions an Answer string and a Message string (note its length) and sets the Line Number (LN) variable to zero.

Line 2130 sets the value of A to correspond (inversely!) to the graphics mode wanted. This allows it to be used in line 2310 to input 10 or 20 lines when you are writing a message.

Line 2210 inputs B, C, D, E as your color values. To see what's happening here, look at line 2550 where the values are read into SETCOLOR

```
2010 DIM A$(1),M$(20):LN=0
2020 ? CHR$(125):POKE 752,1:POSITION 11,
     8:? "MESSAGE GENERATOR"
2030 ? :? :? "{9 SPACES}BY DENNIS HARKIN
     S"
2040 FOR DELAY=1 TO 500:NEXT DELAY
2050 ? CHR$(125):POKE 752,0:POSITION 2,8
2060 ? "TO WRITE MESSAGES, TYPE W"
2070 ? :? "TO RUN YOUR MESSAGES,TYPE R"
2080 ? :? :? "THEN HIT RETURN"
2090 ? :? :INPUT A$:IF A$="R" THEN 2500
2100 IF A$<>"W" THEN 2050
2110 ? CHR$(125):POSITION 2,8
2120 ? "LARGE OR SMALL LETTERS?"
2130 ? :? :? "TYPE 1 FOR LARGE, 2 FOR SM
     ALL":? :? :INPUT A
2140 IF A<1 OR A>2 THEN GOTO 2110
2150 ? CHR$(125):POSITION 2,4
2160 ? "NOW ENTER YOUR COLOR CHOICES"
2170 ? :? :? "REMEMBER TO ENTER FOUR NUM
     BERS{8 SPACES}SEPARATED BY COMMAS"
2180 ? :? :? "EXAMPLES:"
2190 ? :? "{10 SPACES}1,14,7,8"
2200 ? :? "{5 SPACES}OR{3 SPACES}7,8,1,1
     4"
2210 ? :? :? "THEN HIT RETURN":? :? :INP
     UT B,C,D,E
2220 LN=LN+1
2230 ? CHR$(125)
```

COMPUTE!'s Second Book Of Atari

After only three years on the market, the Atari 400/800 microcomputers have become among the most popular personal computers ever made. So it was no surprise when *COMPUTE!'s First Book of Atari*, a collection of the best Atari articles published during 1980-81 in **COMPUTE!** Magazine, also became a "bestseller" with Atari enthusiasts. The first printing sold out in just a few months.

That's why we've followed up with *COMPUTE!'s Second Book of Atari*. Available immediately, the *Second Book of Atari* continues **COMPUTE!**'s tradition for personal computer users.

But the *Second Book of Atari* differs from the *First Book* in one important respect – all the articles are totally new and previously unpublished. The *Second Book of Atari* includes such interesting articles as "Page Flipping," "Fun With Scrolling," "Perfect Pitch," "Player-Missile Drawing Editor," and "TextPlot Makes a Game." Whole chapters are devoted to subjects such as "Advanced Graphics and Game Utilities," "Programming Techniques," and "Beyond BASIC." With 250 pages – more than 25 percent thicker than the *First Book* at the same price – the *Second Book of Atari* is crammed with information and ready-to-type program listings. And the book is spiral-bound to lie flat and is fully indexed for quick reference.

Best of all, *COMPUTE!'s Second Book of Atari*, like **COMPUTE!** Magazine itself, is written and edited to appeal to all computer enthusiasts – beginners and experts alike. Priced at only $12.95.

Available at computer dealers and bookstores nationwide. To order directly call TOLL FREE 800-334-0868. In North Carolina call 919-275-9809. Or send check or money order to **COMPUTE! Books**, P.O. Box 5406, Greensboro, NC 27403.

Add $2.00 shipping and handling. Outside the U.S., add $4.00 for air mail, $2.00 for surface mail. All orders prepaid, U.S. funds only.

```
2240 ? "{DOWN}";LN;" DATA ";A;",";B;",";
     C;",";D;",";E
2250 ? :? :? "CONT"
2260 POSITION 0,0
2270 POKE 842,13:STOP
2280 POKE 842,12
2290 IF A=1 THEN LN=LN+9
2300 IF A=2 THEN LN=LN+4
2310 FOR G=1 TO 10*A
2320 ? CHR$(125):POSITION 6,4:? "TYPE LI
     NE ";G;" OF MESSAGE HERE"
2330 POSITION 10,8:? "------------------
     --"
2340 POSITION 2,10:? "ONLY 20 CHARACTERS
      WILL BE ACCEPTED"
2345 POSITION 2,13:? "REMEMBER THAT YOU
     CANNOT USE COMMAS{3 SPACES}IN YOUR
     MESSAGE"
2350 POSITION 9,7:POKE 752,0:INPUT M$:SE
     TCOLOR 1,7,4:POKE 752,1:? "
     {CLEAR}"
2360 ? "{DOWN}";LN;" DATA ";M$
2370 ? :? :? "CONT"
2380 POSITION 0,0
2390 POKE 842,13:STOP
2400 POKE 842,12:? "{CLEAR}":SETCOLOR 1,
     7,10
2410 ON A GOTO 2420,2425
2420 IF G<>10*A THEN LN=LN+10:GOTO 2430
2425 IF G<>10*A THEN LN=LN+5
2430 NEXT G
2440 ? CHR$(125):POSITION 13,8:? "MESSAG
     E ENTERED"
2450 POSITION 1,14:? "DO YOU WANT TO SEE
      THAT PAGE (Y OR N)";:INPUT A$
2460 IF A$="Y" THEN 2700
2480 ? :? :? :? "DO YOU HAVE ANOTHER PAG
     E (Y OR N)";:INPUT A$
2485 IF A$="Y" THEN GOTO 2110
2490 IF A$<>"N" THEN 2480
2495 GOTO 2050
2500 ? CHR$(125):? :? :? "PAGES TO BE RU
     N";:INPUT Z
2510 RESTORE :FOR W=1 TO Z
2520 READ A,B,C,D,E
2530 IF A=1 THEN GRAPHICS 2
2540 IF A=2 THEN GRAPHICS 1
2550 SETCOLOR 0,B,C:SETCOLOR 2,D,E:SETCO
     LOR 4,D,E
2560 FOR N=0 TO 10*A-1
2570 READ M$
2580 LET P=(20-LEN(M$))/2
2590 POSITION P,N
2600 PRINT #6;M$
2610 NEXT N
2620 FOR T=1 TO 3000:NEXT T
2630 POKE 77,0:NEXT W
2640 GOTO 2510
2700 RESTORE LN-99
2710 READ A,B,C,D,E
2720 IF A=1 THEN GRAPHICS 2
2730 IF A=2 THEN GRAPHICS 1
2740 SETCOLOR 0,B,C:SETCOLOR 2,D,E:SETCO
     LOR 4,D,E
2750 FOR N=0 TO 10*A-1
2760 READ M$
2770 LET P=(20-LEN(M$))/2
2780 POSITION P,N
2790 PRINT #6;M$
2800 NEXT N
2810 ? "THIS IS PAGE ";LN/100
2820 ? "HIT RETURN TO CONTINUE":INPUT A$
2830 GRAPHICS 0:POSITION 2,10:? "DO YOU
     WANT TO RETYPE THAT PAGE{7 SPACES}(
     Y OR N)";:INPUT A$
2840 IF A$="Y" THEN LN=LN-100:GOTO 2110
2850 IF A$<>"N" THEN 2830
2860 GOTO 2480
```

