# THE COMPUTE!'S GAZETTE DISK

# JULY 1994

```
*****  *****  *   *  *****  *   *  *****  *****    *   *****
*      *   *  ** **  *   *  *   *    *    *       *   *
*      *   *  * * *  *****  *   *    *    ***    *    *****
*      *   *  * * *  *      *   *    *    *             *
*****  *****  * * *  *      *****    *    *****        *****
```

**GAZETTE ON DISK** — **JULY 1994**

( 2 ) = Table Of Contents

Disk Magazine

**PROGRAMS:**

( 4 ) = Mutate!

( 11 ) = 15 Solitaire

( 13 ) = Color Master

( 17 ) = List Maker

( 19 ) = Gazette File Converter

( 20 ) = Globe (PD)

( 20 ) = Warp Field (PD)

**COLUMNS:**

( 21 ) = 64/128 View

( 22 ) = Feedback

( 25 ) = D'Iversions

( 28 ) = Beginner Basic

( 32 ) = Machine Language

( 36 ) = Programmer's Page

( 40 ) = G.E.O.S.

( 42 ) = PD Picks

**FEATURES:**

( 45 ) = Assembly Language

( 52 ) = 6502 Mnemonics

To Load Disk:

Load "Menu",8,1 and press <Return> --- Commodore 64

GAZETTE DISK *** July 1994

Features:

ASSEMBLY LANGUAGE
By David Pankhurst

You can add years of enjoyment to your Commodore by learning how to program in machine language. With a little knowledge in that area, you may discover whole new aspects of your computer.

6502 MNEMONICS
By David Pankhurst

Before you can master any new language, you first have to learn the vocabulary. Learning to program in the language of the 6502 chip is no different. These 56 commands are the vocabulary of machine language.

Columns:

64/128 VIEW by Tom Netsel
We'd like to know more about our readers. Please take a minute to print out, complete, and return this readership survey.

FEEDBACK
Comments, questions, and answers.

D'IVERSIONS by Fred D'Ignazio
The Fork in the Road.

MACHINE LANGUAGE by Jim Butterfield
Screen Zap.

BEGINNER BASIC by Larry Cotton
Rounding.

PROGRAMMER'S PAGE by David Pankhurst
Make Room for Data.

GEOS by Steve Vander Ark
Mouse Hunt.

PD PICKS by Steve Vander Ark
Warp Field and Globe.

128 Programs:

Space Pirates by John LeDoux

See how much wealth you can accumulate as you explore the galaxy. Be careful to avoid bounty hunters, renegades, and the Federation Space Police.

Gazette/SpeedScript File Converter
Convert Gazette Disk or SpeedScript text program (PRG) files into sequential (SEQ) ASCII files.

64 Programs:

15 Solitaire by Arthur Moore
You are dealt 16 cards in a 4 x 4 grid. Use your joystick to group cards of a like suit so that they total 15. This solitaire variation is challenging and fun to play.

Color Master by Daniel Lightner
Use this utility to convert Doodle graphics into multicolor masterpieces.

Listmaker by Robert Nellist
Everybody makes lists. Let Listmaker help you create and manage lists of all types: shopping lists, team rosters, inventories, club memberships, mailing lists, and more. You can then sort them, store them, and print them in a number of helpful ways.

Mutate! by Gary Noakes
You can feed 64 video directly into a VCR to create professional-looking titles and sequences for your home videos. This version of Mutate!, which ran originally as a bonus on the April 1993 Gazette Disk, offers dozens of new fonts for you to use.

Gazette/SpeedScript File Converter
The text files on Gazette Disk are in SpeedScript PRG format. If you'd like to load, read, and print them with a word processor that uses SEQ files, here's a utility that should help. This program converts PRG files to Commodore ASCII or true ASCII sequential files. It can also convert Commodore ASCII files to PRG format.

Warp Field by Amenophis
Control eight variables and see how each change affects an animated mathematical plot. This one is fascinating to watch.

Globe by John Crider
As the earth spins in the center of your monitor, you can control its speed and the angle from which you view it. Look down from either pole or from anywhere in between. The globe is detailed with colors to show the earth's topography.

Gazette, July 1994

MUTATE!

By Gary Noakes

A video titling program for the 64 that appeared on the April 1993 Gazette Disk as a bonus prgram. This version has 20 new fonts in its library.

With its' built-in capability to connect directly to a TV set or VCR, the C-64 can easily be used to make title or credit screens for your home videos. Mutate! is designed to create professional-looking video sequences on your VCR while retaining a user-friendly interface. All main functions are menu-driven for convenience and fully error-checked before execution for foolproof use. Help screens are also available from the menus.

RUNNING THE PROGRAM
Although it is written entirely in Basic, Mutate! uses a boot program to load data files and to reconfigure memory. To load the program, type LOAD "MUTATE!BOOT",8 and run it. The boot loads two character sets and the help screens and then checks to see if a 1541 or 1571 in native mode is connected as drive 9.

If drive 9 is present, you will be given the choice of using it to load and save text files while reserving drive 8 for the character sets. Just press the Y or N key at the prompt. Finally, the start of Basic is lowered and the main program is loaded and run.

Important Note: If you have a second drive connected and the device number has been changed through software, it will be reset to drive 8, causing the system to "hang" when loading the main program. If the second drive is a 1571, it will be reset to native mode. Use only a hardware-changed drive 9 or turn the drive off before loading the program.

When the main screen appears, you'll see there are two menus on the left: Function and Design. To operate them, use the up and down cursor keys to move the arrows to your choice and then press Return. At these menus, the f7 or Stop keys can be used to exit to Basic.

On the right side of the screen is a Font box displaying a sample of the currently loaded font. The Prompt box is directly below. The Current Status box at the bottom of the screen displays the filename, the number of screens in the file, the video translation status and the name of the current display font.

Keep an eye on the Prompt box. In addition to displaying information, any time a function is selected that requires some input, a bell will sound and request information. If you have selected a menu option that cannot yet be performed or one that will kill your text, a buzzer will sound and the program will ask for confirmation to continue.

For absolute safety, all prompts will accept pressing any key except Y

the same as an N key press. On two drive systems, the Directory function will request a drive number before continuing. It defaults to drive 8 if any key but 8 (or 9) is pressed without selecting a drive number.

The Function Menu is used to access the most commonly used features. Here is a description of the Function Menu options.

DESIGN MENU
This menu is used to access all text-related options, described below.

SELECT FONT
Use the Cursor up and down keys to scroll the font names within the menu. Press Return to load the font indicated between the arrows. Press the Clear key to read a new font index from the disk in drive 8.

All of these commands are displayed on screen. Not listed in the commands is the Back arrow key, which resets the colors to the default of white characters on a black background and border. The colors selected are stored and will remain in effect until reset.

The character sets used by Mutate! are all prefixed with M!, both so Mutate! can find them and so they aren't confused with other fonts. The prefix will not be visible in the menu. There are also four suffixes used to mark special font types:

RVS for reversed sets
HLF for half sized sets
FNE for fine line sets
OTL for outline sets

The available characters are displayed in the lower screen area. Fonts may be loaded at any time without affecting either the text or the video translation status.

In order to compare fonts against different backgrounds, the F-keys perform these functions:

f1/f2: Cycle Border color forward and backward
f3/f4: Cycle Background color forward and backward
f5/f6: Cycle Character color forward and backward
f7: Return to the Function Menu

VIEW DISPLAY
Before the video screens can be displayed, they must first be translated from the text screens. The miniscreen will appear and the text for each screen will be displayed and translated into the video display characters. The translation may take up to several minutes,

depending on how many screens and how much text there is to be translated. As each character is read, it is replaced with a dash, giving a visual display of the progress. Translation only needs to be done once unless the text is altered by editing or adding a new screen.

Once completed, the View Display menus will appear. The first menu is Display and it includes teh following.

Blank Screen: Blanks the screen between each display.

Blank & Wipe: Blanks the screen between each display and allows a choice of screen clearing method from the Wipe menu.

Scroll Screen: Scrolls each screen up from the bottom. It adds a blank line between screens and a blank screen at the end.

Scroll & Wipe: Scrolls each screen up from the bottom. It allows a choice of screen clearing method from the Wipe menu and adds a blank screen at the end.

Line Scroll: Performs a line-at-a-time scroll, eliminating the separation between pages in the display. It adds a blank screen at the end.

The Wipe Menu offers seven methods of clearing the display. The names are as descriptive as possible:

Left To Right: Clears a column at a time from left to right.
Right To Left: Clears a column at a time from right to left.
Top To Bottom: Clears a line at a time from top to bottom.
Bottom To Top: Clears a line at a time from bottom to top.
Left & Right: Clears columns from both left and right sides to the center.
Top & Bottom: Clears lines from both top and bottom to the center.
Four Sides In: Clears from all four sides to the center.

The last menu is Mode. Video Tape is for making title or credit screens on your VCR. Billboard is used to continuously display the same message, useful for attracting attention at a user group meeting or for advertising purposes.

