COMMODORE WORLD

Issue 10
Volume 2, Number 5
$4.95 U.S.

THE NEWS MAGAZINE FOR COMMODORE 64 & 128 USERS

Machine Language for Beginners

Learn to Program in ML

Reviews

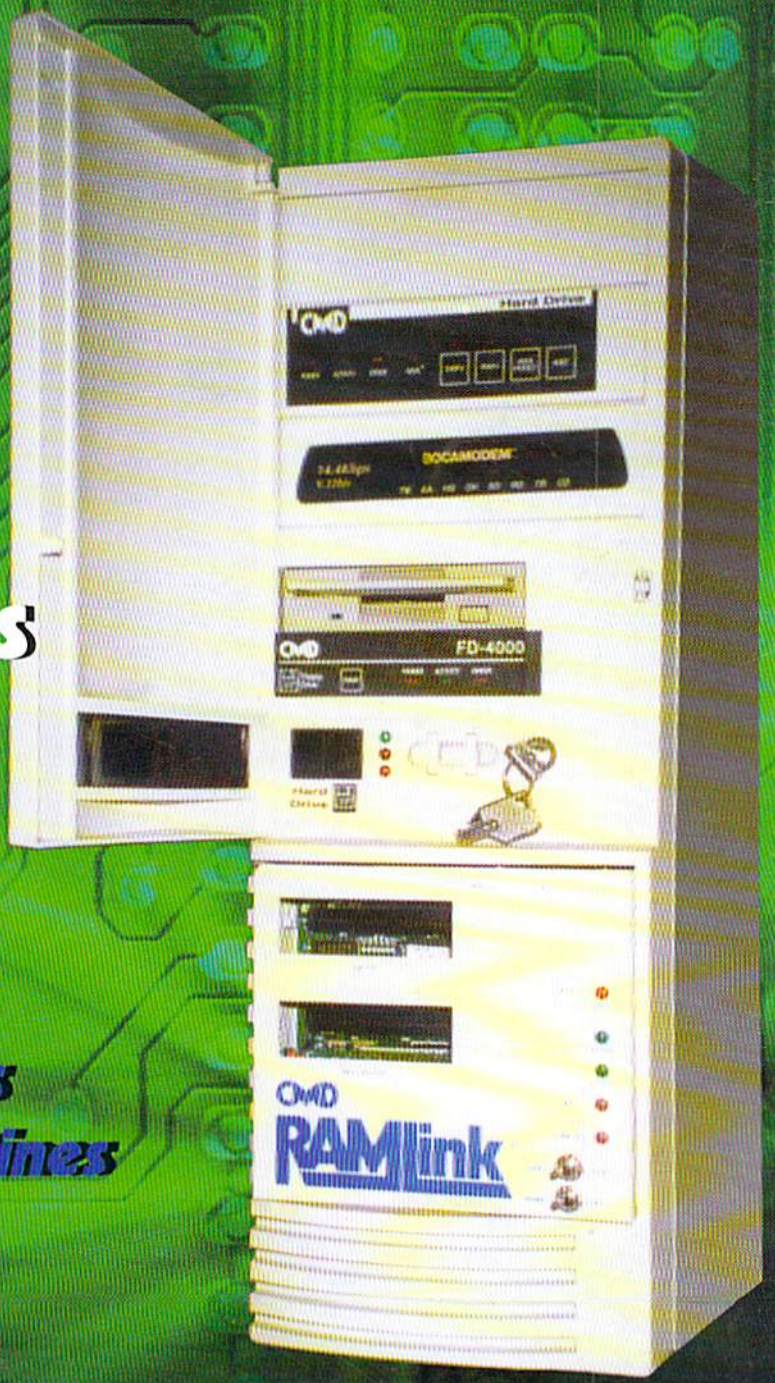
GeoFax
HandyScanner
Buddy 64/128
Assembler

More New Games

Reviews of the latest releases

Plus...

Customized Commodores
Serial Device Kernal Routines
Commodore Trivia



SECOND CLASS

SOFTWARE SUPPORT INTERNATIONAL

We Carry America's Largest Selection of C-64/C-128 Software!

ENTERTAINMENT

10 Game Grab Bag	$9.95
Arkanoid 2	$7.97
Beyond Dark Castle	$7.97
Beyond Zork 128	$12.97
Defender of the Crown	$9.97
Double Dragon 2	$9.97
Heavy Metal	$12.97
Heros of the Lance	$14.97
Keys to Maramon	$9.97
Mean Streets	$9.97
Monopoly	$12.97
Ms Pacman	$9.97
Ogre	$9.97
Pacman	$9.97
Plundered Hearts	$12.97
Pool of Radiance	14.97
Portal	$14.97
Questron 2	$9.97
Rampage	$7.97
Realms of Darkness	$12.97
Roadwars	$7.97
Sidearms	$7.97
Steel Thunder	$9.97
Strip Poker	$14.97
Trump Castle Casino	$12.97
Wheel Fortune 1-2-3	$9.97

SPORTS & FLIGHT

4th & Inches	$7.97
Champshp Baseball	$7.97
Champn Basketball	$7.97
Dave Winfield Batter Up	$9.97
Fast Break	$7.97
Hardball	$7.97
Jack Nicklaus Golf	$9.97
Jordan vs Bird	$7.97
Leaderboard Golf	$9.97
Pro Football Facts	$9.97
Pro Soccer	$7.97
Pure Stat Baseball	$9.97
Sporting News Baseball	$9.97
Star Rank Boxing	$7.97
Tony LaRussa Baseball	$7.97
Tournament Tennis	$7.97
WWF Wrestling	$7.97
Acrojet	$9.97
Apache Strike	$7.97
Blue Angels	$7.97
F-14 Tomcat	$9.97
F-19 Stealth Fighter	$12.97
Flight Sim Games	$4.97
High Roller	$7.97
Jet Combat Sim	$7.97
Skyfox	$7.97
Super Huey I	$7.97
Top Gunner	$9.97

ACCESSORIES

C-128 RGB Cable	$17.95
Computer Hand 2	$6.95
Convert A Com	$24.95
Disk Bank 10/3.5"	$2.95
Disk Bank 10/5.25"	$2.95
Disk Bank 100/3.5"	$12.95
Disk Bank 100/5.25"	$12.95
Disk Bank 70/5.25"	$7.95
Disk Mailers	$0.39
Drive Cleaners	$6.95
Serial Cable 6ft	$8.95
Serial Cable 10 ft	$9.95
Drive Power Cable	$7.95
User Port Cable	$15.95
Com Modem Adapter	$15.95
Floppy Disk Notcher	$6.95
Dust Covers - specify	$7.95
Ergostick Joystick	$16.95
CBM 1200 Baud	$9.95
Aprotek 2400 Baud	$49.95
1351 Smart Mouse	$44.95
MW 350 Interface	$44.95
64 Power Supplies	$34.95
128 Power Supplies	$49.95
Printer Ribbons	CALL
Userport Expander	$24.95
Video Ram Upgrade	$49.95
3.5" 10 cnt.Floppy	$7.95
5.25" 20 cnt.Floppy	$4.95

PRODUCTIVITY

1750 Super Clone	$99.95
Basic Compiler 64	$12.97
Big Blue Reader	$29.97
B/W Prog Tools	$14.97
B/W Power C	$9.97
B/W Turbo Cartridge	$17.97
C128 Graphics Bundle	$29.97
Christmas Model Kit	$9.97
CSM Protection Man 1	$14.97
CSM Protection Man 2	$19.97
Data Manager 2	$14.97
Designer's Pencil	$9.97
Drive Align 1541/71	$12.97
Easy Working Tri Pak	$9.97
Geos 64 v2	$39.97
Geos 128 v2	$44.97
Graphic Label Wizard	$14.97
Graphics Basic	$9.97
Home Designer 128	$24.97
Manager, The	$12.97
Maverick v5	$24.95
Model Diet	$9.97
Newsroom	$14.97
On Line Help	$9.97
Outrageous Pages	$19.97
Postcards	$14.97
Printmaster Plus	$19.97
Superbase 64	$19.97
Superscript 64/128	$14.97
Swiftcalc w/Sideways	$14.97

EDUCATIONAL

Early Learning Friends	$9.97
European Nations & Loc	$9.97
Keys to Typing	$9.97
Memory Manor Cart	$9.97
Magic Spells	$9.97
Snoopy Sky Scramble	$9.97
Spellicopter	$9.97
Stickybear Math	$9.97
Stickybear Numbers	$9.97
Typing Tutor 4	$9.97
Word Attack	$9.97
Word Spinner	$9.97

REFURBISHED HARDWARE

C-64 Keyboard w/ PS	$79.95
C-128 Keyboard w/PS	$139.95
C-128D Computer	$174.95
1541 Disk Drive	$64.95
1571 Disk Drive	$129.95
1541 Clone Drive	$39.95
40 Col Monitor	$99.95
80 Col Monitor	$139.95
Printers/Call for Make	$39.95
Misc.	CALL

Items Listed Above Do Not Include Shipping. U.S 48 States - Add $5.50 per order. Alaska, Hawaii & Canada - add $5.50 for the first piece and $1.00 per each additional piece per shipment. Second Day Air shipping is available. Call for shipping charges. Call Or Write For Your Free c-64/128 Catalog Listing HUNDREDS Of Commodore Products And Special Offers For Your Computer. Our Order Takers Are On Duty 6:00 a.m. - 5:00 p.m. M - F and 7:00 a.m. - 3:00 p.m. Sat. - Pacific Time.

SOFTWARE SUPPORT

Software Support Int.
2700 N.E. Andresen Rd.
Suite D-4
Vancouver, Wa 98661
(360) 695-1393

CALL TOLL FREE TODAY! **1-800-356-1179**

Major Credit Cards Accepted.

CONTENTS

ISSUE 10 • VOLUME 2 • NUMBER 5 • OCTOBER 1995

COMMODORE WORLD
THE NEWS MAGAZINE FOR COMMODORE 64 & 128 USERS

GENERAL MANAGER
Charles R. Christianson

EDITOR
Doug Cotton

ASSISTANT EDITOR
Jenifer Esile

ADVERTISING SALES
Charles A. Christianson
(413) 525-0023

PHOTOGRAPHY
Wayne Wrubel

GRAPHIC ARTS
Doug Cotton
Jenifer Esile

ELECTRONIC PRE-PRESS & PRINTING
Mansir/Holden, Inc.

Cover Design by Jenifer Esile
Cover photo c/o Al Anger (see page 4)

 Commodore World is published 8 times annually by Creative Micro Designs, Inc., 15 Benton Drive, East Longmeadow MA 01028-0646. Second-Class Postage Paid at East Longmeadow MA. Annual subscription rate is US$29.95 for U.S. addresses, US$35.95 for Canada or Mexico, US$45.95 for all EC Countries, and US$57.95 to all other addresses worldwide. All subscription payments must be provided in U.S. Dollars. Mail subscriptions to CW Subscriptions, c/o Creative Micro Designs, Inc., P.O. Box 646, East Longmeadow MA 01028-0646.

POSTMASTER: Send address changes to: CW Address Changes, c/o Creative Micro Designs, Inc., P.O. Box 646, East Longmeadow MA 01028-0646.

FEATURES

REVIEWS

COLUMNS

DEPARTMENTS

See Our CHK-LIST Utility On Page 54

From The Editor

THE HOUSE THAT WINDOWS BUILT?

s it over? I believe it probably is. With a careful look around, I tried to determine just how much the world had changed with the release of Microsoft's new OS, Windows '95. After all the media hype, I expected the sun to rise in the west, birds to fly backwards, peace to reign throughout the world, and even more. Oddly, nothing much seemed to change.

Okay, that isn't entirely true. There were a lot more messages in the 'local' message base on the BBS I use to access the Commodore Fidonet echoes. Looking them over, I noticed a trend... all of the messages regarding Windows '95 seemed to be negative! I checked closer. Sure enough, I couldn't find a single positive response posted concerning the OS release that would shake the computing world to its foundation. What I did see, though, wasn't entirely unexpected. Users who had found some way to crash their hard drives during installation, others with applications that they could not get to work right anymore; not surprising at all, this stuff happens all the time when users upgrade their OS and applications, and the fact that quite a few more users were doing this at the same time easily accounted for the extra dose of problem messages.

But I did find something I didn't expect: several MS-DOS users were actually decrying the new release, and complaining about the endless upgrading that kept emptying their wallets. I had to chuckle a little as I thought back on similar sentiments expressed in past issues of Commodore World. But being the polite telecommunicator that I am, I bit my tongue (fingers?) and avoided posting any "I could have told you so" responses.

Now, Windows '95 really can't be all that bad. Like I said, problems are going to occur with upgrades of any kind. But it isn't even close to what the expectations of all the media-hype would have had us believe.

And only two days after the release, dealers were complaining that it hadn't brought them the extra business they had hoped it would. By the weekend, most retail computer stores were back to business as usual, which means slow this time of year. Odds are that this was mostly due to the majority of sales going through mail-order for all those software upgrades and additional RAM folks needed, as the apparent lack of interest at the retail level hadn't affected Microsoft's sales projections for Window's, which appear to be right on track.

Meanwhile, Microsoft head honcho Bill Gates got caught up in another situation a little closer to home—his home, as a matter of fact. Apparently, a Seattle PR firm decided that providing a web page with pictures of the new $50 Million mansion Gates is having built on Lake Washington. Ironic that Gates' most recent New York Times column talked about people's rights to privacy. Not ironic that it came at a time when he felt that his privacy was being threatened, but that it hadn't been that long since Microsoft itself had been accused of spying on individuals by collecting information on what applications users had on their computers via the Windows '95 beta electronic registration. Perhaps Bill grew up too fast to learn that old saying, "What comes around, goes around."

Now, since all of this brought me around to the subject of Bill's column, let me quote for you something he said a couple of columns back: "If a new computer or a software upgrade costs more than it's worth to you, don't buy it. After all, you don't have to upgrade. Software will run forever and computer hardware will work as long as it is kept in good repair."

Thanks, Bill. I couldn't have said it better myself.

Doug Cotton
Editor

THE COUNTDOWN HAS STARTED...

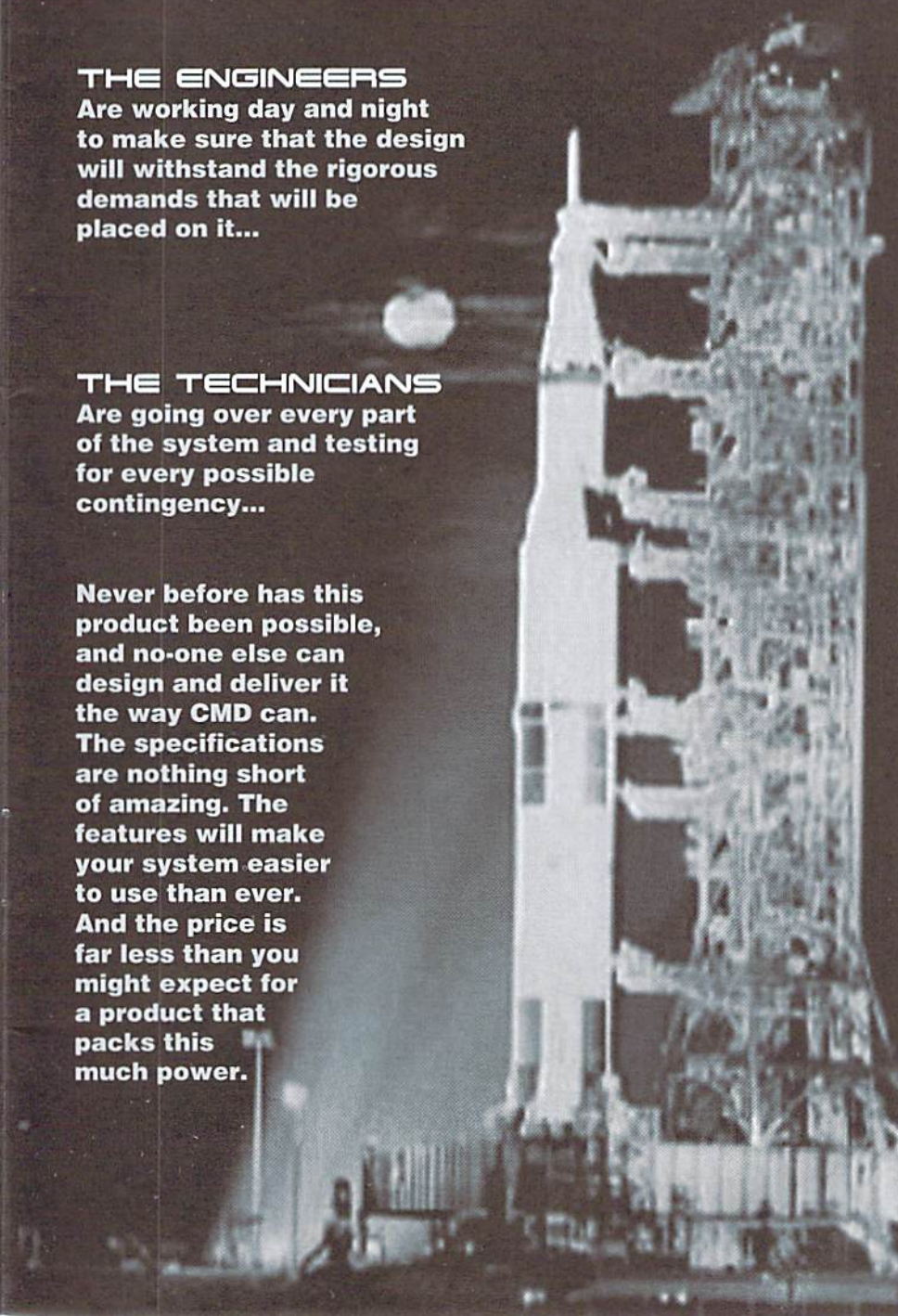
T-4 MONTHS AND COUNTING...

CMD is once again hard at work, doing what they do best: designing a hardware product that will push the capabilities of your computer to the very edge of current technology.

THE ENGINEERS
Are working day and night to make sure that the design will withstand the rigorous demands that will be placed on it...

THE TECHNICIANS
Are going over every part of the system and testing for every possible contingency...

Never before has this product been possible, and no-one else can design and deliver it the way CMD can. The specifications are nothing short of amazing. The features will make your system easier to use than ever. And the price is far less than you might expect for a product that packs this much power.



In just four short months, you'll have the opportunity to push your system to the limit. Prepare yourself for

SUPER64 CPU

Processor:	**10 or 20 MHz 65C816S**
RAM:	**64K Fast Static RAM**
ROM:	**64K w/JiffyDOS Kernal**
Features:	**Cart. Expansion Port**
	Enable/Disable Switch
	Turbo/Normal Switch
	Software Switchable

Compatible with C64, C64c, C128/C128D (in 64 mode), Commodore REU's, GEORAM, RAMLink, all Commodore serial drives (stock or JiffyDOS-equipped).

	Speed	MIPS[1]	RL/REU Compatible	Cart./Exp. Port
TurboMaster CPU	4 MHz	2	No	No
Flash-8	8 MHz	4	No	No
Super64/10	10 MHz	5	Yes	Yes
Super64/20	20 MHz	10	Yes	Yes

[1]Millions of Instructions per Second. The Super64/20 beats even a 25 MHz 386SX (8 MIPS).

The COMMODORE Customizer!

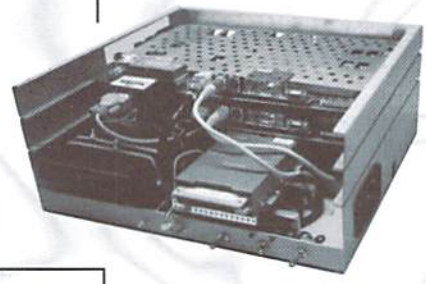
This month's cover features a C-128 Tower built by a true Commodore aficionado, Al Anger of Miami, Florida. Other projects by Al are shown below. If you'd like to contact Al, you can send him Internet e-mail via d0141066c@dcfreenet.seflin.lib.fl.us or write to him at 13841 SW 139 Ct., Miami, FL 33186.

Dual 1581

C-128 Tower

Dual C-128

Inside View of Dual C-128



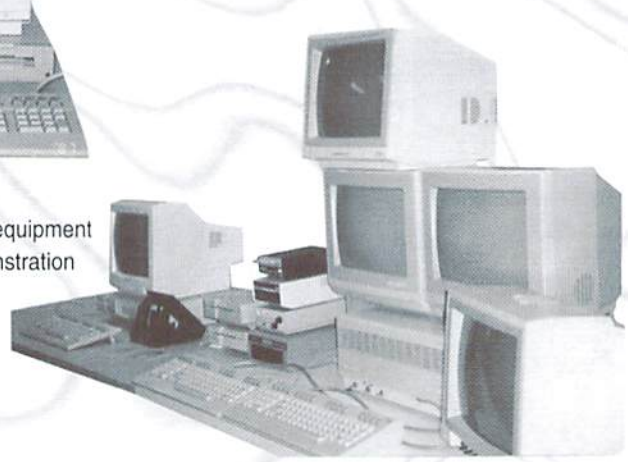
Serial, printer and computer selector box

CMD HD/FD-4000

128 style case with a RAMLink, a CMD FD-4000 and a CMD Hard Drive

128 D with a 1581 Replacing the 1571, expansion port moved to the front of the unit

Setup of Commodore equipment for user group demonstration

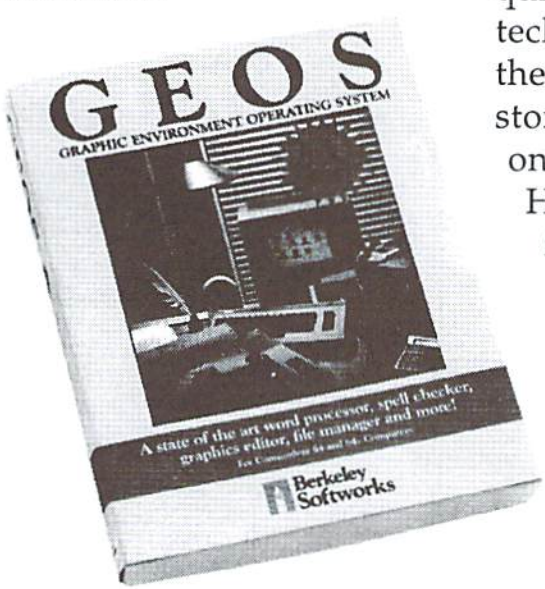


NO MATTER WHAT YOU DO ON A COMMODORE, DO IT WITH GEOS.

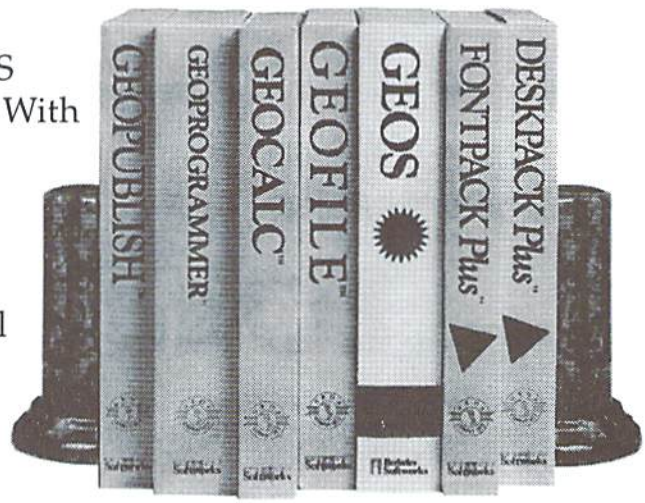
And with the whole GEOS™ family to choose from, you're bound to be able to do a lot of things you've always wanted to do. From word processing to desktop publishing, database management to programming, there's a GEOS application for nearly everything imaginable.

Do it easily.

Not only will you find a host of applications ready to use with GEOS, you'll also find out they're all easy to use. Why? Because GEOS applications share one very important thing in common...



...the intuitive GEOS graphical interface. With a simple point and click operating system, pull-down menus, and easy to use dialog boxes, all GEOS applications will make you feel instantly comfortable with selecting options, entering data, printing, and everything else.



Do it fast.

With a wide range of devices now supported, GEOS can operate at speeds you'll hardly believe. Get quick response from newer technology disk drives, like the CMD FD Series™ and store mega amounts of data on a CMD HD Series™ Hard Drive. And for even faster operation, GEOS works with RAM-based devices such as the Commodore REU or CMD RAMLink™.

Do it better.

GEOS has always supported a wide range of printers. Now, more than ever, GEOS is the logical choice for getting your ideas onto paper. For modern 9- and 24-pin IBM- or Epson-compatible printers, Perfect Print™ for GEOS provides output quality unrivaled by any other software for the Commodore.

Do it now.

So what are you waiting for? GEOS is ideal for most anything you want to do. And there's no better time than the present for doing it!

CMD
Creative Micro Designs

COMMODORE TRIVIA

by Jim Brain

Welcome to another edition of Commodore Trivia. As many of you may know, these trivia questions and answers have been donated by me to the Commodore community at large. Unlike other articles in Commodore World, these trivia questions have been placed in the public domain. I ask only that the trivia questions remain intact and unchanged, and that my name and address appear somewhere so users can contact me. The trivia is also used for a contest I run on the Internet; contact me at the included address for more information. Because curiosity has the best of me, I always welcome a note or postcard detailing where the trivia goes. I always welcome new questions—provided they come with answers. Enjoy.

Jim Brain
Brain Innovations, Inc.
602 North Lemen
Fenton, MI 48430

COMMODORE TRIVIA #9 QUESTIONS

$080 During the days of the Commodore 64 and the VIC-20, Commodore produced at least two Commodore magazines. What were their names?

$081 Back in the PET heyday, another magazine was produced by Commodore Canada. This magazine was later sold and showed up as a hardware journal. Name the magazine.

$082 The Commodore 128 has a VIC-II compatible chip inside it. Can this chips be switched for a VIC-II from a Commodore 64?

$083 What does the video encoding standard PAL expand to?

$084 How many buttons were present on the earliest of Commodore tape decks?

$085 Earlier SID chips had a distinctive "clicking" sound that some demo coders used to an advantage. Commodore subsequently removed the click, and then later reintroduced it. When does the telltale click occur?

$086 What does CP/M stand for?

$087 What is the highest line number allowed for a program line in Commodore BASIC V2?

$088 What symbol, clearly printed on the front of a key on the Commodore VIC, 64, and 128 keyboard, is not available when the lower case character set is switched in?

$089 How do you get the "checkmark" character ?

$08A On the PET computers, what memory location holds the Kernal ROM version?

$08B The Commodore computers have 2 interrupts, called IRQ and NMI. What does IRQ stand for?

$08C What does NMI stand for?

$08D The 6502 line of microprocessors has a number of flags that can be used to test for certain conditions. One of then is the N flag. What does it stand for?

$08E How about the D flag?

$08F The shorthand for the BASIC keyword PRINT is '?'. What is the shorthand equivalent for PRINT#?

COMMODORE TRIVIA #8 ANSWERS

$070 Revision Level 2 ROMs (the ones with more bugs) power up with: *** COMMODORE BASIC ***, with '*' in place of the more familiar '#' character.

$071 General Purpose Interface Bus. Another name is Hewlett Packard Interface Bus (HPIB), since HP developed this standard for its instrumentation device networking.

$072 The Commodore D9060 and D9090. From the cbmmodel.txt file:

* CBM D9060 5 MB Hard Drive, DOS3.0, Off-White, IEEE-488 (GP)

* CBM D9090 7.5 MB Hard Drive, DOS3.0, Off-White, IEEE-488 (GP)

The following model has been said to be in existence, though no one has one on hand to prove it:

* CBM D9065 7.5 MB Hard Drive

And this model may never have made it past the prototype stage:

CBM D9062 Dual D9065

$073 It looked just like a old-style C-64. It had a "home" computer look that the schools didn't care for. They liked the "business" look of the PET series, so Commodore put refurbished and new 64 motherboards in PET cases and sold them as PET 64s. The repackaging suited the schools.

$074 An array can have a cumulative total of 256 elements. For single dimension arrays, that means D(0) to D(255), but a 2D array can only go from DD(0,0) to DD(1,127) etc. All types of arrays had this limitation.

$075 3 bits were transmitted at a time. I assume that each byte had a parity bit tacked on for error detection, so it would have taken 3 transfers to transmit a byte of information from the drives.

$076 300 RPM.

$077 73,CBM DOS V2.6 1541,0,0

$078 73,CBM DOS V2.6TDISK,0,0. Notice that the new text JUST fits!

$079 #5. The Commodore 1525 has a switch to do this, but not all printers have such a switch.

$07A The 6510T. It is a slight variant on the 6510 microprocessor used on the C64. Some say it runs at 2 MHz, but the drive's spec sheet doesn't say.

$07B Let's go back to question $04F:

> $04F What was the primary reason Commodore went to a serial bus with the introduction of the VIC-20?

Jim Butterfield supplied me with this one:

> $04F As you know, the first Commodore computers used the IEEE bus to connect to peripherals such as disk and printer. I understand that these were available from one source: Belden cables. A couple of years into Commodore's computer career, Belden went out of stock on such cables (military contract? who knows?). In any case, Commodore was in quite a fix: they made computers and disk drives, but couldn't hook 'em together! So Tramiel issued the order: "On our next computer, get off that bus. Make it a cable anyone can manufacture". So, starting with the VIC-20, the serial bus was born. It was intended to be just as fast as the IEEE-488 it replaced.

And here is what Jim Butterfield followed up with:

> "Technically, the idea was sound: the 6522 VIA chip has a "shift register" circuit that, if tickled with the right signals (data and clock) would collect 8 bits of data without help from the CPU. At that time, it would signal that it had a byte to be collected, and the processor would do so, using an automatic handshake built into the 6522 to trigger the next incoming byte.
>
> Things worked in a similar way outgoing from the computer, too. We early PET/CBM freaks knew, from playing music, that there was something wrong with the 6522's shift register: it interfered with other functions. The rule was: turn off the music before you start the tape! (The shift register was a popular sound generator). But the Commodore engineers, who only made the chip, didn't know this. Until they got into final checkout of the VIC-20.
>
> By this time, the VIC-20 board was in manufacture. A new chip could be designed in a few months (yes, the silicon guys had application notes about the problem, long since), but it was TOO LATE!
>
> A major software rewrite had to take place to change the VIC-20 into a "bit-catcher" rather than a "character-catcher". It called for eight times as much work on the part of the CPU; and unlike the shift register plan, there was no timing/handshake slack time. The whole thing slowed down by a factor of approximately 5 to 6.
>
> When the 64 came out, the problem VIA 6522 chip had been replaced by the CIA 6526. This didn't have the shift register problem which caused trouble on the VIC-20, and at that time it would have been possible to restore plan 1, a fast serial bus. Note that this would have called for a redesign of the 1540 disk drive, which also used a VIA. As best I can estimate (an article in the IEEE Spectrum magazine supports this) the matter was discussed within Commodore, and it was decided that VIC-20 compatibility was more important than disk speed. Perhaps the prospect of a 1541 redesign was an important part of the decision, since current inventories needed to be taken into account.
>
> But to keep the Commodore 64 as a "bit-banger", a new problem arose. The higher-resolution screen of the 64 (as compared to the VIC-20) could not be supported without stopping the CPU every once in a while. To be exact: Every 8 screen raster lines (each line of text), the CPU had to be put into a WAIT condition for 42 microseconds, so as to allow the next line of screen text and color nybbles to be swept into the chip. (More time would be needed if sprites were being used). But the bits were coming in on the serial bus faster than that: a bit would come in about every 20 microseconds! So the poor CPU, frozen for longer than that, would miss some serial bits completely! Commodore's solution was to slow down the serial bus even more. That's why the VIC-20 has a faster serial bus than the 64, even though the 64 was capable, technically, of running many times faster.
>
> Fast disk finally came into its own with the Commodore 128."