statements.

Lines 2230-2280 use the "dynamic keyboard" routine to enter the first data line.

Lines 2310-2340 set up the loop to enter each line of the message, again using the "dynamic keyboard" routine. Notice the way the LN variable increments depending on the number of lines in your message. This makes a lot of little options easier. LIST 201-300, for example will always LIST page 3, no matter what length each page is.

Lines 2500-2640 are the actual display loop. Note that line 2580 centers the data string in a line of 20 spaces.

Lines 2700-2820 are the same display loop modified for the "PREVIEW" option.

To use the program as a titler for your home video productions, change line 2620 to:

```
2620 GET#1,KEY
```

and line 2500 to:

```
2500 PRINT CHR$(125):? :? :?
  "PAGES TO BE RUN" ;:INPUT
  Z:OPEN #1,4,0,"K:"
```

You'll be able to prepare your titles in the same way as you create messages and hit any key to change to the next credits. ©

Mass-delete lines, renumber, and search – many programming aids become available when you put the Assembler Cartridge to work on a BASIC program.

Editing BASIC Programs With The Atari Assembler/Editor Cartridge

Dennis Allen
San Jose, CA

It's probably a safe bet that the majority of Atari owners do most if not all of their programming in BASIC. However, this article will show you how to put the Atari Assembler/Editor Cartridge to work providing some BASIC programming aids.

The Editor/Assembler consists of three programs: the Assembler, the Debugger, and the Editor. The Assembler converts source programs into object code which is executable by the computer. The Debugger is a programming aid which does just that – debugs machine language programs. The Editor is a fairly simple line-oriented text editor through which you can enter and correct or modify source programs – *or any other ASCII text files*. It is the Editor which will allow you to modify your BASIC programs.

Apart from the I/O commands which allow you to SAVE or LOAD source and object code files to and from the cassette or disk, there are four major commands of interest to the BASIC programmer: Renumber, Delete, Replace, and Find. *Renumber* is a simple renumbering utility. It allows you to specify the starting line number and the increment. *Delete* lets you erase any line or group of lines. *Find* will show you all of the statement lines where a certain string of characters appears. *Replace* will allow you to substitute any string of characters for any other string – a line at a time or throughout the entire program at once.

A Major Typing Task Avoided

Program 1 is a simple demonstration of the use of these editing features. Suppose you've just CLOADed this program into your computer. The first thing you will notice is the line number increment. You would like to insert some graphics routines in the middle of the sound routines, but you can't because a thoughtless programmer used a line number increment of one. Now pretend this is a very large program and the routines you want to insert will extend the overall program beyond your memory limit. You now have two problems which would require many hours of retyping to solve. Not with the Editor!

Step one is to get the program in a form the Editor can LOAD. To do this you type LIST"C" [*For disk, use LIST"D:Name"*]. At this point, it is a good idea to make a hardcopy listing of the program you are editing, especially if it's a long one. Once the program has been listed to cassette, you simply replace the Atari BASIC Language cartridge with the Editor/Assembler, power up and type ENTER# C: (Note the #,NOT". [*For disk:* ENTER#D:Name]). Once the cassette stops, type *LIST*, and the program is LISTed to the screen, just as if you were still in BASIC.

Step two is to get rid of all the REM statements, since we are concerned with saving memory. Referring to Program 1, the first 11 REMs are easy to find. To get rid of these, we type the command: *DEL1,11*. Now do a LIST and you will find that the first 11 lines have vanished. This is much easier than typing 11 separate line numbers followed by carriage returns. Assuming this were a much longer program (with many REM statements embedded throughout), we would want a convenient way to list all the lines with the word REM. Simple. Type: *FIND/REM/,A*. The computer responds:

```
16 REM SOUND VALUES
23 REM SOUND SUBROUTINE
```

Now type: *DEL16*, followed by *DEL23*, or use the BASIC method – simply type the line number followed by a RETURN. The list will show the deletion of these two lines as well. The ",A" following the FIND command is a qualifier meaning "all occurrences of." Thus "FIND/REM/,A" means: find *all* lines which contain the letters REM and list them to the screen. It does not matter where in the statement the string is to be found. If the program you are editing has a line such as:

```
1010 PRINT "THIS IS A REMINDER":
     GOTO 3050
```

the FIND command will find the REM in "REMINDER" and list that line as well. Examine the results of the FIND listing before you delete any lines.

Using REPlace

Like the FIND command, REP has an "all occurrences" option. REP also has a "query" option which allows you to step through each line containing the string to be replaced. The computer will pause, then "ask" you whether or not you want the REP command to be executed on that line. This option is very convenient when you have to change, say, a variable name which may also appear in a PRINT statement.

To scan through the program for all of the occurrences of TIMER, and to optionally replace them with the shorter variable T, type: *REP/TIMER/T/1,1000,Q*. (The "1,1000" determines the range of line numbers for which the command is valid; choosing a very large number will insure scanning of the entire program.) The computer responds:

```
13 FOR TIMER=1 TO 100:NEXT TIMER
?■
```

To exercise the REP command for this line, type *Y* followed by a return. The computer responds:

```
13 FOR T=1 TO 100:NEXT TIMER
?■
```

The first occurrence of TIMER in line 13 has been changed to T. The computer is now querying you about the second occurrence. To change this one and all of the following occurrences, just type Y after every prompt. When the Editor has scanned through the entire program, it will display the EDIT prompt, meaning the function is complete. Use the same procedure to change WAIT to W and LOOP to L. Since the variables T and D are already used, we will change TONE to A and DURATION to X. When through, do another list to the screen just to make sure you didn't make any mistakes.