As each selection is made it appears in the Sequence box so you can keep track of what you've selected. If you've made a mistake, press f7 to exit to Function and reselect View Display from the menu.

When all selections are complete, the border, background and font sample will change to the current colors. Use the F-keys to change them and press Return when they are satisfactory.

The next prompt will ask "Set Time For Each Display?" Answer yes if you wish to adjust the time each screen is displayed. If you answer no, the default time of ten seconds is used. If Line Scroll has been

selected, this prompt will be skipped. Line Scroll uses a predefined timer.

At "How Many Seconds For Each Display?", enter any number between 1 and 99. Pressing Return at the prompt resets the timer to the default.

Pressing any key at the last prompt clears the screen and puts the program into pause so you can set your VCR to record. When you're ready, press a key to begin the display sequence.

If Video Tape has been selected, the display will go through a single cycle and end. If Billboard has been selected, the display will repeat until you tell it to end. To exit, keep the Shift (or Shift-Lock) key depressed and wait for the current cycle to end.

Once the cycle is complete, a bell sounds and the screen blanks. Pressing any key returns to the Function Menu. If no key is pressed, the program automatically returns to Function in about 15 seconds. This should allow you adequate time to stop your video recording.

DIRECTORY
This will give a directory of Mutate!-related font and text files on the disk in drive 8 (or drive 9 if it is present and selected). Press Return to begin, Shift (or Shift-Lock) to pause or the C= key to abort the listing. When finished, press any key to return to the Function Menu.

HELP SCREENS
There are three screens that give a brief description of each of the menu functions. Press Return to page the screens or press f7 or Stop at any screen to exit.

EXIT PROGRAM
This will request confirmation to exit. Pressing Y will go back to BASIC by resetting the computer. Pressing any other key will continue.

DESIGN MENU
Enter Text: This displays a proportional miniscreen with the lines numbered. Each screen is limited to twelve lines of eighteen characters each.

At the prompt "How Many Screens?" enter a number between one and nine or press Return to exit. When each line is entered, it appears in the miniscreen in light red, formatted exactly as typed. If the line is less than eighteen characters long, you will then have the opportunity to justify the line.

The justify commands consist of Left, Right, Center, Right Cursor, Left Cursor and Return. Left, Right, Center and Return are immediate commands; they perform their desired function and then go to the next

line. The Left and Right cursor commands allow moving the line back and forth a space at a time. When Return is pressed, the line will turn cyan and you can go to the next line.

Typeable characters are limited to the available characters of the font. The set consists of the unshifted letters A through Z, numbers 0 through 9, space and shifted-space and punctuation consisting of period, question mark, exclamation point, colon, ampersand and apostrophe. One graphic character is also included, the Shift-asterisk (a bar character).

The shifted-space is used as a "continuity" character; when entering or editing text, it produces a small graphic character so you can see it is not a normal space. When displayed with an unreversed Mutate! font, the shifted-space will be invisible, but if used with a reversed font, it produces a graphic space in the same style of the font currently selected.

Once all lines for all screens have been entered, you will asked to enter a filename (limited to 14 characters). The program then returns to the Design Menu, where the Prompt box now displays:

C=+ Adds Screens
C=- Deletes screens

(C= denotes the COMMODORE key)

This allows adding more screens if the number of screens entered was insufficient or deleting screens if too many were requested. Screens can only be added or deleted at the end of the text file. Text for any added screens must be entered from Edit Text.

If Enter Text is selected and a file is already in memory, you will be asked to confirm your choice. If you answer yes at the prompt, memory is cleared before going to the text entry screen. The font name and color selections will be retained.

EDIT TEXT
This will once again display the miniscreen and the text for the first screen. At the prompt "Edit This Screen?", press Y or N. If you answer no, the next screen (if there is more than one screen) will be displayed.

If you answered yes at the prompt, it is replaced by "Line Number To Edit?". Type the number displayed next to the line you wish to edit. The prompt will then ask "Enter New Line or Justify Line?". Press E if you wish to enter new text or J if you only wish to justify the line.

This cycle repeats until you have had the opportunity to edit all of the screens. Once completed, you will be returned to the Design Menu.

LOAD TEXT
This loads previously saved text files from the disk in drive 8 (or drive 9 if it is present and selected). Type the filename, limited to 14 characters, and press Return. The program adds the T!-prefix to the name so typing it is unnecessary.

If text is already in memory, you will be asked to confirm your choice. If you answer yes at the prompt, memory is cleared before a filename is requested. The font name and color selections will be retained.

SAVE TEXT
This will save the text currently in memory to drive 8 (or drive 9 if it is present and selected). The filename entered at the Enter Text screen will be printed at the prompt. Press Return to save the file with that name or use the Delete key and enter a new name. The program prefixes the filename with T! before the save.

HELP SCREENS
Same screens and directions as above.

FUNCTION MENU
This returns to the Function menu.

NOTES ON FONTS
The Mutate! program disk comes complete with 20 indexed fonts. Different fonts came with the original program.

If you are artistic and/or good with a font editor, you can design your own Mutate! fonts. Any good character editor can be used, but editors using an 8-row x 32-character layout (such as COMPUTE'S Ultrafont+, written by Charles Brannon) give a better picture of the character arrangement used by Mutate!.

Newly designed or edited fonts can be saved using any load address. Mutate! will automatically relocate the data to the address it needs when the font is loaded. Just make sure you use the M! prefix when saving a new set.

New fonts must also be added to the font index (this appears on the disk as MUTATE!INDEX). This is done by running Mutate!Indexer, a short utility program included with Mutate!. Since the main program is limited to 30 fonts in the Select Font menu, the indexer reads and alphabetizes only the first 30 M!-prefixed fonts it finds, scratches the old index and then writes the new index back to disk.

If more than 30 fonts are on a disk the extras will be skipped, so they should be copied to a new disk and the indexing program run on the new disk.

# 15 SOLITAIRE

By Arthur Moore

It's 3:00 a.m., but you're determined to win. A game this simple shouldn't be so frustrating.

You'll find that 15 Solitaire is a game that's easy to learn and one that quickly becomes addictive. It's written for the 64; a joystick is optional. If you use a 128 in emulation mode, you'll enjoy an accelerated initialization phase, thanks to the 8502's dual speed microprocessor.

## HOW TO PLAY

You are dealt 16 cards that are arranged in a 4 x 4 grid. The object is to group cards of like suits that total 15. Aces are counted as 1. Once grouped, the cards will be erased, and more cards are dealt to replace them. Complete this for the whole deck, and you win. Of course, there is one little catch.

## FACE TO FACE

Groups of 15 must be built using the cards ace through 9. The four face cards of each suit (10, jack, queen, and king) cannot be grouped until they are all present. Face cards become an annoying obstacle before long.

## GETTING CONTROL

Use your joystick in port 2 or the cursor keys to move the flashing X. Once you've decided on a group, place the cursor on each of the cards and press either the fire button or Return. This will put a colored frame around the card. Should you make a mistake or change your mind, press the button again to erase the frame. To the right, a counter totals the value of all framed cards. Once this counter equals 15, those framed cards will be replaced with cards from the deck (which are also tracked). If face cards are framed, no total is given.

## NO CHEATING!

Since groups must consist of the same suit, no card will be framed if it doesn't match the suit of the other framed cards. Also, face cards and numbered cards cannot be framed simultaneously. Remember: The 10 is considered a face card, not a numbered card.

Should you find yourself starting a group that simply will not work, you may cursor to each card to remove the frame or move to the green box in the lower right corner marked CLR ALL. This box will clear all frames when selected.

In the upper right corner is a blue box marked REDEAL. Once you've reached a point where no more moves can be made, select this box to start a new game. You will be asked to confirm your selection. Press either the fire button or Return to confirm, or push the joystick in any direction to abort the function. Gamers using the keyboard can press any key to cancel it.

CONGRATULATIONS!
Complete the deck, and you will be commended for your work. Press either the fire button or Return to start a new game.

BEWARE
The author has lost his temper, many clumps of hair, and countless hours of sleep playing 15 Solitaire. You've been warned.

Arthur Moore lives in Orlando, Florida.

Gazette, July 1994

COLOR MASTER

By Daniel Lightner

A graphics utility for the 64.

Color Master is a utility program that aids you in converting artwork created from powerful design programs such as Sketch-Pad or Doodle into true multicolor masterpieces. Color Master is not a sketch or paint program, but it is possible to create great artworks with it from scratch. Sketch-Pad allows you to have a paint and a background color throughout the whole screen, while Doodle allows a paint and a background color for each byte of color memory. Color Master lets you have three paint colors in each 8 x 8 pixel space and a background color as well.

Color Master is written in machine language, but it loads and runs just like a BASIC program. When run, Color Master displays the title screen. Press the space bar to continue. The bitmap is cleared, and a blank screen appears.