$07C 192 bytes is used as a tape buffer. Blocks of data on tape are 192

$07D #3

$07E #0

$07F (This was not a Commodore specific question) Commodore computers use this notation to represent integer quantities. In 2's complement notation, a -1 looks like 11111111 (binary) or $FF (hex).

ON THE HORIZON

COMMODORE AND COMPUTER INDUSTRY NEWS

Updates to geoFAX

Click Here Software has performed a minor upgrade to geoFAX that corrects problems with specific modems. The original release version of 1.5 presented problems to users with 28.8Kbps modems. This has been corrected in the newly released version 1.6. The new version also contains a work-around for a problem in Australia, where the dial tone is odd enough to keep modems from properly detecting it.

Registered owners experiencing either of these problems should contact CHS to obtain an upgrade.

GeoFAX is available directly from the author ($39.95 plus $4.00 s/h): Maurice Randall, P.O. Box 606, Charlotte, MI 48813, (517) 543-5202. It can also be purchased from dealers, such as CMD.

The 128 Gets MODified...

Nate Dannenberg, author of Sound Studio, has recently announced that he'll be releasing a C128 MOD music file player. Such player programs have been popularized over the last two or three years on a variety of platforms, including the Amiga and MS-DOS computers. Dannenburg excitedly announced this past month that his 128 program was finally coaxed into playing a MOD file created by the Amiga Protracker program.

While the program is still in development, Dannenburg unveiled these details: "Only a few Protracker commands are supported, and the sample rate is only 4.1 KHz, but the program is working, and it plays most of my MODs with little loss of quality!" He added, "Currently for the C128, the MOD player requires a Ram Expansion Unit (it will use up to 1 MB if available) and Stereo SID chips."

No release date was set, but anyone with ideas or comments can contact Dannenberg via Internet e-mail (tron@onyx.southwind.net).

Creative Micro Designs Announces Super64 CPU Accelerator Series

CMD has now officially announced that they are developing a new series of accelerators for use with Commodore computers. Scheduled for release in early 1996, two models are currently planned. Both will use high-speed 65C02S processors supplied by Western Design Center. The two models, designated as the Super64/10 and Super64/20 will operate at 10 MHz and 20 MHz respectively.

CMD points out that the high speed of the Super64/20 coupled with the pipelined architecture of the 65C02S enables it to achieve a MIPS (Millions of Instructions Per Second) rating higher than a 25 MHz 386SX Intel processor. This speed also requires using support chips on or near the leading edge of technology.

The accelerators will use methods similar to those employed by the Commodore 128 series computers to sync to slower components in the computer. These methods slow the acceleration to 1 MHz approximately 10 percent of the time, yielding effective speeds of up to 9 MHz and 18 MHz on the two models. CMD, however, claims that higher effective speeds may be possible through special configuration methods being considered for the final production units.

The accelerators are also to offer an expansion port compatible with Commodore REU's, GEORAM, and CMD's own RAMLink devices. Some other cartridges, such as CMD's SwiftLink and SID Symphony should also be compatible, but most other utility and game cartridges will not be compatible in accelerated modes.

Other features mentioned are an Enable/Disable switch, Turbo/Normal Mode switch, and software control of Turbo/Normal modes. Both models will also provide the JiffyDOS computer Kernal to speed operation with JiffyDOS-equipped drives (stock Commodore and Commodore-compatible drives are also fully supported).

The two models are scheduled to be available in February, and CMD has estimated that the retail prices will both be under $200.00.

Point Survey Gives Top 5% Rating to Site with Commodore Content

Jim Brain and Brain Innovations, Inc., announced today that its Commodore World Wide Web Site had been rated in the top 5% of all sites on the Internet by Point Survey, a World Wide Web Site rating service sponsored by Point Communications Corporation. The Commodore site includes informational material, pointers to on-line resources, pictures, and historical documentation on the popular home computer, manufactured in the early 1980's by Commodore Business Machines, now Amiga Technologies. The site is at http://www.msen.com/~brain/cbmhome.html.

This information is presented on the World Wide Web, a graphical multimedia hypertext service on the Internet, a large collection of networked computer systems encompassing much of the world. The World Wide Web, or WWW, presents textual and graphical information from areas called "sites" and displays the information in "pages". The concept of WWW allows a variety of content to be gathered into a single source to perusal by on-line users.

Point Survey is a free service which rates and reviews only the best sites on the World Wide Web. Point provides Internet users with a standard of excellence: a catalog of the most lively, useful, and fun sites on the Internet. Point Survey is on the World Wide Web at http://www.pointcom.com/ Point's ratings are based solely on merit as judged by Point's reviewers.

Point Survey ratings are made available to media around the world, and

Point's Top Ten list has been featured on CNN and in many publications. In addition to the rating, Point Communications will include the Brain Innovations site information and a screen shot of the site in its upcoming book, tentatively entitled, "The 1000 Best Sites on the Internet".

Brain Innovations President Jim Brain said the Point Survey rating will provide more exposure for the Commodore site, which is maintained by his company as a free service. The site, which currently tracks over 700 visitors daily, combines the technology of the World Wide Web with the popularity of the Commodore computer systems in one dynamic multimedia exhibition. Brain notes that the both new and experienced Commodore owners can find information of interest and pointers to suppliers and repair facilities from the site.

Brain Innovations, Incorporated is an Internet consulting and embedded hardware/software development company based in Fenton, Michigan.

Seagate and Conner Peripherals Reach Merger Agreement

Seagate Technology Inc. has reached a preliminary agreement to buy Conner Peripherals Inc., creating the world's largest independent disk drive maker. The agreement is based on a stock swap valued at about $1.11 billion.

Seagate claims that it is pursuing the agreement to gain access to Conner's manufacturing operations, considered by many in the industry to be among the most efficient and technologically advanced. Analysts believe that Connor's facilities and reputation might also make them attractive to other suitors.

Seagate is already the largest independent drive maker in dollar value of sales. The acquisition would boost its disk drive production to nearly 7 million units a quarter, overtaking Quantum Corp., who is currently the number one supplier in terms of units shipped, with almost 5 million units shipped in the last quarter.

The announcement came as a surprise in the disk drive industry, where a rivalry between the outspoken founders of the two companies has been ongoing since the Conner Chief Executive Finis Conner stormed out of Seagate 10 years ago. Seagate Chairman Al Shugart denied that there was any ill-will between himself and Finis Conner, who helped Shugart found Seagate in 1979. Finis Conner also joined Shugart Associates, the predecessor to Seagate, when it was founded in 1973. Despite the denial, analysts agree that Finis Conner will likely leave the company once the merger is completed.

The two companies haven't yet reached a definitive agreement and are still in talks, according to Seagate Chairman Al Shugart. The transaction is subject to completion of due diligence, signing a definitive agreement and approval of both boards.

Conner, who had posted the best first-year earning for a U.S. company at the time back in 1987, lost $445.3 million in 1993 because of devastating price wars in the storage industry and outdated inventory, and has been struggling to recover. Analysts said there had been speculation that Samsung Corp. would buy Conner, and that Seagate may have stepped in to prevent the South Korean electronics company from gaining a bigger hold in the disk drive industry.

All Modems will NOT be Boca-compatible...

In Commodore World Issue #9 we reported that Boca had reached an agreement to purchase Hayes. Shortly after we went to press, the deal fell through. Boca claims that Hayes was still out courting other offers, and that they no longer felt they could work with Hayes toward a final agreement.

Fee Now Required for Domain Name Registration

Beginning in mid-September, Network Solutions, the InterNIC Registrar began a new policy requiring a fee of $100 for registration of new Domain names in the "COM", "ORG", "NET", "EDU", and "GOV" domains. An annual maintenance fee of $50 will also be collected for each existing domain names.

Since March 1, 1993, the National Science Foundation has funded the administration of the "COM", "ORG", "NET", "EDU", and "GOV" and root domains through a Cooperative Agreement with Network Solutions, the InterNIC Registrar. The funds received from the fees will replace the funding provided by the National Science Foundation, and will provide "program income" which will offset costs related to the intellectual infrastructure of the Internet.

To further explain the need for these fees, information on the InterNIC web site states that the exponential growth of the Internet, due mostly to the connecting of commercial organizations to the Internet over the past couple years, has had a directly proportional affect on the registration activity of the Registrar. The increased activity, with the corresponding growth of operating costs, have resulted in funding requirements exceeding the National Science Foundation's budget. InterNIC feels it is appropriate that Internet users, instead of the U.S. Federal Government, pay the costs of domain name registration services.

New domain names are valid for two years from the date that the Registrar activates the domain name. The Registrar will activate domain names upon request, on a first-come, first-serve basis. Payment of the Registration Fee is due on the 30th day after the activation date, and is non-refundable.

For all registered domain names, the annual maintenance fee will be due upon the anniversary date of the domain name activation. This fee will keep the domain name valid for one year. Payment must be made in advance on an annual basis, and is non-refundable.

The Registrar will remove domain names from the database upon the request of the domain name holder, and will also remove domain names for which registration or maintenance fees have not been received by the due date. Domain names deleted from the database will be available for reuse after a waiting period of 60 days.

Domain name holders will be notified via e-mail 60 days prior to the due date of their annual maintenance fee. Additional notices will be sent at 30 and 15 days prior to expiration.

For further details, visit the InterNIC web site (http://www.internic.net/).

ERROR CORRECTIONS

In Issue 9 we reported an error regarding our book review of *RAM DOS 128 Case Study*. We incorrectly stated that the correction was for an error in Issue 9, whereas the initial error had actually been in Issue 8. Despite the ribbing our editorial staff took over blundering on a correction to a previous error, we decided to keep our jobs and put out another issue.

In Issue 9 (really, it was Issue 9 this time) there was an error in the Commodore World Sweepstakes Winner box, which reported that we were giving away an FD-400. This should have read FD-4000.

TOP TIPS

UPGRADE TO GEOS 2000 NOW!

Recently, an Associated Press article in my local paper got me thinking about the turn of the century; specifically, how the year 2000 will affect me as a GEOS user.

The article addressed the concerns and problems of a computerized society, where only two numbers represent the year. As long as the century stays the same, there is no problem. When a new century arrives, however, many problems could occur. According to the article, Boeing Co. expects to have over 1000 workers correcting this flaw in its own systems.

Many other businesses will experience similar problems that may affect each of us. Credit card balances and phone bills are mentioned as possible victims of calculations that could be off by 100 years, the possibilities for problems seem endless. Unfortunately, the article did not state that the bank would give me one hundred years extra interest on my balance. Go figure.

What this has to do with GEOS may be obvious to you by now. Like all these business computers, GEOS only allocates two digits for the year. Look in any info box and the file date looks like 12/29/87, for example. Even the clock in my CMD FD-2000 uses this format. While we can't change the basic storage format of the date in the system, we can adjust the programs that use it.

The most obvious adjustment required is the Calendar Desk Accessory. This program will let you look ahead into the 21st century, but if your date is set to 01/01/00 the calendar that comes up is January 1900. Upgrading your Calendar for the new millennium only requires changing two bytes. The program included in this article (see box at right) will install the patch very easily. To use this patch program, copy the Calendar DA to a fresh, blank disk. This step is very important, as the patch program only works on the very first program on the disk in drive 8, and does no checking to see if it is the Calendar DA.

This program works on the Calendar Vl.2 from the Deskpack Plus collection (mine is dated 12/29/87 4:08 PM). If you have Calendar V1.0 from the original Deskpack (mine is 10/9/86 3:00 PM) you must make a change to the program.

Enter the program as above, but enter line 180 as follows:

```
180 z(15)=2:z(20)=0:gosub2000
```

While most GEOS users likely don't use their systems for purposes that could cause million dollar mistakes, this little fix will make the 21st century less of a nuisance by eliminating the need to reset the Calendar date every time you use it. Also, when you select TO PRESENT in the date set menu, the calendar will go to the proper century.

You may want to rename your new Calendar to something like CAL 2000. enGEOy!

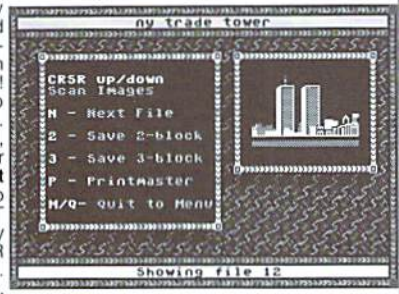
- Bruce Thomas

```
90 rem patch prog for geos calendar for
21st century default
100 dimz(256)
110 open15,8,15:open2,8,2,"#"
120 tr=18:se=1:gosub1000
130 tr=z(3):se=z(4):gosub1000
180 z(8)=2:z(9)=0:gosub2000
260 close2:close15:end
1000 print#15,"u1"2;0;tr;se
1010 forx=0to255
1020 get#2,a$:a$=a$+chr$(0)
1030 z(x)=asc(a$)
1040 next
1050 return
2000 print#15,"b-p:"2;0
2010 forx=0to255
2020 print#2,chr$(z(x));
2030 next
2040 prlnt#15,"u2"2;0;tr;se
2050 return
```

LOADSTAR SOFTWARE VALUES!

The Compleat* Series

The Compleat PRINT SHOP I: Over 1300 artistic and never before published PRINT SHOP images. The smart, fast software package included allows you to quickly scan through the many PRINT SHOP images sequentially, by name, or by group number. Press a key and save the graphic you want in 2-block, 3-block and even PRINTMASTER graphic files! All that plus a printed guide to your new sea of graphic files. And if you like the first volume, you'll probably want to get your hands on **The Compleat PRINT SHOP II**, which also contains over 1300 PRINT SHOP images, previously published on LOADSTAR issues over the past ten years. Included is the same smart, fast software package included in The Compleat PRINT SHOP I. Each volume is $20.00. **Volume 1**: C-64/128 3.5-inch disk item #0001d3. 5.25-inch disks item #0009d5. **Volume 2:** C-64/128 3.5-inch disk item #0002d3. 5.25-inch disks item #0010d5.



The Compleat Programmer: Become the Commodore programmer you've always wanted to be! Megabytes of knowledge crammed and stuffed onto eight 5.25-inch disks or two 1581 disks! Plus we include all the tools, extensions, languages, assemblers, tutorials and utilities you'll need to create the same type of software you see on LOADSTAR! This massive collection is way over two megabytes of instructional text and valuable tools. 5.25 set #0005D5. 3.5-inch disk #0005D3. For $5 more, get C= Hacking MAG #0006D3 (on 3.5-inch disks only and NOT available separately) to complete your programming set. $20.00 postage paid.

The Compleat Maurice: A compilation of 26 solitaire card games written by Maurice Jones, the acknowledged master of card game simulations for the C-64/128. There's even a brand new, never before published game called Boomerang. Two 5.25 inch disks #0007D5 or one 3.5 inch disk #0007D3. $20.00 postage paid!

The Compleat Dave: Two 1581 disks or three 1541 disks crammed with SID music. Over 250 classic melodies from yesteryear, arranged and transcribed by the Master of Music, Dave Marquis, and now they're available in one gigantic 8-hour collection. If you enjoy SID music, you owe it to yourself to get THE COMPLEAT DAVE. Two 1581 disks #070523 Three 1541 disks #070525. $20.00 postage paid!

The Compleat Walt: During LOADSTAR's first ten years we have published 24 of Walt Harned's slideshows and multimedia events. Now we've gathered them into one huge collection: seven 5.25 inch disks or three 3.5 inch disks! There are over 250 pictures, including some that have never been published. The greatest one-man collection of art on any computer platform! As this example picture, taken from *The Clowns Of LOADSTAR* shows, Walt knows how to push a C-64 to its limits and create stunning art. 5.25-inch disks order #070425 3.5-inch disks order #070423. $20.00 postage paid!

The Compleat Roger: 25 educational quiz programs, each carefully crafted by Roger Norton, an educator who uses C-64's. These programs come crammed on two 5.25-inch 1541 disks or one 1581 disk. 5.25 set #0004D5 3.5-inch disk #0004D3. $20.00 postage paid!

The Compleat Prosequest: A 1541 disk with all of the entries in the 1993 short story writing contest on it, including the three grand winners. $4.95.

*** No, we didn't misspell "complete." Compleat is the ten dollar spelling of complete.**

What Is LOADSTAR?

LOADSTAR is a monthly "magazine on disk" for the Commodore 64/128. Subscribers receive two 1541 disks (or one 1581 disk) in their mailbox every month filled with news, articles and programs. These non-PD, high-quality programs are written by the best home-based programmers in the field and edited by the crack LOADSTAR team of Fender Tucker and Jeff Jones. Subscription prices are at an all-time low of $69.95 for a 12-month subscription, or $19.95 for a three-month subscription. You may also elect to subscribe "by the month," where we charge your credit card $6.95 for each issue *after* it's shipped.

LOADSTAR's track record of over 11 years of uninterrupted publication (135 monthly issues, each available as a back issue) is unmatched by any Commodore computer magazine, disk or paper. As long as there are devoted Commodore 64/128 fans, there will be a LOADSTAR Tower. Don't miss out! **Call 1-800-594-3370 and subscribe!**

Other Products

Geopower Tools - 19 Geos utilities: Calendar Printer, Fast Format, Geo Fetch (grab any portion of a screen as a Photo Scrap), Phoenix (resurrect a trashcanned file), Programmer's Calculator are just a few of the handy tools. Side Two is filled with Clip Art (in Photo Album format) and fonts. $9.95 (C-64/128) Item #080525

Songsmith - LOADSTAR's own music-making program. With this deluxe music editor/player you can easily transcribe music from sheet music or make up your own tunes. Songsmith comes with a slick 30-page manual and a jukebox player with eight tunes. $9.95 (C-64/128) Item #069525

Game Star #1 - Eight games from LOADSTAR #70 - #100). The Tenement, Stack 'Em, The Sherwood Open, Gems, Stealth Bomber, Eagle Eyes, Moonraker and Circuitry. $9.95 (C-64/128) Item 080825

Just For Fun - Eight original games. There are arcade games, educational games, puzzle games and just games that are just plain fun on this disk. $9.95 (C-64/128) Item #073525

Fun Four - Four original games. A huge maze game, trivia game, solitaire and a space shoot 'em up -- all runnable from a menu. $9.95 (C-64/128) Item# 080725

Sport -This is a full novel on C-64 disk by author, Jeff Jones, about a crazed superhuman creature that kills humans for sport. Over 500 terrifying pages. Completely automatic presentation software included. Optional printing capability. Bookmarks. Warning! This is a *real* novel with strong content, frank language, violence and adult situations. If the uncensored, hard-hitting action of *real* sci-fi/horror novels turns your stomach, please don't buy this book. Must be 18. One CMD HD disk #070327. Two 1581 disks #070323. Three 1541 disks #070325. $5.95 $1.00 Shipping.

Brainpower/Brainstorm - 80-column word processor and idea processor for the C-128. These two programs are together on one disk. It comes with a detailed 32-page manual. $9.95 (C-128 only) Item #069421

Best Of Loadstar Compilations:

SEE LOADSTAR'S GROWTH FROM HUMBLE BEGINNINGS WITHOUT BUYING ALL 135 BACK ISSUES!

#5 Anthology disk. $9.95 (C-64/128) Item #049525
#4 Anthology disk. $9.95 (C-64/128) Item #049425
#3 Anthology disk. $9.95 (C-64/128) Item #049325
#2 Anthology disk. $9.95 (C-64/128) Item #049225
#1 Anthology disk. $9.95 (C-64/128) Item #049125

Master Base - Database of users' groups for the 80-column C-128. This is a fast, powerful database program for handling addresses and mailing labels (includes barcode printing). The disk also has a file of over 600 users' groups addresses. $4.95 (C-128 only) Item #081025

Still to come! The COMPLEAT GEOS, a treasure chest of GEOS clip art encompassing over a dozen 3.5-inch disks. Our GEOS clip art was created by computer graphics professionals whose only job is to create great art. The COMPLEAT STRATAGEMS with over 300 game genies to help you finally beat those games. Each set will be $20 postage paid. Available this fall.

Send Check or money order to:

Softdisk Publishing

Box 30008, Shreveport LA 71130-0008

VISA/MASTERCARD/DISCOVER/AMEX Call Toll-free 1-800-594-3370

Just For Starters

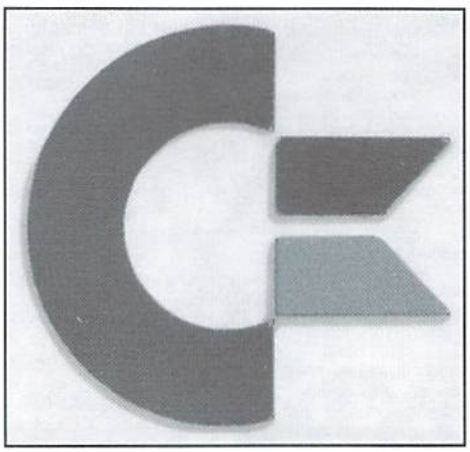
by Steve Vander Ark

TELECOMMUNICATING FOR THE BEGINNER

When you're new to computing, one of the hardest things to do is get a straight, understandable answer to your questions. A lot of the folks you'll meet at user groups or on line have been involved in computing for many years and have forgotten what it's like to start from zero. Their understanding of how a computer works and of "what affects what" inside that little box is formidable. As a result, their answers may assume a lot of prior knowledge on your part or rush past some of those details in a flurry of techno-speak. If you're lucky, you'll find a few people who are either fairly new themselves or who have a knack for explaining things in terms anyone can understand. If you can't find someone like that, you may find yourself getting more and more confused and decide that your worst fears were true: computers are just too darned complicated.

Don't despair. Before you get to that point, write me a letter, ask your question. I would really like to start answering beginning readers' questions as a regular feature in these two pages. You can send me a letter by US Mail to: Steve Vander Ark, 6730 Brad Ave. SE, Grand Rapids, MI, 49548. You can email me as well if you have an America OnLine account, to my screen name, which is SteveVArk (no spaces). On GEnie, my screen name is STEVE.VA. If you'd rather use the Internet, email me at steve.va@genie.geis.com. You never know, you may see your name in this column one of these days.

Once again, I'll be talking about programming in BASIC, and discussing some things about program flow, which is a fancy way to refer to something very simple. Before I get to that, I want to spend a few hundred words talking about a subject which is becoming more and more important in our world: telecommunications. That is also a fancy sounding word, but one which you are likely to have read before. Telecommunications is part of a staggering



change which is currently taking place in our society. Your computer is part of that change, whether or not you're taking advantage of it.

In a sense, this discussion is becoming a moot point. Telecommunications is starting to become invisible, and the definition is changing rapidly to encompass more areas of our lives. For example, when you access your bank account using an ATM machine, you're using telecommunications; you just don't see it happen. It won't be long before your cable TV connection will become a two-way interactive connection more like the Internet's World Wide Web, and even that will be telecommunications. In a few more years, we will be telecommunicating all the time. Then I won't have to tell you how to do it; any more than I'd have to explain how to dial a phone or chose a television channel.

But we're going to stick with the older designations here, in a large part because the hardware and software which will make the new ideas happen are still pretty much in the prototype stages. Particularly if you are using a Commodore, telecommunications requires some explanation. To communicate with your 64 or 128, you see, you need to jump through a few hoops. You can do it, but it will take some thought on your part and probably a small outlay of cash. There are three main pieces to the telecommunications puzzle on your computer, each of which you'll have to buy in some form.

The first piece is called a modem. This is a little box which you will most likely plug into the user port on the back of your computer. A modem can be thought of as a translator which can interpret your computer's language into a series of signals which can be sent to your telephone to another computer. Signals sent back to you are interpreted by the modem back into the kind of information your computer can display, usually as letters and numbers. This little device is essential; without it, your computer is dumb and illiterate. You'll have to buy one, but you'll find that the prices for Commodore modems are surprisingly low. Check the CMD catalog for more details. Modems are capable of various "baud rates," which generally refers to the speed at which information travels through it. You will probably want to get the fastest one you can afford. Typical baud rates are 300 (pretty slow), 1200, and 2400.

The second part of the telecommunications puzzle is a terminal program. This program provides you with the means of entering text to send out and of seeing the text others send to you. A good telecommunications program will include a lot of powerful features which you will find extremely useful, but in its most basic form it will dial the phone, make the connection with another computer, then allow you to type out and look at the text coming back in.

Before I talk about the third piece of the puzzle, let me explain a couple of things that you'll certainly run into. For two computers to connect properly, they have to agree on certain things right off the bat, including how fast they will send information back and forth, how they will

know when the other machine is done sending, and so on. These details must be set the same for each computer or the connection won't happen. It is in this area that the most confusion will happen, so it's worth taking a second to think about them.

Let me just say right here and now that I myself don't really understand what some of these things mean, terms like "stop bit" and so on. I tell you that to reassure you that you don't need to understand them all either. You just have to make sure they're set properly. Any term program will include a way to adjust these settings and often you can set them separately to apply to a particular phone number. Then, save those settings to automatically go into effect every time you call that number. How do you know how they need to be set? Usually, the computer you are calling will be set up already in a certain way to accept calls, and when you are given the number you'll see their settings included right with it. It might say "8-N-1," for example, which means that you will set the "data bits" to 8, the "parity" to "N" or "none," and the stop bits to 1.

The third piece of the telecommunications puzzle is the most exciting. Once you have the modem and a term program, you'll need a place to call. You'll need a phone number of some other computer which also has a modem installed and will answer your call. There are many of these set up all over the country called "bulletin boards", or BBS's. These are usually run by private individuals for the fun of it out of their homes. Some are run by teenagers who use them as a forum for talking about everything from role-playing games to rock and roll. Others are run by clubs or businesses to keep in contact with customers or members. Here you might find message areas filled with ideas and tips for using your computer as well as areas for more wide ranging discussions. In the Grand Rapids area, there are BBS's devoted to science fiction topics, to home-brewing, and to general chat about politics and issues of the day. It's also possible to call the county library systems computer and scan their files.

There are larger services you can connect to for a fee, which you may have heard of. Often you need to use their terminal programs (Macintosh and Windows as a general rule) to access them, making them off limits to Commodore users. Those include America OnLine and Prodigy. Some services allow access by standard terminal programs as well as with their own software, such as on GEnie and CompuServe, where you will find an active Commodore presence. All of these services charge a monthly fee for access, plus usually an hourly charge once you're connected more than a certain number of hours in a month. The vast array of services they offer is worth the cost of the service.

Moving Along

Inside your computers operating system is something called a "BASIC interpreter." That's how your computer can understand the commands you've been typing into it. This interpreter knows what all the BASIC commands mean and how to make them work with your computer. It also keeps track of where it is as it moves through your list of commands.

That's important because if your computer doesn't execute things in the right order, the program won't run. Moving in a consecutive manner from one command to the next is called "program flow." It's easy to see where your programs so far have been flowing, since they simply move from one numbered command to the next in numerical order. As we noted, the line numbers can skip, say by tens, but as long as they're in order numerically, the BASIC interpreter will just move along from one to the next, as nice as you please.

There are times when you don't want the program to execute in order of commands. Sometimes you'd like the program to jump over to another line which is out of numerical order. An example of this is when you ask the person using your program to make a choice from several options. Depending on which option they chose, you'd want the computer to jump to one set of commands or another. BASIC includes several useful commands to let you change the program flow as desired.

The simplest example is the command GOTO. It means "go to," not surprisingly, and it is always followed by the name of the place you want the BASIC interpreter to go at that point. You specify this using a line number, so if you wanted the program flow to jump from line 100 to line 200, you'd enter this command:

```
100 GOTO 200
```

If your program has no line number 200, your computer will tell you it's confused with an error message. Once you jump to line 200, your program will continue from that point and all the line numbers you might have between 100 and 200 will be ignored. To try using this command, see the box at the end of the page.

Several things happened here which are worth noting. First of all, did you see that IF command? The commands after the IF on the line only happen if the IF tests out true, in this case if the number entered in line 10 was a 1. If not, the program moves on to the next line and checks if the user entered a 2. This is a very powerful command and we'll use it more as we go along. Also, you will see that I placed an END command in the program at several points. This is helpful to keep the program from moving on into areas we didn't want it to go. Unfortunately, it also stops our program cold in its tracks. If we wanted to keep going and start the program over to choose another number, we could replace that END with another GOTO. For lines 110 and 210 we could put GOTO 10 and the program would ask for a number again.

Commands which change the program flow are very important in any programming language. We'll spend more time with IF...THEN and with some alternatives to GOTO in our next column.

SAMPLE PROGRAM USING GOTO

```
10 INPUT"ENTER 1 OR 2";X          (this line asks the user for a number and calls it "X")
20 IF X=1 THEN GOTO 100           (this line checks if X is a 1 and if it is, jumps the
                                   to line number 100)
30 IF X=2 THEN GOTO 200           (this line checks if X is a 2 and if it is, jumps the
                                   to line number 200)
100 PRINT "YOUR NUMBER IS ONE"    (this line responds to a 1 entered; if you enter 2,
                                   this will never happen)
110 END                           (if I didn't put this here, the program would
                                   now go on to line 200 , even though we
                                   never entered a 2!)
200 PRINT "YOUR NUMBER IS TWO"
210 END
```

Foreign Exchange

By Joseph Gaudl

WAITING FOR GODOT...

Back in 1952, the Irish-born poet, novelist, and foremost dramatist of the theater of the absurd, Samuel Beckett, wrote his masterpiece play, "Waiting for Godot." Two tramps spend the entire play contemplating and waiting for the arrival of a certain Mr. Godot, who never arrives. Beckett's purpose for writing the play was to show the human need for hope.

Forty years later, two German programmers got tired of waiting for the perfect graphic program and decided to pool their resources and write the best graphic program ever available for the C64/128. Arndt Dettke and Wolfgang Kling were introduced to me as "Children of the Beat Era" (which means they are both in their forties) and classic Commodore freaks.

Arndt still has his first C64, which he bought in 1983, and uses on a daily basis! As a highly skilled and successful programmer, in 1986 he rewrote the well known but buggy "Simon's Basic", which was then released as part of a school software package and called "Tuned Simon's Basic". In August 1987 he joined the staff of the German 64'er magazine and has written articles and software non-stop for this highly successful periodical.