One of the reasons for using the query option is that the Editor signals the completion of a command with the EDIT prompt whether or not the function was actually performed. In the above example, had you tried to do a *REP/TIMER/T/,A*, and typed *TIMET* by mistake, the Editor would respond with its usual prompt, although, since there is no TIMET string in the program, nothing was REPlaced. If there *were* another variable named TIMET – you get the picture.

Having shortened all of our variable names, we are ready for step four: renumbering the lines to make room for our patches. This is where the hardcopy of the original program comes in handy. If you have one, fine. If not, you will have to struggle along with screen lists (like I do). The RENumber command will effortlessly renumber all statement numbers, but unfortunately will do nothing with all your GOTOs and GOSUBs. These will have to be handled separately.

First, identify where all of your GOTOs and GOSUBs are GOing to. Make a note of the target statements; i.e., GOSUB 17 points to a statement which starts "A=81", which follows a GOTO statement. Your "target notes" can be anything which uniquely places the location in the program of the targets; there can be no confusion. My notes looked like this:

17 A=81 (after GOTO)
24 T=2 (after RET)
32 FOR T=1 (after RET)
34 SO.0 (after L=1)

Having fixed the relative position of our target lines, we can execute the RENumber option. Type: *REN100,10*. The "100" tells the Editor what the new number of the first line of our program will be; the "10" specifies the increment. Now we can fix our GOTOs and GOSUBs. To do this, you can either do a screen list or use the FIND command. Referring to the target notes, do a *FIND/A=81/,A*, to determine old line number 17's new number.

Now simply REPlace all GOxx references to line 17 with GOxx 140. (You don't need to worry about the leading zeros in the Editor's listing. BASIC will ignore them.) Execute the following, responding with Y to all of the queries:

REP/GOSUB 17/GOSUB 140/1,1000,Q
(Don't forget the space after the GOxx!)
REP/GOSUB 24/GOSUB 200/1,1000,Q
REP/GOTO 32/GOTO 280/1,1000,Q
REP/THEN 34/THEN 300/1,1000,Q

Now do one last screen list, checking it carefully for errors. Your new program should look like Program 2, ready for your additions or patches. If there are no errors, type: LIST#C:, to produce a cassette file, which you can *ENTER"C:* using the BASIC cartridge. Once the modified prográm is ENTERED into BASIC, it can be RUN, CSAVEd and treated exactly as if it were written using the BASIC cartridge. If you have followed this procedure using Program 1 and have RUN both versions, note the great difference that shortening the variable names causes in the sound routines.

Program 1.

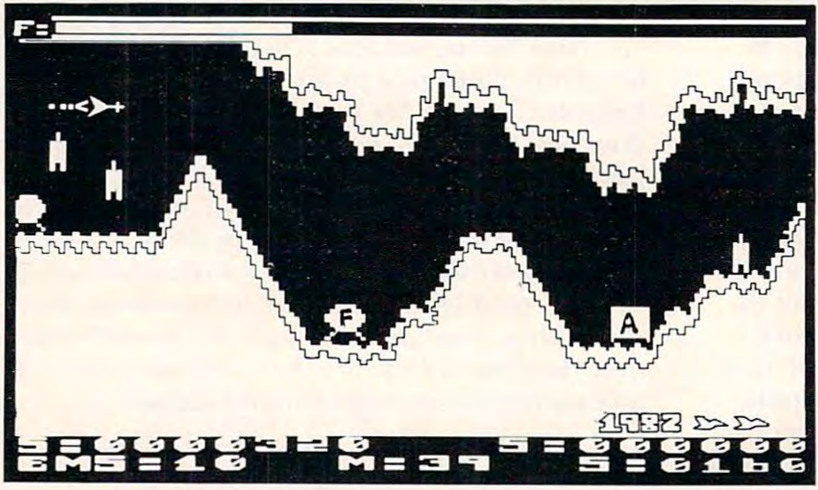
```
1 REM DEMONSTRATION PROGRAM
2 REM
3 REM by
4 REM
5 REM Dennis Allen
6 REM
7 REM
8 REM
9 REM
10 REM
11 REM 1981
12 GOSUB 17
13 FOR TIMER=1 TO 100:NEXT TIMER
```

```
14 GOSUB 17
15 GOTO 32
16 REM SOUND VALUES
17 TONE=81:D=20:GOSUB 24
18 TONE=64:D=20:GOSUB 24
19 TONE=50:D=20:GOSUB 24
20 TONE=43:D=95:GOSUB 24
21 TONE=50:D=20:GOSUB 24
22 RETURN
23 REM SOUND SUBROUTINE
24 TIMER=2
25 SOUND 0,TONE,10,15
26 SOUND 1,TONE,12,4
27 FOR DURATION=1 TO D:NEXT DURATION
28 SOUND 0,0,0,0
29 SOUND 1,0,0,0
30 FOR WAIT=1 TO 10:NEXT WAIT
31 RETURN
32 FOR TIMER=1 TO 50:NEXT TIMER
33 LOOP=1
34 SOUND 0,162,10,8
35 SOUND 1,162,2,4
36 FOR TIMER=1 TO 15:NEXT TIMER
37 SOUND 0,128,10,8
38 SOUND 1,128,2,8
39 FOR TIMER=1 TO 15:NEXT TIMER
40 LOOP=LOOP+1:IF LOOP<10 THEN 34
41 END
```

Program 2.

```
100 GOSUB 140
110 FOR T=1 TO 100:NEXT T
120 GOSUB 140
130 GOTO 280
140 A=81:D=20:GOSUB 200
150 A=64:D=20:GOSUB 200
160 A=50:D=20:GOSUB 200
170 A=43:D=95:GOSUB 200
180 A=50:D=20:GOSUB 200
190 RETURN
200 T=2
210 SOUND 0,A,10,15
220 SOUND 1,A,12,4
230 FOR X=1 TO D:NEXT X

240 SOUND 0,0,0,0
250 SOUND 1,0,0,0
260 FOR W=1 TO 10:NEXT W
270 RETURN
280 FOR T=1 TO 50:NEXT T
290 L=1
300 SOUND 0,162,10,8
310 SOUND 1,162,2,4
320 FOR T=1 TO 15:NEXT T
330 SOUND 0,128,10,8
340 SOUND 1,128,2,8
350 FOR T=1 TO 15:NEXT T
360 L=L+1:IF L<10 THEN 300
370 END
```

©



This program can be used to ring school bells, to scare crows, and to do many other timing tasks.