In the upper left corner of the screen, you'll see a small square box. This box is your screen pointer. The pointer can be moved around the screen using the cursor keys or a joystick plugged into port 2. The pointer color can be changed simply by pressing the * key. Move the pointer to the center of the screen and change the color. The + and - keys control the speed of the pointer. Pressing + will speed it up a little each time it is pressed, and - slows it down. Slow the pointer down before continuing.

Function key f7 is the keyboard equivalent of the joystick's fire button. When the fire button is pressed, the area of bitmap beneath the pointer is captured, and Color Master goes into Zoom mode. When you enter Zoom mode with a blank screen, all you will see is a large box filled with the background color. Press the number 1 key to change the screen color. When you have selected a new color, you should see four columns of 1s. The number 1 indicates that the area is full of color 1, the screen background color. Number 2 would indicate paint color 2, and so on through color 4. Color Master's default colors are 1 (black), 2 (red), 3 (white), and 4 (blue).

Press the space bar, and a ball appears in the corner. You can move the ball around in the edit box just as you did the pointer. The fire button or f7 will toggle the numbers and their colors.

Notice that two spaces indicate one color. Instead of eight bits across the box, there are only four. When using multicolor graphics, the bits drawn on the bitmap screen actually indicate what color will appear. Since a bit can be either on or off, one bit could only represent two colors. So two bits are used, which allows four different settings. Bit pattern 00 indicates background, which is the actual screen color at 53281. Bit pattern 01 is color 2, and 10

indicates color 3. These two colors are stored as one byte in actual screen memory starting at 1024. Color 2 is stored in the upper four bits, and color 3 is stored in the lower four bits. Last is bit pattern 11, color 4. Color 4 is stored in screen color memory starting at 55296.

Using the above procedure, draw two rows of each color. If the screen is the same color as one of the four, just readjust it with the 1 key. Exit Zoom mode by pressing Return.

You will see your color pattern under the pointer. If you press f2, the pattern will be memorized. Move the pointer and press f4 to stamp it in another place. This procedure works well for filling an area with a repetitious pattern. Stamp a couple of short rows of these.

Move to one of the patterns and press the 2 key. Notice that one of the colors in the pattern has changed. Continue to press the 2 key and watch as it scrolls up through the colors. The 3 and 4 keys work the same way with the other colors. Using these keys in conjunction with the Commodore key will cause them to scroll down through the colors. Shift and the 1 key (Shift-1) will change the screen color.

Move to other places where you stamped the pattern and adjust the colors so that you have an assortment of different color patterns.

You can move to any of these parts of the screen and zoom in to see that color 2 may be one color in one place and another color in another place. The same applies to color 3 and color 4. If you wanted a color, such as 2, to be blue throughout the screen, you would press Shift-2 from the bitmap, not from the Zoom mode, and enter into Equalize Color mode.

Before entering Equalize mode, press Shift-B. This stores the current colors into a buffer so that if you don't like the changes, you can recall them as they were before. Press Shift-2. Notice a string of colors with a arrow pointing to the current setting for color 2. Use the joystick or cursor keys to move to the desired color. When you're ready, press the fire button, or f7. You will be asked if Color Master should equalize color 2. If you press N, you will be returned to the bitmap without any changes being made. If you press Y, then Color Master changes all of the colors represented by 2, or bit pattern 01, to the selected color and returns you to the bitmap. Notice the change.

Shift-3 and Shift-4 work similarly with those respective colors, bit patterns 10 and 11. To recall the buffer after an equalize, press Shift-P.

While drawing using Sketch-Pad or Doodle, each bit represents a color other than the background. So you can understand how messed up a bitmap created by one of those drawing programs might look in multicolor mode.

Color Master has another mode that lets you equalize the actual bits. A Sketch-Pad bitmap is drawn in one color. In Color Master, since colors are represented by the placement of the bits, all the bits would have to be rearranged to be one pattern, or colors 2 to 4 would have to be set to the same color to make the Sketch-Pad display appear correctly. Setting all the bit patterns to the same color would defeat the purpose of Color Master.

The logical thing to do would be to make all the bit patterns the same. You will lose some of the fine detail of standard high resolution, but after you've touched up here and there, your screen will explode with color and detail. That's what Color Master is all about.

To adjust the bit patterns, press Shift-C to enter Bit Change mode. The bit settings will be displayed, and Color Master will wait for you to make a selection. Select the pattern that you would like to change. You will then be prompted to select a pattern to change to.

The directory of drive 8 can be viewed at any time by pressing the left-arrow key. Press the space bar to halt or restart the listing. Press Return to exit the directory.

Any file that loads into address 8192, such as a Sketch-Pad file, can be loaded. Just use Shift-L. You will be prompted for a filename. Enter the name just as it appears on the directory listing and press Return. If you would like the directory again at this point, simply enter a $ and press Return. The file will be loaded directly into the bitmap without being altered.

Doodle files load differently. Color Master tries to mimic Doodle by utilizing Doodle's color memory and doing a couple of bit changes.

When you are ready to save your completed or partially completed work, press Shift-S. Enter a filename and press Return. Color Master affixes the prefix CM. to all saved files.

To clear the bitmap, press Shift-Clr/Home key. It should be noted that you will not be prompted and your map will be completely lost unless you have it saved on disk.

To exit Color Master press Shift-Q.

SUMMARY OF COMMANDS

| | |
|---|---|
| Shift-1 | Screen Color |
| Keys 2-4 | Scroll Colors |
| C= Key and 2-4 | Scroll Down Through Colors |
| Fire Button or f7 | Zoom Mode |

| | |
|---|---|
| Cursor Keys or Joystick | Move Pointers |
| Key 1 in Zoom Mode | Screen Colors |
| Shift-2-4 | Enter Equalize Color Mode |
| Shift-C | Change Bit Pattern Mode |
| Shift-B | Put Color Memory in Buffer |
| Shift-P | Pull Color Memory from Buffer |
| f2 | Memorizes Memory Under Pointer |
| f4 | Stamps Pattern to Bitmap |
| * | Scrolls Pointer Color |
| + | Moves Pointer Faster |
| - | Moves Pointer More Slowly |
| Shift-Clr/Home | Clear Bitmap |
| Left Arrow | Disk Directory |
| Shift-L | Load File |
| Shift-S | Save File |
| Shift-Q | Exit Color Master |

Daniel Lightner lives in Sidney, Montana.

Gazette, July 1994

LISTMAKER

By Robert Nellist

A utility for the 64 to make and manage almost any kind of list.

Everyone makes lists. Housewives make grocery lists. Little League coaches and Scout leaders require rosters. Avid readers keep track of the books they've read. Travelers need a list of items to take on a journey. Just as Santa has his gift list, con artists keep a sucker list, and the Mafia even has a hit list. The list of people who need lists is endless. It's hard to imagine anyone going through life completely listless.

MAKING A LIST

Listmaker is designed to be versatile and easy to use. When you load it, you should have one of two things in mind: You will either want to start a new list or work with a previously saved one.

This choice is presented as soon as you run the program. You have the option of pressing E to enter a new list or L to load an old one. If you've already created a list and saved it to disk, press L to load it. At this time, you'll have the option to view the disk directory if you can't recall your list's filename.

If you select E to enter a new list, you'll be asked to enter a title for it. (It's a good idea to save the Listmaker program to a work disk. You'll need such a disk for your lists.) The title will be centered above your list in the printout. The Entry screen appears next. It can accommodate as many as 224 entries. Each entry can be any length up to a maximum of 74 characters and spaces. When this limit is reached, further typing will not be accepted.

To prevent problems, the cursor keys are disabled during entry. You can make corrections within the current entry by erasing text with the Del key and then retyping. Don't worry if a mistake slips by. A Delete option will let you clean up things later. All punctuation is valid with the exception of the quote mark, which has been disabled. When you've finished entering items for your list, press the up-arrow key on a blank line. This will return you to the main menu.

CHECKING IT TWICE

You can easily make additions or deletions by selecting the appropriate option from the main menu. To delete, press D and use the space bar to page through the list. When you've located the offending entry, place the cursor on it and vaporize it by pressing Ctrl-D. Each time you delete an entry, the listing will restart. Return to the main menu at any time by pressing the up-arrow key. Entries that occupy more than one screen line will be abbreviated in the Delete mode. To see a complete listing, choose the View option.

PUTTING IT IN ORDER

Pressing A from the main menu will alphabetize the entire list. This

can take quite some time when dealing with a long list. Keep in mind that entries beginning with an uppercase letter will be alphabetized separately from those beginning with a lowercase letter. Remember, also, that alphabetizing or deleting will change the entry numbering. These numbers are for reference only and do not appear in the printout.

STORING IT

The Load and Save options are conventional, and both permit viewing of the disk directory, if desired. Either option can be canceled by pressing Return before typing a filename. Saves are made as a precaution after scratching any existing file of the same name. This lets you replace a list with an updated version simply by saving under the original filename.