Wolfgang is a graphic expert who conceived the idea for their joint program, showing up at Arndt's door and asking if they could work on the project together. They discussed the concept for the graphic program and came up with some guide lines. First of all, the program had to have professional qualities and tools which were similar to those of other computer types. Second, the program must be compatible to every C64 configuration. Those users who only owned a

C64 with a 1541 should be able to get good results. Users with expensive and extensive peripherals would be able to produce even better results with the program. Third, this had to be the program that all C64 users have been hoping and waiting for.

Coming up with a name for their program - even before it was written - was relatively easy. Both Arndt and Wolfgang are full time teachers of the arts and familiar with English literature. They figured that just like the two tramps in Beckett's play, all C64 users had been hopefully waiting for a graphic program, which until now has never arrived. Why not name the program after the guy the tramps were waiting for: Godot? Thus for C64 fans, the waiting is over. GoDot has finally arrived!

Wolfgang's program concept began with the transformation of every graphic file into the 4-Bit Format. Almost every other typical graphic editing module uses the same GoDot concept. Godot is so flexible, that it can then create all other formats. Arndt designed the graphic environment and hardware connections based on Wolfgang's graphic ideas. Arndt and Wolfgang developed GoDot's environmental design by basing their programming on the (then) standard Amiga program Art Department Professional and its GUI. The entire environment was programmed in only 700 bytes! The entire file handling is contained in another 700 bytes. The rest of the program consists mainly of graphic routines.

GoDot allows the user of many graphic programs and platforms the ability to import, edit, and export graphics from and to each other. Every type of graphic file finds support in GoDot; Commodore 64, Amiga and PC. Those of you that own and work with another type of computer along with your C64/128 will appreciate its import and export capabilities. True graphic freaks will go wild with GoDot's editing functions. This baby can do everything from masking to digitizing!

The program supports scanning and color print out and new modules hit the market on a regular basis. Wolfgang is currently working on a full screen animation module and Arndt is working on adapting the program to work with CMD hardware. The graphic environment is written in English and the handbook will be translated by the US distributor.

Unfortunately, when and where the program will be available in the States has not been determined yet, but we will certainly keep you informed as soon as we know the details. You can expect to hear more about GoDot in this column. Possibly the authors themselves could write a few columns describing the advanced techniques that the program has to offer.

Getting Ready for MACHINE LANGUAGE

by Jim Butterfield

Before diving into the workings of machine language programs, let's look at a few related topics. Some of them will clarify terminology, others will help you get ready.

What is Machine Language?

Machine language is the *only* program code that runs on your computer. It consists of simple instructions that run at lightning speed. BASIC seems to run on your computer. But what's really happening is that a machine language program (the "interpreter") is whizzing along, picking up the code from BASIC statements and performing whatever actions the BASIC code specifies. BASIC doesn't do it, the machine language "BASIC interpreter" does the actual work.

Other languages (for example, "C") may be used to program your computer. But C or Fortran or whatever doesn't run the code, (written by the programmer, often called the "source" program) it must be translated. The translator, usually called a "compiler", produces a machine language program. So in the final analysis, it's all machine language.

Machine Language vs. Assembly Language

Machine language is the stuff that actually runs in your machine. A related term, assembly language, talks about code that's very close to machine language. A program coded in assembly language needs to be translated, or "assembled", into machine language.

In our series here, we'll talk in "assembler" terms. For example, to decrement the value in the X register by 1, the machine code is hexadecimal CA. Don't worry what all this means yet, we'll get to registers and hexadecimal later. But it's easier for me to call that instruction DEX, which sounds and looks much more like Decrement X. DEX is called a "mnemonic", easy to remember; it will translate easily to its real machine language value.

Simple assembler programming is extremely close to machine language in style and size. More complex assemblers introduce other things that might confuse the beginner: macros, conditional assembly, and even libraries and relocating linkers.

We'll keep our attention carefully focused on machine language. We'll use only the simplest kind of assembler, the one that comes as part of your machine language monitor (MLM).

Why Learn Machine Language?

You may be just curious. But there are three good reasons for learning machine language. The most obvious is that machine language programs run incredibly fast. Such a program will speed up your computer and impress your friends.

Another good reason is that machine language programs have no built-in limitations. For example, BASIC doesn't permit such things as strings with length greater than 255, or twenty-digit numbers. More subtly, you can have only one BASIC program in memory at a time. Machine language doesn't have such limitations. If you want it, you can program it.

Perhaps the most important reason for learning machine language programming is the insight it gives you into the workings of the computer. You get to handle the fabric of the machine itself; the memory, the input/output mechanisms, even the pre-coded ROM routines. Even if you never code a line of machine language, it can help you understand how things work.

Who Can Learn Machine Language?

Anyone. You don't need to be a super brain to write M/L. Nor do you need to know advanced math or electronics. It's useful to have an orderly mind and a little patience. Each machine language instruction does a small task and you need quite a few of them for most jobs. So you must be prepared to work your way through a fair number of instructions. It will help you to keep them orderly.

But there's even room for wild-eyed "spaghetti coders" who dive in and write instructions at a furious pace with no advanced planning. Their programs may "crash out" the first hundred attempts, but with perseverance, they will get there.

What Tools Will I Need?

Bring along your brain. That's the most important tool of all. You will need a MLM - a machine language monitor program. There's one of these built into every 128. Typing the command MONITOR will take you there (typing X gets you back to BASIC). On the Commodore 64, you'll need to load an MLM program, such as SuperMon. A number of these types programs are available for free in the public domain. If you don't have one, check with your user group or BBS system. There are also some commercial sources, such as CMD's JiffyMon (requires JiffyDOS), and the built-in monitors in the Super Snapshot and Action-Replay cartridges.

Reference books are good, but they are getting hard to find. "Commodore 64 Programmer's Reference Guide" was published in 1982 in conjunction with Howard W. Sams; "Commodore 128 Programmer's Reference Guide" in 1986 by Bantam Books. Either one is a gold mine of useful data. For those who just can't find anything, we'll be publishing occasional reference material here as needed.

You don't need to buy a full assembler (a "symbolic" assembler) yet. It will be useful in the future, but we'll do all our work using the tiny assembler built into the MLM. On the other hand, if you have trouble locating an MLM program, or want to do some more extensive experimenting on your own, an assembler might be a good idea.

Can I Read Ahead?

By all means. There are a couple of areas where some study will be especially useful: binary/hexadecimal numbers, and system peeks and pokes.

In your computer's memory, there are "electronic switches" that turn on and off called "bits". The word "binary" means "based on two", and your binary computer is full of these bits that have only two states: on or off, 1 or 0. Bits are grouped together, eight at a time, into "bytes". So we might describe the contents of the eight bits in a certain byte this way: 01000001. You can see that two of the bits are "on", and the remaining six are "off". The value is a binary number. This is often signaled with a leading "%" character, thus, %01000001. This combination of bits might represent the letter A, or it could be used as the value 65; or it might even be a machine language instruction. By the way, that right-hand ("low order") bit is called bit number 0 within the byte. Counting right to left, the left-hand ("high order") bit will be number 7. In the above example, we could say that bits 0 and 6 are "on" (or "set"), while the others are "off" (or "clear").

It's awkward (and boring) to write the contents of a byte as eight binary values, and it gets worse when you need to write two-byte values, such as %1101000000100001. To help this situation, a method of grouping bits together, four at a time, has been devised: hexadecimal. In hexadecimal code, %1101 would be written as "D", %0000 as "0", %0010 as "2" and %0001 as "1". Prefixing these digits with "$" to indicate hexadecimal, we could write the above sixteen-bit number as $D021. Same number, just a different way of writing it down that's much more compact, and errors are less likely to occur when you transcribe numbers in "hex".

Read up on binary and hexadecimal numbers. It's useful to know the methods which allow you to translate a number between the two systems, and to or from decimal. And remember that in machine language terminology, a program that starts at $801 isn't expensive, that's just the address (in "hex") where the program begins.

PEEK and POKE

The BASIC commands PEEK and POKE allow you to view the contents of selected bytes of memory. There are similar commands in machine language, of course. My main message today is: look carefully at lists of useful PEEK and POKE addresses. There are addresses for the screen, addresses for sound, addresses for testing the keyboard. We'll use these addresses and more when we start coding machine language.

Do you want to change the screen background color? In BASIC, you know it's a POKE to address 53281. In machine language, we'll do a "store" to the same address, although we might supply that address in hexadecimal. We will be identifying the important addresses as you need them. But it never hurts to read ahead.

Machine Language Skills

When you learn machine language, you learn at least three important skills. Some of these skills will be useful even if you need to deal with other types of computers.

First, you learn the mechanics of machine language coding, which commands you use to add, to store values into memory, to test for a condition, and even to stop. Second, you'll learn how to use the tools that come with machine language. How do you use a Machine Language Monitor program, which on other computers may be called a Debugger? In using it, how do you track down bugs in your program? And you'll learn how an simple assembler works. Finally, you'll discover more about the architecture of your machine. For example, how it performs input and output, and the means of reaching disk files. You'll have a better understanding of the working of your computer than ever before. And you can even look into ROM code to see how the computer goes about certain jobs.

Get ready for a fun ride. There's challenge and delight in putting together your own machine language program.

A BEGINNERS GUIDE TO MACHINE LANGUAGE ELEMENTS

Jim Butterfield

Part 1: The Processor Registers

The 6510 & 8502 microprocessors are low-cost computing devices based on the popular 6502. The 6510 is the heart of the Commodore 64, while the more recent 8502 can be found in Commodore 128 computers.

Understanding the microprocessor used in a specific computer system is the key ingredient to gaining proficiency at programming the system in machine language. Since the 65xx family of microprocessors are register-based devices, learning what the registers are and how they can are used to process information and control program flow is a logical starting point.

Microprocessor registers are similar in many ways to the memory in your computer. Information can be stored in registers for later retrieval. The similarities end here, however, as the registers are also capable of being used to perform a wide range of operations including mathematical calculations, comparisons, and bit manipulations. Other control and logic elements within the microprocessor are brought into play by the various instructions, but it's the registers where you'll place your data—and it's the registers where you'll get the results.

The following paragraphs will describe the registers in the 65xx family of microprocessors. Different registers have different purposes; some have general applications, while others have specific purposes. You'll discover this yourself later when we examine the instruction set, but you'll grasp those instructions quicker by getting a better understanding of the registers first.

Accumulator (A). The Accumulator is known as the "A" register, and it's a good place to perform addition and subtraction. It's an eight-bit register, which means it holds values ranging from zero to 255 (hexadecimal $00 to $FF). Since most of the instructions can work with the Accumulator (and many of them actually require it), this register is the one most commonly used within machine language programs.

Index Registers (X and Y). The X and Y registers are called "index" registers because the values that they hold can be used by some instructions to adjust (or "index") computer memory addresses. However, X and Y are also good registers for holding data. Like the A register, X and Y can load their data from memory, store their data to memory, or compare the data. And, like A, they are eight-bit registers.

Program Counter (PC). This special-purpose register indicates where the next program instruction will come from. In other computers, it might be called the IP (instruction pointer). This pointer generally takes care of itself, tracking along from one address in memory to the next, or switching to a new value when a jump, branch, or subroutine call is invoked. The PC can reach the whole 64K of memory, so it's a sixteen-bit register, holding values from 0 to 65535 (hexadecimal $0000 to $FFFF).

Stack Pointer (SP). The Stack Pointer helps keep track of housekeeping values. Such values are used, for example, to note the return address of a subroutine call. All values are stored in page 1, the memory area from 256 to 511, hexadecimal

0100 to 01FF; the stack pointer says where the data is being stored within that area. The stack generally takes care of itself. If your program wants to use it directly, it should be neat and careful. The stack pointer is eight bits in size, selecting from a range of 256 locations.

Processor Status Register (P). The Processor Status Register holds seven items of information called "flags". Four of these flags provide information about recent data operations. Using certain instructions, your program can test the condition of these flags and use that information to control program flow. The four flags which can be tested in this manner are the *Negative flag (N)*, *Carry flag (C)*, *Zero flag (Z)*, and the *Overflow flag (V)*. The other three flags (there are a total of seven) control or signal the way the processor is working. These are the *Interrupt flag (I)* , the *Decimal flag (D)*, and the *Break flag (B)*. You'll make use of the Processor Status register mostly by means of "branch" instructions, which test flags. Each of the seven flags are a "bit" that can be on or off; these are all held within an eight-bit byte.

Part 2: The Instruction Set

Now that you're familiar with the registers, the next step is to get to know the command set. Naturally, you won't remember all of this information—even experienced machine language programmers have to check references now and then. Thus, in addition to this text we've included a handy Instruction Set reference chart at the end of this article, which provides the full list of instructions and addressing modes.

What you should try to focus on as you read the descriptions in this section is how the individual instructions fall into categories. If you strive to understand the categories, the instructions themselves will come naturally as you combine this with your knowledge of the registers.

Here, then, are the categories, the general rules that apply to the instructions within.

Commands that copy data into registers. Two flags are affected: the Z flag is set if the resulting register value is zero, otherwise it's clear; the N flag is set if the high bit (bit 7) of the register is set, otherwise it's clear.

- LDA Load A
- LDX Load X
- LDY Load Y
- TAX Transfer (copy) A to X
- TAY Transfer A to Y
- TXA Transfer X to A
- TYA Transfer Y to A
- TSX Transfer the Stack Pointer to X

Commands that copy data from registers. These commands do not affect any flags.

- STA Store A to memory
- STX Store X to memory
- STY Store Y to memory
- TXS Transfer (copy) X to the Stack Pointer

Commands that manipulate the individual bits in the A register. Each bit is matched against the corresponding bit of the operand. Flags Z and N are set based on the resulting value in A (see group A instructions, above). These "logical operations" take a little getting used to, but are useful.

- AND Logical AND. Set "1" if both corresponding bits were "1", otherwise "0".
- ORA Logical OR. Set "0" if both matching bits were "0", otherwise "1".
- EOR Logical Exclusive-OR. Set "0" if the matching bits were alike, otherwise "1".

Arithmetic operations. The A register is normally used, although the shift/rotate instructions may act upon a memory location instead. The C, Z, and N flags are affected; ADC and SBC also affect the V flag.

- ADC Add, with carry.
- SBC Subtract (with borrow, an "inverted" carry).
- ASL Arithmetic shift left. All bits move up one place.
- LSR Logical shift right. All bits move down one place.
- ROL Rotate Left, via the Carry flag. Bits move up/around.
- ROR Rotate Right, via Carry. Bits move down and around.

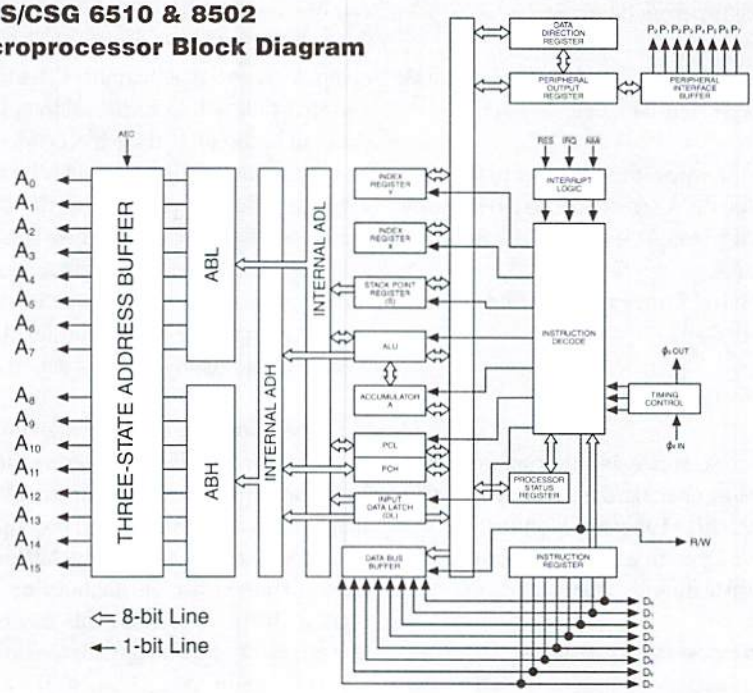
Comparison instructions. They do not affect the contents of any data register, but will set values of the C, Z, and N flags depending on the result of the comparison. The BIT instruction also affects the V flag.

- CMP Compare the A register with a value.
- CPX Compare the X register.
- CPY Compare the Y register.
- BIT Perform a curious BIT test, often on a data port.

Branch instructions. These test a selected flag, and cause the program to branch to a new address depending on what is found. Branch instructions can take only "short hops" so they sometimes need to work together with a JMP instruction.

- BCC Branch if Carry flag Clear.
- BCS Branch if Carry flag Set.
- BEQ Branch if Equal (if Z flag set).
- BNE Branch if Not Equal (if Z flag clear).
- BMI Branch if Minus (if N flag set).
- BPL Branch if Plus (if N flag clear).

MOS/CSG 6510 & 8502
Microprocessor Block Diagram



BVC Branch oVerflow Clear (if V flag clear).
BVS Branch oVerflow Set (if V flag set).

Counting instructions. The contents of registers X or Y or a selected memory location are increased or decreased by one. The Z and N flags are affected according to the resulting value.

DEC Decrement memory.
DEX Decrement the contents of X.
DEY Decrement the contents of Y.
INC Increment memory.
INX Increment the contents of X.
INY Increment the contents of Y.

Instructions which set or clear individual flags in the Status Register.

SEC Set the Carry flag.
CLC Clear the Carry flag.
SED Set Decimal Arithmetic; affects ADC and SBC only.
CLD Clear the Decimal Arithmetic flag.
SEI Set the Interrupt-disable flag.
CLI Clear the Interrupt-disable flag.
CLV Clear the oVerflow flag.

Instructions for changing the address at which instructions are being executed. No flags are affected except for instruction RTI.

JMP Jump to the address given.
JSR Jump to a SubRoutine at the address given.
RTS Return from Subroutine.
RTI Return from Interrupt.

Instructions to put ("push") a byte of data onto the stack, or pull a byte of data from the stack.

PHA Push (copy) the A register to the stack.
PLA Pull the A register data from the stack.
PHP Push (copy) the Processor Status to the stack.
PLP Pull the Processor Status from the stack (all flags).

Miscellaneous instructions.

BRK Break, create an interrupt condition. Most often used to return you to your Machine Language Monitor.
NOP No Operation. Do nothing, other than waste time (2 cycles per).

Part 3. Addressing modes

Many instructions need data, which often comes from memory. When you write a program in machine language, you'll either provide this data after the instruction, or you'll supply an address that the microprocessor can interpret in order to find the data. Interpreting where the data comes from is the job of the addressing mode. Here's an example to help illustrate this:

```
LDA $2000
```

The above example tells the microprocessor to read the contents of memory location $2000, then copy that value into the Accumulator. This example, by the way, isn't actually machine language per se, but *Assembly Language*. What's the difference? Well, here's the same example, but this time in true machine language:

```
AD 00 20
```

Those are hexadecimal numbers, by the way. See the difference? Computers only understand numbers, and if you run the first example (*source code*) through an assembler, it will generate what you see in the second example (*object code*). Most machine language programmers will write in Assembly Language—it's just simpler.

Looking at the source code example, you'll see the LDA instruction, followed by an address of $2000. This address (also called an *argument*), is the location where the data can be found. How does the computer know that this is an address and not the data itself? In Assembly Language, the syntax of the argument determines this.

Looking now at the object code example (which is what the computer will actually operate on), you can see that the *Absolute* addressing mode form of the LDA instruction is present. Thus the computer knows that two more bytes will follow, and that they will form the address where the data can be found. It also expects the address to be supplied in low-byte/high-byte format.

In the following paragraphs, the addressing modes will be described, and we'll give you an example of the Assembly Language syntax used for that particular addressing mode by showing a sample source code instruction for that mode. Again, use our handy chart for details.

Implied. No address (argument) required, since the instruction itself implies where the data can be found. Check the chart for the instructions that use this addressing mode. [example: PHA]

Accumulator. The rotate and shift instructions may operate either on memory or on the A register. If the A register, this mode is called 'Accumulator addressing'. Only four instructions use this mode: ASL, LSR, ROL, and ROR. [example: ASL A or just ASL]

Relative. Branch instructions test a flag. Depending on the result of this test, program execution may move (branch) by up to 127 bytes forward, or go backwards up to 128 bytes. There are eight branch instructions: BCC, BCS, BEQ, BNE, BMI, BPL, BVC, and BVS. [example: BNE $2000]

Immediate. Within the instruction, the actual data byte is supplied, rather than a memory address. Most data instructions other than "store" can use this mode. [example: LDA #$01]

Absolute. A two-byte address specifies the data address in memory. This is used by many instructions: [example: STA $2000]

Zero Page. When the address in memory is located within the popular page zero (address 0-255, or hex $00 to $FF), one byte can specify the location. This mode is widely available. [example: LDA $BA]

Absolute, X-indexed. The contents of X will be added to the absolute address supplied by the instruction. That gives the instructions a "range" of 256 bytes to reach. [example: LDA $2000,X]

Zero page, X-indexed. Same as above, but the supplied address is one byte, and the resulting adjusted address will be in zero page. [example: LDA $B8,X]

Absolute, Y-indexed. The address will be adjusted by the contents of the Y register. [example: LDA $2000,Y]

Zero page, Y-indexed. The one-byte address is adjusted by the contents of Y. Only two instructions use this: LDX and STX. [example: LDX $B8,Y]

Indirect. The JMP instruction most often uses an absolute address, which specifies where to jump. But it occasionally uses an "indirect" address, an address where it will get the real jump address. JMP is the only instruction that can use this addressing mode. [example: JMP ($2000)]

Indirect, Y. A two-byte location in zero page is specified where the base address may be found. The contents of the Y register is added to this base address. This addressing mode is a popular way to reach a large range of memory; its use is limited to eight instructions: LDA, STA, AND, ORA, EOR, ADC, SBC, CMP. [example: LDA ($90),Y]

X, Indirect. A zero-page location is indexed by X; the result points to a two-byte address which tells which memory location we want. This addressing mode is not often used in computer programs, but it's popular when used in control devices such as disk drives. Only a few instructions can use this addressing mode: LDA, STA, AND, ORA, EOR, ADC, SBC, and CMP. [example: LDA ($90,X)]

MOS/CSG 65xx/75xx/85xx Family Microprocessor Instruction Set

Mnemonic	Operation	Immediate		Absolute		Zero Page		Accumulator		Implied		(Indirect,X)		(Indirect),Y		Zero Page,X		Absolute,X		Absolute,Y		Relative		Indirect		Zero Page,Y		Flags					
																												N	Z	C	I	D	V
ADC	A+M+C→A (1)(4)	69	2	6D	4	65	3					61	6	71	5	75	4	7D	4	79	4							•	•	•	-	-	•
AND	A∧M→A (1)	29	2	2D	4	25	3					21	6	31	5	35	4	3D	4	39	4							•	•	-	-	-	•
ASL	C←[7 ... 0]←0			0E	6	06	5	0A	2							16	6	1E	7									•	•	•	-	-	-
BCC	BRANCH ON C=0 (2)																					90	2					-	-	-	-	-	-
BCS	BRANCH ON C=1 (2)																					B0	2					-	-	-	-	-	-
BEQ	BRANCH ON Z=1 (2)																					F0	2					-	-	-	-	-	-
BIT	A∧M			2C	4	24	3																					M_7	•	-	-	-	M_6
BMI	BRANCH ON N=1 (2)																					30	2					-	-	-	-	-	-
BNE	BRANCH ON Z=0 (2)																					D0	2					-	-	-	-	-	-
BPL	BRANCH ON N=0 (2)																					10	2					-	-	-	-	-	-
BRK	FORCE BREAK									00	7																	-	-	-	1	-	-
BVC	BRANCH ON V=0 (2)																					50	2					-	-	-	-	-	-
BVS	BRANCH ON V=1 (2)																					70	2					-	-	-	-	-	-
CLC	0→C									18	2																	-	-	0	-	-	-
CLD	0→D									D8	2																	-	-	-	-	0	-
CLI	0→I									58	2																	-	-	-	0	-	-
CLV	0→V									B8	2																	-	-	-	-	-	0
CMP	A-M (1)	C9	2	CD	4	C5	3					C1	6	D1	5	D5	4	DD	4	D9	4							•	•	•	-	-	-
CPX	X-M	E0	2	EC	4	E4	3																					•	•	•	-	-	-
CPY	Y-M	C0	2	CC	4	C4	3																					•	•	•	-	-	-
DEC	M-1→M			CE	6	C6	5									D6	6	DE	7									•	•	-	-	-	-
DEX	X-1→X									CA	2																	•	•	-	-	-	-
DEY	Y-1→Y									B8	2																	•	•	-	-	-	-
EOR	A∀M→A (1)	49	2	4D	4	45	3					41	6	51	5	55	4	5D	4	59	4							•	•	-	-	-	-
INC	M+1→M			EE	6	E6	5									F6	6	FE	7									•	•	-	-	-	-
INX	X+1→X									E8	2																	•	•	-	-	-	-
INY	Y+1→Y									C8	2																	•	•	-	-	-	-
JMP	JUMP TO NEW LOC			4C	3																			6C	5			-	-	-	-	-	-
JSR	JUMP TO SUB			2D	6																							-	-	-	-	-	-
LDA	M→A (1)	A9	2	AD	4	A5	3					A1	6	B1	5	B5	4	BD	4	B9	4							•	•	-	-	-	-
LDX	M→X (1)	A2	2	AE	4	A6	3													BE	4					B6	4	•	•	-	-	-	-
LDY	M→Y (1)	A0	2	AC	4	A4	3									B4	4	BC	4									•	•	-	-	-	-
LSR	0→[7 ... 0]→C			4E	6	46	5	4A	2							56	6	5E	7									0	•	•	-	-	-
NOP	NO OPERATION									EA	2																	-	-	-	-	-	-
ORA	A∨M→A	09	2	0D	4	05	3					01	6	11	5	15	4	1D	4	19	4							•	•	-	-	-	-
PHA	A→M_S S-1→S									48	3																	-	-	-	-	-	-
PHP	P→M_S S-1→S									08	3																	-	-	-	-	-	-
PLA	S+1→S M_S→A									68	4																	•	•	-	-	-	-
PLP	S+1→S M_S→P									28	4																	(RESTORED)					
ROL	C←[7 ... 0]←C			2E	6	26	5	2A	2							36	6	3E	7									•	•	•	-	-	-
ROR	C→[7 ... 0]→C			6E	6	66	5	6A	2							76	6	7E	7									•	•	•	-	-	-
RTI	RETURN FROM INT									40	6																	(RESTORED)					
RTS	RETURN FROM SUB									60	6																	-	-	-	-	-	-
SBC	A-M-$\bar{C}$→A (1)	E9	2	ED	4	E5	3					E1	6	F1	5	F5	4	FD	4	F9	4							•	•	(3)	-	-	•
SEC	1→C									38	2																	-	-	1	-	-	-
SED	1→D									F8	2																	-	-	-	-	1	-
SEI	1→I									78	2																	-	-	-	1	-	-
STA	A→M			8D	4	85	3					81	6	91	6	95	4	9D	5	99	5							-	-	-	-	-	-
STX	X→M			8E	4	86	3																			96	4	-	-	-	-	-	-
STY	Y→M			8C	4	84	3									94	4											-	-	-	-	-	-
TAX	A→X									AA	2																	•	•	-	-	-	-
TAY	A→Y									A8	2																	•	•	-	-	-	-
TSX	S→X									BA	2																	•	•	-	-	-	-
TXA	X→A									8A	2																	•	•	-	-	-	-
TXS	X→S									9A	2																	-	-	-	-	-	-
TYA	Y→A									98	2																	•	•	-	-	-	-

Instruction Values (left) & Execution Time (right) per Addressing Mode

(1) ADD 1 CYCLE IF PAGE BOUNDARY IS CROSSED
(2) ADD 1 CYCLE IF BRANCH OCCURS TO SAME PAGE
ADD 2 CYCLES IF BRANCH OCCURS TO DIFFERENT PAGE
(3) CARRY NOT = BORROW
(4) IF IN DECIMAL MODE, Z FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X INDEX X
Y INDEX Y
A ACCUMULATOR
M MEMORY PER EFFECTIVE ADDRESS
M_S MEMORY PER STACK POINTER

\+ ADD
\- SUBTRACT
∧ AND
∨ OR
∀ EXOR

• MODIFIED
\- NOT MODIFIED
M_7 MEMORY BIT 7
M_6 MEMORY BIT 6
1 FLAG VALUE 1
0 FLAG VALUE 0

A Machine Language Program for Beginners

Jim Butterfield

Let's write a simple machine language program—one that isn't too hard, but will demonstrate the dazzling speed that's possible. You can use an assembler, or you could enter the program via your Machine Language Monitor (MLM). You could even do the whole thing on paper, and enter the final program byte by byte, but I doubt you'll want to go that far.

Here's our program objective: read a key from the keyboard and echo it to the screen as a full line of characters. On a 40 column screen, that's 40 characters out for each character input.

System calls

To read the keyboard, we'll call the GET subroutine at hexadecimal $FFE4. To print to the screen, we will use the CHAROUT subroutine at $FFD2 (the dollar symbol indicates hexadecimal).

Master plan: this could be drawn as a logic outline:

START: Read a key by calling GET;
  if it's the RETURN key, quit;
  if it's "no key", go back to START;
Set a counter to zero, to prepare for looping;
LOOP: print the key to the screen by calling CHAROUT;
Add one to the counter;
Check to see if we've reached 40 (or 80);
If not, go back to LOOP and do more;
If we have reached 40/80, go back to START.

Assembly Code

You'll need to use an editor to prepare your "source code". After this is complete, the assembler will take what you have written and translate it to machine language.

We should start with some comments, and continue by defining constant values to be used in the program:

```
; THIS IS ONLY A COMMENT
; SPEED WRITING PROGRAM
; TYPE YOUR NAME, DATE,
; WHATEVER, HERE
COLMS     EQU  4Ø
; change 4Ø above to 8Ø if
; desired
GET       EQU  $FFE4
CHAROUT   EQU  $FFD2
```

We could just pop the appropriate values into the program as we went, but the above "equate" commands make for easier reading and updating if we decide on a change.

We must choose a location where our finished program will reside in memory. Hex 2000 isn't a perfect spot, but that address is available on many Commodore 8-bit machines, including the Commodore 64 and 128. Most assemblers use an asterisk character to signal the current "working point". We'll tell the assembler to site the code at hexadecimal 2000 with:

```
*         =     $2ØØØ
```

On to the program instructions. At location START, we must read the keyboard. We do this by using instruction JSR, Jump Subroutine, and the address of the subroutine is noted above and assigned to the symbol GET. Here's the code:

```
START:    JSR  GET
```

When the system subroutine returns control to our program, we will either have a key in the A register, or there will be a binary zero there. The key will be in ASCII code; that's fine by us, since it will be perfect for printing in a moment.