Perform A Task At Equally Spaced Intervals

Marvin L. De Jong
Department of Mathematics-Physics
The School of the Ozarks
Pt. Lookout, MO

A short time ago we required an observation of a certain experiment in our laboratory to be made approximately every 30 minutes over a period of 30 hours. Not wishing to stay up night and day for that period of time, we designed a timing program to do the observation with the aid of a movie camera. The movie camera had a remote switch, so all we had to do was close the switch for a period of three seconds every 30 minutes, over a time interval of 30 hours.

Perhaps you will encounter a similar timing problem sometime. This program should handle most such problems. It is designed to perform some task at equally spaced programmable intervals ranging from 0.01s to 99 days, 23 hours, 59 minutes, and 59.99 seconds. The task is performed for the first time at 0.01s after the program is initiated. It is then performed at equally spaced intervals until execution is terminated. The desired time interval between tasks is determined by the contents of locations $0000 (tenths and hundredths of seconds), $0001 (tens and units of seconds), $0002 (minutes), $0003 (hours), and $0004 (days). The appropriate *decimal* quantities must be entered into these locations.

Whatever task the computer is to perform is accomplished by subroutine TASK, which we located at $OF16. We include our subroutine in the program to illustrate the use of the T2 timer on the 6522 VIA. The task must take less time than the interval between tasks, by about 200 microseconds.

Any microcomputer system with a 6522 Versatile Interface Adapter can be used to execute the program. The program itself is easily relocated. The comments should explain most of the details regarding its operation, which is very similar to a clock routine. It is important to note that the T1 timer is operating in its free-running mode with interrupts. In order for the program to work, the IRQ interrupt vector *must* be loaded to point to the interrupt routine at $0F4D in Program 1.

Although our task consisted of simply switching a device on for a few seconds, the task might be more complex, such as performing an analog-to-digital conversion, measuring the activity of a radioactive source, making a temperature or voltage measurement with T/F or V/F converter, starting the coffee pot in the morning, firing a gun to scare the crows from the corn, or running the school buzzers or bells to let the kids out of class. Let your imagination go wild.

Program 1: A program to perform a task at equally-spaced programmable intervals.

$0000-$0004 = TIMTBL; five locations whose contents determine the time interval in hundredths of seconds, seconds, minutes, hours, and days.

$0005-$0009 = CNTTBL; five locations used to count until [CNTTBL]=[TIMTBL].

$000A = FLAG; a location that is decremented to zero when [CNTTBL]=[TIMTBL].