GETTING IT ON PAPER

When you press P to select Printout, the program automatically checks the entire list and calculates the length of the longest entry. This number is then used to determine what column options, if any, will be offered to you in the printout. Short lists of ten entries or less will always print in a single column. If the Column Choice screen appears, select the desired number of columns by pressing the appropriate number. The Form Feed option is set to occur whenever any one column contains 56 entries.

SAVING YOU FROM YOURSELF

Since the Quit and Start New List menu choices erase any lists that may be in the computer's memory, these selections require verification before execution. With the lone exception of Alphabetizing, you can escape from any option you may have entered accidentally. It almost goes without saying that you must avoid pressing the Run/Stop key. If your menu choices appear to be ignored, check to be sure the Shift Lock is off.

Robert Nellist lives in Brockport, New York.

Gazette, July 1994

# GAZETTE FILE CONVERTER

A conversion program for Gazette Disk and SpeedScript files. Versions for the 64 and 128.

The text files on Gazette Disk are in SpeedScript format and these documents are saved on disk as program (PRG) files. Normally, you can load and run PRG files from BASIC, but you can't run this type of text file. The characters are stored in their screen code (Poke) equivalents. Several other word processors store text similarly, including Word Pro and Paper Clip. That means you can load Gazette text files into these processors for reading, printing, or editing.

Many other word processors store their documents as sequential (SEQ) files, either as Commodore ASCII or true ASCII. You can Gazette File Converter to convert Gazette files for use with these word processors.

The file converter can also convert a Commodore ASCII sequential file into a screen code Gazette or SpeedScript file (or vice versa.) It can also translate Gazette or SpeedScript files into true ASCII. This is a convenient feature whenever you want to send a SpeedScript file via modem to an IBM computer.

With this conversion program, you can save the converted files to disk, print them to the screen, or send them to your printer in the desired format.

Gazette, July 1994

GLOBE

By John Crider

Globe and Warp Field are public domain programs discussed in this issue's "PD Picks."

Globe consists of a spinning globe, and its controls are simple. A joystick in port 2 controls the angle from which you view the globe. The plus and minus keys control the speed of rotation.

WARP FIELD

By Amenophis

This graph program plots the values of eight variables. Use a joystick to change the variables to create a variety of interesting patterns.

Gazette, July 1994

64/128 VIEW: Readership Survey

By Tom Netsel

Now that the new Gazette Disk has been out for a few issues, I think it's time we found out more about our subscribers. We want to make this product as useful and entertaining to you as possible. So here's a chance to tell us about yourself, your equipment, and your interests. Please take a minute to print out and complete this survey. You may write in your replies by hand or load this file into your word processor and use it to reply.

Mail your survey to Gazette Survey, COMPUTE Publications, 324 W. Wendover Ave., Ste. 200, Greensboro, NC 27408. Feel free to use the back for additonal comments.

You may also send your surveys by fax to (910) 275-9837, or by E-mail to Gaz on Quantum Link or TomNetsel@aol.com via the Internet.

1. What is your primary computer? 64____ 128____ Both_____

2. If you use a 128, do you have an 80-column monitor? Yes____ No____

3. How long have you owned a Commodore 8-bit machine?

4. Before the switch to disk format, had you ever used a Gazette Disk? Yes____ No____

5. Do you program in BASIC? Yes____ No____

6. If so, do you consider yourself to be Beginner, Intermediate, Advanced programmer? (Circle one.)

7. Do you program in machine language? Yes____ No____

8. If so, do you consider yourself to be Beginner, Intermediate, Advanced programmer? (Circle one.)

9. What is your age? ____ 10. What is your sex? M____ F____

11. How long have you subscribed to Gazette or Gazette Disk?

12. What types of programs do you want on Gazette Disk?
More Games____ More Utilities____ A Mix of Both____

13. Do you use an REU? Yes____ No____ If so, what type?

14. Do you use a 1541 disk drive? Yes____ No____

15. Do you use a 1571 disk drive? Yes____ No____

16. Do you use a 1581 disk drive? Yes____ No____

FEEDBACK

Questions and comments from our readers.

BUG-SWATTER
Bug-Swatter is where we print program modifications and corrections to earlier programs and disks.

Joseph Koen of Avalon, New Jersey, reports a problem loading the SpeedCalc instructions on the March 1994 Gazette Disk. The instructions stop before reaching the end of the file. It appears that the 124-block instruction file for SpeedCalc is too large for TextReader64 to handle with the rest of the menu program in memory. The instructions for Batch Files may also have this problem.

We've modified Textreader64, the program used to display the text files on this disk, and installed a new version of it called Text Reader. This new version should handle large files without any problems.

To read the entire SpeedCalc documentation, there are two solutions. If you have SpeedScript, you can run it, then place the March disk in your drive and load the SpeedCalc documentation into SpeedScript as you would any text file.

If you don't have SpeedScript, you can use the new Text Reader with a short program on the flip side of this disk called Filereader. When you run Filereader, you'll be asked to insert a disk that contains the new Text Reader. When that's done, Text Reader is loaded into memory.

Next, you'll be asked to insert a disk that contains the Gazette PRG file that you want to read. Filereader will then search for and attempt to load that file. Make sure that you enter the filename exactly as it appears on disk.

In order to read the SpeedCalc documentation, load and run Filereader on side 2 of this disk. Then insert side 1 where the program can find the new Text Reader. Then insert side 1 of the March disk and enter the filename SPEEDCALC. The entire documentation should load properly.

To cut down on disk swapping, you may want to put all three files on one work disk. To do this, you can use the File Copier program from the May Gazette or any other copy program to load and save the programs to another disk.

HELP WITH MULTIRAID
MultiRaid (February 1994) sounds like a good game, but I am unable to get it to load. Can you offer any help?
LARRY NESSRALLAH
LAVEL, PQ
CANADA

We're sorry to learn that you're having problems with MultiRaid. It's a rather large public domain program and it can be temperamental. It takes several minutes to load, and the screen should changes color throughout the loading process. Watch for the light on your drive to indicate that it is still loading.

Try loading the program without using the Gazette menu. You can load the game from side 2 by typing LOAD "MR",8 or by typing LOAD "MULTIRAID" ,8,1 and pressing Return.

Do you have any Fast Load or other cartridges plugged into your 64? If you do, unplug them, reboot, and try loading MultiRaid again. That should solve the problem. Be sure to unplug any cartridges whenever you have any problems loading a program.

Also, this public domain program probably originated in Europe where the TV and monitor standards are slightly different from those found in the U.S. There appears to be a score box at the bottom of the screen that will not fit on U.S. monitors. MultiRaid is still an interesting program that can help young children practice their math skills.

READER TO READER HELP

I was recently typing some text using Word Writer 4, and I found that I could not enter the symbol for pi.

Fortunately, I have three other word processing packages: geowrite, SpeedScript, and WordPro. I was sure one of those programs would let me use the symbol for pi when typing text. Much to my dismay, I found that none of them would.

Can anyone tell me if it's possible to alter any of these programs to allow me to do so?

FLORA DAIGNEAU
RR#1
ATHENS, ON
CANADA K0E 1B0

You can define a programmable key to print most any ASCII character with SpeedScript. (Other readers may be able to tell you how to define a key for the other word processors.) You define a key and assign to it the ASCII value of the character that you wish to print. In this case, the ASCII value for pi is 126. When you press that printkey, the Greek letter pi won't appear on the screen, but it will appear when you print it on paper.

To define a printkey, put your cursor at the very top of your document and press Ctrl-3. You'll see a prompt at the top of the screen telling you to press a format key. The format key is the one that you'll use to print the pi symbol. You can use most any uppercase letter. Since pi on the Commodore keyboard is on the up arrow key (↑), you may want to select that key as the printkey for pi.

Next, hit the equal sign (=) and then type 126, the ASCII value for the pi symbol. Whenever you want to print pi in your text, press Ctrl-3 and the up arrow key.

This is the process for defining a printkey to print most any desired ASCII character, but it may vary somewhat depending on your printer. On some, you may first have to switch into uppercase mode before printing the pi symbol, and then switch back out to continue with your normal upper and lower case printing. Check your printer manual for any special codes that it may require.

GAZETTE FILE CONVERSION
I'd like to print the instructions on Gazette Disk with my own word processor, but it won't load Gazette's PRG text files. Isn't there a way to convert them to SEQ file format?
DENISE WARRELL
CHICAGO, IL

As mentioned above, the text files on Gazette Disks are in SpeedScript format. If you have a copy of SpeedScript, Gazette's word processor, you can use it to load, read, and print any of these PRG files from the disk. Several other commercial word processors, such as Word Pro and PaperClip, can also load and read PRG text files.

If your word processor requires sequential (SEQ) files, however, there is a way to make the conversion. In the program menu, look for a utility to convert Gazette's PRG files to SEQ format. SpeedScript/Gazette File Conversion converts SpeedScript files to Commodore ASCII or true ASCII sequential files. It can also be used to convert Commodore ASCII files into SpeedScript format. There are versions for the 64 and 128. This handy utility can also be found on the SpeedScript disk.