Check our program flow outline (above) and we see that the program must next test to see if RETURN has been pressed by the user. Aha! A comparison is needed, and instruction CMP is ideal for checking the contents of the A register. We're looking for the RETURN character that's hexadecimal 0D.

Using "immediate" mode addressing, the program can check for value $0D rather than the contents of a memory address. The "#" symbol is often called an "octothorpe." No matter what

you call it, the assembler will see it as a request for immediate mode addressing. So here's the comparison, followed by a branch-if-equal:

```
CMP  #$ØD
BEQ  EXIT
```

We could read the above two instructions as "Compare the contents of A with the immediate value hex 0D, and branch if equal to location EXIT." We have not yet defined where the code for EXIT will be, we must remember to do that.

The above Compare instruction didn't just test for equal. It also tested for less-than, and set the C (carry) flag according to the results of that test: C will be clear if the value in A is less than $0D, else C will be set. Now: if we didn't branch to EXIT, the C flag is still there and available for testing. And the flag will be clear if the A register contains a value from hex 00 to 0C inclusive.

None of the values in the range hex 00 to 0C represent printable characters. And a value of 00 represents "no key", which needs special handling. It looks like it's safe to do a "branch less than" action.

```
BCC  START
```

Setting Up The Loop

The program flow indicates that we need to count our output characters. Register X or Y would be ideal for this kind of thing; we could use the INX or INY (increment-X or -Y) commands to count. Either will do, so I'll pick X. First, we must set the count to zero, so we load X with immediate value zero:

```
LDX  #Ø
```

Now we can loop and count. After we mark the loop point, we must print the key in the A register with a subroutine call to $FFD2, CHAROUT:

```
LOOP:    JSR  CHAROUT
```

A wonderful thing about the CHAROUT subroutine is that it does not disturb any of our three data registers (A, X, and Y). That is not true of other system calls. But in this case, it means that we can expect that our ASCII character in A and our counter in X will still be there.

Our next task is to add one to our counter, and check to see if we have reached the limit:

```
INX
CPX  #COLMS
```

Note that "#" symbol for immediate mode. If we had forgotten to use it, our program would have checked the contents of address 40, rather than the value 40 itself! The program should continue looping if the count has not reached 40; BNE means "Branch Not Equal":

```
BNE  LOOP
```

If we have reached 40 (or 80, if that's what you chose) we want to loop back to start. Well, if we didn't take the branch with BNE, Branch Not Equal, we're guaranteed to take it with BEQ (Branch Equal); both instructions check the same Z flag. Thus:

```
BEQ  START
```

There's one loose end to tie up. Somewhere above, we branched to code at location EXIT. We'd better make such a location containing the appropriate instruction:

```
EXIT:    RTS
```

RTS means "Return from Subroutine" In this case, it will take us back to the calling program, which is Basic. Later, we'll use the Basic SYS command, to call our program as a subroutine, giving us an easy way back.

Assembly and Implementation

If you have an assembler, the above code will translate easily into a machine language module. That in turn can be loaded into memory (remember to use LOAD "..",8,1) and then triggered with the Basic command: SYS 8192. Decimal value 8192 is the same as hexadecimal $2000. When the program first starts, there will be a scary moment when nothing happens, just touch a key to see the action.

If you don't have an assembler, you can still do the job using a Machine Language Monitor. Load one into your 64, or command MONITOR on your 128.

Using the Machine Language Monitor

A simple MLM assembler won't take symbols, and many of them insist that you supply all numbers in hexadecimal. To do the job, go into the monitor and prepare to type a first line that starts A 2000. After you have typed each line and press RETURN, you may be surprised by the screen changes, but it won't take long to get used to it. Here's the program:

```
A 2ØØØ JSR $FFE4   (read keyboard)
A 2ØØ3 CMP #$ØD    (is it RETURN?)
A 2ØØ5 BEQ $2Ø15        (yes, exit)
A 2ØØ7 BCC $2ØØØ   (less, try again)
A 2ØØ9 LDX #$ØØ    (counter=0)
A 2ØØB JSR $FFD2 (LOOP, print it!)
A 2ØØE INX              (count it!)
A 2ØØF CPX #$28     (reached 40?)
A 2Ø11 BNE $2ØØB   (no, print more)
A 2Ø13 BEQ $2ØØØ   (yes, start over)
A 2Ø15 RTS        (EXIT, back to Basic)
```

We must make a separate note of where LOOP is located, and will likely have to guess the address of EXIT. When we get there, we can go back and fill in the proper address. We must give our column count in hexadecimal, so decimal 40 becomes $28. For an 80 column machine, you would use $50.

Exit to Basic (MLM command X) and give the Basic SYS 8192 command. Then try typing, ending with RETURN.

The Machine Language

If you look at the program itself as it lies in memory, you'll see a stream of bytes such as: 20 E4 FF C9 0D. These 22 bytes are the real program. You can save the bytes or peek and poke them. If you want to see the program while in the MLM, you could command D 2000. And you can impress your friends with your speed typing.

ASSEMBLY LINE

SERIAL DEVICE KERNAL ROUTINES

by Doug Cotton

Of all the machine language routines I've ever written, the ones which were most difficult to write and debug were undoubtedly those which had to access serial bus devices. Even now, when I find I need to access a drive in ML, I have to go back and check my old programs or dig out a stack of reference books to remind myself of how everything works.

Certainly I can't be the only one who runs into this—so I've prepared a chart and some routine outlines for publication here. The chart brings together information from a number of different references, as well as some info which you might be hard-pressed to find anywhere else. I'll also tackle explaining how to use the routines, but this will come in either two or three parts. We'll begin with an overview, and also cover the "high-level" routines in this first installment.

Routine Levels and Vectors

As you look at the chart, you'll notice that the available Kernal routines have been divided into two groups: low-level and high-level (there's also a separate category for calls unique to the 128. So what's the difference?

The main purpose of the high-level calls is to simplify the process of working with devices—they let you accomplish your tasks with a little less code. In addition, you should always keep in mind that certain devices may not support the direct use of the low-level Kernal routines. This is because most of the important high-level routines are *vectored* through RAM locations, while the low-level routines are not.

Some devices—most often those which use special interfaces to attach to your computer—can only function with your computer through the use of vectors. The Xetec Leutenant Kernel hard drive is one example of a device which requires the use of high-level Kernal routines. If compatibility with a wide range of devices is important to you, don't use the low-level routines directly in your programs.

By the way, if you're not familiar with what vectors are, take a look at the Kernal OPEN routine in the chart. Notice that the routine is called at $FFC0. But this is just a Kernal Jump Table location, and not the address at which the actual routine resides. Now, if we were to disassemble the code located at this address in the Commodore 128 Kernal ROM, we'd see the following:

```
JMP ($Ø31A)
```

This forms an indirect jump, which means the processor uses the two bytes stored beginning at location $031A to form the actual jump address. The defaults you'll find at $031A and $031B are $BD and $EF, respectively, which form the address $EFBD, the true entry point for the Kernal OPEN routine. By changing the bytes at locations $031A and $031B, any programmer can redirect what happens when OPEN is called. In turn, we refer to $031A as the OPEN routine vector.

So, getting back to the discussion at hand, why use the low-level routines at all? Well, you may find it easier to get specific jobs done with the low-level routines in certain situations. There's also a slight speed advantage to the low-level routines, and when you're optimizing a routine to squeeze the most performance out of every last cycle—low-level routines can help. Finally, the low-level routines can help you bypass the limitations on the number of files you can have open when dealing with devices with extensive file-handling capabilities such as the CMD HD hard drive. It all comes down to a matter of knowing when it is or isn't appropriate to use them, though. As an example, I personally wouldn't use them in a BBS program, where compatibility with a wide range of devices is an important factor; on the other hand, I wouldn't hesitate to use them when writing a partitioning utility for RAMLink or a CMD HD hard drive because I know that these devices support the low-level calls.

Avoiding Problems

Serial device routines always seem to be a magnet for code problems, at least in my experience. They can also be tough to debug, because many of the mistakes that can be made will lead to odd behavior as opposed to outright failure. The best way to avoid such problems is to exercise great care when writing the routines. I've collected a few tips here to help you stay out of trouble.

Follow the correct order of events. Pay close attention to the information given in the chart—especially the information concerning pre-requisites. In particular, call CLRCH *before* you call CLOSE, and clear the STATUS byte before calling LISTN or TALK. And whatever you do, make sure every TALK or LISTN gets an UNTLK or UNLSN before you get another talker or listener going; if you don't, you'll end up with routines that will almost always work, but will fail on somebody's system down the line. Making sure that everything in the right order will vastly increase your odds of getting routines that run correctly the first time.

Don't skip error checking. Be sure to check for errors when they can occur. Most of the Kernal serial device routines either return a value in the STATUS byte ($90) or in the .A register (see the chart for details). If you're using one of the routines that uses the latter method, the processor's Carry flag (.C) will be set to 1 if an error has occurred (use BCS or BCC to determine program flow when checking for these kinds of errors). Also, be sure to check for errors via the command channel after sending commands to a device. Consider what kind of errors could occur, check for them, and plan ahead on how to deal with them when they actually do crop up.

Don't use registers that get trashed by routines you call! This is a common error. If you're going to be using drive routines in the middle of a loop, make sure the register(s) you're using won't be affected by the routines you call. If there's any doubt, save your registers and use a loop variable in memory.

Serial Bus Device Kernal Routine Reference Chart

Kernal Routine	Jump Table Address	RAM Vector	Calling Parameters .A	.X	.Y	Returned Parameters .A	.X	.Y	Status	Pre-requisites
Low-Level Routines										
SECND	$FF93 (65427)	n/a	SA+$60[1]	~	~	-	-	-	ST	LISTN
TKSA	$FF96 (65430)	n/a	SA+$60[1]	~	~	-	-	-	ST	TALK
ACPTR	$FFA5 (65445)	n/a	~	~	~	DATA	+	+	ST	TALK,TKSA
CIOUT	$FFA8 (65448)	n/a	DATA	~	~	+	+	+	ST	LISTN,SECND
UNTLK	$FFAB (65451)	n/a	~	~	~	-	-	-	ST	(TALK,TKSA)
UNLSN	$FFAE (65454)	n/a	~	~	~	-	-	-	ST	(LISTN,SECND)
LISTN	$FFB1 (65457)	n/a	DEV	~	~	-	-	-	ST	CLEAR ST
TALK	$FFB4 (65460)	n/a	DEV	~	~	-	-	-	ST	CLEAR ST
High-Level Routines										
READSS	$FFB7 (65463)	n/a	~	~	~	STATUS	+	+	ST[2]	None
SETLFS	$FFBA (65466)	n/a	LFN	DEV	SA	+	+	+	n/a	None
SETNAM	$FFBD (65469)	n/a	FNLEN	FNAL	FNAH	+	+	+	n/a	None
OPEN	$FFC0 (65472)	$031A (794)	~	~	~	ERROR	-	-	.C	SETLFS,SETNAM[3]
CLOSE	$FFC3 (65475)	$031C (796)	LFN	-	-	ERROR	-	-	.C	(OPEN (CLRCH))
CHKIN	$FFC6 (65478)	$031E (798)	~	LFN	~	ERROR	-	-	.C	OPEN
CKOUT	$FFC9 (65481)	$0320 (800)	~	LFN	~	ERROR	-	-	.C	OPEN
CLRCH	$FFCC (65484)	$0322 (802)	~	~	~	-	-	+	n/a	(CHKIN,CKOUT)
BASIN	$FFCF (65487)	$0324 (804)	~	~	~	DATA	+	+	ST	(OPEN,CHKIN)
BSOUT	$FFD2 (65490)	$0326 (806)	DATA	~	~	ERROR	+	+	ST	(OPEN,CKOUT)
LOAD	$FFD5 (65493)	[$0330 (816)]	LV[4]	(SAL)	(SAH)	ERROR	(EAL)	(EAH)	.C	SETLFS,SETNAM[3]
SAVE	$FFD8 (65496)	[$0332 (818)]	SAP[5]	EAL	EAH	ERROR	-	-	.C	SETLFS,SETNAM[3]
GETIN	$FFE4 (65508)	$032A (810)	~	~	~	DATA	-	-	ST	(OPEN,CHKIN)
CLALL	$FFE7 (65511)	$032C (812)	~	~	~	-	-	+	n/a	(CHKIN,CKOUT)
128 Unique Routines										
SPIN_SPOUT	$FF47 (65351)	n/a	~	~	~	-	+	+	n/a	.C[6]
CLOSE_ALL	$FF68 (65384)	n/a	DEV	~	~	-	-	-	n/a	None
SETBNK	$FF68 (65384)	n/a	BA	FNBANK	~	+	+	+	n/a	None

Reference Chart Notes & Definitions

SYMBOL DEFINITIONS

- ~ No parameter required
- - Register is not preserved during operation
- + Register is preserved during operation

NOTES

1. Add $F0 to SA instead of $60 to open file, $E0 to close file
2. STATUS byte (ST) is not cleared unless current device is 2 (RS-232)
3. SETBNK also required for 128
4. 0 for LOAD (requires address in .X and .Y if SA=0, returns ending address in .X and .Y); non-zero for VERIFY (address not required)
5. Pointer to zero page location holding starting address in low byte/high byte format
6. Clear .C (CLC) to set fast serial input, set .C (SEC) to select fast serial output

VARIABLE DEFINITIONS

- .C = Processor Carry Flag
- ST = STATUS byte ($90)
- LFN = Logical File Number
- SA = Secondary Address
- DEV = Device Number
- BA = Bank for LOAD/SAVE/VERIFY (128 only)
- LV = LOAD/VERIFY Flag
- SAP = Starting Address Pointer
- FNBANK = Bank where filename for LOAD/SAVE/VERIFY is stored (128 only)
- FNLEN = Length of filename in bytes (0 if no name is required for an operation)
- FNAL = Filename Address Low
- FNAH = Filename Address High
- SAL = Starting Address Low
- SAH = Starting Address High
- EAL = Ending Address Low
- EAH = Ending Address High

ACCUMULATOR ERROR CODES

$01	Too Many Files
$02	File Open
$03	File Not Open
$04	File Not Found
$05	Device Not Present
$06	Not Input File
$07	Not Output File
$08	Missing Filename
$09	Illegal Device Number
$10	Illegal LOAD (past $FEFF) on 128

STATUS BYTE VALUES

$01	Print Time-out
$02	Input Time-out
$40	EOF (End Of File)
$42	Read past EOF
$FF	Device Not Present

FILE SECONDARY ADDRESSES

$00-01	Reserved for LOAD/SAVE
$02-0E	Input/Output Files
$0F	Command Channel

Don't close the command channel if you have other files open that you want to continue using. Closing the command channel causes all other open files on the same device to be closed as well!

Opening a File

Whenever you want to read data from a serial device, you'll have to open an input file. Likewise, writing to a file requires opening an output file. Oddly enough, the steps for opening an input file are the same as those used to open output files when the high-level commands are used; parameters at the end of the filename are used to specify whether the file will be input (,R) or output (,W), except when special secondary addresses are employed.

The command channel is one of these exceptions; it requires no filename and can be accessed as either input or output. The command channel, is used to send commands to your serial bus peripherals, and to check for and read back error messages after commands are sent.

Different secondary addresses must be used for each file opened on a device, and logical file numbers must be unique regardless of device.

Here are the steps you'll need to follow to open a file using the high-level commands.

1. Load .A with the Logical File number, .X with the device number, and .Y with the secondary address, then call SETLFS. (Note: SA=$0F for command channel)
2. Load .A with the length of the filename ($00 for opening the command channel with no filename), .X with the low byte of the filename location, and .Y with the high byte of the filename location, then call SETNAM. Now is a good time to make sure that the filename string (if required) is in place if you haven't already done so. (Note: You can send a command on the command channel when you open it by using the command for the filename string)
3. 128 only: Load .X with the bank number where the filename is located, then call SETBNK.
4. Call OPEN.
5. Check the Carry flag (.C) for an error and process if necessary.

Data Input

To get data from an opened file, you need to set that file as the current input file. This is done by using the Kernal CHKIN routine. Once this is accomplished, data can be read from the file by using either BASIN or GETIN. However, GETIN calls BASIN when serial bus input is requested, adding unnecessary overhead to the process; so I'd recommend just using the BASIN routine for serial bus device input.

Here are the steps you'll need to follow to input data from an opened input file:

1. Load .X with whatever Logical File number you assigned to the file you want to read data from, then call CHKIN.
2. Check the Carry flag (.C) for an error and process if necessary.
3. Call BASIN. One byte of data will be transferred from the input file to the Accumulator (.A).
4. Store the received data byte in a more permanent place in memory.
5. Check STATUS byte ($90) for an error and process if necessary.
6. Repeat steps 3 through 5 as necessary to input all of your data.
7. Call CLRCH to restore the default input and output files.

Step 4 bears further examination: you need to store your data somewhere as you input each byte. There are a number of different ways to approach this, since you can't be sure if the data is valid until you check the STATUS byte. You might store the data temporarily until you verify that the data is good. You might instead store the data where it is intended to go by using a pointer, then update the pointer only if the data is good. While checking for errors, you'll also have to watch for an EOF (End Of File) indication.

Data Output

To send data to an opened file, you need to do the following:

1. Load .X with whatever Logical File number you assigned to the file you want to send data to, then call CKOUT.
2. Check the Carry flag (.C) for an error and process if necessary.
3. Load the data byte you want to send to the file into the accumulator (.A), then call BSOUT.
4. Check STATUS byte ($90) for an error and process if necessary.
5. Repeat steps 3 and 4 as necessary to output all of your data.
6. Call CLRCH to restore the default input and output files.

Closing Files

Once all the hard stuff is over with, all you're left with is closing the file (or files). This is a simple matter of calling CLOSE, but before you proceed you should first make absolutely sure that the default I/O has been reset with CLRCH. If you followed the steps given above, and haven't branched out of the middle of one of your routines for reading or writing, then you should be okay.

Bear in mind that you still need to check for errors when closing files. The user may have removed the input disk, and an output disk will be at a point where it still has to write the last block of data and update certain file and BAM information. Also, save closing the command channel on a given drive until after all the other files on that drive have been closed.

Here are the steps you'll need to follow when closing files:

1. Load .A with the Logical File number of the file you want to close, then call CLOSE.
2. Check the Carry flag (.C) for an error and process if necessary.

One last note: don't bother with using the CLALL Kernal call, and certainly don't use this in place of closing files individually. CLALL can be used to make sure that the file table held in memory is completely clear, but it doesn't do everything that CLOSE does.

The Sample Program

To further illustrate the techniques described in this article, I've included a sample program that takes the form of a dedicated file copier.

This copier, as is, will only copy a sequential file named "TEST1" from device 8 to device 9. Of course, you can modify the source code to copy other files, or to use other devices. You'll see some of these modifications in further installments of this column.

One other thing you'll notice about the sample program is that no attempt has been made to determine reasons for errors if they should occur. Again, we'll cover this later; it was important to first familiarize you with the serial routines.

What's Coming?

We have learned where we need to check for errors, and the program has the appropriate checks in place. But to process the errors, we'll need to study what errors to look for at specific times, and that will take more room than I could steal in this issue. So look for that in a future installment.

We'll also get into those low-level routines, and we'll take a look at how we can use machine language drive routines to access files previously opened by BASIC. So stay tuned!

```
; SAMPLE DRIVE CODE PROGRAM
; USING HIGH-LEVEL KERNAL ROUTINES
;
            .ORG  $2000
            .OBJ  FILETEST.O
;
SETBNK      =     $FF68
SETLFS      =     $FFBA
SETNAM      =     $FFBD
OPEN        =     $FFC0
CLOSE       =     $FFC3
CHKIN       =     $FFC6
CKOUT       =     $FFC9
CLRCH       =     $FFCC
BASIN       =     $FFCF
BSOUT       =     $FFD2
;
ST          =     $90
;
START       LDA   $FFFD       ;CHECK TO SEE IF
            CMP   #$FF        ;COMPUTER IS 128
            BNE   +           ;BRANCH IF 64 MODE
            STA   MODE        ;ELSE STORE MODE
;
+           LDA   #$01        ;LOGICAL FILE #1
            LDX   #$08        ;DEVICE 8
            LDY   #$0F        ;SECONDARY ADDRESS 15
            JSR   SETLFS
            LDA   #$00        ;FILE LENGTH 0
            TAX               ;ZERO OUT .X
            TAY               ;& .Y
            JSR   SETNAM
            LDA   MODE        ;GET MODE
            BEQ   +           ;BRANCH IF 64
            LDX   #$00        ;ELSE BANK 0 FOR NAME
            LDA   $C6         ;GET LOAD BANK DEFAULT
            JSR   SETBNK
+           JSR   OPEN        ;OPEN SRC CMD CHANNEL
            BCC   +           ;BRANCH IF NO ERROR
            JMP   ERROR       ;ELSE PROCESS ERROR
;
+           LDA   #$02        ;LOGICAL FILE #2
            LDX   #$09        ;DEVICE 9
            LDY   #$0F        ;SECONDARY ADDRESS 15
            JSR   SETLFS
            LDA   #$00        ;FILE LENGTH 0
            TAX               ;ZERO OUT .X
            TAY               ;& .Y
            JSR   SETNAM
            LDA   MODE        ;GET MODE
            BEQ   +           ;BRANCH IF 64
            LDX   #$00        ;ELSE BANK 0 FOR NAME
            LDA   $C6         ;GET LOAD BANK DEFAULT
            JSR   SETBNK
+           JSR   OPEN        ;OPEN DEST CMD CHANL
            BCC   +           ;BRANCH IF NO ERROR
            JMP   ERROR       ;ELSE PROCESS ERROR
;
+           LDA   #$08        ;LOGICAL FILE #8
            LDX   #$08        ;DEVICE 8
            LDY   #$08        ;SECONDARY ADDRESS 8
            JSR   SETLFS
            LDA   #ODF-SNAM   ;GET NAME LENGTH
            LDX   #<SNAM      ;NAME ADDRESS LOW
            LDY   #>SNAM      ;NAME ADDRESS HIGH
            JSR   SETNAM
            LDA   MODE        ;GET MODE
            BEQ   +           ;BRANCH IF 64
            LDX   #$00        ;ELSE BANK 0 FOR NAME
            LDA   $C6         ;GET LOAD BANK DEFAULT
            JSR   SETBNK
+           JSR   OPEN        ;OPEN SRC FILE
            BCC   ODF         ;BRANCH IF NO ERROR
            JMP   ERROR       ;ELSE PROCESS ERROR
;
MODE        .BYT 0
SNAM        .BYT  'TEST1,S,R' ;SOURCE NAME
;
```

```
ODF         LDA   #$09        ;LOGICAL FILE #9
            LDX   #$09        ;DEVICE 9
            LDY   #$09        ;SECONDARY ADDRESS 9
            JSR   SETLFS
            LDA   #COPY-DNAM  ;GET NAME LENGTH
            LDX   #<DNAM      ;NAME ADDRESS LOW
            LDY   #>DNAM      ;NAME ADDRESS HIGH
            JSR   SETNAM
            LDA   MODE        ;GET MODE
            BEQ   +           ;BRANCH IF 64
            LDX   #$00        ;ELSE BANK 0 FOR NAME
            LDA   $C6         ;GET LOAD BANK DEFAULT
            JSR   SETBNK
+           JSR   OPEN        ;OPEN DEST FILE
            BCC   COPY        ;BRANCH IF NO ERROR
            JMP   ERROR       ;ELSE PROCESS ERROR
;
DNAM        .BYT  'TEST1,S,W' ;DEST NAME
;
COPY        LDX   #$08        ;SET FILE #8 AS
            JSR   CHKIN       ;INPUT SOURCE
            BCC   +           ;BRANCH IF NO ERROR
            JMP   ERROR       ;ELSE PROCESS ERROR
;
+           JSR   BASIN       ;GET A BYTE
            STA   DATA        ;AND STORE IT
            LDA   ST          ;GET STATUS BYTE
            STA   TEMPST      ;AND STORE IT
            BEQ   +           ;BRANCH IF ST IS OK
            CMP   #64         ;ELSE CHECK FOR EOF
            BEQ   +           ;AND BRANCH IF TRUE
            JMP   ERROR2      ;ELSE PROCESS ERROR
;
DATA        .BYT  $00         ;TEMP AREA FOR DATA
TEMPST      .BYT  $00         ;TEMP STATUS
;
+           JSR   CLRCH       ;RESET I/O DEFAULTS
            LDX   #$09        ;SET FILE #9 AS
            JSR   CKOUT       ;OUTPUT FILE
            BCC   +           ;BRANCH IF NO ERROR
            JMP   ERROR       ;ELSE PROCESS ERROR
;
+           LDA   DATA        ;GET SAVED DATA
            JSR   BSOUT       ;AND OUTPUT TO DEST
            LDA   ST          ;GET STATUS BYTE
            BEQ   +           ;BRANCH IF ST IS OK
            JMP   ERROR2      ;ELSE PROCESS ERROR
;
+           JSR   CLRCH       ;RESET I/O DEFAULTS
;
            LDA   TEMPST      ;GET SAVED INPUT ST
            CMP   #64         ;CHECK FOR EOF
            BEQ   DONE        ;AND BRANCH IF TRUE
;
            JMP   COPY        ;ELSE RESUME COPY
;
ERROR       NOP
ERROR2      JSR   CLRCH
;
DONE        LDA   #$09
            JSR   CLOSE
            BCC   +           ;CHECK FOR ERROR
            JSR   ERROR3      ;BRANCH IF ERROR
+           LDA   #$08
            JSR   CLOSE
            BCC   +           ;CHECK FOR ERROR
            JSR   ERROR3      ;BRANCH IF ERROR
+           LDA   #$02
            JSR   CLOSE
            LDA   #$01
            JSR   CLOSE
;
            RTS
;
ERROR3      RTS
;
            .END
```

CMD • HARDWARE • ORDERS: 1-800-638-3263 • SOFTWARE • CMD

HD Series SCSI Hard Disk Drives

HD-40, 42 MB (Limited Supply)	$329.00
HD-85, 85 MB (Limited Supply)	**$359.00**
HD-170, 170 MB (Special Edition)	$399.00
HD-340, 340 MB (Special Edition)	$449.00
HD-500, 500+ MB (Special Edition)	$499.00
HD-1000, 1 GB (Special Edition)	$779.00

FD Series 3.5" Floppy Disk Drives

FD-2000 (800K and 1.6 MB)	$179.95
FD-4000 (800K, 1.6 MB and 3.2 MB)	$249.95
FD Real-Time-Clock Option/Kit	$20.00/$29.00
Box of 10, High Density Disks (1.6MB)	$14.95
Box of 10, Enhanced Density Disks (3.2 MB)	$29.00

JiffyDOS

(Specify computer serial number and drive model)

JiffyDOS C64/SX-64 System	$49.95
JiffyDOS C-128/128-D System	$59.95
Additional Drive ROM's	$24.95

Miscellaneous Hardware

80-column Monitors (Refurb)	CALL
Aprotek 3-Way User Port Expander	$30.00
Aprotek User Port Extension Cable	$19.00
C-64/64-C Computers (Refurb, w/JD)	$89.00
C-64, C-64C Power Supply (Repairable)	$39.00
C-128 Power Supply (Repairable)	$49.00
Cannon BJ-200ex Bubble Jet Printer	$289.00
Cannon BJC-4000 Color Bubble Jet Printer	CALL
Commodore 1541 Disk Drives (NEW, w/JD)	$119.00
Commodore 1541 Disk Drives (Refurb, w/JD)	$75.00
Commodore 1541-II Disk Drive (Refurb, w/JD)	$109.00
Commodore 1802 40-column Monitor (Refurb)	$139.00
CMD EX2+1 3-Port Cartridge Port Expander	$34.95
CMD EX3 3-Port Cartridge Port Expander	$29.95
CMD Gamepad/Joystick Controller	$24.95
CMD GeoCable II Print Cable 6ft./15ft.	$29.00/$34.00
CMD SmartMouse (1351 Compatible Mouse)	$49.95
CMD SmartTrack (1351 Compatible Trackball)	$69.95
Monitor Cables	CALL
Mouse Pad	$2.95
MW-350 Printer Interface (0K/8K Buffer)	$49.00/$60.00
Samsung SP-0912 9-pin Epson-comp. Printer	$139.00
Samsung SP-2412 24-pin Epson-comp. Printer	$179.00
Samsung SP-2417 24-pin Color Printer	**$249.00**

Utilities

Big Blue Reader V4.10 (SOGWAP)	$39.00
CMD Utilities	$24.95
JiffyMON-64 (ML Monitor)	$19.95
The Compression Kit '94 (Mad Man)	$39.00

GEOS

Collette Utilities (Handy Geos Utilities)	$19.95
Desk Pack Plus	$29.00
Dweezils Greatest Hits (NewTools2,Stamp,Label128)	$30.00
FONTPACK Plus	$25.00
gateWay 64 or 128 (Specify Version)	$29.95
geoBASIC	$20.00
geoCalc 64/128	$40.00/$45.00
geoChart	$29.00
geoFAX	**$39.95**
geoFile 64/128	$40.00/$45.00
geoMakeBoot (Makes Bootable copies)	$12.95
geoProgrammer	$45.00
geoPublish	$40.00
GEOS 64 v2.0	$44.00
GEOS 128 v2.0	$49.00
geoSHELL V2.2 (CLI for GEOS)	$24.95
International FONTPACK	$25.00
Perfect Print LQ for GEOS (Laser-like output)	$49.95
RUN GEOS Companion	$20.00
RUN GEOS Power Pak I or II (Specify)	$20.00

Books

Anatomy of the 1541	$12.00
Basic Compiler Design for the C-64	$12.00
C-64 Science & Engineering	$12.00
C128 Computer Aided Design	$12.00
C128 BASIC Training Guide	$12.00
Cassette Book for C-64 and Vic 20	$12.00
Commodore 64 Tricks and Tips	$12.00
GEOS Programmers Reference Guide	$35.00
Graphics Book for the C-64	$12.00
Hitchhikers Guide to GEOS	$35.00
Ideas for Use on Your C-64	$12.00
Printer Book for the C64	$12.00
Mapping the C64	$12.00
Simple Internet	$16.95
Superbase - The Book	$15.00

Scanning & Video

Handyscanner	$249.00
Pagefox	$139.00

RAMLink Power-Backed RAM Disk

RAMLink Base Model (ØMB, No RAMCard)	$149.00
RAMLink w/1 MB RAMCard (Limited Time)	$199.00
RAMLink w/4 MB RAMCard (Limited Time)	**$339.00**
RAMCard RTC Option/Kit	$10.00/$19.00
RAMLink Battery Back-up (Optional)	$24.95
Parallel Cable (RAMLink to HD)	$14.95