IRQ VECTOR = $0F4D

```
$ØE98 A9 Ø1       START    LDA #$Ø1       Initialize Port B for the task.
 ØE9A 8D ØØ AØ             STA PBD        Port B, pin PBØ starts at logic one.
 ØE9D 8D Ø2 AØ             STA PBDD       Data Direction Register set to
 ØEAØ A9 40                LDA #$40       make PBØ an output pin.
 ØEA2 8D ØB AØ             STA ACR        Put T1 in free-running mode.
 ØEA5 A9 CØ                LDA #$CØ       Enable interrupts by setting
 ØEA7 8D ØE AØ             STA IER        IER, bit six.
 ØEAA A9 ØE                LDA #$ØE       Set up T1 timer to run free with
 ØEAC 8D Ø4 AØ             STA T1LL       a period of ($27ØE + 2) $T_C$ = Ø.Ø1s.
 ØEAF A9 27                LDA #$27
 ØEB1 8D Ø5 AØ             STA T1LH       Start timing.
 ØEB4 58                   CLI            Clear interrupt flag to allow interrupts.
 ØEB5 A9 Ø1                LDA #$Ø1       Set FLAG to be non-zero.
 ØEB7 85 ØA                STA FLAG
 ØEB9 F8                   SED            All arithmetic functions done in
 ØEBA 38                   SEC            decimal mode.
 ØEBB A2 FB                LDX #$FB       Start by setting CNTTBL = TIMTBL.
 ØEBD B5 Ø5       BACK     LDA TIMTBL,X
 ØEBF 95 ØA                STA CNTTBL,X
 ØEC1 E8                   INX
 ØEC2 DØ F9                BNE BACK
 ØEC4 A5 Ø5                LDA CHSEC      Now make CNTTBL = TIMTBL - Ø.Ø1.
 ØEC6 FØ Ø6                BEQ ONE        Then task will be performed for
 ØEC8 E9 Ø1                SBC #$Ø1       the first time after the first
 ØECA 85 Ø5                STA CHSEC      time out of the T1 timer
 ØECC BØ 34                BCS ONWRD
 ØECE A9 99       ONE      LDA #99
 ØEDØ 85 Ø5                STA CHSEC
 ØED2 A5 Ø6                LDA CSEC       Get seconds from CNTTBL.
 ØED4 FØ Ø6                BEQ TWO        If it is not zero, subtract one.
 ØED6 E9 Ø1                SBC #$Ø1
 ØED8 85 Ø6                STA CSEC       Result into seconds place.
 ØEDA BØ 26                BCS ONWARD     No borrow, so get out.
 ØEDC A9 59       TWO      LDA #59        If it is zero, borrow one from
 ØEDE 85 Ø6                STA CSEC       the seconds plce.
 ØEEØ A5 Ø7                LDA CMIN       Get minutes.
 ØEE2 FØ Ø6                BEQ THREE      Is it zero?
 ØEE4 E9 Ø1                SBC #$Ø1       No. Subtract one and get out.
 ØEE6 85 Ø7                STA CMIN
 ØEE8 BØ 18                BCS ONWRD
 ØEEA A9 59       THREE    LDA #59        Yes. There was a borrow from
 ØEEC 85 Ø7                STA CMIN       the minutes place.
 ØEEE A5 Ø8                LDA CHRS       Get hours.
 ØEFØ FØ Ø6                BEQ FOUR       Is it zero, then borrow from days.
 ØEF2 E9 Ø1                SBC #$Ø1       Otherwise, subtract one.
 ØEF4 85 Ø8                STA CHRS
 ØEF6 BØ ØA                BCS ONWARD
 ØEF8 A9 23       FOUR     LDA #23
 ØEFA 85 Ø8                STA CHRS
 ØEFC A5 Ø9                LDA CDAYS
 ØEFE E9 Ø1                SBC #$Ø1
 ØFØØ 85 Ø9                STA CDAYS      Finished doing [CNTTBL]=[TIMTBL]-0.01s.
 ØFØ2 A5 ØA       ONWRD    LDA FLAG       Wait in this loop until FLAG is zero.
 ØFØ4 DØ FC                BNE ONWRD
 ØFØ6 A2 FB                LDX #$FB       Clear CNTTBL, then do TASK.
 ØFØ8 A9 ØØ                LDA #$ØØ
 ØFØA 95 ØA       UPWRD    STA CNTTBL,X
 ØFØC E8                   INX
 ØFØD DØ FB                BNE UPWRD
 ØFØF 2Ø 16 ØF             JSR TASK       Perform TASK.
 ØF12 C6 ØA                DEC FLAG       Set FLAG to non-zero number.
 ØF14 DØ EC                BNE ONWRD      Wait for the next interval to elapse.
 ØF16 A9 3C       TASK     LDA #$3C       Count $3C = 6Ø intervals of .Ø5s
 ØF18 85 3Ø                STA COUNT      each.
```

```
ØF1A CE ØØ AØ           DEC PBD         PBØ to logic zero to start device.
ØF1D A9 4F    HERE      LDA #$4F        Load T2 for Ø.Ø5s.
ØF1F 8D Ø8 AØ           STA T2LL
ØF22 A9 C3              LDA #$C3        $C34F + 1 = 5ØØØØ
ØF24 8D Ø9 AØ           STA T2CH
ØF27 AD ØD AØ WAIT      LDA IFR         Read interrupt flag register
ØF2A 29 2Ø              AND #$2Ø        to see if T2 has timed out.
ØF2C FØ F9              BEQ WAIT
ØF2E C6 3Ø              DEC COUNT
ØF3Ø DØ EB              BNE HERE
ØF32 EE ØØ AØ           INC PBD         PBØ to logic one to turn device
ØF35 6Ø                 RTS             off.
  .                      .
  .                      .
  .                      .
ØF4D 48       INTERRUPT PHA             Save accumulator.
ØF4E 8A                 TXA             Save X.
ØF4F 48                 PHA
ØF5Ø AD Ø4 AØ           LDA T1CL        Clear T1 interrupt flag, IFR6.
ØF53 A9 Ø5              LDA #$Ø5        Set FLAG to $Ø5.
ØF55 85 ØA              STA FLAG
ØF57 F8                 SED             Set decimal mode for subsequent
ØF58 38                 SEC             BCD arithmetic. Set carry to
ØF59 A2 FE              LDX #$FE        add one to the least significant
ØF5B B5 Ø7    ADD1      LDA CNTTBL,X    location of the CNTTBL.
ØF5D 69 ØØ              ADC #$ØØ        Add carry.
ØF5F 95 Ø7              STA CNTTBL,X
ØF61 E8                 INX             The two lowest locations of CNTTBL
ØF62 DØ F7              BNE ADD1        have now been incremented by Ø.Ø1s.
ØF64 C9 6Ø              CMP #$6Ø        Have seconds reached 6Ø yet?
ØF66 DØ 26              BNE OUT         No. Branch to see if CNTTBL=TIMTBL.
ØF68 A9 ØØ              LDA #$ØØ        Yes. Clear seconds counter and
ØF6A 85 Ø6              STA SEC         increment minutes counter.
ØF6C A5 Ø7              LDA MIN         Carry is set from CMP instruction,
ØF6E 69 ØØ              ADC #$ØØ        so minutes are incremented with ADC.
ØF7Ø 85 Ø7              STA MIN
ØF72 C9 6Ø              CMP #$6Ø        Have minutes reached 6Ø yet?
ØF74 DØ 18              BNE OUT         No. Branch to see if CNTTBL=TIMTBL.
ØF76 A9 ØØ              LDA #$ØØ        Yes. Clear minutes counter and
ØF78 85 Ø7              STA MIN         increment hours counter.
ØF7A A5 Ø8              LDA HRS
ØF7C 69 ØØ              ADC #$ØØ
ØF7E 85 Ø8              STA HRS
ØF8Ø C9 24              CMP #$24        Have hours reached 24 yet?
ØF82 DØ ØA              BNE OUT         No. Branch to see if CNTTBL=TIMTBL.
ØF84 A9 ØØ              LDA #$ØØ        Yes. Clear hours counter.
ØF86 85 Ø8              STA HRS
ØF88 A5 Ø9              LDA DAYS        Increment days counter using
ØF8A 69 ØØ              ADC #$ØØ        set carry flag.
ØF8C 85 Ø9              STA DAYS
ØF8E A2 FB    OUT       LDX #$FB        Subtract the five bytes of CNTTBL
ØF9Ø 38                 SEC             from the five bytes of TIMTBL.
ØF91 B5 Ø5    HERE      LDA TIMTBL,X    Get a byte from TIMTBL.
ØF93 F5 ØA              SBC CNTTBL,X    Subtract a byte from CNTTBL
ØF95 DØ Ø2              BNE THERE       If result is not zero, leave FLAG alone.
ØF97 C6 ØA              DEC FLAG        Otherwise decrement FLAG.
ØF99 E8       THERE     INX             Repeat for the remaining bytes of
ØF9A DØ F5              BNE HERE        TIMTBL and CNTTBL. If all five bytes
ØF9C D8                 CLD             are equal, FLAG contains zero to
ØF9D 68                 PLA             flag main program to do task.
ØF9E AA                 TAX             Restore registers.
ØF9F 68                 PLA
ØFAØ 4Ø                 RTI             Return.
```

©

These screens enter the FORTH machine and bring back two lists: what words are and are not being used in a particular program. These lists can be used to streamline an application by removing (metacompiling) all unreferenced words. Or you could analyze a program to discover areas where large numbers of words were being called and then speed things up by replacing those sections with machine language.

The FORTH Page

REFS

Richard Mansfield and Lou Cargile

The question seems simple at first. What words are *not* used in a FORTH program? Say you've written a simple game and you decide to find out what words in the dictionary are not necessary to run this game. Since these words are never referenced in this game, they could be left out of this FORTH application entirely. (See "Headless Metacompilation," **COMPUTE!**, July 1982, p. 174.) Unfortunately, you can't just look at the screen which holds the game and then make a list of all the words on it.