64 FOR SALE
I have a complete 64 system with lots of software that I no longer need and would like to sell. Can I advertise it through your disk magazine?
ADAM PARKS
ALEXANDRIA, VA

Yes you can. Starting with this issue of Gazette Disk, we now offer classified ads at sufficiently low prices to appeal to individuals, user groups, and small businesses. You can find these ads and information about placing your own ad in the Advertisement section of Gazette Disk.

D'IVERSIONS: The Fork in the Road

By Fred D'Ignazio

During the decade from the 1980s to the early 1990s, cyberspace was growing rapidly, but it was still behaving nicely. It had divided into four major road systems of PLUs (People Like Us).

Highway 101: The personal computer hobbyists and enthusiasts.
Highway 202: The serious business computing users.
Highway 303: The hitchhikers, truckers, cabbies, and road warriors.
Highway 000: The hackers, nerds, rappers, and videogame gangstahs.

Each PLU had its own language, lifestyle, and comfort zone. Each group saw computers in a unique and insulated light.

The hobbyists on Highway 101 carried Apple IIs, Commodore 64s, and Atari computers under their arms into classrooms and living rooms. They often spoke of the educational value of computers.

The business users on Highway 202 spoke of a magical world called the desktop in which all reality could metaphorically be transformed into pictures (or semi-religious icons) which resembled items on the desktop. All waste could be neatly tucked into a little trash can. All information could be tidily sorted into cute little folders. The nasty messiness of the world could be tamed, cleaned up, color coded, and quickly processed in desktop land.

The road warriors on Highway 303 represented the first true denizens of cyberspace--people with accounts on Internet, Prodigy, CompuServe, GEnie, America Online, and so on. These folks had long ago abandoned the quaint notion that personal computing was conducted in an isolated one-computer cul-de-sac, tucked away like a hermit crab in an office or bedroom. These folks spent their time on the open road searching the company, the community, and the globe for new interactions with people, data, and systems.

Then there were the other folks, arcade piranhas and evil hackers, lurking inside middle-class bedrooms and suburban shopping malls, masquerading as innocent children. These interlopers represented the real challenge to the elasticity of cyberspace. They traveled down Highway 000 on garish, brightly painted vehicles that looked nothing like the cute little 64s of the hobbyists; the prim, proper office appliances of the business users; or the neat text-based, bookish computers of the road warriors. Instead, they hurtled through cyberspace sitting on computers that looked more like motorcycles, spaceships, and army tanks and manipulated screens which looked like the nightmarish cross-dressed imaginations of Stephen King, Jean-Claude Van Damme, Madonna, and Saddam Hussein.

And they didn't obey any of the road signs of cyberspace, drive defensively, or use proper manners.

What happens when Highways 101, 202, 303, and 000 all run together in a mega-intersection? Cyberspace shock. The point where realities collide.

This explains the effect of my game-arcade videotape on the teachers and children in the elementary school ("D'Iversions," May 1994). The kids identified blindly with the videogames as part of the mid-1990s youth culture. They were clearly the travelers on Highway 000 (or else they identified with older kids and kids on TV who are actually out on the road).

The adults were equally blind in reacting against the games. They were obviously motorists on Highway 101. They identified the games as being part of a hostile, antischool, antiadult youth culture. This was foreign to their Highway 101 notion of computers as educational, wholesome tools for quiet, thoughtful classroom use. Well-dressed, well-behaved students use classroom computers. Badly dressed, ill-mannered outlaw children are found at game arcades.

The teachers were good Highway 101 motorists. They had been trained by a certified driving instructor. They knew the rules.

But the kids in the video arcade out on Highway 000? Forget it! Didn't they know that the proper control panel for a cyberspace motorist was a keyboard? That the only language spoken in cyberspace was a simple language of monosyllabic words or long columns of data fields and numbers?

Highway 101 motorists believed it was OK to be a little experimental. Some of the more daring motorists had even learned to operate a small device known as a mouse. You could roll it slowly around on a desk to make simple point-and-click selections on the computer screen. Mouseketeers were held in high regard by fellow navigators on Highway 101.

There were deeper points of disagreement. It wasn't just the joysticks and the display screens that affronted the motorists on Highway 101 and made arcade games look so un-computerlike. Just look at the awful way kids in arcades used their computers while standing. Didn't they know that when you enter cyberspace you're supposed to be seated at a keyboard, two and a half feet from the display screen, with your bottom resting snugly on a comfortable chair? With the gangs out on Highway 000, proper cyberspace posture has gone down the tubes. The kids stand; they sit; they lie down; they play multiplayer games in darkened rooms like denizens of a sinister 1950s pool hall. Don't they realize that cyberspace is a well-lit world, with fluorescent lighting pertly shining on brightly tiled and carpeted floors?

"And what about the smile you wear in cyberspace?" asked the corduroy and cardigan crowd on Highway 101? "And the educational thoughts you think?"

"And how about the work that must get done, the columns of numbers totaled, and the records to be filed? How about the serious business ahead?" The clipboard-toting, water-cooler team on Highway 202 had lots of questions.

"And how about the prosocial, multicultural, anonymous conversations you can have and the necessity of expressing your thoughts through words?" ask the hitchhikers, road warriors, and taxicab drivers on Highway 303.

None of today's groups (with perhaps the exception of some road warriors) sees beyond the games to a point of multidimensional convergence: where all groups' concepts of computing rush together toward ultimate, unknowable impact?...fusion?...blending?...destruction?

Gazette, July 1994

# BEGINNER BASIC: Rounding

By Larry Cotton

One of the challenges I gave you last November was to submit a homegrown program that utilizes number rounding. Well, the programs are in, the votes are tallied, and the winner is (insert drumroll here) Rex Merritt. Rex sent me a program that not only rounds dollars and cents, but it also places the figures in neat columns at the edge of the screen.

As you probably know, to round a number is to shorten it to a more manageable size. If a mathematical calculation produces a number such as 34.36437, we can round it to any lesser degree of accuracy we wish.

If we want to round a number to two decimal places, we look at the third digit to the right of the decimal point--in this case, 4--and compare it to 5. If the number is equal to or greater than 5, the digit to its left (6 in this case) is increased by 1. If the examined digit is less than 5 (as in this example), the number to its left is not changed. Thus, 34.36437 rounded to two decimal places becomes 34.36.

To round numbers in BASIC, we use the INT (integer) command and raise a number to a power. Don't let that scare you. Raising a number to a power simply means to multiply the number by itself a certain number of times. To raise 5 to the second power, multiply 5 x 5 to get 25. To raise 8 to the third power, multiply 8 x 8 x 8 to get 512.

The general formula for rounding--and the one Rex used in his program--can be expressed by this line of code.

```
R=INT(N*10↑D+.5)/10↑D
```

R is the rounded number that we seek, N is the number to round, and D is the number of decimal places to which to round it. Here's a minimalist rounding program.

```
10 PRINT"[CLR]"
20 PRINT:INPUT" NUMBER TO ROUND";N
30 PRINT:INPUT" TO HOW MANY PLACES";D
40 R=INT(N*10↑D+.5)/10↑D
50 PRINT:PRINTR
```

Run the program and enter 453.567552 at the first prompt and 3 at the second. What does the program do with these numbers? You may recall from previous columns that there's an order of math operations. Here's the order in which operations are performed.

1. Parentheses

2. Raising a number to a power
3. Multiplication
4. Division
5. Addition
6. Subtraction

Our rounding formula will be executed in the above order, so let's attack the parentheses first.

(N*10↑D+.5)

Within the parentheses, first the 10 will be raised to the power of D (10 will be multiplied by itself D times). Since the value for D in our example is 3, 10 will be multiplied by itself three times; it becomes 10 X 10 X 10 or 1000. Then the multiplication within the parentheses takes place: 453.567552 X 1000 = 453567.552.

We're still inside the parentheses. Now we add .5 to 453567.552 to get 453568.052. Now apply INT to that number. Remember that INT simply drops all decimal places to leave a whole number. In this example it's 453568. Our formula has now been reduced to this.

R=453568/10↑D

What's next? Dividing by 10? Or dividing by 10↑D? Remember from the order of operation that powers are computed first. Therefore, raise 10 to the third power again to get 1000. Now our formula is reduced to this.

R=453568/1000

Dividing by 1000 yields 453.568, which should have been the result of running the program and entering the two suggested values. Line 50 prints the answer.

Try entering various numbers and rounding them to different decimal places. If you type a value for D greater than the number of decimal places, you may get an overflow error.

Don't confuse rounding a number with finding its integer value. Rounding usually leaves decimal places, and a particular digit may increase by 1. Finding a number's integer value just drops the digits after the decimal, leaving a whole number. Here are some examples.

* 10.583 rounded to one place is 10.6.
* The integer value of 10.583 is 10.
* 10.583 rounded to a whole number (zero decimal places) is 11.