MIDI & Sound

Digimaster	$34.95
SID Symphony Stereo Cartridge	$44.95
Sonus 64 Sequencer, MIDI Interface, Cables	$99.00
Sonus 128 Sequencer, MIDI Interface, Cables	$99.00

Languages & Compilers

BASIC 64 Compiler (Abacus)	$17.00
BASIC 128 Compiler (Abacus)	$25.00
Blitz! 64 Compiler (Skyles)	$30.00
Blitz! 128 Compiler (Skyles)	$30.00
Buddy 64/128 Assembler	$39.00
Cobol 64 (Abacus)	$17.00
Fortran 64 (Abacus)	$17.00
Pascal 64 (Abacus)	$17.00
Power C 64 (Spinnaker)	$14.00

Productivity

Cadpak 64 (Abacus)	$22.00
Cadpak 128 (Abacus)	$25.00
Chartpak 64 (Abacus)	$17.00
Chartpak 128 (Abacus)	$25.00
I Paint v1.5 (128, 80-col., 64K VDC) (Living Proof)	$39.00
I Port v1.54 (128, 80-col., 64K VDC) (Living Proof)	$29.00
Outrageous Pages (Batteries Included)	$19.00
PaperClip 3 64/128 (Batteries Included)	$35.00
Personal Portfolio Manager (Abacus)	$16.00
Pocket Writer 2 (64) (Digital Sol.)	$65.00
Pocket Writer 3 (64 or 128) (Digital Sol.)	$70.00
Pocket Planner 2 or Pocket Filer 2 (Digital Sol.)	$35.00
PowerPlan 64 (Abacus)	$16.00
RUN Productivity Pak I, II, or III (Specify)	$15.00
RUN Super Starter Pak 1541 or 1581	$20.00
RUN Works	$20.00
SEC Check Register 128	$29.00
SuperScript 64 (Precision)	$15.00
SuperScript 128 (Precision)	$20.00
Suberbase 64 Version 3.01 (Precision)	$35.00
Superbase 128 Version 3.01 (Precision)	$35.00
SwiftCalc 64 (Timeworks)	$16.00
Tax Perfect 64 (Free '94 Upgrade)	$69.00
Tax Perfect 128 (Free '94 Upgrade)	$79.00
TWS 64 w/Speller (Busy Bee)	$29.00
TWS 128 w/Speller (Busy Bee)	$39.00
TWS Modules (HD/RL/Illustrator)	each $5.00

Telecommunications

Aprotek MiniModem C-24 (C= ready, 2400 baud)	$69.00
Aprotek MiniModem C (C= ready, 1200 baud)	$50.00
Aprotek Modem adapter (C= to Ext. PC Modem)	$20.00
BOCA 2400 Baud Modem	$69.00
BOCA 2400 w/SwiftLink and Cable	$99.00
BOCA 14.4K bps FaxModem	$129.00
BOCA 14.4K w/SwiftLink & Cable	$159.00
BOCA V.34 28.8K bps FaxModem	$259.00
BOCA V.34 w/SwiftLink & Cable	$289.00
Dialogue 128	$29.00
SpeedTerm (Abacus)	$25.00
SwiftLink RS-232 Cartridge (Up to 38.4K baud)	$39.95
SwiftLink Modem Cable (DB9-DB25)	$9.95

Games

Atomino	$17.00
Ballistix	$13.00
Blood Money	$15.00
Chomp!	$10.00
Day in the Life of Prehistoric Man	**$19.00**
Escape Route	**$19.00**
Heavenbound	**$19.95**
Island of the Dragon	**$19.00**
Laser Squad	$13.00
Lions of the Universe	**$19.95**
Mainframe	$13.00
Menace	$15.00
Navy Seal	$10.00
Rings of Medusa	$16.00
RUN C128 Funpak	$15.00
RUN C64 Gamepak	$15.00
Skate or Die	$13.00
The Amazing Spider-Man	$15.00
The President Is Missing!	$10.00
The Three Stooges	$10.00
Tie Break Tennis	$16.00
Total Eclipse	$10.00
Ultima V	$17.00
Wings of Circe	**$19.00**
Wizardry 5: Heart of the Maelstrom	$24.00

JiffyDOS

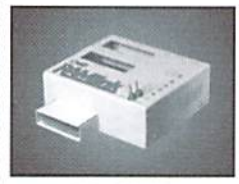
Increase Speed Up to 1500% while retaining 100% compatibility

- Speeds up Loading, Saving, Verifying, Formatting and Reading/Writing of Program, Sequential, User and Relative files unlike cartridges which only speed up Loading and Saving of PRG files
- Built-in DOS Wedge plus 17 additional features including file copier, text dump, printer toggle, and redefinable function keys make using your computer easier and more convenient
- ROM upgrade installs easily into most computers and disk drives. Supports C-64, 64C, SX-64, C-128, 128-D, 1541, 1541C, 1541-II,1571, 1581 and more.

FD Series

3.2 and 1.6 MB 3.5" Floppy Drives

The FD-2000 and FD-4000 disk drives utilize today's latest 3.5 inch disk drive technology. FD-2000's support 800K (1581 style) and 1.6 MB (High Density) formats, while the FD-4000 also offers support for the 3.2 MB (Enhanced Density) disk format. Fast and reliable, they support 1541, 1571 and 1581 style partitions, Native Mode partitioning and can actually read and write 1581 disks. FD's feature built-in JiffyDOS, SWAP button and optional Real Time Clock. High capacity, speed and compatibility make the FD right for every application, including GEOS.

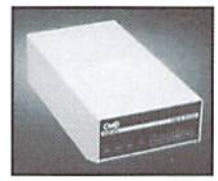


RAMLink

Power-Backed Expandable REU

The fastest possible form of storage, RAMLink provides instant access to files and retains data while your computer is turned off. Easy to use and expandable up to 16 MB, RAMLink utilizes the same powerful operating system found in CMD Hard Drives. Unlike Commodore REU's which are compatible with less than 1% of commercial software, RAMLink supports more than 80% of the commercial titles. RAMLink also offers built-in JiffyDOS Kernal, SWAP feature, reset button, enable/disable switch, pass-thru port and RAM port for REU/GEORAM use. RAMLink offers maximum speed, expandability and compatibility with all types of software and hardware including GEOS.

HD Series



SCSI Hard Drive for the C-64/128

HD Series Hard Drives are available in capacities up to 1 GB, are fully partitionable, and can emulate 1541, 1571, & 1581 disks while Native partitions utilize MS-DOS-style subdirectories. HD's connect easily to the serial bus or parallel via RAMLink. Includes built-in JiffyDOS, SWAP feature and RTC. HD's offer superior compatibility with most commercial software including BBS, Productivity and GEOS. And with new pricing, HD Series drives offer the lowest cost per megabyte of any C64/128 storage device.

Shipping and Handling Charges

Use the chart below to match your order subtotal with your shipping zone and method.

	Continental United States					
Order Subtotal	UPS Ground	2nd Day	Next Day	AK HI, PR	Canada	Foreign
$0.01 - $19.99	$3.00	$8.00	$17.00	$12.00	$5.00	$15.00
$20.00-$29.99	$5.00	$9.00	$18.00	$14.00	$7.00	$20.00
$30.00-$59.99	$6.00	$10.00	$20.00	$15.00	$9.00	$25.00
$60.00-$149.99	$8.00	$12.00	$23.00	$19.00	$12.00	$35.00
$150.00-$299.99	$10.00	$14.00	$27.00	$21.00	$20.00	$50.00
$300.00-$799.99	$15.00	$20.00	$34.00	$27.00	$25.00	$55.00
$800.00 +	$20.00	$25.00	$40.00	$32.00	$35.00	$60.00

UPS C.O.D. add $5.00 (U.S./PR only)

Payment and Delivery: CMD accepts MC, Visa, Money Orders, COD and Personal Checks. Personal Checks are held for up to 3 weeks. Most items are stock, contact CMD for current delivery schedules. Returns for merchandise credit only within 30 days with prior authorizations. All prices and specifications are subject to change without notice.

Creative Micro Designs, Inc.
P.O. Box 646
East Longmeadow, MA 01028
Info: (413) 525-0023
Fax: (413) 525-0147

JiffyDOS™ SPEED ENHANCEMENT SYSTEMS

- **An easy to install ROM chip upgrade for your computer and disk drive**
- **Increases speed of all disk operations up to 1500%**
- **Provides a built-in, easy-to-use DOS wedge with 17 additional JiffyDOS commands**
- **Versions available for all Commodore 64 and 128 computers and serial disk drives**
- **Built-in two drive file copier works with all drives and file types**
- **Compatibility guaranteed or your money back**

What Is JiffyDOS?

JiffyDOS is a Disk Operating System (DOS) enhancement which gives your C64 or C128 the disk access speed it has always needed. A chip-for-chip replacement for the Kernal ROM in your computer and the DOS ROM in your disk drive(s), JiffyDOS achieves levels of performance and compatibility unmatched by other disk speed-enhancement products. Outstanding speed, solid compatibility with virtually all hardware and software, and a new set of desperately-needed commands and features give your system the power to compete with today's newer, more-expensive machines.

JiffyDOS should not be confused with Cartridges, Turbo ROMs, Burst ROMs or "Parallel" systems. Ultra-high-speed multi-line serial technology enables JiffyDOS to outperform these products without any of their inherent disadvantages. JiffyDOS leaves all ports on your computer open, works with virtually all software, speeds up PRG, SEQ, REL and USR files, and does not require any extra cabling. In short, JiffyDOS is working whenever your computer accesses your disk drive.

10th Anniversary Sale-a-bration

THIS OFFER IS FOR CW SUBSCRIBERS ONLY AND HAS BEEN EXTENDED THROUGH OCTOBER 31, 1995

For ordering and shipping information, please refer to our main ad on the adjoining page. You must request this special offer at the time you place your order. This offer may not be combined with any other offer.

Need Input? It Doesn't Get Any Better Than This...

SmartTRACK

For years, Commodore set the standard with the 1351 Mouse. Sure, it was vastly superior to using the old digital input devices like the 1350 mouse, joysticks or imitation trackballs. But everything can be improved. Guaranteed 100% 1351-compatible, SmartMouse and SmartTrack do everything the C-1351 does and more! These highly intelligent, three-button input devices include a built-in battery-backed Real-Time Clock, along with double-click and Turbo features for GEOS. Plus, they come with a complete set of utilities for using the clock in GEOS and BASIC applications. Modern ergonomic designs make these devices smooth operators that are a pleasure to use. So, if you're tired of the slow, erratic movement of your current input device, make the intelligent choice and pick up a SmartMouse or SmartTrack today!

SmartMOUSE

SmartMouse and SmartTrack... the Best C-64/128 Input Devices Ever!

- **Three buttons means convenience! If you're a GEOS user, the left button is configured as single click, the right as a handy double click and the center button is the TURBO button. When depressed, it doubles the speed at which the pointer moves across the screen. Additionally, programmers can assign their own functions to all three of the buttons.**
- **Unlike other third party mice or trackballs, the CMD SmartMouse and SmartTrack trackball use the same custom gate array chip as the Commodore C-1351 mouse. This guarantees 100% compatibility.**
- **Switches easily into joystick emulation mode on power-up by holding down the right button.**
- **SmartMouse/SmartTrack utilize the same advanced technology used in today's powerful 486 and Pentium PC's, providing you with unparalleled accuracy and smoothness.**
- **Built-in battery-backed Real-Time Clock automatically sets the GEOS clock, displays time and can be used in your own programs.**
- **Includes utilities disk and detailed manual explaining the utilities and programming information.**
- **Attention Lefties! SmartMouse can be altered for left handed use.**
- **Don't be fooled by the old style digital "trackballs". These only perform as well as a joystick. SmartTrack uses analog inputs for greater accuracy.**

SOFTWARE IN REVIEW

GEOFAX

geoFAX
Click Here Software; $39.95

If there's anything that makes a Commodore user feel good, it's being able to do something that would normally require a "more powerful" computer. One such capability is to send and receive fax documents via fax modems. The arrival of geoFAX, however, signals that we're no longer left out when it comes to this task.

The geoFAX package contains a two-sided 5-1/4" disk. On the front side you'll find geoFAX; a Read Me file; custom printer drivers for Epson-compatible 9- and 24-pin dot-matrix printers, HP & Canon ink-jet printers, and Postscript™ laser printers; a photo scrap; and a couple of fax documents. It also contains a special printer driver called INTERCEPTOR that outputs to disk in geoPaint format, and a copy of Maurice's goeMORPH program. The back of the disk has a second Read Me file, and the documentation for geoFAX in geoWrite format (you'll have to print this out yourself).

Producing fax software proved difficult under the limitations of the fax protocols, the slow clock speed of the 64, and GEOS' disk drivers. Optimizing time taken by certain routines, and relying on some help from the modem's own hardware made it all possible. But because of these factors, geoFAX requires a modem that has certain features.

Of the common 14.4Kbps modems, those that have been used with success are: Best Data Smart One 14.4K, Boca M144EW, Supra FaxModem (9600 & 14.4K), and Zoom VFX Model 350. Modems made by GVC, Reveal, LineLink, and USRobotics are among those which have failed to work with geoFAX. Some don't have the necessary Class 2 protocol; others don't support certain commands used by geoFAX (&K4 is one), or are lacking in RAM buffer capacity.

According to the author, modems with less than 16K of internal RAM buffer space are likely to have problems, and should be avoided. You can check the buffer size of most modems by sending an AT+FBUF? command from a terminal program. You'll get four sets of numbers back, separated by commas. The first number is the important one, and would read 16834 for a 16K buffer.

Provided you have a SwiftLink interface (required) and a modem that meets the requirements of geoFAX, you can use practically any storage devices supported by GEOS; your fax speed will, however, be governed by the device you use for storing your fax documents. Floppy drives are the slowest, ranging from 7200 to 9600 on send, and 2400 to 4800 on receive. A hard drive fares a little better (9600 send, 7200 receive), with the faster RAM devices coming out at the top (up to 14.4K send, 9600 receive).

For review purposes, I copied geoFAX to my RAMLink, and after completing the brief installation requirements, I ran it through its paces using a Boca M144EW modem. I began by sending one of the sample fax documents to our fax machine, located in another room. The fax came through cleanly at 9600. I next sent a fax from the fax machine to my 128, using a receive speed of 7200 (as suggested for RAMLink). Again, no problem.

I printed the resulting document to a Panasonic KX-P1123 printer, using the highest-quality driver supplied for this printer in the present version of geoFAX. The output quality was extremely good, and the size was identical to the original document. This is very impressive, as I've seen fax programs on other platforms that didn't maintain the size nearly as well.

I tested several more documents in both directions, using various devices and speeds; I also tested the

Tips for Fixing Fax Problems

A new software program, no matter how bug free, will still have a small number of users that have trouble with it. The new program geoFAX is no exception. After all, fax machines in general are sort of fickle. In questioning several businesses with heavy fax usage, there were reports of several faxes a day which didn't go through.

Many of these are probably due to line noise. This is the first area to check; you should be sure that you have a "clean" line. A noisy line can cause the receiving fax not to "hear" the incoming signals from your fax machine, whether it is a computer fax, or an actual facsimile machine. Your local phone company can usually test your line for noise, and they can also fix it in most cases if it turns out to be faulty. They'll also probably recommend that you invest in a dedicated fax/modem line. It's true that this is the best way to ensure a strong signal, though you'll almost always pay an added premium for it. Be aware that you can still have problems if the party you're trying to send to has a noisy fax line.

Another thing that can cause you problems are poor connections. Make sure that connections to telephones, wall jacks, and modem jacks are secure. While you're at it, check to make sure that your phone line is in the correct jack on your modem. I'm embarrassed to say this was one of my problems. My excuse is that I couldn't see the back of the modem to plug it in the line jack.

Possibly the most important cause of problems for those of us who want to fax from a Commodore begins with an important observation: all modems are not created equal. Some lack the required Group III Class 2 protocol, while others may not support the necessary commands. Some of the less expensive modems may not have a large enough RAM buffer. You may pay a little more for a modem that has everything required, but the old saying, "you get what you pay for" seems to ring true when choosing a modem for geoFAX.

There's one final problem that can affect any hardware configuration: file corruption. This can usually be resolved by simply recopying the application to your work disk.

Fortunately, there is great product support from the author of geoFAX. If all else fails, get in touch with him. In many cases, he'll have already dealt with a similar problem, and will have you up and faxing in no time!

- Timothy R. Hewelt

built-in conversion utility for converting documents between geoFAX and geoPaint (users with Commodore-compatible 60 dpi printers will need to use this utility to print their fax documents). Everything was simple to use, and worked as described.

While geoFAX lacks the frills of fax programs on other platforms, it performs the basic task of faxing well. If you want or need the ability to send and receive fax documents, and have (or are willing to get) the required equipment, and don't mind working in the GEOS environment, geoFAX delivers good quality at a reasonable price.

- Doug Cotton

Don't Just Fax... Scan!

While geoFAX was created to provide users with the ability to send and receive faxes, it can have another purpose for users who own a standard fax machine, or have access to one. Since geoFAX includes a utility to convert received faxes into geoPaint files, it can be used to scan graphics! And the results can be pretty impressive. Even full page pictures can be reproduced with surprising detail.

Optimum scans are produced by using a FAX machine that has halftone capabilities, which produces superior results when scanning photographs, clip art, drawings, magazine covers, or most any image that isn't just black and white.

Using the scanning process online involves having the geoFAX set up to automatically receive faxes, and dialing and sending the document from a remote fax machine. Just make sure you saving the incoming fax documents to a disk (or partition) with plenty of room since fax files have more resolution than a geoPaint file (204x196 dpi vs. 80x72 dpi), which means that they do take up a lot of space.

Many modern fax machines will connect directly to a faxmodem without having to tie into an actual phone line. The scanning process when using a fax machine that is directly linked to the geoFAX system just involves putting geoFAX in manual mode, sending the fax from your fax machine, and hitting the start button in geoFAX. It's best to set geoFAX to receive at its slowest rate (2400 baud); this helps ensure that the best possible copy will be received.

Regardless of which method you use to receive the "scanned" document, the next step is to convert it from a fax to a geoPaint file. You can do this in geoFAX's DISK functions. Just select the 'CONVERT' icon associated with converting a geoFAX document to a geoPaint document. Once you've accomplished this, you'll have a geoPaint image of the faxed document, which can be edited or used as you see fit.

- Timothy R. Hewelt

GRAPHICS MASTER

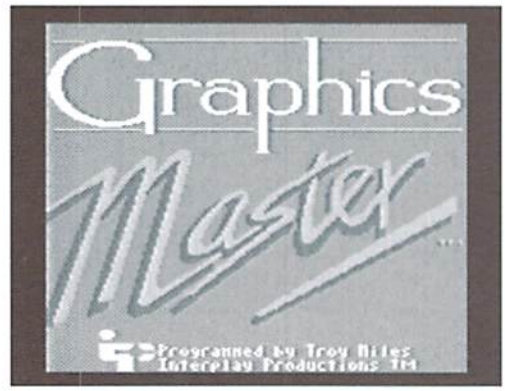
Graphics Master
ShareData, Inc.; Available from SSI.

Graphics Master provides complete control of computer art on a Commodore 64. With Graphics Master, Commodore users can produce almost any desired project from posters and signs to graphs and charts. Basically, if you can imagine it, it can be created.

Graphics Master must possess several unique features in order to compete and survive against today's multitude of available Commodore graphics programs. One of Graphics Master's biggest advantages is its flexibility. This is one program which does not dictate which input device you'll have to use. You get to decide which device you feel the most comfortable with: a keyboard, joystick, Wico Trackball, or Koala Pad.

Yet another key characteristic is the ability to create custom character sets exclusively for use with Graphics Master creations. These characters can then be used to label charts, signs, etc. Off the top of my head, I can't think of another graphics package including this unique feature.

Now let's get down to the basics. Graphics Master employs two modes: One for drawing or revising pictures, and the other for creating or modifying character sets. Let's start with Picture Mode. Here, the upper portion of the screen consists of a drawing board where the actual drawing is displayed or created. Located below the drawing board are three rows of icons. Drawing board operations are activated with the arrow shaped cursor. Of course, the cursor is controlled through your selected input device (keyboard, joystick, trackball, or Koala Pad).



There are two types of icons: Command Icons and Status Icons. The Command Icons are selected to perform such operations as draw, color, fill, magnify, etc. These operations perform in the same manner as those found in most other graphics packages. Activate the disk icon to save or load files and view directories. When you're done, select the printer icon to produce a hard copy of your project. Printers supported by Graphics Master include: Commodore 1525, Commodore MPS801, Epson MX-80/100, Epson FX-80/100 with Cardco+G Interface, and an Okidata 92 with Cardco+ G Interface. The Status Icons are located to the right of the icon rows. These icons indicate the active Command Icon as well as the current cursor location.

The screen in Character Set Mode is very different from the screen in Picture Mode. The drawing board contains eight sectors used to modify or create characters or symbols. On the right side of the drawing board is a list of available commands. Below the drawing board are four rows of characters and symbols comprising the current character set. To modify or delete any of these characters, highlight the desired character and press the input device button. The character can then be placed in any one of the eight sectors. Once it is in one of the sectors, the commands on the right can be used to delete or modify it. When all the changes to the character have been completed, it must then be placed back into the character set below the drawing board. You continue in this manner until all the characters and symbols have been altered to your liking. The new character set can then be saved to disk for future use.

The Graphics Master program is very easy to use. A manual is included with detailed information on all the various icons and their usage. The disk also contains a few sample pictures, posters, and character sets to get you started. Graphics Master provides you with all the necessary tools to complete almost any graphics project. The possibilities are only limited by your own abilities to transfer your imagination to the screen.

- Sherry Freedline

PAPER MODELS: The Christmas Kit

Paper Models:
The Christmas Kit
Activision; Available from SSI

Activision places a host of creative Christmas projects at your fingertips with the Christmas Paper Workshop. The program comes complete with a glue stick, ruler, card stock paper, red and green markers, and even a few small jingle bells. Aside from your own artistic imagination, the only items left for you to supply are scissors and a Commodore 64/128 with a disk drive, joystick and printer.

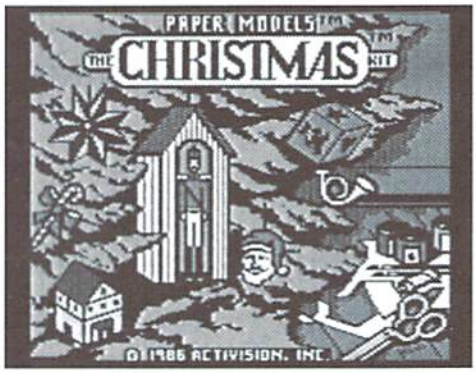
For this review, I enlisted the help of my nine year old daughter. After reviewing the manual, we selected a sleigh for our first holiday project. Since the manual lists the required files and instructions for assembling each of the included projects, you'll want to keep it in a safe place.

Loading the program revealed a black and white screen with a work area in the center and a menu box on the left side of the screen. Our selection required the printing of three separate files: the right, left and bottom of the sleigh. We loaded the right side of the sleigh first. It seemed rather bare so we consulted the manual again and discovered included holiday clip art could be used to decorate the sleigh.

The Workshop actually has two work areas which you toggle between by pressing the 1 and 2 keys. We loaded the clip art into the second work area, filling up with the art.

Then we chose some decorations and moved them from the second work area to the screen containing our sleigh. This was done by selecting the copy and paste commands from the menu with the joystick. Other commands allowed us to draw, erase, fill, zoom, and add text to our creations.

Finally, it was time to print that section of the sleigh. Of course printing is done by selecting the output command from the menu. After all the required sleigh portions finished printing, my daughter colored them. Once she finished adding her artistic touches, I cut out each piece, glued it to the card stock, then cut each piece out again. We scored each piece with a ruler where necessary and glued them together where indicated. Voila... we had a nice little decoration to place beneath our tree this Christmas.



Besides the user-friendliness aspect, I liked this program because it can be used by the whole family. The Workshop contains files to complete over 30 different projects, from Christmas villages to gift boxes to a holiday train! It's the perfect program to dig out on boring Saturday afternoons, or just to get you in the mood for Christmas.

-Sherry Freedline

BUDDY 64/128 ASSEMBLER

Buddy Assembler
Chris Miller; $34.95; Available from CMD

You can't do a job right if you don't have the right tools. I discovered the truth in this when my wife and I decided to do a little remodeling. It looked good on paper: new wallpaper, new windows in the dining room, a new bathroom where the utility room is; it seemed to be do-able in a weekend each, at most. After nine trips to various hardware stores, three trips to return and exchange things for other items, almost two months of work, and numerous experiences in "measure once, cut twice," my house is a wreck. It's not that I can't do remodeling, I just don't seem to own the tools that would make each job easier.

The same holds true for my programming on the Commodore 64. I sketch out a great idea for a program, and assure myself that all the functions can be written in short order, with little effort. In earlier days, I would to pull out the machine language monitor and start manually programming the C64 to turn my great idea into reality. As usual, my efforts would end up fruitless. Programming the C64 with a machine language monitor was hard, and I tired of programming that way. I wondered, "How do other people manage to program in machine language with the ML monitor?" The answer: they don't. I discovered that sane folks use an assembly language development system to make life easier.

Now, with the development system, my ideas can be more easily turned into reality. I have one such system in front of me now: The BUDDY 64/128 Assembly Development System. Let's take it for a test drive.

The BUDDY system is a full-featured assembly language development system that has versions for the 64 and the 128. It offers the user the options of compiling for the C64 environment, the C128 native environment, or the C128 CP/M environment. In addition, a version of the assembler that will interface with C-Power 128 is included. So, the user gets 4 assemblers in one package deal.

Only one decision need be made before tackling your next great software idea: which editor to use. BUDDY allows the user to either write programs in the BASIC interpreter, much like the way BASIC programs are written, or enter source code in a full screen editor, of which 64 and 128 versions are included.

The C64 editor provides a 40x25 "window" to view source code, while the C128 editor provides 2 40x25 "windows" for source entry and perusal. With each source code entry environment, all options in BUDDY are available, and the resulting assembled program is identical. The user's preference will dictate which is the editor of choice.

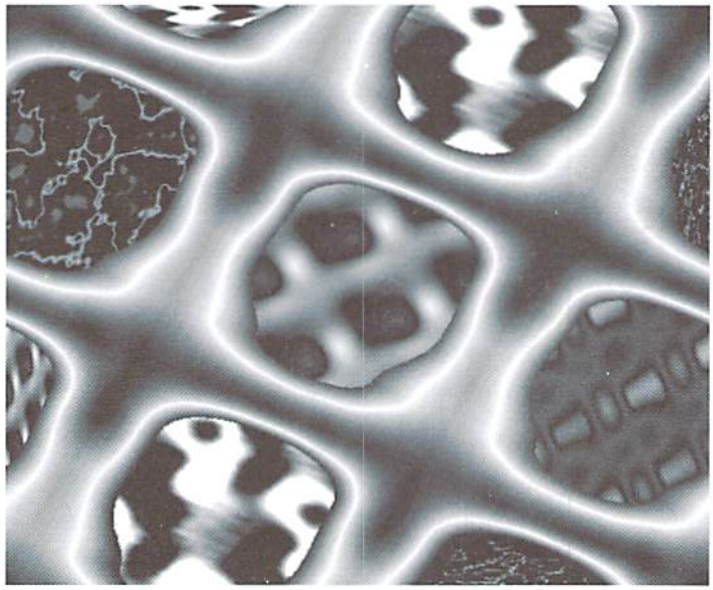
BUDDY contains the standard items one would expect to find in a well-written assembler, so I won't bore you with the generic qualities. I will point out that BUDDY has a rich set of pseudo-

opcodes, special statements in the source code that tell the assembler to perform certain operations. Each pseudo-opcode is prefaced by a period (.), to distinguish it from normal opcodes.

Some of the special opcodes are easy to understand: ".ORG" to set the origin address of the assembled program, and the ".MEM" opcode to assemble right to memory, but others require more thought, like ".OFF", which temporarily changes the address to assemble to, and ".PSU", which turns on the recognition and compilation of the special "illegal opcodes" in the 65XX CPU.

To make it easier for the machine language programmer, BUDDY also contains the following features:

- The ability to assemble files straight from sequential files, for those who already use a preferred editor or source entry environment.
- A special pseudo-opcode (.BUF) to create reserved areas in the assembled program for buffers.
- The ability to assemble code to one location that is destined to be moved to another before execution. (the .OFF opcode)
- The option of displaying a full listing of the assembly process to either screen or printer, or merely a listing of errors encountered.
- The ability to systematically include BASIC commands in a program. (i.e. The infamous "SYS 1234" at the beginning of programs).
- Chaining of source code modules, to allow for large programs to be developed in multiple files.
- Macros. BUDDY will allow the programmer to create functions (with parameters), which can be used like regular opcodes in the source code, and will be expanded to the macro definition at assembly time.
- Flexibility in specifying data elements in programs. Operators exist to store bytes, words, ASCII, and screen code data in to data areas in programs.
- Full utilization of the Commodore 128 system. Pseudo-opcodes like ".BURST" and ".FAS" are included to take advantage of burst disk speeds and 2 MHz operation for those people using BUDDY on a C128.
- Conditional assembly. The if-then-else set of pseudo-opcodes make tailoring program to multiple environments much easier.
- User written opcodes. The BUDDY system allows the programmer to define up to 5 new opcodes, which can then be used in source code just like regular opcodes.
- The reusable "-", "+", and "/" temporary labels. Small loops and other similar code fragments will benefit from this way to label a loop without assigning a unique name.
- Multiple assemblers. Versions are provided to assemble 65XX source from the BASIC interpreter, 65XX source from sequential files, 65XX source from within the C Power 128 environment, and Z-80 source code.
- Efficient use of memory. For small projects, the entire source file, the assembler, the editor, and the resulting assembled code will fit into the Commodore 64 memory space The disk that holds the editors and the different assemblers also includes a number of sample macro definition files, special opcode files, and some simple programming aids, including a rudimentary unassembler. Although each is not that significant, the total of them adds greatly to the usefulness of the entire BUDDY system.



You may ask, "So, what's not to like?" Even in a system as complete as BUDDY, a few rough edges still protrude. The first is the user manual. Although manual is mostly error-free, and all the commands in the product are detailed, the manual spends too little time explaining some of the more esoteric ones. It dives into the specifics of the assembler a little too fast, giving the manual reader the feeling of being "rushed" into the product. While a "real programmer" would never stoop to reading a manual, users who begin the assembly language programming journey will find the manual somewhat lacking in beginner focus.

Another item of concern is the age of this product. While some Commodore software is classic and cannot be improved, I don't believe a programming environment qualifies. Although I was able to use my REU with BUDDY via RAMDOS, BUDDY doesn't take advantage of the REU to allow "hot-switching" between the assembly environment and the test project under development. Also, BUDDY does not support cross assembly to or from other platforms. Some may argue the necessity of the latter enhancement, but the former would reduce the wait in assemble-test-reset-edit sequences. Even with all the files in the REU, the system was too complex to be stable after many iterations, not to mention that RAMDOS is designed to be as unobtrusive as possible at the expense of speed. CMD, who distributes the product, should update the product to keep it current with other assembly development systems on the market.