The words on the game screen are the highest level definitions. Within each of these definitions is a group of other words, and within those words are yet others below. Imagine that you defined : SCOREFORMAT ." SCORE IS –" ; At first glance, it might seem that you could eliminate most FORTH words since this definition uses only the word ." and that's all. But what words are necessary to support ." itself?

FORTH has hidden levels, words within words, radiating below what you can see. The simple ." uses many other words. It uses EXPECT, for one. What's more, EXPECT itself uses several words. Within EXPECT is +ORIGIN. And +ORIGIN uses + which is finally an end point. We can come to a full stop with +, a code definition, complete within itself, referencing nothing. Higher level words, however, all depend upon a hidden support network. What's visible reveals only a fraction of the words actually being used. Beneath, like a desert plant, there is a root system many times the size of what appears on the surface.

List Management

The function of the REFS screens is to analyze a particular FORTH program and report back to you with a list of words representing that hidden root system. Any word which is referenced within an application is added to the list. When you load the REFS screens, a buffer area is created (REF.LIST) with room for 512 CFAs. This buffer is cleared out whenever you execute XREF. To accumulate a list of references you execute +REFS. Accumulation continues during whatever FORTH operations take place up to and including the execution of -REFS.

Accumulation occurs in the order in which each word was first referenced. The list will not duplicate references: a word goes onto the list only the first time it is called.

After the accumulation is stopped with -REFS, you can see the list of words by executing the word REFS. Perhaps equally useful is the list of words which were not needed and never made the REFS list. NONREFS will list out the words which are unnecessary for your application, beginning with the top of the context vocabulary. These words would be candidates for omission in a compacted, headless, dedicated FORTH program.

They would only be candidates, though. The words VARIABLE, CONSTANT, :, etc., will *always* appear in the NONREFS list, but they must be included so that the address of metacompiler utilities DOCOLON (or DOVARIABLE or DOCONSTANT, etc.) will be that of the compacted FORTH, not the host FORTH. And the words intended to be executed in the application will in general be on this list. This is because the execution of a word from the input stream does not of itself put the word on the REFS list. Only when a word is called by another executing word does it become a part of the REFS list. For example, compiling

```
: TEST2 TEST1 ;
```

would not put either TEST2 or TEST1 on the REFS list. Executing TEST2, however, would put TEST1 on the REFS list.

Other Uses

The listing of NONREFS is somewhat slowed because each word printed has to first be compared with the entire REFS list. If the list were maintained in sorted order, this listing could be made faster. In any case, it is worthwhile having the REFS listed out in the order in which the words were first interpreted.

The approach used in these screens intercepts the action of the inner interpreter. It is possible to create a debugging "trace" routine in a similar way and watch FORTH at work during execution.

Program 1. REFS Screens

```
SCR # 244
 Ø  ( REFS --1 )  FORTH DEFINITIONS HEX
 1 Ø VARIABLE REF.LIST 3FE ALLOT ( SPACE FOR 512 WORD CFA'S)
 2  ' REF.LIST 4ØØ FF FILL ( INITIALIZE REF.LIST ARRAY )
 3 CODE STORE.REF  ( CODE TO BE PATCHED TO INNER INTERPRETER )
 4   T REF.LIST 2 - 1ØØ /MOD
 5   # LDA, N 1+ STA, # LDA, N STA,
 6   Ø # LDY,
 7   BEGIN, INY, INY, Ø=  IF, N 1+ INC, THEN,
 8     N )Y  LDA, W CMP,
 9       Ø= IF, INY, N )Y LDA, W 1+ CMP,
1Ø         Ø= IF, FF # LDA, THEN, DEY,
11       THEN,  FF # CMP,
12   Ø= END, N )Y LDA, FF # CMP,
13   Ø= IF, W LDA, N )Y STA, W 1+ LDA, INY, N )Y STA, THEN,
14   Ø # LDY,
15   CLC, IP LDA, ( OVERWRITTEN INTERPRETER CODE ) RTS, -->

SCR # 245
 Ø  ( REFS --2 )
 1  : REFS 8Ø OUT ! ( PRINT A LIST OF REFERENCED WORDS )
 2     CR REF.LIST BEGIN OUT @ C/L > IF CR Ø OUT ! ENDIF
 3              DUP @ FFFF = Ø=
 4               WHILE DUP @ 2 + NFA ID. SPACE 2 + REPEAT
 5  DROP CR ;
 6  : NONREFS  ( PRINT A LIST OF NON-REFERENCED WORDS )
 7  8Ø OUT ! CONTEXT @ @ >R
 8  BEGIN  OUT @ C/L > IF CR Ø OUT ! ENDIF
 9     REF.LIST BEGIN DUP @ FFFF = OVER @ R PFA CFA = OR Ø=
1Ø                    WHILE 2 + REPEAT
11     @ FFFF =
12       IF R ID. SPACE SPACE ENDIF
13     R> PFA LFA @ DUP >R Ø= ?TERMINAL OR
14  UNTIL R> DROP ;  -->
15

SCR # 246
 Ø  ( REFS --3 )
 1 CODE +REFS  ( PATCH STORE.REF CODE TO INNER INTERPRETER )
 2                    2Ø # LDA, NEXT ØB + STA,
 3   'T STORE.REF 1ØØ /MOD # LDA, NEXT ØD + STA,
 4                       # LDA, NEXT ØC + STA,
 5    NEXT JMP,
 6 CODE -REFS  ( RESTORE INNER INTERPRETER )
 7       18 # LDA, NEXT ØB + STA,
 8       A5 # LDA, NEXT ØC + STA,
 9       Ø6 # LDA, NEXT ØD + STA,
1Ø       NEXT JMP,
11  : XREFS    ( EMPTY REFERENCE BUFFER )
12    ASSEMBLER NEXT ØB + C@ 18 =
13      IF REF.LIST 4ØØ FF FILL
14      ELSE ."  CANT--IN +REFS MODE" ENDIF ;
15  FORTH DECIMAL  ;S
```

REFS could also be used to help pinpoint areas which are retarding a FORTH application. You could insert the sequence XREFS +REFS at some key point in a colon definition (perhaps just before a loop that appears to be operating slowly) and further in the definition place the word -REFS (perhaps at the completion of the loop). The REFS list would then contain the candidates which could be coded in machine language to speed up the loop.