The uses for rounding are varied, but probably one of the most common is in dealing with money. Since in America we usually round to two places (hundredths of a dollar, or cents), we can specialize our rounding formula.

Start with this equation.

```
R=INT(N*10↑D+.5)/10↑D
```

Substitute 2 (two decimal places) for D.

```
R=INT(N*10↑2+.5)/10↑2
```

Raise 10 to the second power, or square 10, which yields the following.

```
R=INT(N*100+.5)/100
```

It's a temptation, but we can't add 100 and .5 to get R=INT(N*100.5)/100 because that's not the correct hierarchy of operations. N must be multiplied by 100 before we add .5.

Since 100 appears twice in the formula, let's assign a variable name H (hundred) to it.

```
H=100:R=INT(N*H+.5)/H
```

Here's a program that computes the average price to the nearest cent when we buy a certain quantity of eggs, screws, or whatever. It uses our new rounding formula.

```
10 PRINT"[CLR]"
20 INPUT" HOW MANY TO BUY";Z
30 PRINT"[DN][SPC]PRICE FOR"Z";
40 INPUT"PIECES";P
50 N=P/Z: REM ACCURATE PIECE PRICE
60 H=100: R=INT(N*H+.5)/H: REM SPECIALIZED ROUNDING FORMULA
70 PRINT:PRINT" AVG. PRICE EA. ="R
```

Another common use for rounding is in scoring games. We can create a specialized formula that uses 0 as the value for D, since we want our result rounded to a whole number, with no decimal places.

We start with our general formula.

```
R=INT(N*10↑D+.5)/10↑D
```

Substitute 0 for D.

```
R=INT(N*10↑0+.5)/10↑0
```

This yields an interesting result based on a strange mathematical rule: Any number raised to the power 0 is 1! X to the power 0 is 1; 14 to the power 0 is 1; 5,137,335 to the power 0 is 1. Therefore, 10 to the power 0 is 1.

```
R=INT(N*1+.5)/1 or R=INT(N+.5)
```

Now we have a new formula for rounding to whole numbers. Let's write a short program to calculate average shots per hole in three rounds of golf. Let's assume that each round consists of 18 holes.

```
10 PRINT"[CLR][BLK][DN]ASSUMING 18-HOLE COURSE[DN]"
20 INPUT" FIRST ROUND";R1
30 IFR1<>INT(R1)THENGOSUB200: GOTO20
40 Z=R1:GOSUB300
50 INPUT" SECOND ROUND";R2
60 IFR2<>INT(R2)THENGOSUB200: GOTO50
70 Z=R2:GOSUB300
80 INPUT" THIRD ROUND";R3
90 IFR3<>INT(R3)THENGOSUB200: GOTO80
100 Z=R3:GOSUB300
110 TS=R1+R2+R3:PRINT" TOTAL SHOTS FOR 3 ROUNDS:"TS
120 N=TS/54:REM 54 HOLES IN 3 ROUNDS
130 R=INT(N+.5):REM SPECIALIZED ROUNDING FORMULA
140 PRINT:PRINT" AVG. SHOTS PER HOLE FOR 3 ROUNDS:"R
150 END
200 PRINT:PRINT" NO DECIMALS, PLEASE!
210 PRINT:RETURN
300 N=Z/18:REM 18 HOLES PER ROUND
310 R=INT(N+.5): REM SPECIALIZED ROUNDING FORMULA
320 PRINT:PRINT" AVG. SHOTS PER HOLE:"R
330 PRINT:RETURN
```

Golf scores don't contain decimal values, so lines 30, 60, and 90 check for this by comparing the INPUT numbers with their integer values. If they're not the same, a decimal value must have been entered, and the questions are repeated.

Lines 40, 70, and 100 assign another value Z for R1, R2, and R3 in succession, so one subroutine at lines 300-330 (to calculate average shots per hole) can be used repeatedly.

The rounding formula, custom-tailored for whole numbers, is used in line 310 to round scores to values that contain no decimals.

This program, Golf Scores, can be found on the flip side of this disk.

Gazette, July 1994

Machine Language: Screen Zap

By Jim Butterfield

Here's a beginning level program that demonstrates the amazing speed of a machine language program. Screen Zap works on the 64 and the 128 in 40-column mode. Its function is to read from the keyboard and poke the raw key value to every location of screen memory.

Screen memory may be found at addresses 1024 to 2023 on the 64 and 40-column 128. If you haven't met screen memory before, you might like to try typing the BASIC command: POKE 1024,1. You'll see the letter A appear at the top left of the screen. Number 1 is the screen code for the letter A. Bear in mind that the screen codes are not the same as regular ASCII codes, where the letter A would be coded as number 65. What you will see on the screen will not correspond precisely with the key you have pressed.

A BASIC program to do the job that Screen Zap performs would be coded along the following lines.

```
100 GET X$ : IF X$="" GOTO 100
110 X = ASC(X$)
120 FOR J=1024 to 2023
130 POKE J,X
140 NEXT J
150 GOTO 100
```

If you'd try this BASIC program, you'd notice that it's sluggish. You would see the characters arrive as they were written by the loop at lines 120 to 140. Machine language makes the effect instant.

Let's trace the BASIC code and its equivalent in machine language. GET X$ will be matched by a call to the GET subroutine at hexadecimal address FFE4. The character will arrive into the A register. If no character is waiting, a binary 0 will be placed in the A register. Using the JSR (Jump Subroutine) instruction, we code:

```
  JSR $FFE4.
```

That's the first instruction; but a beginner might have missed a couple of steps such as deciding where to put the instruction and learning how to get it into the computer.

I often choose address 8192 (hexadecimal 2000) as a site for simple machine language programs. It's not an ideal location, but it's available for use on most Commodore 8-bit computers. And it saves me the trouble and possible confusion of outlining different code for different machines. We'll stay with $2000 (the dollar sign signifies hexadecimal) for this example.

To enter a machine language command, you'll need a machine language monitor program. On the 128 (and the Plus/4), there's one built in.

Just type the command MONITOR, and you're there. On the 64, you'll need to load in a separate MLM (machine language monitor) program. There are many public domain MLM programs available, such as SuperMon, which appeared on last month's Gazette Disk.

Assuming you have your monitor program in place and active, you're ready to enter your first instruction, JSR $FFE4, into the computer at address $2000. To assemble this instruction, just type the following line.

```
A 2000 JSR $FFE4
```

If you get a SYNTAX ERROR message, you're still in BASIC. Go to your MLM on the 128 with MONITOR or SYS 8 (or RUN on the 64), and try it again. If you were in the monitor system and typed the line correctly, it should have changed to the following when you pressed Return.

```
A 2000 20 E4 FF  JSR $FFE4
A 2003
```

The computer has taken your JSR $FFE4 command and has translated it to the three bytes: 20 E4 FF. Those three bytes are placed in memory. You might correctly guess that 20 is the code for JSR; further inspection will show address FFE4 has been flipped for storage in two bytes.

The line you typed has been rewritten to show this translation, called an assembly. Further, the computer expects that you will probably want to type in more instructions and helps you by typing the start of the next line. The next memory address that you will use will be $2003, and that's the value that the computer types.

So far, we've entered the equivalent of GET X$. Let's proceed to the BASIC statement IF X$="" GOTO 100. The input character is in A; we can compare A to 0 with the CMP (Compare) instruction. If A is equal to 0, BEQ (Branch Equal) will take us back to address 2000. Here are your next two lines.

```
CMP #$00
BEQ $2000
```

Again, the lines will change when you press Return. If you make a mistake, you'll see a question mark (?) at the end of the line. To fix it, just type over it.

By the way, the CMP line has a number (#) symbol in front of the 0 value. The symbol indicates immediate addressing; it instructs the computer to use the value 0, not the contents of address 0.

Next, we want to store the value in register A into our screen memory addresses 1024-2023. In fact, we'll go a little further and cover the range from 1024-2047, which won't hurt a thing. We will use a technique called indexing, in which the X register will count over the locations to be changed. Start by setting X to 0.

```
LDX #$00
```

Again we use the number (#) symbol to signal we want value 0, not the contents of address 0.

Note that the next address to come is hex 2009. We'll need to remember that address; that's the place to which the program will loop back to store to the next screen address.

The X register cannot step through 1000 values. Like any other 8-bit register it has only 256 possible values, from 0 to 255. How can we cover the whole screen range?