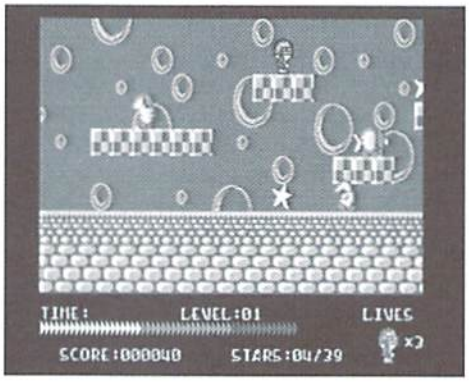
In the "trivial but annoying" department, I wish the editors included an 80 column option. True, a C128 80-column editor is supplied, but it displays 40 columns of two source files. I rarely needed the two file option, and although useful, got in my way at times. I' d like to see it made into an option rather than a mandate. In the 40 column editor, a pseudo 80-column display would be nice, to allow me to see more of my source code on screen at one time. I noted that both editors would scroll to 250 columns, but it just isn't the same. Since BUDDY allows the use of external editors, these are not crucial to the usefulness of this product, but are enhancements that could prove useful.

In spite of the user manual faults and the inability to fully exploit my REU, the BUDDY system is very complete. It automates a number of tedious tasks that every program includes, and is flexible enough to satisfy a wide variety of programming tastes, explaining by the large number of people who use it to develop software, and why many programs available as source code are in "BUDDY" format. Additionally, Craig Bruce's ACE assembler, and Karma 128 will accept BUDDY source files and operations. The inclusion of programming aids, editors, multiple assemblers, and sample macros and special opcodes makes for a very well integrated package. BUDDY can help the new or experienced assembly language programmer surmount many of the hurdles in software development and move on to the more pleasant experiences in programming.

Now, if I could just find an equivalent for the same price for my remodeling project...

-Jim Brain

New GAMES



Slaterman

Threshold Productions

Issue 9 of *Commodore World* announced quite a lineup of new games to come from Threshold Productions of Seattle, Washington. Slaterman, a one-player arcade game, is the first of these new games to fall into my hands. I am amazed at the number of games being released for the Commodore in recent months. This is almost too good to be true!

The only negative thing about Slaterman is the time required to load the game. But once you enter the world of Slaterman, you may find it very hard to drag yourself away from it. You may have read in the last issue of *Commodore World* that Slaterman is similar to the infamous Super Mario Games. Personally, I think the only portions of this game resembling Super Mario are the layouts of the game screens. There are no similarities in the actual game play.

The game screen is comprised of a variety of platforms constructed of items such as blocks, cones, pillars, and almost anything you can imagine. Stars are randomly placed on top of the many platforms. The status area is located at the bottom of the screen and contains the timer, number of lives left, your score, and the star indicator. The star indicator displays the number of stars you've collected, along with the number you'll need to collect to finish the game screen. Of course, there are a host of odd looking enemies roaming the platforms. Naturally, their purpose is to give you a hard time. Luckily, your gun is armed with a never-ending supply of bullets.

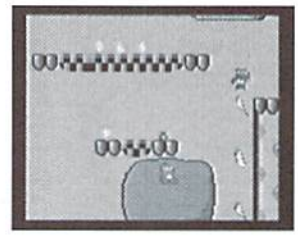
The object of the game is to collect all the stars on each game screen within an allotted amount of time, indicated by the timer at the bottom of your screen. You'll advance to the next game screen once you've collected all the stars. Complete five games screens and you'll advance to the next level. At the end of each level, a password is displayed allowing you to easily begin game play at the level last mastered.

You are given three Slatermen at the beginning of the game. A Slaterman is lost when time runs out, when he touches an enemy creature, or if he happens to fall into the water. Game play is fairly easy up until about the second game screen of level 2. Then the jumps get trickier due to increasing distances between the platforms and the placement of the bad guys. The trickiest creatures I've encountered so far have been the flying birds. They are harder to shoot because Slaterman has to jump just right to hit them, plus, they're hard to hit since they're in the air.

At the beginning of the game, you have a choice of selecting both music and sound effects, only music, only sound effects, or silence. A password may also be entered at this same screen to bypass previously conquered levels.

Slaterman contains a variety of games screens to keep the game fresh, along with a toe-tapping tune. The best feature of the game is that it can be played straight from the box without the need to read directions; it's that easy to play! So, if you need an escape from everyday pressures, allow Slaterman to whisk you away!

-Sherry Freedline



The Magnificent Six

Retros

The Magnificent Six is a disk containing six different programs for the Commodore 64. Of the six programs, five are games and one is a utility program.

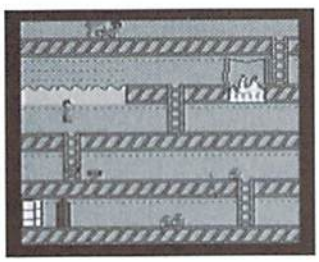
The first game, Fojamin Fum, is the best of the lot. Fojamin Fum is a platform style game, where you guide your character continually up. Along your climb you must avoid the droppings of flying birds (yes, you read that right!) along with flying boulders thrown by other fearsome creatures. Fortunately, you're capable of shooting and killing the menacing enemies. Unfortunately, I was unable to get very far in this game due to an allotment of only two lives per game.

In the next game, Body Count, you'll find yourself lost inside a 3-D maze. The 3-D effect is nicely rendered. You travel through the labyrinth with the use of a joystick or gamepad. The keyboard is used to access different weapons and also to position yourself at different angles. This game suffers from lack of documentation. The only way to find out the controls is to explore your keyboard.

The third game on the front of the disk is Blitz 43. While loading, the author states this game is the outcome of a thirty minute BASIC venture. And that's exactly what this game resembles. The object of Blitz 43 is to shoot planes which fly overhead at various altitudes. Your bomber is located at the bottom of the screen and can be moved left and right with the Z and X keys, respectively. The spacebar is used to launch the bomb. This game is strictly a matter of timing. It contains no sound effects and lo-res graphics.

Flip the disk over and you'll find the second best game of the group: Kongman 2. In case you haven't guessed, it's a take off on the popular Donkey Kong game. With a joystick you must shoot or climb over flying logs thrown by the Big Ape at the top of the screen. The game screen consists of platforms connected by ladders. Your goal is to rescue Glynnis, who is imprisoned by the Ape sitting at the top of the screen. Upon her rescue, you are taken to the second level of this two level game. The game contains average graphics and sound effects.

The last game, Tots TV, is a sequel to Smash TV, which during its time, was a popular Nintendo arcade game. At the beginning of the game, you find yourself locked in a room full of various enemies. You must shoot them before they shoot you and dodge the rotating, shooting machine found in most rooms. Once all the enemies have either left or been destroyed, you can exit the room and enter the next room with even more challenging creatures, daring to be obliterated. The graphics in this game were a little below average. Because of the fast paced nature of this game, it provides you with quite a challenge.



The sixth program is Super Typewriter 64, a bare bones word processing program. This program is actually pretty cute, and is great for the children in the family. It looks and sounds much like a typewriter. The program is extremely user-friendly and easy to use. Text may even be printed out on a printer.

That sums up the Magnificent Six disk from Retros. The title may be a little misleading, but if you have youngsters in your family, this would be an ideal disk for them. About half of the programs are worthwhile. Plus, it's nice to see a product that offers such a variety of programs to choose from.

-Sherry Freedline

Lazer Duel

Threshold Productions

The year is 2045. Corporations control the world, fusion powered vehicles are introduced to the masses, and the Lazer Duel is invented to draw a quick (albeit violent) end to "fierce disputes." As one of the duelists, you must obliterate your opponent or lose your own life.

Lazer Duel is yet another freshly released game from Threshold Productions. It's a unique one-player arcade game which can be played straight from the box. By not reading the enclosed instruction sheets, however, you may miss the atmosphere they create, detracting from your total enjoyment and complete understanding of the goals of the game.

Lazer Duel's game screen contains a square with tanks placed on the right and left sides. Located at the top and bottom are "bonus givers". Diamond-shaped, lazer-defracting obstacles are randomly placed in the center of the square. Three lazer-absorbing squares are located behind each players tank. Be careful not to block these squares, because it's better to have the square absorb the lazer beam than hit your tank. With a joystick, you control the tank on the left side of the screen, while the computer controls the opposing tank.

You and the computer start the duel with three tanks. Through a series of alternating turns, you have the opportunity to move your tank up or down, and to fire a single shot from your lazer, aiming at your competitor. Actually hitting your opponent is complicated by the placement of the obstacles. If your lazer beam strikes one of the obstacles, it will ricochet until it hits one of four items. Obviously, the best possible target is your opponent, closely followed by one of the two "bonus givers". Hitting one of these results in a reward of an additional tank, 1000 points, or 500 points. Resulting in no additional points, (but causing you no harm) is to have the lazer strike on of the six lazer-absorbing squares. Shooting your opponent results in the loss of one of his tanks, and 1000 points for you.

The worst-case scenario occurs when the lazer beam strikes your own tank, resulting in the loss of the tank. Once you've completed your turn, your opponent takes aim. During your opponent's turn, you are frozen in place and left virtually defenseless. A tactic I found quite helpful, if at all possible, is to refrain from blocking any of your lazer absorbing squares. The turns continue until you or the computer opponent runs out of tanks.

If you survive, a display will appear indicating your score for the prior game screen. The game will continue with the next game screen of the level. Each of Lazer Duel's eight levels consists of five game screens. After winning the fifth game screen of a level, you'll be rewarded with a password which allows you to bypass levels you've previously survived. However, if you lose to the computer, you'll be returned to Lazer Duel's initial game screen where you can either start a new game or enter a password to start at the level where you last dueled.

I have never played or even heard of a game similar to Lazer Duel. Initially I didn't feel I'd be challenged by this game. But, as I've been told many times, you can't judge a book by its cover. It didn't take long before I found myself totally engrossed in Lazer Duel; eagerly playing through the game screens, plotting my next move and anticipating the challenges awaiting me on the next level.

- Sherry Freedline

HARDWARE IN REVIEW

HANDYSCANNER: For the C-64

Scantronnik; $249.00, Distributed in North America by Creative Micro Designs, Inc. P.O. Box 646, East Longmeadow MA 01028-0646, 1-800-638-3263.

Hand scanners have been around for a long time. Other computer platforms have them ad nauseam, recently being replaced by the dropping prices, and the improved function of flatbed scanners.

Commodore users have less models to fret over. In fact, there is (and always has been) only one model of hand scanner available, the HandyScanner 64, by Scantronnik of Germany. This product has been available since the late '80s, but disappeared when its distributor, Rio Computers, went under in the early '90s. Not surprisingly, the Commodore gurus at Creative Micro Designs have picked up the ball and supplied the demand. They now distribute the HandyScanner, with a reduced price to boot!

If you need the capabilities of a scanner for graphic work, then you need the HandyScanner. It does what any other scanner can do, limited only by the resolution of the 64. Since we only have one choice, it's fortunate for us that the product lives up to its billing.

When you open the box, you'll notice three items of hardware, a disk, and a loose-leaf manual for any three-ring binder. The hardware consists of the scanner, a power supply and an interface.

The scanner itself looks like any scanner on the market, which is a little misleading. It's fairly big, and has two selector switches, a button, and a contrast dial. "Primax 400 DPI" is printed on the top, but don't think for a second that you are going to get 400 dots per inch from this scanner. I don't even get that from my laser printer! It's obvious that the scanner was made for another platform, but has magically been made to use with our 8-bits. I love ingenuity like this, and it's this kind of engineering that has kept our computers useful. Just look at the success of the Parallel Printer Interface, and you'll see where I'm coming from.

The interface (which is about the size of those old Commodore game cartridges), plugs into the user port. Two receptacles lie at the back of the interface, one for the scanner, the other for the common-looking power supply. There's nothing tricky about installation, it's simple and painless.

The next step is to boot the driver software (which is completely CMD-device compatible). After doing so, you are asked if you want to clear screen memory. If you don't, you begin with pixel filled, black screen, so I always respond with a "yes" to this query. The main screen then appears, and you are ready to begin scanning.

The environment in which your scanning days will begin, is actually two things: a scanning interface, and a paint program. The interface is actually quite simple to use, if you can get past the awkward manual. Translated from German, the manual is disjointed and hard to follow at best. Persistence, careful reading, and experimentation are your best bets to solving this minor mystery. Once you get the basics down, however, things move along rather well.

You have access to several menus within the driver. Iconical options line the bottom of the screen, and change with the press of the left mouse button. The right button activates the icon's function. Sound backwards? Well, it is if you're used to Geos. Pressing RESTORE and the left button simultaneously will reverse the mouse buttons functions.

If you have only have a joystick, you must move the pointer off of the screen and press the button to change menus. Pressing the button when pointing at an icon will activate whatever that particular icon does. Keyboard shortcuts are also available for most functions, but not all.

To activate the scanner, click on the open eye icon, or press the F1 key. You are then allowed to enlarge or reduce the potential image from 30 to 300 percent. Once selected, the screen turns blue, and you can begin pulling the scanner downward to record your image. If your scanner is on, white bars will blink on the screen as you pull. If it is off, you need to press the button on the left side of the scanner, and repeat your pulling move.

Once the end of available memory is reached, the computer will go into a "thinking mode", displaying your scanned image after 10 seconds or so. You can default to this "think-then-display" option early by pressing the RUN/STOP key.

You may notice after your image is displayed, that you are only seeing a portion of it. This is because your image takes up 4 cells (8 if you have the optional PageFox module), which you can scroll around in via the cursor keys, or move directly to with the number keys ("1" takes you to the first cell, "2" to the second, etc.). If you want to see the entire image, click on the preview icon, or press the UP ARROW key (the one used for exponents in BASIC) to get the same thing.

There are several adjustments which lie on the sides of the scanner, but only one has a positive effect. The contrast wheel is the most useful, allowing you to control the amount of light that your scanner sees. Detail can be added or removed from an image using this adjustment. A four-position toggle switch on the left side of the scanner is supposed to turn photos into halftones, but results are horrible. All this seems to do is impose a stipple pattern over pictures, making them unrecognizable. The switch is labeled "Text/Photo", and ironically, the scanner works best when this switch is all the way to the left in "Text" mode. Finally, another four-position switch (labeled "4-3-2-1"), gives the user different sized scanning areas to choose from. Don't ask why (because I have no idea), but the best results seem to come from the "2" setting.

So, what kind of quality do you really get from a scan? Well, a lot depends on the material you are scanning. The bottom line is contrast. The higher the contrast in the image to be scanned, the better the result. As a result, line art (such as comic drawings), translate very well. Photographs, on the other hand, are a bit trickier. If there is ample contrast, the picture may work fine. If the image is flattened or "washed out" due to head-on camera flash, don't count on much detail from the scanner. The worst result is probably from halftone pictures (used in newspapers and some magazines), which look awful. The scanner can pick up all those tiny white spaces between the dots.

The second part of the software package is the paint program, which isn't bad as far as these programs go. All of the basic functions are here, allowing you to draw, fill, edit, cut and paste, and print. I won't go into details about the program, but it is quite functional. Again, some careful study of the manual is in order.

Probably the best use of the paint program is the ability to paste scans together. Why paste? Because even though the physical scanner has a 4" window, the computer only uses half that. As a result, if your image is wider than 2", you're going to have to scan both sides of it, save each one separately, load them into different places, and paste them together. The trickiest part of this process is getting both scans to match, so when you hook them together, your end result doesn't look crooked. One nice option we have is the ability to load a scanned image into Geos, via Joe Buckley's Handy Import. I prefer geoPaint and its many add-ons to the program provided, and this public domain program does the job quite well. While I did not get this program from CMD, they know about it, and it should be shipping with all current scanner orders.

Is there anything I didn't like about the HandyScanner? Not really. Sure, I could nit-pick about the limited scanning area, or the limited resolution, or that cryptic manual, but in this case, the bad here doesn't outweigh the good. My only real complaint is that you cannot use an REU to expand scanning memory. To do this, you must purchase the Pagefox module, for another $150. Ignoring a standard such as the very-popular 17xx series REUs is backward in my opinion, but is the case nonetheless. What's really ironic is the fact that the Pagefox is only 100K, meaning that even the low-memory 1700 REU (128K) could have been used, as well as the 1764 (256K), and the 1750 (512K)

I doubt the HandyScanner has wide appeal. It will probably only interest those such as myself that want to spice up their desktop publishing efforts. It's nice, however, that the product is available, does what it claims, and offers a luxury to the Commodore community that other platforms take for granted. For these reasons, it's a nice addition to any serious users hardware arsenal.

- Scott Eggleston

Graphic Interpretation

by Steve Vander Ark

GETTING STARTED ON USING GEOBASIC

For the last few months, I have been looking forward to working on this column. In a way, it's kind of like when I write reviews of really cool games. I can blithely tell my wife that, yes, I am working, even though it's obvious that I'm having a ball. This column and the next will be devoted to geoBASIC, and I think geoBASIC is loads of fun.

Now, I realize that sounds kind of strange. I'm talking about a form of BASIC, a programming language. What's so exciting about that? Well, let's face it: if you're dead set against the idea of writing your own programs, you won't think geoBASIC is much fun at all. But if the prospect of creating your very own GEOS programs does sound intriguing, you'll find that geoBASIC is as exciting as a new set of Legos.

Before I get down to the nitty gritty of creating a geoBASIC program, I should probably mention one or two other options you might try for GEOS programming. One is geoProgrammer, the high-powered tool folks like Maurice Randall use to create miracles. If you think you're ready to use assembly language and really to dig into the guts of GEOS, this is the package for you. Check out Maurice's excellent column for more about that kind of full-tilt GEOS programming. In this column, though, we'll be sticking with BASIC, which is a more user-friendly programming language with more English-like commands. With geoBASIC, you don't need to know nearly as much technical stuff about the inside of your computer and the GEOS operating system. For a lot of us, including me, that's a good thing.

There is another GEOS compatible version of BASIC called BeckerBASIC. This package, originally released by Abacus Software, is a very detailed version of the BASIC language with many commands for graphics, structured programming, and so on. BeckerBASIC is an

interesting system, one which can be used to create very powerful programs. Unfortunately, the BeckerBASIC system uses a separate module for editing and another for running your program. These modules take time to load each time you want to see how you're doing, which takes away from the interactive nature of programming. Those separate modules slow down the process considerably.

That's not all. BeckerBASIC does include commands to create GEOS-style drop-down menus and dialog boxes, but in order to use them you need to specify things; like how many pixels tall to make them. That means you really have to plan every bit of your menu out in advance, maybe even on graph paper, to know what numbers to feed the commands. On the other hand, geoBASIC provides you with an editor which creates the menu as you go. If you want to change the text on the menu item, just type it. Decide to add a menu entry? Just increase the number by clicking on an arrow and another entry slot appears. As you make changes, your menu appears at the top of the editor screen so you can actually try it out and change things as necessary. The whole process is very interactive. In fact, for die-hard GEOS folks like myself, being able to create real GEOS menus and dialog boxes on the fly is a kick. (Remember, I said this was going to be fun!)

There's an even more important distinction to make here between geoBASIC and BeckerBASIC. It has to do with the way you structure a program. BASIC usually is a very linear language, with your computer executing one command after another in the order they appear in the program listing. Once in a while the computer may stop for user input and it will likely loop or jump around a little, but it's still essentially working in one long sequence of commands.

GEOS itself, on the other hand, doesn't work that way. The structure of a GEOS program isn't linear and the computer spends relatively little time executing commands. Most of the time, GEOS just sits there, waiting. What is it waiting for? For something to change. That change might be a mouse click or a keypress or any number of things, often generated by the user. When GEOS sees that change, it checks what that means it's supposed to do, then jumps to the appropriate routine and runs it. GEOS might be sitting there waiting, for example, when you click your mouse pointer on a button. Immediately GEOS jumps out of its wait mode and hops to the series of commands assigned to that particular button. Something happens—maybe a dialog box closes—and then GEOS goes back to waiting for the next event to occur.

I chose the word "event" intentionally, by the way, since programs designed this way are called "event-driven" programs. Event-driven programs are the kind you create with geoProgrammer as well. In fact, the GEOS operating system itself is event-driven. With geoBASIC, that's the kind of program we can write: an honest-to-goodness, event-driven, GEOS program.

Okay, let's get to work. If you're an experienced BASIC programmer, you'll feel right at home using geoBASIC. Many of the commands are exactly the same as those used with any other form of BASIC. In fact, if we wanted to, we could simply write this program the way we'd write any other BASIC program. Like I said, though, we're going to write a "real" GEOS program, so it's going to take on a new, event-driven structure. The program we'll create is very simple, but it will demonstrate the way a geoBASIC program is laid out. We'll include a drop-down menu, a dialog box, and some other GEOS-style features.

The program will start with an opening screen. This could be simply the main program screen itself, but since we want to try a few tricks, we'll create a title screen first, with a button to press to start the program itself. That's when our main program screen will appear, and from then on the computer will sit in MAINLOOP and wait for an event to happen. That MAINLOOP command is the heart of a geoBASIC program, just like I mentioned above. Each section of the program will be labeled so we can tell the computer where to go when we need to. Here's how it will look:

@titleScreen - a routine to create the title screen with two buttons:
  title1 - ties to @quit
  title2 - ties to @mainScreen

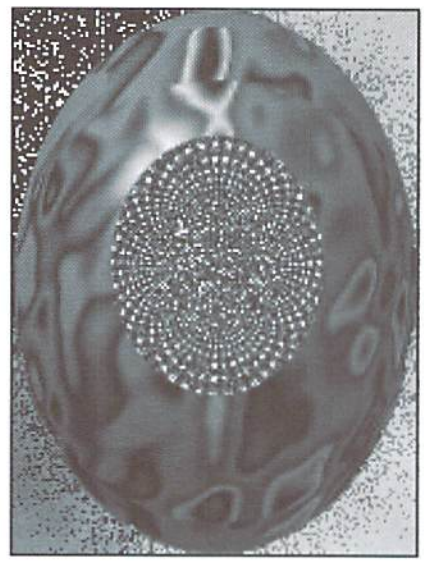
MAINLOOP - the program waits at this point for a button or menu item to be selected. The rest of the program will consist of the routines which those buttons and menu items call up.

@mainScreen - a routine to draw the main program screen and sets pattern and color to their starting values and displays the following buttons and menus:
- a set of three drop down menus:
  - a "geos" submenu that contains:
    - available desk accessories (listed automatically)
    - "program info" - tied to @about
  - a "file" submenu with two choices:
    - "quit" - tied to @quit
    - "start over" - tied to @mainScreen
  - an "attributes" submenu with two choices:
    - "color" - tied to @colChoice
    - "pattern" - tied to @pattChoice
- four icons:
  - draw1 - tied to @drawRect
  - draw2 - tied to @drawLine
  - draw3 - tied to @sayHi
  - draw4 - tied to @scrnClear

@quit
@colChoice
@pattChoice
@drawRect
@drawLine
@sayHi
@scrnClear
@about

If we've planned everything out correctly, this program will run just fine, since for every possible event we have defined a routine. There is also a way to end the program and a way to go back and start over. With this structure in place, we're ready to go.

Once our plan is made and our routines labeled, we can start geoBASIC and use the built-in editors to create some of the pieces. Specifically, we'll create our drop-down menu for the main screen with the menu editor, we'll use the bitmap editor to design our icons, and then we'll use the dialog box editor to create a "do you really want to clear the screen?" message to include in our @scrnClear routine. In order to insert them into our BASIC program, we'll have to name them. We'll use "mainM" for the menu, "icons1" and "icons2" for the set of icons, and "clrD" for the dialog box. These filenames, by the way, can't be longer than five characters.



In the menu editor, accessible from the "utilities" drop-down menu on the geoBASIC text screen, we'll create our new file and call it "mainM." The editor starts us out with a generic menu at the top of the screen with four sub-menus. We'll cut that down to three by using the mouse to adjust the number next to "Number of submenus" on the editor screen. Notice that the menu at the top of the screen immediately adjusts to these new parameters. Using the editor's tools, we'll change the title of the second submenu to "file" and the third one to "attributes." Again, the menu at the top of the screen adjusts. You can go ahead and try it out and you'll see that we also need to change the number and labels of the items in those submenus. We do that using the editor tools as well. Let's change the number of items under both "file" and under "attributes" to two. Now when we try out our menu up on top of the screen, we'll see pretty close to what our plan called for.

All that's left to do is adjust those items to show the correct text and to tell them which routine they're supposed to call up. You can do this by clicking the mouse on the menu at the top of the screen, calling up a submenu's list of items, then clicking on the item itself. Using this method, we'll change the "file" submenu items to "quit" and "start over" and assign them the names that we chose in our plan. We'll do the same with the "attributes" and "geos" submenus. We'll change the "attributes" items to "color" and "pattern" and assign the correct routines to them, then set the "program info" item to our information routine. Then we'll quit the editor so the whole thing is saved under the name "mainM."

From there we'll head over to the bitmap editor. We need to create the icons which will appear on the title screen and on the main screen. To keep things easy, we'll stick with the default size of the editor. For the first set, create two bitmaps under the names "title1" and title2" with the words "quit" and "go" on them. Next, we're going to need four tool icons, which we'll name "draw1," draw2," "draw3," and "draw4."

Now I'm sorry, but there's no way for me to tell you exactly how to make these bitmaps. Here's where you'll have to just use your own creativity to create images of four buttons: the first one for drawing a rectangle, the second one for drawing a line, the third one for printing "HI" on the screen, and the fourth one for clearing the drawing area. When those four button pictures are created and saved, we'll switch over to icon list editor to specify their location on the screen and tell them which routines to call up. We'll place them at Y position of 24 and X positions of 8, 32, 56, and 80. This final file we'll save under the name "icons2."

From this point on we'll just be sticking all these pieces together into one program, following our initial plan. Everything will plug right in, just like an electronic Lego set. Or will it? GeoBASIC can throw you for a loop now and then. Next issue we'll try to get everything working the way we want it.

GEO PROGRAMIST

by Maurice Randall

BACK TO BASICS

Let's step back a bit and talk about basic program development. We must remember that GEOS programs are event driven. That is the biggest step to overcome for a new GEOS programmer. When writing a program outside of GEOS that needs user input, we would normally create a loop that checks for whatever input it is that we are looking for, that normally being the press of a key. It might go something like this:

Step A: Check for a key to be pressed.
Step B: Was a key pressed?
Step C: If yes, then handle the keypress.
Step D: Do some other stuff that needs to be done.
Step E: Return to Step A.

The example above is a simple flowchart of what is referred to as a 'MainLoop'. In most cases, it is much more elaborate than that. GEOS has its own elaborate MainLoop so that we don't need one in our GEOS application. The software for watching the keyboard is already there. It would be wasted programming to write our own MainLoop. In fact, the GEOS MainLoop also helps keep an eye on the mouse pointer for us.

Once an application is running on the screen, it will get to a certain point and then it is not really running anymore. That's right, the application might be visible on the screen, but the code that is now running is actually the GEOS MainLoop. When the application is first loaded, GEOS will jump to the start of the application, wherever that might be and the program code found there will be executed. Your program code there would normally be code that would initialize the application. There would naturally be routines called to draw the screen as needed, a variable area might get initialized or cleared, and maybe some additional code from another file or a VLIR record could be loaded at this point.

This initial routine would eventually end with an RTS. Outside of GEOS, what would happen after this RTS was encountered? We would bounce right back to BASIC and that would be the end of our program. Outside of GEOS, we would not want an RTS there but instead would have to loop back or jump to somewhere and wait for the user to do something, depending on the nature of the program. Not so with GEQS. This RTS will take us into the GEOS MainLoop. GEOS will now wait for the user to do something.

That's fine, but how does GEOS know what to do if the user clicks the mouse or presses a key? It knows by examining a series of address vectors for valid addresses. We would load these various vectors with addresses during that first routine that is called in our application. We would also set up some tables and inform GEOS of the location of these tables. There are two types of tables, one for icons and one for menus. If the mouse is clicked, GEOS will check the current location of the mouse and then examine these icon and menu tables. If it determines that an icon or menu was clicked on, it will call a routine that we have listed in the table that corresponds with that icon or menu. The routine in our application performs what it is designed to do and then upon exit, the GEOS MainLoop will once again continue.

If the user presses a key, GEOS will examine the vector known as 'keyVector'. If we have put an address of one of our routines at this vector, then our routine will get called. Let's take a look at a very simple example, in which we merely want to check for the user to press the 'Y' or the 'N' key. Here's the routine that will set things up:

```
YNSetup:
   jsr i_PutString
   .word 100
   .byte 100
   .byte "Continue? Y/N",0
   LoadW keyVector,#TestYN
   rts
```

This routine will display the 'Continue? YN' message beginning 100 pixels down and 100 pixels across on the screen and then load keyVector with the routine that we want called if the user presses a key. The RTS took us back to the GEOS MainLoop. GEOS is watching the keyboard for us now. Prior to this routine, keyVector contained the address $0000. This is how GEOS initializes it just before our application first starts up. If the user presses a key and keyVector is pointing to $0000, then nothing will happen. But, if it points to anything else, GEOS will call the routine at that address. In our case, it is going to call 'TestYN' as soon as the user presses a key. Now all we need is a routine to check the key that is pressed. Any keypress will cause our routine to be called. But the only keys we are interested in are the 'Y' and the 'N' key. Prior to calling our routine, GEOS MainLoop will store the keypress in a GEOS variable known as 'keyData'. The value that we will find there is actually the ASCII value that is represented by that keypress.

```
TestYN:
   lda keyData
   cmp #96
   bcc 10$
   and #%11011111
10$
   cmp #'Y'
   bne 20$
   jmp NextPhase
20$
   cmp #'N'
   bne 30$
   jmp ThisPhase
30$
   rts
```

The example first converts the keypress to uppercase prior to testing it. Our example doesn't care if the CAPS LOCK key is pressed or not. If the keypress is a 'Y', then the routine will jump to our examples NextPhase or wherever you want. If the 'N' key is pressed, then we will jump to ThisPhase, which might be the point of our program that is already running. If neither key is pressed, then the routine does nothing and the RTS will return us to the GEOS MainLoop.

There are other vectors that we can use, some useful and some that are very rarely used or needed except for very special applications. Let's take a look...

appMain - This allows us to add routines to the GEOS MainLoop. When the GEOS MainLoop ends, the routine that application Main points at will get called. For most purposes, this vector is not needed since most functions are quite well handled already. When our routine is finished, GEOS MainLoop will once again return to its beginning.

inputVector - This is another very rarely used vector. It is called if any change occurs with the input device. If the mouse is moved, the routine at this vector will get called. You might use this if you need to move an object around on the screen.

mouseFaultVec - When an area has been defined by your application in which to confine the mouse, this vector is used if an attempt is made to move the mouse outside of the defined region.

otherPressVec - If GEOS finds that the mouse was pressed and the location of the mouse is not over an icon or a menu, then this vector is used. You might take advantage of this vector and supply a routine to test for a mousepress on certain parts of the background or any region on the screen.