Try executing the following:

XREFS +REFS -REFS REFS

and see how little it takes for FORTH just to keep body and soul together.

The FORTH Interest Group 6502 Fig-FORTH inner interpreter (its address is given by the Assembler constant NEXT) is shown disassembled in Program 2. In this particular implementation of FORTH, NEXT is $0642 and the code that is overwritten by a JSR is at $064D-$064F. The three bytes overwritten form the last three bytes of the patch jumped to (line 15, Screen 244). To use REFS on other FORTH implementations, changes would be made here and in lines two through four and lines seven through nine of Screen 246.

REFS Functions

```
+REFS    begin adding to accumu-
                lated list of
                referenced words.
-REFS    stop adding to the list.
XREFS    clear out the list.
REFS     print the list.
NONREFS  print all other words in
                the dictionary.
```

Program 2. 6502 Inner Interpreter

```
Ø642 AØ Ø1     LDY #$Ø1
Ø644 B1 Ø6     LDA (IP),Y
Ø646 85 Ø2     STA W+1
Ø648 88        DEY
Ø649 B1 Ø6     LDA (IP),Y
Ø64B 85 Ø1     STA W
Ø64D 18        CLC
Ø64E A5 Ø6     LDA IP
Ø65Ø 69 Ø2     ADC #$Ø2
Ø652 85 Ø6     STA IP
Ø654 9Ø Ø2     BCC $Ø658
Ø656 E6 Ø7     INC IP+1
Ø658 4C ØØ ØØ  JMP $ØØØØ
```

©

CAPUTE!:

Modifications Or Corrections To Previous Articles

PET Compactor

The following lines were inadvertently left out of "Machine Language Compactor," July 1982, p. 159.

```
Ø5Ø8 52 ØC A9 5Ø 9D 51 ØC A9
Ø6Ø8 88 ØB C9 16 BØ 48 AD 87
Ø7Ø8 2Ø 63 F5 2Ø F7 Ø9 2Ø CE
Ø8Ø8 2Ø FØ DB C9 8F DØ ØD 2Ø
Ø9Ø8 FF 2Ø CC FF EE 7F ØB 4C
ØAØ8 E8 4C FE Ø9 CA 8E 3Ø ØA
ØBØ8 ØØ ØD ØD 4F 55 54 5Ø 55
```

The author sent in the following modification to Compactor which allows a BASIC program to be reduced even further in size by altering the way that ON GOTO is compacted. To add this patch, change the following two lines:

```
Ø7E9  4C 63 Ø3
Ø89C  4C 8A Ø3
```

and add these lines:

```
Ø363 AD 79 ØB C9 91 DØ Ø5 A9
Ø36B Ø1 8D 9Ø ØB AD 79 ØB C9
Ø373 89 FØ Ø3 4C FØ Ø7 AD 9Ø
Ø37B ØB DØ Ø6 AD 79 ØB 4C EC
Ø383 Ø7 AD 79 ØB 4C FØ Ø7 A9
Ø38B ØØ 8D 9Ø ØB AD 82 ØB 4C
Ø393 9F Ø8 ØØ ØØ ØØ ØØ ØØ ØØ
```

Gold Miner Game, Atari Version

(also see note in box below)

Two RESTORE commands should be changed in the Atari version of this game (see p. 28, **COMPUTE!**, July 1982). Change line 840 to RESTORE 850 and the RESTORE in line 1130 should read RESTORE 1190.

Our July issue featured an action game and accompanying article entitled "Gold Rush!" This game should not be confused with an arcade-graphics game for both the Apple and Atari of the same name produced and marketed by Sentient Software of Aspen, Colorado. No comparison or confusion was intended regarding the products of Sentient Software, and readers should be aware that these two games are entirely different products. In any future use of this article and action game, we will refer to it as "Gold Miner." ©

COMPUTE!'s Listing Conventions

Many of the programs which are listed in **COMPUTE!** use special keys (cursor control keys, color keys, etc.). To make it easy to tell *exactly* what should be typed in when copying a program into the computer, we have established the following listing conventions.

For The Atari

In order to make special characters, inverse video, and cursor characters easy to type in, **COMPUTE!** magazine's Atari listing conventions are used in all the program listings in this magazine.

Please refer to the following tables and explanations if you come across an unusual symbol in a program listing.

Atari Conventions

Characters in inverse video will appear like: INVERSE VIDEO
Enter these characters with the Atari logo key, {人}.

When you see	Type	See	
{CLEAR}	ESC SHIFT <	↖	Clear Screen
{UP}	ESC CTRL -	↑	Cursor Up
{DOWN}	ESC CTRL =	↓	Cursor Down
{LEFT}	ESC CTRL +	←	Cursor Left
{RIGHT}	ESC CTRL *	→	Cursor Right
{BACK S}	ESC DELETE	◀	Backspace
{DELETE}	ESC CTRL DELETE	◧	Delete character
{INSERT}	ESC CTRL INSERT	◨	Insert character
{DEL LINE}	ESC SHIFT DELETE	▣	Delete line
{INS LINE}	ESC SHIFT INSERT	▣	Insert line
{TAB}	ESC TAB	▶	TAB key
{CLR TAB}	ESC CTRL TAB	▣	Clear tab
{SET TAB}	ESC SHIFT TAB	▣	Set tab stop
{BELL}	ESC CTRL 2	▣	Ring buzzer
{ESC}	ESC ESC	⇤	ESCape key

Graphics characters, such as CTRL-T, the ball character ● will appear as the "normal" letter enclosed in braces, e.g. {T}.

A series of identical control characters, such as 10 spaces, three cursor-lefts, or 20 CTRL-R's, will appear as {10 SPACES}, {3 LEFT}, {20 R}, etc. If the character in braces is in inverse video, that character or characters should be entered with the Atari logo key. For example, {♥} means to enter a reverse-field heart with CTRL-comma, {5 ▣} means to enter five inverse-video CTRL-U's.

For PET/CBM/VIC

Generally, any PET/CBM/VIC program listings will contain bracketed words which spell out any special characters: {DOWN} would mean to press the cursor-down key; {3DOWN} would mean to press the cursor-down key three times.

To indicate that a key should be *shifted* (hold down the SHIFT key while pressing the other key), the key would be underlined in our listing. For example, S would mean to type the S key while holding the shift key. This would result in the "heart" graphics symbol appearing on your screen.