To cover a wide range of memory, we often turn to a programming technique called indirect addressing, but we won't need to do that here. We'll just write four separate store commands, each covering a quarter of the screen. If we multiply 4 times 256, it gives us 1024 locations. We use the STA (Store A) command.

```
STA $0400,X
STA $0500,X
STA $0600,X
STA $0700,X
```

That takes care of four locations. Now we'll increment X to the next higher value (INX). Then we can branch back to 2009 and repeat the loop. We use BNE (Branch Not Equal) so that we will take the branch as long as the new value of X is any value other than 0. Eventually, X will go over the top, from 255 to 0. At that time, the branch will not be taken, and the program will proceed to the next step.

```
INX
BNE 2009
```

It would be nice to have a way to stop this program. Let's do it by pressing the space bar. The character value will still be stored in the A register. For a space, that value would be 32 (hexadecimal 20). So we'll compare to that value, and it's not equal. Therefore, back we go to repeat the loop.

```
CMP #$20
BNE $2000
```

If the program does not branch back, the user must have pressed the space bar. In this case, we quit. More exactly, we return to whatever called this program, often RTS (Return from Subroutine). For this program, that means a return to BASIC.

```
RTS
```

Our program is finished, but the monitor still prompts us for the next line, with A 201D. Just press Return to signal you have no more code.

Check the listing on the screen. It should look something like the following.

```
A 2000  20 E4 FF JSR $FFE4
A 2003  C9 00    CMP #$00
A 2005  F0 F9    BEQ $2000
A 2007  A2 00    LDX #$00
A 2009  9D 00 04 STA $0400,X
A 200C  9D 00 05 STA $0500,X
A 200F  9D 00 06 STA $0600,X
A 2012  9D 00 07 STA $0700,X
A 2015  E8
  INX
A 2016  D0 F1    BNE $2009
A 2018  C9 20    CMP #$20
A 201A  D0 E4    BNE $2000
A 201C  60       RTS
A 201D
```

Press the X key to return to BASIC. Now you're ready to run the program. Since our program begins at address 8192 (hex $2000), we type SYS 8192. Remember that SYS calls use the BASIC address, not the hexadecimal.

Now play with the keyboard and watch the screen change like lightning. Try the shift and Control key combinations, too. When you're finished, tap the space bar.

If you don't have either a 128 with its built-in monitor or an MLM for your 64, you can still load and run Screen Zap on the flip side of this disk. It's a BASIC loader that pokes the equivalent machine language code into memory.

Gazette, July 1994

PROGRAMMER'S PAGE: Make Room for Data

By David Pankhurst

No matter what program you're using or writing, data is what makes it tick. And finding places to store that data is a never-ending chore for a programmer. This month we're going to look at some tried-and-true tips for handling data, and we'll also examine some novel ways to stash info.

THE UBIQUITOUS DATA

The most common way of handling data is with BASIC's DATA statement. It's simple to use, effective, and it's found in all versions of BASIC. The following program lets you take a block of memory and convert it into DATA statements. Enter the location of the first and last byte to copy, and supply a line number from which to start the data lines.

```
1 REM  DATA CONVERTOR
2 INPUT"MEM START, END, & 1ST LINE
#";A,Z,L:X$=STR$(A)+","+STR$(Z):GOTO6
3 IFA>ZTHENEND
4 Z$=STR$(PEEK(A)): X$=X$+RIGHT$ (Z$,LEN(Z$)-1): A=A+1
5 IFA<=ZANDLEN(X$)<65THENX$=X$+",": GOTO3
6 PRINT"[CLR][DOWN][DOWN]"L"D
[SHIFT-A]"X$:PRINT"A="A":Z="Z":L="L+10":G[SHIFT-O] 3: [HOME]";:
POKE631,13: POKE632,13
7 POKE198,2:END
8 PRINT"[CLR][DOWN][DOWN]"X: PRINT "X="X+1": G[SHIFT-O] 8:[HOME]";:
POKE631,13:POKE632,13: POKE198,2:END
9 READX,Y:FORI=XTOY:READX:POKEI,X:NEXT
```

Line 8 provides a quick way to delete these lines. Just type RUN 8. Line 9 remains behind as the finished code for use with your DATA statements. You can simply renumber these lines and use them in you own program.

READER TIP

Kirk Fontenot, of Lafayette, Louisiana, sent in a tip that can help you when working with DATA lines. When an error occurs as your program is reading DATA statements, you often need to know what item of data did the evil deed. To print the current DATA line number, enter the following line in immediate mode.

```
PRINT PEEK(63)+256*PEEK(64)
```

TUCKING IT BEHIND

DATA statements offer programmers quite a few advantages, but they also have some serious disadvantages. This downside is especially true if you're using DATA statements to store byte values. They're slow to read and take up a lot of room (on the average, more than 3.5 bytes for each byte value being poked). Even for string data, the

information takes twice as much room as it should. The string is stored first in the DATA line and then again at its destination. All this adds up.

Machine language can be the answer to this problem. The next program takes data bytes from memory and attaches them to the program. It also provides a machine language copy routine.

```
0 REM  COPY DATA BEHIND PROGRAM
1 INPUT"START, END ADDR.";S,E:
GOSUB63999:B=X:X=X+46+E-S:Y=INT(X/256):Z=X-256*Y
2 PRINT"[CLR][DOWN][DOWN][DOWN]P [SHIFT-O] 45,"Z":P[SHIFT-O]
46,"Y":C[SHIFT-L]:S="S:PRINT"[DOWN][DOWN]E="E":B="B":G[SHIFT-O]
4[HOME]"
3 POKE198,2:POKE631,13:POKE632,13:END
4 FORI=BTOB+42:READX:POKEI,X:NEXT:
Y=INT((E+1)/256):Y=E+1-X*256:POKEI,Y
5 POKEI+1,X:I=I+2:FORJ=STOE:POKEI,PEEK
(J):I=I+1:NEXT:END:DATA165,20,24,105,43
6 DATA133,95,169,0,101,21,133,96,160,
0,177,95,133,88,200,177,95,133,89,230,95
7 DATA230,95,144,2,230,96,165,45,133, 90,165,46,133,91,76,191,163
63990 GOSUB63999:SYSX:PRINT"FILE COPIED":RETURN
63999 X=PEEK(61)+256*PEEK(62)+30:RETURN
```

The placement of the data relies on a little trick of the system. A BASIC program has two pointers associated with it: the start-of-program pointer and the end-of-program pointer. Everything between these two locations is considered program. What we do is move the end-of-program pointer up higher in memory and create an area that is still considered part of the program but follows the highest BASIC line.

When the program is run, you're prompted for the starting and ending locations of the data in memory. Room is made behind the program, and the data, along with a copy routine, is placed there. All of the BASIC lines except 63999 can then be deleted. This line points to the copy routine and shouldn't be edited. As well, it must remain the last line in the program.

Line 63990 does the calling of the machine language and the copying of the file to where it should go. Change the line number, or just add GOSUB 63999:SYS X to your own code.

This file is safely protected as part of the BASIC program, with one exception. Some programming utilities (such as MetaBASIC) can erase this section of memory when renumbering or deleting lines. If your utility does that, plan to make the copying the last thing you do before you complete your program.

LIKE AN IBM

A little known spot to pass data to a program is with the RUN or GOTO command from the screen. In the IBM DOS world, it's been a common

practice for years to pass parameters to the program that way. Although I've never seen it done on the Commodore, it's actually quite easy to set up. Here's how.

```
10 REM THIS READS IN INPUT LINE INTO X$
20 REM
30 X=512:X$=""
40 IFPEEK(X)THENX$=X$+CHR$(PEEK(X)): X=Y+1: GOTO40
50 PRINTCHR$(34)X$CHR$(34)
```

When a line is typed in direct mode on the 64 screen, the information is copied and manipulated in a buffer beginning at location 512. If the line contains any BASIC keywords, they are compacted, leading and trailing spaces are taken out, and a 0 is placed at the end of the text. Lines 30 and 40 read the line into string X$ until the 0 byte is encountered.

To use it, add a colon after you type RUN to avoid the ?UNDEF'D STATEMENT error, and then type your text.

Since trailing spaces are removed, it's easy to read in one-character values. For example, RUN:D could be read by executing the previous routine, followed by Y$=RIGHT$(X$,1).

STRINGING ALONG INFO

Ever thought of putting machine language directly in your program? Here's an example. Note line 30, the machine language code.

```
10 REM CALCULATED RESTORE LINE   - DON'T EDIT LINE 30 !
20 :
30 X$=" [CTRL-A][CMD-£]
[f4][CMD-Z][SHIFT-SPACE]1[f8][CTRL-A][CTRL-A][SHIFT-SPACE][CMD-Y][f8][C
TRL-C][CTRL-A][SHIFT-£][CMD-Y][CTRL-I]@[CMD-£][f8][CTRL-B][CTRL-A]
[CTRL-A][CTRL-A] [HOME][CMD-+][CMD-I][CTRL-A][CMD-J][LEFT
ARROW][CMD-£][CMD-+][LEFT ARROW]
[SHIFT-P][CTRL-A][F7][SHIFT-J][f4]8L$[CMD-£]"
40 :
50 SYS45195X$: X=PEEK(780)+256*PEEK (781): VG=PEEK(X+1)+256*PEEK(X+2)
100 REM DEMO
110 FORI=1TO10
120 X=INT(RND(0)*10)+520: PRINT: PRINT"READING FROM LINE"X"...":SYS
VG,X
130 READX:IFX<999THENPRINTX;:GOTO130
140 FORJ=1TO800:NEXT:NEXT:END
520 DATA 520
521 DATA 521
522 DATA 522
523 DATA 523
524 DATA 524
525 DATA 525
526 DATA 526
527 DATA 527
```

```
528 DATA 528
529 DATA 529
530 DATA 999
```

Although line 30 takes some getting used to, it is a perfectly functional machine language routine. It performs a calculated RESTORE, an appropriate program for this month's column. Line 50 looks up X$ and returns the pointer to the string contents, which is then placed in VG. Unlike a normal RESTORE, the command SYS VG,X then restores to line X, instead of to the beginning of the program.