StringFaultVec - If you are displaying text to the screen using GEOS routines such as PutString, the address at this vector will get called if the text hits the right margin. The right margin defaults to the right edge of the screen but may also be changed within your program.

There are also two vectors that are checked during each interrupt. These are intTopVector, which is called at the start of the interrupt sequence, and intBotVector, which is called at the end of the interrupt sequence.

As you can see, there are many ways to get access to different parts of our application. It all depends on the design of the application and what it is intended to perform. GEOS makes it easy to get a response from the user through the mouse or the keyboard and will interact with our application in whatever manner we desire.

HARD TIPS

DISABLE THE BUILT-IN DRIVE ON YOUR COMMODORE 128D

By Al Anger

Okay, so you've installed device switches for the 1571 in your C 128D. Maybe you've also installed a momentary on switch to reset the drive after changing the device number (to replace the hard to use drive reset switch that comes with the D). What if you're lucky enough to have a RAMLink, a CMD Hard Drive, a CMD FD-4000, and a C 1581, and you'd like to use them with Geos which has a limited number of drives allowed? What if you'd like to not have internal drive recognized by your computer? With the faster, larger drives and RAM devices available, the 1571 in your 128D might need to be disabled now and then. Well folks, here's the answer.

Legal Mumbo Jumbo

Before we get started, a word of caution and a disclaimer. Inside your C 128D are voltages which can harm you. Also, you can have static charges in your body which can harm your 128D. If your unsure if you can complete this short project, get help from a friend, or go to a computer dealer/repair center. Neither Creative Micro Designs, Inc. nor the author of this article shall be responsible for the use or misuse of information in this article. The information provided has been tested and is believed to be correct. Now on to the fun stuff.

One More Switch

Here's the procedure for installing a drive disable switch on the front panel of the C 128D. Parts required for this job are:

- Soldering pencil
- Fine electronic solder
- 1 Miniature Toggle Switch (spdt)
- 1 resister, 10K Ohm 1/4W
- 3 ft. (approx.) of 28 gauge stranded hookup wire
- A drill and bit to provide the hole to mount the switch in
- A small pair of diagonal cutters (nippers)
- An Exacto Knife

Lets get started. Open the 128 D. Remove the screws from the rear and bottom of the unit. Slide the cover back, and remove it. Next, remove the 1571 drive unit. Remember the orientation of the connectors and remove them. Remove the drive mount screws and slide off the latch lever. Slide the drive unit out, and set it aside. If you've installed device selection switches before, all of this will be easy. If not, take your time and make notes.

One of the six lines in Commodores famous (or infamous) serial bus is labeled ATN for "attention". This line is used to get the attention of all devices on the serial bus. The host computer brings this line low, which in turn generates an interrupt on the drive controller board. The ATN signal is followed by a device number. It works a little like a roll call. The computer calls all the serial devices to attention and looks for the device requested. If the device doesn't respond, the computer assumes it isn't there. You can switch the ATN line in and out for any serial device. I use a couple of rotary switches to allow me to select from a 1541, a 1571, and a 1581 as device 8, and the same for device 9. These rotary switches allow me to bring the ATN line to the drive of my choice. Our goal is to switch out the ATN line for the internal 1571 so it doesn't answer the roll call.

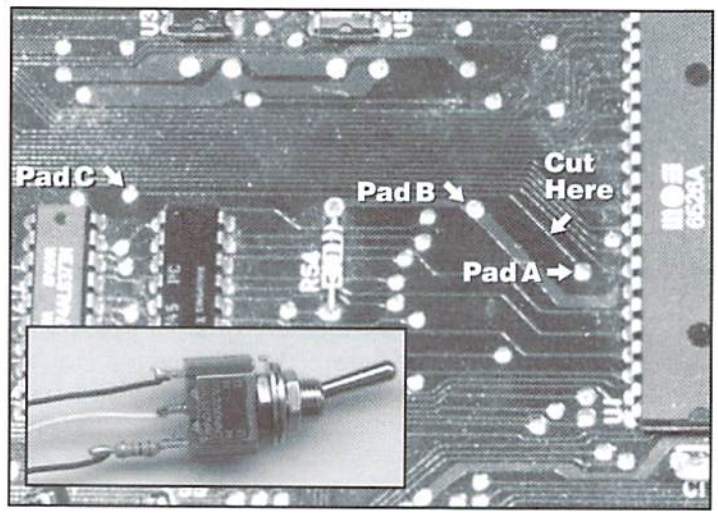
Enough theory, let's solder something! First, solder the resistor to one of the outside terminals on the switch. Next, cut the hookup wire into three equal sections of approximately 1 ft. each, strip the ends back about 1/16th of an inch, and solder one end of two of the wires to the two remaining terminals on the switch. Solder one end of the remaining wire to the open end of the resistor. Cut away any excess leads or wires so that your switch and resistor assembly look like the one in the inset picture below.

Now refer to the picture and locate IC U1 on your 128D motherboard. This is a large (40 pin) 6526 chip near the right center of the board (shown at the far right in Figure 2). Using IC U1 as a reference point, locate the three through-hole solder pads identified as Pads A, B and C in the picture. Notice that a trace runs between Pads A and C. Using an Exacto knife, cut this trace somewhere near Pad A (there's a little more clearance at that end of the trace). Make sure your cut creates a good solid break in the trace (you can verify this with an ohm-meter or continuity-checker if you have one). Now solder the wire connected to the center terminal of the switch to Pad A; solder the wire connected to the resistor to Pad B; finally, solder the remaining wire to Pad C.

Locate an appropriate location for mounting the switch on the front panel of your 128D, drill a hole, and mount the switch (you can feed the switch through one of the large holes in the metal frame of the 128D).

Reinstall the drive, and front panel if you removed it, and test your work. Then close up the 128D. If you close up the 128D before testing, something will be wrong and you'll need to reopen it (Murphy's Law being what it is.)

If you have questions or comments, or would like this or some other 'hack' done to your computer for no charge (you pay shipping, insurance, and parts), I can be reached at (305) 233-4689 between 4 p.m. and 8 p.m. ONLY (eastern standard time). I can also be reached on GEnie (A.Anger1) or A.Anger1@genie.geis.com or on DiamondBack BBS at (305) 258-5039 as Wile E. Coyote.



BASIC INSTINCTS

by Gene Barker

USING BASIC'S LOAD: PART II

In this two part series, we focus on BASIC's powerful LOAD statement and its use in our programs. This second issue focuses on using LOAD to retrieve data and place it memory for use by our programs. This data could be anything from small machine language programs to sprites to custom character sets. In fact, in this issue we create a custom screen maker in BASIC which allows us to create and save custom screens for use in our programs. Enough talk, let's program!

LOAD With A Twist

Have you seen this command before?

```
LOAD"DATA",8,1
```

What in the world does that ",1" mean? Good question. The ",1" tells BASIC to load the "DATA" file back to the same place in memory from which it was saved. If we forgot to add the ",1" BASIC would load "DATA" as if it were a BASIC program and place it in program memory and crash!

Now that we have a general idea of what that ",1" means, let's review the next example:

```
100 REM PROGRAM ONE
110 LOAD"DATA",8,1
120 PRINT"FILE LOADED"
```

Using what we learned from the last issue, we would guess that Program One would never make it to line 120. If the file "DATA" is a BASIC program the example would work as we discovered in the last issue. So what would happen if the file "DATA" is not a BASIC program? Surprisingly, Program One would restart from the beginning after loading "DATA" resulting in an infinite loop; loading the file over and over again. To solve the problem using the technique we reviewed in the last issue, we might code:

```
100 REM PROGRAM TWO
110 IF A=999 GOTO 130
120 LOAD"DATA",8,1:A=999
130 PRINT"FILE LOADED"
```

However, sometimes this technique can be difficult to follow; especially when your LOAD statement is dozens of lines away from the first line of your program. Even more daunting, what if your program loaded several different data files depending upon which menu you were in? Keeping track of those GOTO statements would be a nightmare.

I Have A Better Way

Thanks to the tight integration between Commodore BASIC v2.0 and its Kernal, we can access your Commodore's LOAD routines directly. See the example below:

```
100 REM PROGRAM THREE
110 POKE 147,0
120 SYS 57812 "DATA",8,1
130 SYS 62631
140 PRINT"FILE LOADED"
```

The advantage to Program Three is that the program executes line by line without any need to detect whether the file has already been loaded (as does Program Two). You will notice that we used this improved method in our demonstration program.

Custom Screen Maker Explained

The demonstration program, "LOAD2.BAS" allows you create and save custom screens for your BASIC programs. We also included "SCRSUB.BAS". This is a small subroutine which you can add to the end of your own programs so that custom screens made with the "LOAD2.BAS" can be loaded and displayed at will. Additionally, you will notice that these programs use several of the techniques we covered in the last couple of issues.

Screen Data File

In order to save a screen we need to save the characters on the screen located from PEEK(1024) to PEEK(2023), the colors of those characters located from PEEK(55296) to PEEK(56295), the border color at PEEK(53280), the screen color at PEEK(53281), the menu text color, and the current character set. Let's review how we store these values in memory. Review the Screen File Layout.

Screen File Layout

```
MA = 49152 (or $C000)
MA + 0 Screen Memory Buffer 40 columns x 25
lines (0-255)
MA + 1000    Color Memory Buffer 40 columns x 25
lines (0-15)
MA + 2000    Border Color (0-15)
MA + 2001    Screen Color (0-15)
MA + 2002    Menu Text Color (0-15)
MA + 2003    Character Set CHR$ Value
```

14 Lowercase Character Set

142 Uppercase Character Set

Color Conversion

Unfortunately, the value to change the cursor color using CHR$ is different than the value we poke to color memory. In order to account for these differences, we created a color conversion chart in the form of a BASIC integer array CO%. Please review the Color Conversion Chart. See if you can follow how "LOAD2.BAS" uses the CO% array to implement this conversion in lines 135 to 165 and line 315. See Color Conversion Chart.

Character And Color Values

When poking characters and colors to memory, it's important to keep these values within a range that is meaningful to your Computer. For a character, there are 256 possible characters you can POKE to screen memory (values 0 through 255). For a color, there are only 16 possible colors you can POKE to color memory (values 0 through 15). Furthermore, BASIC can not POKE an integer less than 0 or greater than 255 to memory. In lines 410 to 465, great care is taken to insure that we adhere to these rules.

Notes

Put the SAVE code in lines 640 to 660 and the cursor code in line 1060 on the shelf for the moment until we have a chance to cover them in more detail at a later time.

As you type in this issue's programs, take it one section at a time. Try to get a general idea of what the section is trying to accomplish. See if you can follow how the section is using its variables. If you see an unfamiliar BASIC statement, take a quick look at it in your BASIC manual. If you are still confused, move on to the next section; often times the next section helps explain the previous one. Above all, don't forget to BACKUP your work frequently.

Entering The Program

Before entering this issue's programs, load and run the CHK-LIST utility (located elsewhere in this issue). CHK-LIST insures that you enter the programs correctly the first time. Also, remember to SAVE the programs before you attempt to RUN them.

Color	POKE Value	CHR$ Value
Black	0	144
White	1	5
Red	2	28
Cyan	3	159
Purple	4	156
Green	5	30
Blue	6	31
Yellow	7	158
Orange	8	129
Brown	9	149
Light Red	10	150
Dark Gray	11	151
Medium Gray	12	152
Light Green	13	153
Light Blue	14	154
Light Gray	15	155

(Program listings are located on the following page)

LOAD2.BAS

```
5000 100 rem--------------------------------
6dc2 105 rem commodore world magazine
c67c 110 rem basic instincts w/gene barker
ad3e 115 rem using basic's load part ii
0e6b 120 rem
2d24 125 rem (c)1995 creative micro designs
92ad 130 rem--------------------------------
1315 135 rem-
ff19 136 rem initialize chr$ color array
d9e4 137 rem-
3a97 140 dim co%(15)
7fbb 145 for i=0 to 15
69ba 150 : read co%(i)
7453 155 next
1ca8 160 data 144,5,28,159,156,30,31,158,129
2523 165 data 149,150,151,152,153,154,155
7e06 180 rem-
f280 181 rem allow all keys to repeat
7e06 182 rem-
adc4 185 poke 650,128
c1cc 200 rem-
fa3e 201 rem set screen defaults
c1cc 202 rem-
c670 210 s0 = 11:rem border color
1bb9 215 s1 = 0 :rem screen color
369e 220 mc = 13:rem menu text color
e8c5 225 cs = 14:rem lower/upper case chr$
4478 245 ma = 49152:rem set memory address
f963 250 gosub 10000:rem clear intial screen
5291 300 rem-
1e56 301 rem display main menu
5291 302 rem-
6bf8 305 poke 53280,s0:rem set border color
d573 310 poke 53281,s1:rem set screen color
ef46 315 print chr$(co%(mc)):rem set text co
620b 320 print chr$(cs):rem select char set
af66 350 print"{CLEAR/HOME}{CRSR DN}{CRSR RT}
     main menu"
9624 355 print"{CRSR DN}{CRSR RT}e)dit screen
     {4 spaces}1)border color"
863d 360 print"{CRSR RT}v)iew screen{4 spaces
     }2)screen color"
166a 365 print"{CRSR RT}c)lear screen{3 space
     s}3)menu color"
7f02 370 print"{CRSR RT}l)oad screen{4 spaces
     }4)char set"
aab3 375 print"{CRSR RT}s)ave screen"
4e18 380 print"{CRSR DN}{CRSR RT}q)uit"
83c7 400 rem-
8450 401 rem get user choice and act
83c7 402 rem-
78b1 405 get x$:if x$ = "" then 405
cb2e 410 if x$<>"1" then 430
45ec 415 : s0 = s0 + 1
15a0 420 : if s0>15 then s0 = 0
54cb 425 : goto 300
1db0 430 if x$<>"2" then 450
1b4a 435 : s1 = s1 + 1
470d 440 : if s1>15 then s1 = 0
9bda 445 : goto 300
61ef 450 if x$<>"3" then 470
0f9a 455 : mc = mc + 1
39de 460 : if mc>15 then mc = 0
58ec 465 : goto 300
b81c 470 if x$<>"4" then 490
565d 475 : if cs = 14 then cs = 142:goto 300
a5f4 480 : cs = 14:goto 300
6d23 490 if x$<>"e" then 500
1c6f 495 : goto 1000
4acd 500 if x$<>"v" then 520
5c00 505 : gosub 10100
```

LOAD2.BAS *(cont.)*

```
7059 510 : get x$:if x$="" then 510
1318 515 : goto 300
7eaa 520 if x$<>"c" then 540
44da 525 : gosub 10000
abc5 530 : goto 300
9069 540 if x$<>"l" then 600
d48e 545 : print"{2 CRSR DN}load screen - ";
9c93 550 : gosub 10200
d5d1 555 : if f1$="" then 300
dbb8 560 : rem load command
c82a 565 : poke 147,0
c36b 570 : sys 57812 f1$,f1,1
427a 575 : sys 62631
4a87 580 : s0 = peek(ma+2000)
94d7 585 : s1 = peek(ma+2001)
9f96 587 : mc = peek(ma+2002)
67e2 590 : cs = peek(ma+2003)
8596 595 : goto300
e9df 600 if x$<>"s" then 700
cca2 605 : print"{2 CRSR DN}save screen - ";
8901 610 : gosub 10200
dee9 615 : if f1$="" then 300
e8a6 620 : rem the save command
3ded 625 : poke ma+2000,s0
738e 630 : poke ma+2001,s1
10aa 632 : poke ma+2002,mc
10cf 635 : poke ma+2003,cs
fc34 640 : rem save command
4b47 645 : sys 57812 f1$,f1,1
48ac 650 : poke 193,0:poke 194, 192
1f24 655 : poke 174,0:poke 175, 200
fd83 660 : sys 62957
40e6 665 : goto 300
e563 700 if x$<>"q" then 400
f85a 705 : end
d85a 1000 rem--------------------------------
04cc 1001 rem edit screen routine
d85a 1002 rem--------------------------------
ea9b 1005 rem display exit instruction
dd18 1010 print"{CLEAR/HOME}{10 CRSR DN}{7 CR
     SR RT}{CMDR a}{24 SHFT *}{CMDR s}"
bef3 1015 print"{7 CRSR RT}{SHFT -}Press (ret
     urn) when done{SHFT -}"
c642 1020 print"{7 CRSR RT}{CMDR z}{24 SHFT *
     }{CMDR x}"
5f16 1050 gosub 10100:rem display screen
d47f 1055 print"{HOME}";:rem set cursor at ho
     me
4a25 1060 cp=(256*peek(210))+peek(209)+peek(2
     11)
17a6 1065 oc=peek(cp)
a516 1070 nc=peek(cp)+128
b289 1075 if nc>255 then nc=nc-256
9960 1080 poke cp,nc
9edd 1085 get x$:if x$<>"" then 1100
f01a 1090 for i=1 to 100:next:goto 1070
3a99 1100 poke cp,oc
1d4d 1105 if x$<>chr$(13) then 1120
177e 1110 : gosub 10300
1052 1115 : goto 300
e55c 1120 print x$;
c9fe 1125 goto 1060
548b 10000 rem--------------------------------
ed8d 10001 rem clear the screen in memory
548b 10002 rem--------------------------------
f1a0 10005 for i=0 to 999
b149 10010 : poke ma+i,32:rem clear ch
a9b8 10015 : poke ma+i+1000,mc:rem clear col
2335 10020 next
8841 10025 return
20b8 10100 rem--------------------------------
```

LOAD2.BAS *(cont.)*

```
aa54  10101 rem display the screen in memory
20b8  10102 rem------------------------------
aedd  10105 for i=0 to 999
9549  10110 : poke 1024+i,peek(ma+i)
a5ae  10115 : poke 55296+i,peek(ma+i+1000)
fff5  10120 next
5481  10125 return
378a  10200 rem------------------------------
d812  10201 rem get filename and drive num
3497  10202 rem and place in fl$ and fl
ba56  10203 rem------------------------------
e8ed  10205 print"enter filename"
7dda  10210 print"(return) to abort
84fc  10215 input fl$
1ada  10220 if fl$="" then return
717a  10225 print"enter drive number"
6f3d  10230 print"(return) to abort"
6df2  10235 fl=0:input fl
71b6  10240 if fl<8 or fl>16 then fl$=""
ab74  10245 return
6fc6  10300 rem------------------------------
d49a  10301 rem save screen to memory
6fc6  10302 rem------------------------------
b6c8  10305 for i=0 to 999
f087  10310 : poke m a+i, peek(1024+i)
e646  10315 : poke ma+i+1000, peek(55296+i)
5ece  10320 next
f5ba  10325 return
```

Don't forget to SAVE!

Peripheral Vision

By Jim Butterfield

SERIAL BUS BASICS

Your printer and disk connect to the computer by means of a "serial bus". Let's take a look at how this works. When your BASIC program says, for example:

```
PRINT#2,"X";
```

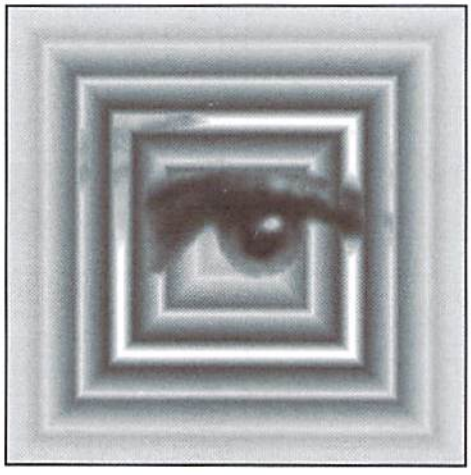
the computer sends out a call to all devices that are hooked up to the serial bus. It sends a signal which says "I want to talk to device number N." N might be a number from 4 to 15; it certainly won't be 2, which is Basic's "Logical File Number" in the example just given.

All connected devices read this signal. One of these devices says, "Hey! That's me!" and stays on the line in "listen" mode. The other devices note that their number was not used, and go back to sleep. The computer then sends the letter X to the bus. The listening device receives it and acknowledges it. All other devices stay asleep. Finally, the computer calls all devices again, and says, "I want device number N to stop listening". The selected device notes this, and switches off its "listen mode." The bus becomes quiet again.

A similar sequence of events takes place when Basic executes a GET# or INPUT# statement. The appropriate device is asked to "talk", information comes in from it, and then the device is told to "un-talk".

Speed and Efficiency

Our example above sent a single character, X, to the printer or disk. Yet the serial bus had to carry at least three characters to do the job: the TALK selection, the data character, and the UNTALK selection. There might be a need to send a fourth character, the secondary address.



The ratio of four characters sent to one delivered isn't the most efficient concept, of course. Things get better when you user a PRINT# line that sends a lot of stuff. The appropriate device is selected at the start of the action, all the data characters are sent, with the de-select coming at the end. Instead of an efficiency of one in four, we're likely to get something like 47 data characters in a transmission stream of 50.

Machine language programs are more specific in the way they trigger TALK, LISTEN, UNTALK, and UNLISTEN. To open a device on the serial bus, you would call either CHKIN (at $FFC6) to prepare for input, or CHKOUT (at $FFC9) to get ready for output. These are the same calls you would use to send to non-serial devices, but the ROM logic spots that the serial bus is involved and sends TALK or LISTEN together with the secondary address if appropriate. The channel is left open for data (in or out) until the program calls CLRCHN (at $FFCC), which will arrange for an UNTALK or UNLISTEN to be sent.

After we pick at serial bus mechanisms a little more closely, we'll look into recommended methods of making the bus more efficient. But first, let's look at a little bus history.

Bus Beginnings

First of all, let's clear up some terminology. In the world of Commodore 64 and 128, we use the term "serial bus" to describe the connection that hooks up our peripherals, printer and disk, to the computer. Our communications system connection is called "the RS-232 bus". Yet, from a technical standpoint, it's also a serial bus.

The term "serial" refers to any system in which the bits of data march along one behind another. That's true of a telecommunications connection and of similar devices; a serial printer or a serial mouse, for example. The term usually contrasts with "parallel", where the information moves along side by side on separate wires.

"Serial" is something of a generic term. So don't expect another brand of computer that brags it has "two serial ports" to be able to hook up to your 64 printer or disk. The general method is the same, but the details are quite different.

The earliest Commodore machines connected to peripherals through a parallel bus known as the IEEE-488. It was an international standard for instrumentation. But the cables needed by this bus were heavy, costly, and clumsy. During one unhappy period of time, they were also hard to get, to the extent that Commodore couldn't hook their computers to their disk drives! The switch was made to the serial bus, which used lighter cables, cost less, was easier to obtain, and was intended to be just as fast. We'll mention the reason for slowdown a little later.

So the protocols of the serial bus, talk, untalk, listen, unlisten, are taken almost exactly from the original IEEE specification. If you happen to have an early-model Commodore computer, you'll see the original bus in action.

Speed Considerations

Every PRINT#, GET#, or INPUT# that reference a serial bus device starts with a TALK or LISTEN command, perhaps followed by a secondary address character, and ends with an UNTALK or UNLISTEN. Even a command such as PRINT#2,"" which sends no data at all, still sends two or three control characters (note that the command ends with a semicolon, so that a RETURN is not sent.)

Few programs send data one character at a time, but it's possible. For example, a simple file copier might do the job character by character. But if you have such a program, it might be well to look at the alternative: grouping the characters into a string and sending them together.

"Difficult" files are often read with the GET# statement, since INPUT# has several limitations. But it seems at first as if you can GET only one character at a time, and that would be a major slowdown. Here's the work around: try grabbing several characters in one statement, for example:

```
GET#2,A$,B$,C$,D$,E$
```

This will speed things up noticeably. The only possible tricky bit is detecting end-of-file. If the EOF indication came as you were bringing in character C$, you'd get "fill" values for D$ and E$, the RETURN character. This is not usually a problem if you know how your file is set up.

If you work in machine language, you'll do well to gather your information into a "buffer" area. For input, send the CHKIN to connect to the device. Read data characters with GETIN or CHRIN ($FFE4 or $FFC6) until the buffer is full, the end-of-file is reached, or a selected character is received. Only then send CLRCHN to tell the device to UNTALK. For output, gather your data into a buffer and when it's ready, send it out using CHKOUT, a number of CHROUT calls (at $FFD2) and finally CLRCHN.

The CMD Command

When the CMD statement names a serial bus device, a LISTEN command is sent out, and the bus is left "open". Any characters that are normally directed to the screen will be sent down the bus. Eventually, the CMD condition is canceled by the use of a PRINT# statement, which always finishes by sending an UNLISTEN to the device if it's on the serial bus.

At first glance, this seems like an ideal command for speed. Send as much as you like, with virtually no overhead. Here's the problem: While the CMD is in force, you won't be able to work any other devices on the serial bus. So CMD is usually reserved for its original purpose: sending program listing to disk or printer.

Machine Language Addendum

If you snoop the Kernal's call set, you'll find several calls that deal with the serial bus, yet I haven't mentioned them above. They include such things as:

ACPTR ($FFA5) - gets a byte from a connected device.
CIOUT ($FFA8) - sends a byte to a connected device.
LISTEN ($FFB1) - tells a device to LISTEN.
SECOND ($FF93) - send LISTEN secondary address to device.
TALK ($FFB4) - commands a device to TALK;
TKSA ($FF96) - sends TALK secondary address to device.
UNLSN ($FFAE) - tells device to stop listening.
UNTLK ($FFAB) - tells device to stop talking.

You will rarely need to use any of these. CHKIN, CHKOUT, and CLRCHN will do the work better and avoid conflict between devices. About the only serious use I have seen involves checking something on a disk command channel, when that channel is not known to be open. In that case, it's a good idea to slip information directly to the serial bus rather than to attempt an OPEN/CLOSE sequence that might muddle up other files in progress.

Quick Tech Notes

As the name "serial" implies, characters are sent one bit at a time over a single wire. Each character is acknowledged with a "handshaking" line. The individual bits were originally intended to be caught in a "shift register" circuit; but at the last moment in VIC-20 design, it was discovered that the 6522 VIA chip couldn't handle the job. As a result, the serial bus suffered a great loss in speed, to allow the processor to catch the bits as they came in. "Selection" signals such as TALK and UNTALK are distinguished from data by means of a special ATN ("attention") line. When this line goes hot, all devices listen for commands.

End of file, or more accurately EOI ("end or indicator") is signaled by slowing down part of the transmitted character. You may recall that this signal may also indicate end-of-record during a relative file read.

It's possible to tell one device to TALK and another to LISTEN at the same time, in which case data will pass directly between the two devices without needing the computer. But all reports say it's tricky and unreliable and, of course, the computer would lose its controlling function.

Closing The Loop?

Commodore's IEEE and serial bus concept was an idea before its time. In the very near future, a single bus to connect all peripherals will be trumpeted as a "brand new concept" for microprocessors. Intel will be building the Universal Serial Bus (USB) and you'll also see the IEEE 1394 "FireWire" bus gaining in popularity. These are much faster than Commodore's bus, but hey, we were there first.

Carrier Detect

By Gaelyne R. Moranec

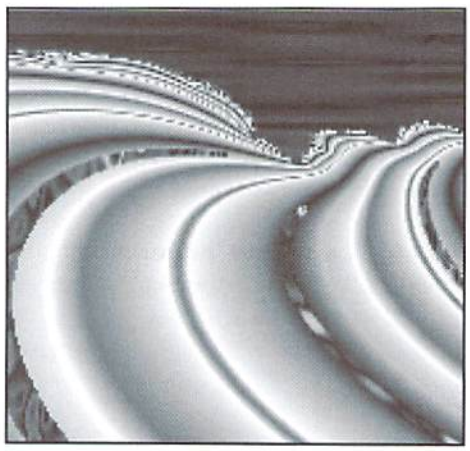
SURFING THE NET HAS A PRICE

Finding an inexpensive on-ramp to the Internet is something we all want, but finding something that's cheap AND easy may be a little more difficult. I found something that's "cheap", but it's a bit like finding a house labeled "a real fixer upper". I had to put some elbow grease in to get what I wanted at a price I could afford.

"Easy" access to the Internet isn't necessarily the better choice in terms of cost, especially if you're being charged by the amount of time you spend online. Almost every major Online Service (i.e. CompuServe, GEnie, Delphi, etc.) charge by time one way or another, whether you're using their Internet services or using other features they offer. As you learn to navigate the system and the Internet, you'll find that the Internet isn't exactly a speed demon.

While I have nothing against using an Online Service, Internet service providers are beginning to pop up everywhere you look, and these give us new alternatives. For clarity, when I use the phrase "Internet service provider", I'm referring to systems which only offer Internet connections without the additional services that are offered by "Online Services" like file libraries, chat and conference areas. How you connect to an Internet service provider can vary. When looking through their ads (in books or magazines about the Internet), you'll note that they usually list the type of "on-ramps" they offer. On-ramps and phrases we can ignore include: SLIP/PPP, UUCP, TCP/IP, Mosaic, as these are methods of Internet connection used by other platform computers and not yet available to CBM users. So what can we use? We're looking for services that let us connect to them with our term programs, so we need to look for key words such as "dial in-terminal", "dial-up" and "Unix Shell Account". These imply that we should be able to connect with and use the service.

Naturally, we need to find out how much access to the Net is going to cost, both initially and once we're on the highway. How you're charged depends on the provider. You might assume that charges are based on what the service provides, but this isn't always true. Some have a joining fee and begin charging from this point on. One service may simply charge an annual fee with unlimited access or may also charge for time spent online.



Another may charge for the amount of data you receive (some even have fees for both). Systems which charge by time may have two different rates, one for "prime time" (daytime) and a less expensive one for "off-peak" (evening) use.

When making your decision about which service to use, take into consideration two "truths" about the Internet:

Internet Truth #1: It can be slow.

It can crawl slower than a stock 1541 running GEOS v.1, slower than a VIC-20 loading from a datasette. It's not always this slow, but Murphy has a way of making sure that when you're spending money by the minute or hour, everything goes slower. Even if you plan to use the service during the "off-peak" times, it won't be much help since there are millions of other people doing the same thing at the same time. It's always "peak time" somewhere in the world.

Internet Truth #2: The Internet is all about data.

Newsgroups, World Wide Web, FTPing to get files, Email... it's all "data", and if you're charged by the amount of data you receive, it can add up very quickly.

Probably the least expensive route to the Internet is any service which offers a reasonable annual or monthly fee (under $100US annually) and allows you to use the system for a fair amount of time per day (say an hour or two) without incurring additional expenses. It helps if you can use the service a few times before officially signing up so you can evaluate the system.