Sometimes in a program listing, especially within quoted text when a line runs over into the next line, it is difficult to tell where the first line ends. How many times should you type the SPACE bar? In our convention, when a line breaks in this way, the ~ symbol shows exactly where it broke. For example:

```
   100 PRINT "TO START THE GAME  ~
       YOU MAY HIT ANY OF THE KEYS
       ON YOUR KEYBOARD."
```

shows that the program's author intended for you to type two spaces after the word *GAME*.

For The Apple

Programs listed as "Microsoft" are written for the PET/CBM, Apple, OSI, etc. Although the programs are general in nature, you may need to make a few changes for them to run correctly on your Apple. Microsoft BASIC programs written for the PET/CBM sometimes contain special cursor control characters. The following table shows equivalent Apple words. Notice that these Apple commands are *outside* quotations (and even separate from a PRINT statement). PRINT"[RVS]YOU WON" becomes INVERSE: PRINT"YOU WON":NORMAL

[CLEAR[(Clear Screen) HOME

[DOWN] (Cursor down)
Apple II +: Call -922
POKE 37,PEEK(37)+(PEEK(37)<23)

[UP] (Cursor up)
POKE 37,PEEK(37)-(PEEK(37)>0))

[LEFT] (Cursor left) PRINT CHR$(8);

[RIGHT] (Cursor right)
PRINT CHR$(21)

[RVS] (Inverse video on. Turns off automatically after a carriage return. To be safe, turn off inverse video after the print statement with NORMAL unless the PRINT statement ends with a semicolon.)
INVERSE

[OFF] (Inverse video off) NORMAL

Shifted characters can represent either graphics characters or uppercase letters. If within text, just use the non-shifted character, otherwise substitute a space. Some "generalized" programs contain a POKE such as POKE 59468,14. Omit these from the program when typing it in. One final note: you will probably want to insert a question mark or colon within an INPUT prompt. PET/CBM and many other BASICs automatically print a question mark:

INPUT "WHAT IS YOUR NAME";N$
becomes
INPUT "WHAT IS YOUR NAME?";N$

All Commodore Machines

Clear Screen	{CLEAR}	**Cursor Left**	{LEFT}
Home Cursor	{HOME}	**Insert Character**	{INST}
Cursor Up	{UP}	**Delete Character**	{DEL}
Cursor Down	{DOWN}	**Reverse Field On**	{RVS}
Cursor Right	{RIGHT}	**Reverse Field Off**	{OFF}

VIC Conventions

Set Color To Black	{BLK}	**Function Two**	{F2}
Set Color To White	{WHT}	**Function Three**	{F3}
Set Color To Red	{RED}	**Function Four**	{F4}
Set Color To Cyan	{CYN}	**Function Five**	{F5}
Set Color To Purple	{PUR}	**Function Six**	{F6}
Set Color To Green	{GRN}	**Function Seven**	{F7}
Set Color To Blue	{BLU}	**Function Eight**	{F8}
Set Color To Yellow	{YEL}	**Any Non-implemented Function**	{NIM}
Function One	{F1}		

8032/Fat 40 Conventions

Set Window Top	{SET TOP}	**Erase To Beginning**	{ERASE BEG}
Set Window Bottom	{SET BOT}	**Erase To End**	{ERASE END}
Scroll Up	{SCR UP}	**Toggle Tab**	{TGL TAB}
Scroll Down	{SCR DOWN}	**Tab**	{TAB}
Insert Line	{INST LINE}	**Escape Key**	{ESC}
Delete Line	{DEL LINE}		

©

Turn To The Future With COMPUTE! Publications

The Beginner's Guide To Buying A Personal Computer

A Novice's handbook of useful, helpful information designed to teach you the basics of evaluating and selecting a personal computer. Written in plain English for the interested beginner. Complete with personal computer specification charts and buyer's guide. Applicable to home, educational, and small business buyers. ISBN 0-942386-03-5. Paperback. $3.95.

COMPUTE!'s First Book Of Atari

192 pages of useful, informative applications and programs from **COMPUTE!** Magazine issues now out of print. Includes previously unpublished information including Memory Map. Contents include such articles and programs as "Adding A Voice Track to Atari Programs," "Designing Your Own Atari Graphics Modes," and "Inside Atari BASIC." Spiral bound for ease of access to listings. For beginner level to advanced Atari users. ISBN 0-942386-00-0. Paperback. $12.95.

Inside Atari DOS

From the authors of the Atari Disk Operating System, an exciting step-by-step guide to the DOS software. Complete with listings of commented source code and detailed explanations of each module of code. Author: Bill Wilkinson, Optimized Systems Software, Inc. Spiral bound for ease of access to listings. For intermediate to advanced Atari users. ISBN 0-942386-02-7. Paperback. $19.95.

COMPUTE!'s Second Book Of Atari

With over 200 pages of unpublished Atari information specially selected by the editors of **COMPUTE!** Magazine, this new release is an excellent resource for Atari owners and users. Spiral bound. ISBN 0-942386-06-X. Paperback. $12.95.

COMPUTE! Books invites dealer inquiries. Call the Toll Free Number below for Dealer Information.

ATARI is a registered trademark of Atari, Inc.
PET and CBM are trademarks of Commodore Business Machines, Inc.

To Order COMPUTE! Books

Send coupon to
COMPUTE! Books. P.O. Box 5406. Greensboro, NC 27403 USA

For fastest service, in the US call
Toll Free 800-334-0868
In NC call 919-275-9809.

All orders must be prepaid (money order, check or charge). All payments must be in US funds. (Outside the US add $4.00 shipping and handling for air mail, $2.00 for surface mail.) NC residents add 4% sales tax.

□ Payment Enclosed
Please charge my □ VISA □ MasterCard □ American Express

Account No. ________________ Expires ____/____

Please send me:

Quan.		**Price**		**Shipping/Handling**	
____	**Beginner's Guide**	$ 3.95 ea.	+	$1.00 ea.	________
____	**First Book of Atari**	12.95 ea.	+	2.00 ea.	________
____	**Inside Atari DOS**	19.95 ea.	+	2.00 ea.	________
____	**Second Book of Atari**	12.95 ea.	+	2.00 ea.	________
				Total	________

Name ________________

Address ________________

City ________ State ________ Zip ________

Country ________________

Allow 4-6 weeks for delivery. Foreign surface delivery allow 2-3 months. Dept. A