For a long time, I avoided an economical and useful choice, the Unix Shell Account, due to something many suffer from: "Fear of the Unknown." I got over this when a bigger fear crept into my consciousness: "Fear of Going Broke While Surfing the Net".

Going UNIX...

Without ever accessing the Internet, having a shell account is like connecting another computer onto your own. You'll have your own "directory" to work in, which is one of the nicest benefits of using a Unix system - you can use someone else's disk "space" as your own which can have some distinct advantages. This gives you the ability to edit large text files, search text for key words, and use utilities which aren't available to us Commodore users yet. I like having the ability to unzip a PKZIP 2.04g file without asking someone else to do it for me, and Unix shell lets me do this.

Before dialing a Unix system, set your terminal emulation for either ANSI or VT-10x (the highest numbered VT emulation your term program offers). When you connect, you will be asked for your login name and password, similar to when you phone a BBS. If you're a new user you may have questions to answer and may be somewhat limited in your access unless the account was set up for you ahead of time. Once you've logged in, you'll either see a menu (if you're lucky) or a prompt with your login name in it like "username%>" and a blinking cursor.

Accessing Unix with a service that has a menu interface is similar to calling a BBS. Any options available to you are on the menu, one of which will be the ability to use the Unix shell. Since the system I use has the more "difficult" option, I'll deal with using a menu-less account.

Using a Unix shell can seem like a bit of an anticlimax. Simply seeing the username%> prompt isn't terribly exciting. Remind yourself that you've simply connected another computer to your own and that YOU are in charge. Unix was designed by people (yes, computer geeks are people too) who decided that prompts to tell you that a command worked would be a waste of time. Instead, they opted to have the system tell you when a command failed, but NOT when a command actually did what you asked it to do. So when you give a command in Unix that works, you won't see anything special happen, just the prompt return. Only when something goes wrong or the command you typed was incorrect will you see a response.

Taking Control

When I turn my computer on, the first thing I do is list the directory so I can see which disk is in the drive, and what files are on it. When I phone the Unix system, I tend to do the same thing. Maybe it's a way of feeling like I'm in control. To list a directory in Unix, you can try: "ls" which will list file names in columns but won't tell you anything more about them (like their size). You can add to this command by adding additional "switches", such as "ls -al" to see ALL files plus information about each file, or "ls -l" which lists only the names of the files but not in columns. The system I use allows "ll". It does the same thing as "ls -al" with less keystrokes. It gives me a full directory listing including file sizes and "hidden" ones. Hidden files begin with a ".", and are usually system files.

A directory listing using either "ll" or "ls -al" looks like this:

```
-rw-rw-r- 1  username   250 Jun 22 12:34 help.txt
drwxrwsr-x 1 username 22942 Jun 21 08:03 Mail
-rw-rw-r- 1  username 32232 Aug 20 21:34 Ridiculously_Long_File_Name
```

The first line is a file which is 250 bytes long. The second is a directory that has 22.9k bytes within it. The "d" in the first column tells you it's a directory. The other letters and dashes can be ignored at this stage. Take a look at the file names. Just as you can have upper and lower case letters for filenames on the Commodore, the same is allowed in Unix. And, like Commodore files, you must be specific about it. If you want to see what's in the "Mail" directory, but type the command using a lower case "m", Unix will tell you there is no such directory with the name "mail". Check out the name for the last file in the directory. Not only does Unix let you use mixed case, it also allows you to create long names (up to 255 characters long on most systems) so that you can give incredibly meaningful names to your files like: "Letter_to_Uncle_John_about_the_4th_of_July_Picnic".

Spaces are allowed in filenames, but because Unix looks at spaces as separators for commands, it's best to use an underscore, dash, or period instead. Now that we've looked at the root directory, let's take a look at what's in the "Mail" directory. To move to another directory, we use the "cd" (change directory) command which is similar in use to the one used by CMD devices. Type "cd Mail", and then list the directory with "ls". You can save time by putting both commands together with a semicolon between them like "cd Mail;ls". To get back to the root of your "home" directory just type "cd".

When I signed up with the service I use, I was told there would be a file for me in my directory called "help.txt". This text file had information about the commands I could use and a list of the utilities available to all users. Assuming other systems offer something similar, the first thing you'll need is a way to read this file. The books on Unix told me to use the command "cat filename", but when I tried this by typing "cat help.txt", the words went by too fast for me to read them. Instead, use the command "more filename" to read it. In my case, I typed "more help.txt", and the file was displayed one page (screen) at a time. To see the next page I used the spacebar (lucky guess). Since this file had a great deal of useful (and needed) information in it, I decided I should download and print it for further reference.

This meant I needed to know how to send the command to begin a file transfer. Another read of the help.txt file gave me what I needed: "sb -a help.txt". This let me download the file as ASCII text with Y-Modem batch as my protocol. I've since learned that "-b" in place of the "-a" lets you download the file as a binary file. To upload files using Y-Modem, the command is "rb" with either the -a (ASCII) or -b (binary) as needed.

UTILITY DEFINITIONS

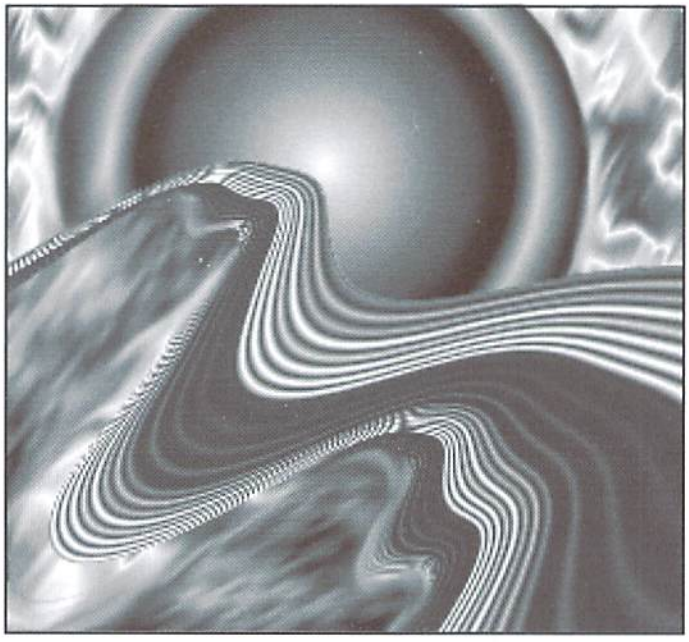
tin	- Usenet Newsgroup reader/poster
elm	- Email reader/poster
pico	- Simple text editor
mg	- More advanced editor
FTP	- File Transfer Protocol: Connect to other services and collect files or software.
IRC	- Internet Relay Chat: Chat with others on the Internet around the world.
gopher	- Information retrieval software: A menu program used to move around the Internet.
lynx	- Text Browser: Browse the Internet, view WWW sites.
gzip	- A file compressor/decompressor: Unzips PKZip files. (Unfortunately it can't be used to create the PKZip 1.x zips which we can dissolve). Some systems have a different Zip program that creates stored (not compressed) files which we can use.
uuencode	- A means of sending binary files to other users in the form of ASCII text. This utility creates a file which is a text version of a binary file.
uudecode	- This utility converts the uuencoded ASCII file back into a binary file.
grep	- A utility to search text for key words or phrases.

The help.txt file gave me a list of utilities that weren't shown in my directory, but were available to use from anywhere on the system. What you have on your system may not be the same, but should be similar and have similar uses. The sidebar on the previous page lists some utilities and their purposes. To use these utilities you simply type the name and press return. Some require other commands after the name. Check the sidebar on the next page for examples.

Mail, Editors and Finding Help

The system you use may have a different Email reader called Pine. Pine comes with an address book for Email addresses and has its own easy to use text editor built in. "elm" is what I have to read and send Email messages with. When you send mail using elm it loads a separate text editor to type your words of wisdom. I mention this because it took me awhile to learn—after using 3 different Unix systems which all had elm, I found that all three used different editors.

It may help to find out what text editors are available on your system and which of these is your "default" editor. If your system has pico you're in luck; it's simple to use, with two lines of commands shown at the bottom of the screen. When using a text editor or sending Email, you can send text from the buffer of your term program into what you're editing while online.



However, it may be easier to upload the text into your directory as an ASCII file, then insert it while in the editor. With pico, the command(s) for this are "ctrl-r" to tell it you want to "read in" text, then "ctrl-t" to show a list of files in your directory. You simply move the cursor to put the highlight bar over the file you want and press return.

If the only choice(s) you have for text editors are "vi" or "emacs", you should be pleased to know that you have a high powered editor to use, but these aren't quite as friendly as pico. You're on your own, because I can't help you with them, but I can tell you how to get the help you need. For almost every Unix command or utility there is a "manual" which you can refer to while online. If you type: "man vi", this will show the text nonstop and probably not be much help. Instead, try "man vi | more" which will let you page through the text.

A better alternative is to have the manual put into your directory as a text file so you can download it. The command for this is "man vi >viman.txt". You can substitute any Unix command or utility for vi, and of course what you decide to name the text file is up to you. To download the file you've just created to your computer, type "sb -a viman.txt" and start your Y-modem transfer.

Cleaning Up

If you've just downloaded the manual for vi, this is a good time to talk about deleting files. On some systems you may be permitted a specific amount of hard drive space to use, and most systems will periodically ask users to delete any unneeded files to make more room (some systems have a chronic problem when it comes to hard drive space). Since you've just downloaded the file viman.txt you no longer need to have this in your directory. The command to remove (delete) the file is "rm viman.txt". Once a file is deleted, it's history; you don't get it back. For this reason, be very careful when using wildcards with this command.

Go Wild!

Yes, wildcards are allowed, and we Commodore users should have an easier time using Unix wildcards than other platform users may have. We know that when we type "s:*" at a DOS prompt that we'll have every file in the directory erased. Unix operates in the same way. The question mark ("?") can also be used, and its use is the same on Unix as it is for CBM DOS commands. It's used as a place marker and can represent any character in that particular position, such as "?at" could be the word cat, bat or hat.

Will Your Commodore Respect You in the Morning?

Remember that long file name called: "Ridiculously_Long_File_Name"? This file would look messy on your Commodore directory if you downloaded it as is: "Ridiculously<Lon", and depending on which graphics mode you're in, may have the CBM graphic characters in place of the R and the L. Yuck! Name files in Unix using the CBM filename standard of 16 characters or less and avoid using the "_" (underscore) character, as these appear as backarrows when listed from our CBM directories. Whether you use upper and lower case letters is up to you.

Changing a filename in Unix is easy. If you want to rename "Ridiculously_Long_File_Name" to something short like "tom.txt". The command is "mv oldname newname", so you would type: "mv Ridiculously_Long_File_Name tom.txt". This wouldn't be as easy to do if you had used spaces in the name.

The "mv" command (the "move" command) can also be used to move a file from one location to another. The file will no longer be in the original location after using this command, so use it with care. A better alternative is the "cp" command which will copy a file from one location to another without affecting the original.

Wrap Up

Congratulations! You now have 100% more information on using a Unix system with your Commodore than I had the first time I logged onto it. With a little time and elbow grease, I've managed to learn to use the system and found that it's not so bad after all. Unix doesn't have to be difficult if you remember that you're in command and are simply "borrowing" the use of another computer.

If you can access the Internet at a low cost using a Unix shell account, it's worth learning a few new computer commands. You'll have a lot more available to you than "just" the Internet. Try it—I not only survived but found some useful tools to use with my Commodore!

Unix Commands

If you'd like to experiment with some of the commands in this article before logging into a Unix based system, try out Craig Bruce's ACE13 for the 64 and 128, which can be found by FTPing to: ccnga.uwaterloo.ca/pub/cbm/os. ACE13 is *not* a Unix system but it uses similar commands and some of the same utilities are included. Utilities/commands that ACE13 also uses are marked with a "+" symbol. Check the documentation of ACE13 for the proper use of the commands. A C64 Unix system is available by FTPing to: server 131.188.190.131 once connected, type: cd pub/poldi/lunix/lunix_v0.1

CTRL-x CTRL-c	Cancel an operation
exit, logout	Log off
passwd	Change password
CTRL-u	Clear command line
Cursor up	Display commands you've used (reuse them!)
ls -al	List directory - all info & all files
ls -l	List directory (names only)
+ ls	List directory (names only & in columns)
pwd	Show path of current directory
cd	Return to home directory
cd ..	Change dir back one level - note space
cd/dirname	Change to a specific directory
cd~/dirname	Change to a dir in your "home" directory
+ mkdir dirname	Create a directory named "dirname"
+ rmdir dirname	Remove an empty directory
+ cp oldname newname	Copy a file
+ cp oldname directoryname	Copy file to another directory
+ cp partofname* dirname	Copy many files to another directory
cp filename ~/dirname	Copy a file to a dir in your "home" dir
mv filename dirname	Move file to another dir (erases original)
+ mv oldname newname	Rename a file
+ rm filename	Delete file (permanent!)
+ grep "this" filename	Search file for any occurrence of "this"
+ more filename	Read a text file
man commandname more	Read instructions for a command
man commandname >file.txt	Create a text file of the instructions
sb -a	Download an ASCII file using Y-Modem
sb -b	Download a BINARY file using Y-Modem
rb (-a or -b)	Upload a file using Y-Modem
sx (-a or -b)	Download using X-Modem
rx (-a or -b)	Upload using X-Modem

Command	**Additional text (or example):**
ftp	ccnga.uwaterloo.ca
lynx	http://www.msen.com/~brain/guest/cwhome/index.html
+ uuencode	filename filename >filename.uue
+ uudecode	filename
unzip	filename
pico	textfile.name

+ Wildcards:

* Represents any number of characters. Do not use this as the first character after a command unless you want it to affect all files in the current or specified directory.

? Represents any one character. Example: t?ll for tall or tell.

Remember:

1) Unix doesn't tell you when a command has worked.
2) Unix is CASE Sensitive!
3) Using a ";" between commands lets you accomplish two or more tasks together.

G.R.M.

How to Type In Program Listings Appearing in Commodore World

While *Commodore World* currently doesn't make it a habit of publishing type-in programs, a number of our columns do require entering sample routines. For this purpose, we have created our CHK-LIST utility for the Commodore 64 and 128. This utility uses a 16-bit CRC checksum method to verify that you have correctly entered each program line, and that each of the characters in the program lines are in the correct order.

You'll notice that program listings appear with a column of values to the left of the program lines. These values are the CHK-LIST values and are not to be entered as part of the program. A similar set of values are generated by the CHK-LIST utility to allow you to verify that everything has been entered correctly.

Enter the CHK-LIST program from BASIC. You can use either a C-64 or a C-128 computer. If you use a C-128, it can be in either 64 or 128 mode. Be sure to enter each line carefully to avoid mistakes—until you actually have CHK-LIST working, finding errors in program entry won't be easy. After you have finished entering the program, be sure to SAVE a copy to disk before you attempt to RUN it, just in case. If you aren't familiar with how to save a program to disk, you can use the following command:

```
SAVE"CHK-LIST",8
```

To use CHK-LIST, load it into your computer and type RUN. Make sure that any program you are currently working on is saved first, or start CHK-LIST before you begin typing in a new program. After you have CHK-LIST in memory and running, type NEW. You may now either load or begin typing the program you wish to have CHK-LIST check on. Whenever you want to check your program, type in the appropriate SYS command given below:

C-64 or C-128 in 64 mode:	SYS49152
C-128 in 128 mode:	SYS4864

Note that when typing in listings, some special characters will appear in braces. For example, {CLR/HOME} means that you should enter the Clear key, which is done by holding down the SHIFT key while you press the HOME key. Other times you may see a number ahead of the key name, such as {3 SPACES} or {5 CRSR L}. This means you should press the key indicated the number of times shown. Most special keys are easy to identify, since the text shown will generally match the text on the key. Exceptions are the space bar {SPACE}, and cursor keys which include directions ({CRSR UP}, {CRSR DN}, {CRSR L} and {CRSR RT}). Be sure to use the correct key combinations for color keys, such as <CTRL><2> for {WHT}.

CHK-LIST

```
A454  10 F=ABS(PEEK(65533)=255):M=49152:IFFTHE
      NM=4864
6E2F  12 C=Ø:PRINT"{CLR/HOME}WORKING";
E35Ø  2Ø READD:IFD=-256THEN4Ø
AD2Ø  3Ø C=C+D:IFD<ØANDF=ØTHEN2Ø
3316  31 IFD<ØTHEND=Ø-D:M=M-1
Ø7FØ  32 POKEM,D:M=M+1:PRINT".";:GOTO2Ø
578A  4Ø PRINT:READCK:IFC<>CKTHENPRINT"ERROR I
      N DATA STATEMENTS!":END
Ø679  5Ø PRINT"DONE.":END
8D92  6Ø :
E7FE  49152 DATA 165,43,-45,133,251,165,44,-46
      ,133,252
B2AE  49160 DATA 169,Ø,141,36,193,-2Ø,169,147,
      32
CD5Ø  49168 DATA 21Ø,255,32,194,192,-19,16Ø,Ø,
      14Ø
C9CD  49176 DATA 37,193,-2Ø,177,251,133,253,2Ø
      8,3
2Ø58  49184 DATA 238,37,193,-2Ø,2ØØ,177,251,13
      3,254
EA9C  49192 DATA 2Ø8,3,238,37,193,-2Ø,173,37,1
      93,-2Ø
6C15  492ØØ DATA 2Ø1,2,2Ø8,1,96,2ØØ,177,251
E7ØE  492Ø8 DATA 17Ø,2ØØ,177,251,32,2Ø5,-5Ø,18
      9,-142,169
6795  49216 DATA 6,133,211,-236,169,61,32,21Ø,
      255
F8ØF  49224 DATA 169,32,32,21Ø,255,16Ø,2,177
9735  49232 DATA 251,32,213,192,-19,2ØØ,177,25
      1,32
Ø734  4924Ø DATA 213,192,-19,2ØØ,177,251,24Ø,6
      ,32
D99D  49248 DATA 213,192,-19,76,9Ø,192,-19,173
      ,191,192,-19
AC3Ø  49256 DATA 32,167,192,-19,173,19Ø,192,-1
      9,32,167
```

CHK-LIST *(cont.)*

```
B343  49264 DATA 192,-19,169,13,32,21Ø,255,165
      ,253
DF3A  49272 DATA 133,251,165,254,133,252,238,3
      6
A6E2  4928Ø DATA 193,-2Ø,173,36,193,-2Ø,2Ø1,2Ø
      ,24Ø,3
936E  49288 DATA 76,18,192,-19,162,Ø,189,1,193
      ,-2Ø
8C3A  49296 DATA 24Ø,6,32,21Ø,255,232,2Ø8,245,
      32
EB74  49304 DATA 228,255,2Ø1,13,2Ø8,249,32
6Ø95  49312 DATA 228,255,2Ø8,251,76,8,192,-19,
      72
AØØ1  4932Ø DATA 1Ø6,1Ø6,1Ø6,1Ø6,32,18Ø,192,-1
      9,1Ø4
FAA2  49328 DATA 32,18Ø,192,-19,96,41,15,17Ø,1
      89
EBFD  49336 DATA 2Ø,193,-2Ø,32,21Ø,255,96,Ø,Ø
E9Ø7  49344 DATA Ø,Ø,169,Ø,141,19Ø,192,-19,141
E8EA  49352 DATA 191,192,-19,169,33,141,192,19
      2,-19,169
A7D7  4936Ø DATA 16,141,193,192,-19,96,162,8,7
      2
6Ø4Ø  49368 DATA 41,127,77,191,192,-19,141,191
      ,192,-19
D24B  49376 DATA 24,14,19Ø,192,-19,46,191,192,
      -19,144
D52F  49384 DATA 18,173,192,192,-19,77,19Ø,192
      ,-19,141
DCA6  49392 DATA 19Ø,192,-19,173,193,192,-19,7
      7,191,192,-19
6Ø32  494ØØ DATA 141,191,192,-19,1Ø4,1Ø,2Ø2,2Ø
      8,215
37C5  494Ø8 DATA 96,13,8Ø,82,69,83,83,32
9A2A  49416 DATA 6Ø,82,69,84,85,82,78,62
AC9Ø  49424 DATA 13,13,13,Ø,48,49,5Ø,51
FE71  49432 DATA 52,53,54,55,56,57,65,66
Ø17E  4944Ø DATA 67,68,69,7Ø,Ø,Ø,-256,37944
```

Over The Edge

by Harold Stevens, Jr.

Remember the Volkswagen Beetle? That little inexpensive car that got great mileage, and took you just about anywhere you wanted to go to? While it was great for traveling, it wasn't worth a dime accelerating uphill, and you couldn't keep it warm in the wintertime. But that didn't stop people from buying the "Bug", as it was an affordable, rugged car that was easy to repair.

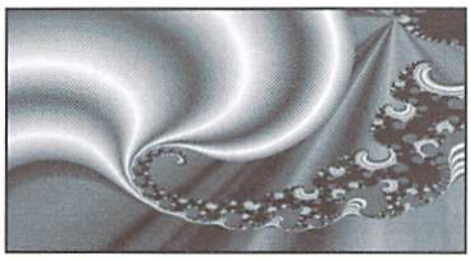
For a long time, the Beetle reigned as king of the small cars; there was even a convertible model, and later a larger model with a real dashboard and curved windshield called the Superbeetle. But technology, the decline of the U.S. dollar, and a shrinking market eventually killed off the Beetle, as people found other cars that were more efficient and powerful. Nowadays it's rare to see a Beetle on the road—they've become classic antiques.

In the world of personal computers, we've seen a similar success story. Like the VW Beetle, the Commodore 64 became one of the most popular personal computers of all time because of its affordability and simplicity.

The C-64 easily earned its role as the most popular of personal computers. It had its own built-in operating system, had great graphics and sound capabilities, and was easy to program using BASIC. These factors, coupled with the C-64's popularity, gained it wide software support, both public domain and commercially produced.

But like the VW, the popularity of the C-64 (and its own Superbeetle version, the C-128) began to wane as prices for the more powerful computers began to fall, and as other machines became more user-friendly through graphic user interface programs like MacOS and Windows. Thus, the technology that had made the C-64 possible, had also made it obsolete.

In the years that the C-64 was at the top, there were programs and peripherals galore, filling every need imaginable; desktop publishing, games, spreadsheets, and even playing stereo music, all were readily available and affordable. By 1990, however, many of the vendors supporting the Commodore were gone, and the



supply of both hardware and software had dwindled down to a trickle. Even new public domain and shareware programs on the old Quantum-Link (abandoned by Quantum Computer in favor of their non-Commodore service *America OnLine*) grounded to a halt. It was then the handwriting appeared plainly on the wall to me: the fate of the C-64 was terminal.

Sure, the C-64 hung on like the old Beetle. Users of the small computer have ended up having search out third parties for parts, software and hardware; some die-hard users even buy old C-64s for parts—again, Volkswagen owners have been doing all these things as well to maintain their beloved classics.

By the way, I'm not the only one who sees the similarity between the C-64 and the Beetle. At a recent conference on GEnie, Steve Main from Geoworks (formerly Berkeley Softworks) made the same analogy. Main is the senior support representative for Geoworks.

In the GEOS conference on GEnie, Main said the C-64 and later, the C-128, reminded him of the couple of Volkswagens he used to own. "I never took auto shop in school or anything, but because the Volkswagen was designed to be a 'user-maintained' car," he said, "I ended up learning how, on my own, to rebuild those beasts, and I rebuilt two from the crank up. The Commodore, for me, was similar. I learned how an OS works, what the hardware does, what a byte is."

So, Commodore users are back to square one. Here we are, stuck with one heck of a computer but no place to go. This means that we have to depend on ourselves if we want more software to push our computer beyond its limitations. For me, that means I have to start learning how to program in BASIC and machine language.

This won't be an easy task for me, as I was lousy at math in school, but I'm going to use the approach that I am learning a foreign language instead. After all, computer languages are languages that the computer understands. So, if I could master the rudimentary concepts of speaking something difficult like Russian in college, it shouldn't be that hard to learn BASIC or machine language.

My primary interest in learning how to program is to improve the productivity applications that I currently use. For instance, I'd like to see geoPaint operate in an environment where I can get 300 dpi graphics instead of 80 dpi, so I can get drawings and graphics without the jaggies on curves and lines. Or see geoWrite in an 80 column environment on a Commodore 64.

I realize that these applications won't have a large commercial market, meaning I'll have to make them shareware or even public domain, strictly so that the serious users of Commodore computers can continue to squeeze more uses out of their wonderful little eight-bit machines.

However, you won't find me among those pronouncing the Commodore 64 dead—not by a long shot. With companies like Creative Micro Designs, Software Hut, and Software Support International still serving the Commodore 64/128 computers, we're a long way from seeing our beloved computers dead and buried.

Volkswagen has announced that the Beetle will be returning to the highway in 1997, although it will no longer be the air cooled sedan we used to drive. Perhaps the same fate awaits our Commodores, particularly after the German computer firm Escom has purchased the assets of Commodore Business Machines and all the talk of once again producing eight-bit computers for the second and third world countries. Who knows, maybe the C-64/128 will rise again?

* CLASSIFIED ADS *

C64/128 PUBLIC DOMAIN. REQUEST FREE CATALOG OR SEND $2 FOR A DEMO & CATALOG. CALOKE IND., P.O. BOX 18477, RAYTOWN, MO. 64133. VISA-M/CARD ACCEPTED.

C-64 FOREIGN-AMERICAN Utilities, Graphics, Hacker, Arcade. 32¢ stamp gets catalog. Home-Spun Software, POB 1064-CW,Estero,FL. 33928

WIN $$$ PLAYING THE LOTTERY! New software will help. **PROVEN SYSTEM!** For details, send SASE to: LOTTOMAN, P.O. Box 44, New Millport, PA 16861, or call 814-236-7615 and leave your name and address at the end of message.

GEOS Publication. One Year Subscription $8.50; two years $16. 713 E. Main Street, Independance, KS. 67301-3726. Monthly.

GRASSROOTS #1. C= history, hardware, help on full 2 sided info disk. Send $3. & system info to Donald Ayers, 75 State Rd. 270W, Sturgis, KY 42459.

Reconditioned C64 and 1541 Disk Drive also some used Commodore parts. For information send a SASE to Chuck 30102 Pacific Island Dr., Laguna Nigel, CA 92677.

RUN magazine, all issues 1987 thru 1992. Commodore MPS 801 Printer, 1541 drive. R. Elliot 228 Star Hill, Swansboro, NC 28584

Wanted to Buy Voice Synthesizer for the C64. Prefer Hearsay 1000, but will take any that can speak AND hear voice commands. 813-914-5410 (beeper).

Commodore World Classified Advertising

Commodore World Subscribers may place non-commercial classified advertising in Commodore World at a cost of $10.00 per issue. Your advertisement may contain up to 150 characters (including spaces). Send your advertisement with payment to: CW Classified Advertising, c/o Creative Micro Designs, Inc., P.O. Box 646, East Longmeadow MA 01028-0646.

ADVERTISERS INDEX

MOVING?

Don't forget to let Commodore World know. Call or write with your change of address 6 to 8 weeks prior to your move so that you won't miss a single issue!

DON'T WAIT UNTIL IT'S TOO LATE— RENEW EARLY!

Is your Commodore World Subscription getting close to running out? There's an easy way to check. Look at the mailing label on the front of your copy. There you'll find your subscription number and the expiration issue number. For example:

```
James Smith          12345EXP09
123 Home Street
Grand Rapids, MI  49502-0123
```

Jim's subscription will run out with issue 9, as indicated by the EXP08 in his subscription code. Jim would be wise to re-subscribe early to avoid missing a single issue of Commodore World!

Commodore World is the publication that will keep you informed in these times when up-to-date information on Commodore computing is so hard to find. Published by Creative Micro Designs, the industry leader in development of Commodore-related products for over six years, Commodore World will supply you with information on what's new, what's still available, and above all else—*where* to get it. If you felt you had nowhere to turn to for Commodore support, turn to the pages of Commodore World for a wealth of resources ready to help you get the most from your computer!

You'll find Commodore World feature articles informative and easy to read; what's more, they're written by leading authorities and experts, many of whom have written for other Commodore-related publications in the past. And Commodore World has something for everyone, whether you're a novice or an experienced programmer.

And while our feature articles help to cover different subjects in each issue, regular columns provide on-going insight into topics of interest to most users. You'll find columns that cover BASIC and advanced programming, and even a column for GEOS programming. And if you prefer being a GEOS user to being a GEOS programmer, you'll find another column devoted to helping you get more out of GEOS. If you want to learn more about using and programming the various peripherals on your system—you guessed it, we've got a column for that as well. Even first-time Commodore users will find a column devoted specifically to their needs.

And there's even more. Departments that cover news, telecommunications, reviews of available hardware and software; even news of what's happening in other Commodore-related publications!

So, if you really want to get the most from your Commodore, there's no better way to get it than Commodore World!

Columns

Just For Starters - An introduction to the C64/128 by Steve VanderArk

Foreign Exchange - An inside look at the market in Europe by Joseph Gaudl

Graphic Interpretation - GEOS, GEOS and more GEOS by Steve VanderArk

geoProgrammist - GEOS programming techniques by Maurice Randall

BASIC Instincts - BASIC tutorials and type-in programs by Gene Barker

Jim Butterfield's ML Column - Probably the best known name in our industry, Jim covers every aspect of programming in ML (coming soon)

Peripheral Vision - Technical insights to C-64/128 hardware peripherals

Carrier Detect - Exploring every facet of the Telecommunications experience

Over The Edge - Editorial covering various computer related topics and news

Departments

From The Editor • BackTalk • On The Horizon
Just Asking • The Connection • Top Tips
User Group Connection • Commodore Trivia
BBS Spotlight • Classified Ads

HEAVENBOUND

T-Fant is missing!

Curiosity has gotten the best of him, and he's off in search of Heaven. But there's a lot of meanies between T-Fant and the promised land, so it's not going to be an easy trip!

In Heavenbound, you'll assume the role of T-Fant–your average pink elephant with an overactive yearning for adventure. Race against time as you climb your way up through the clouds in search of the pearly gates of Heaven. You'll have to battle your way past the strange creatures that would rather see you fail, and you'll also have to be on the lookout for the golden pellets that will increase your time, shot power, and give you extra lives.

Heavenbound is a brand new arcade-style game for the Commodore 64 that features advanced graphics and superb sound. Requires a Commodore 64 or 128, joystick or gamepad, and a 1541 or 1571 disk drive.

LORDS OF THE UNIVERSE

Things have been quiet in the Omacron sector for a suspiciously long time–until *NOW!* Lead the battle against the hoards of marauding invaders in the corridors of death. Face the awesome Mega Guardians who are all too eager to put an end to your pitiful existance. Whatever you do, strap on your seat-belt...

...you're in for the battle of your life!

ELECTRIC BOYS
ENTERTAINMENT SOFTWARE

Distributed Exclusively in North America by Creative Micro Designs, Inc.