A demo of its use follows in lines 100-140. There, it randomly restores to any line from 520-530 and prints out the subsequent values read. To use the calculated RESTORE yourself, just copy lines 30 and 50 into your own programs.

With a calculated RESTORE, you don't have to copy data into an array to access it randomly. Simply place the data on separate lines, with evenly spaced line numbers, and use the calculated RESTORE to call them when you need to. If you have various blocks of data that you need to refer to, this routine lets you read them directly, bypassing all the others.

For a happy union between BASIC and machine language, some guidelines have to be followed. The machine language has to be relocatable, and only certain codes can be placed in a string. The quote symbol (code 34) and BASIC's end-of-line marker (code 0) are definite no-nos. Avoiding these and others can make the code twice as long, as it did in this case.

Also, when strings are used in calculations (unlike line 30 above, which is just an assignment), strings are moved into string memory, above the BASIC program. Once there, they can be moved around without warning, making the calling of code in them much more difficult. Awkward, yes; but for sheer novelty, it's hard to beat.

Look on the flip side of this disk for all of the programs discussed here. They are Run Read, Data Convertor, Calc Restore, Copy Behind. Calc Restore is the program that loads from the Column menu.

If you have a programming hint or tip that might interest other 64/128 programmers, send it to the following address.

Gazette Programmer's Page
COMPUTE Publications
324 W. Wendover Ave., Ste. 200
Greensboro, NC 27408

We pay $25-$50 for each tip that we publish.

Gazette, July 1994

GEOS: Mouse Hunt

By Steve Vander Ark

I had a mouse problem recently. Since this was the first time I'd had this kind of problem, I wasn't sure what I should do. I do watch "Home Improvement" on television however, so I knew where to go for help. I simply hung around my backyard until my neighbor Max wandered by. He nodded as soon as I started to explain my predicament. He recommended peanut butter instead of cheese in the traps.

He was sure about this, although I thought such a tactic was more likely to injure my two-year-old than solve my mouse problem. But I thanked him and went back inside.

Some of you may have figured out that I'm not talking about an excess of rodents in my basement. After all, this isn't "Field and Stream." My problem is with my computer mouse. The ball from my 1531 mouse upped and rolled away like a meatball from a plate of spaghetti. Also, the mouse for my IBM compatible has, for lack of a better way to say it, lost it's grip on reality. Running GEOS without a mouse is a pain on either machine, let me tell you.

Having given up on my neighbor's macho solution, I decided to give my local computer store a try. There is a wonderful one close by with a staff that actually realizes that Commodore users are people too. Of course, they fill their shelves with Mac, IBM, and Amiga stuff, but they came up through the 8-bit ranks and still seem to care. The store has the kind of ambiance that I can relate to. It has lots of technology piled wherever you look and its walls are plastered with printouts of neat graphics, color printer demos, scribbled hacker notes, and listings of local computer shows and club meetings. My wife says it looks like my computer room, only bigger and more high-tech.

I found one of the staff and asked about a mouse. Sure enough, they had a very nice three-button mouse for my PC for less than $40. Then I asked about one for my 128. The fellow just shook his head sadly. There simply wasn't a Commodore mouse to be had. He said they don't make such things anymore. I thanked them and headed back home.

Oh well, I said to myself, I really can't complain. After all, I used GEOS for years without a mouse and got along just fine. I have an Icontroller, which is a mini-joystick that sticks onto the edge of a 128 or 64. It lets me zoom the pointer around the screen one-handed, and that's good enough for geoWrite. And the fastest way around the deskTop is with the keyboard, not the mouse.

Seriously! No pointing device can match the speed of a good set of keyboard shortcuts. If you aren't using them, you're wasting time. They're handy in geoWrite, sure, but they're absolutely dynamite on the deskTop--especially on the 128!

You can access your drives by pressing Commodore-A or Commodore-B, select files by pressing Commodore-1 through Commodore-8 (or Shift-Commodore-1 through 8 for the files on the border), and then move them up and down between the border and the page with Control-U and Control-D. Copying a file, once it's selected, is as easy as pressing Control plus the letter of the drive (A or B) you want it copied to. You can access the menu bar by pressing the left or right arrow. Once you start using these shortcuts, you'll find yourself zipping through file management chores.

I wasn't in such a bad way after all, I reasoned. But then I remembered geoPublish, an application I use most often after geoWrite and geoTerm. There was no way I wanted to go back to using a joystick with geoPublish.

When I first bought my 1351 mouse a couple of years ago, I discovered how intuitive desktop publishing became with a mouse; I was hooked for good. The idea of crawling around the 40-column screen with a joystick, even with the Icontroller, was enough to make me cry. I do own a lightpen and a KoalaPad, which can be sort of fun once in a while, but they're not really very good for serious work. I was depressed.

I took my blue mood online to GEnie, where a lot of my former Quantum Link friends hang out. I asked if anyone had an extra 1351 mouse. No one did. Then someone said, "You know, CMD is coming out with a new Super Mouse."

Of course! Creative Micro Designs to the rescue! I did one of those "I could have had a V-8" head slaps and asked for more information. As it turned out, CMD was putting the finishing touches on a new mouse for the Commodore, and it was one designed with the GEOS-user in mind. The GEOS drivers were being written by Maurice Randall, the programming genius behind geoSHELL and the 80-column publishing program he calls Finally.

The mouse itself would have three buttons and, get this, a battery-operated clock that will set GEOS's time and date automatically. The buttons would be programmed for special tasks in GEOS: left button as a single click, right button as a double-click, and the middle button for turbo mode. I called CMD and found out that in turbo mode the pointer will scoot across the screen almost instantaneously, which will be very handy for working in GEOS.

Now it just figured that CMD would be the ones to work this miracle. I honestly believe that those guys are single-handedly taking the Commodore 8-bit computers into the next millennium. The Super Mouse will sell for $49.95 plus shipping and handling, available from CMD, P.O. Box 646, East Longmeadow, MA, 01028. By the time you read this column, Super Mouse will be shipping and my mouse problem will be solved!

PD Picks: Globe and Warp Field

By Steve Vander Ark

Have you ever spent time wandering through the software libraries on an online service such as GEnie or CompuServe? If your answer to that question is no, you really are missing out. Believe me; I know.

I spend quite a bit of time doing exactly that every month as I work on this column. By far the largest software trove is on QuantumLink, which four years ago claimed to have well over 40,000 files in its libraries, including tremendous collections of SID music and graphics files. When Q-Link logs off for the last time, I sincerely hope that this vast resource won't be lost forever.

The software libraries on GEnie are also pretty impressive. While they're more up-to-date, they're nowhere near as large as those on Q-Link. The libraries on GEnie are extremely well organized and maintained, which makes my time poking around in them all the more enjoyable and efficient.

Browsing through libraries does take time; there's no doubt about it. There's no great problem finding files. I can pretty much choose a topic or a type of program and then log on to GEnie or Q-Link and find plenty of programs that fit the bill. The real problem comes in that there is just so much to choose from. If I want a game that runs on the 128, for example, I can pull 15 or 20 of them off Q-Link just like that. But in order to find the really great ones to recommend in this column, I have to test each one.

That means I get to play games and try out neat utilities. Sounds pretty cool, right? It is. I love it --but good grief! Sometimes it takes a lot of time! When schedules are getting tight and deadlines are looming, even playing games can become a chore.

So you don't exactly feel sorry for me? I understand. I do have to admit that I was very delighted when a package arrived in my mailbox the other day. It had been sent to me by a fellow named Jim Green who lives in Louisiana. Inside the package I found six disks labeled Superior Programs/Commodore Software/Commodore Sixty-Four. I dropped my 128 into 64 mode and fired up one of the disks.

It turned out that, over the years, Jim had been doing a lot of my research. He'd been collecting and organizing onto disks the very best programs he could find on Q-Link, on GEnie, and from public domain software companies. He even improved some of the programs. In his letter he writes, "I decided to start collecting PD's and, where I could, improve them, with instructions, color, sound, etc."

He wasn't kidding. When the menu program booted up and presented me with a beautiful menu from which to launch any of the many programs on the disk, I had to smile. The handiwork of a craftsman was on my

monitor. I was impressed.

I was also relieved. Jim, you see, had done a lot of the legwork for me. I realized as I experimented with the games and the utilities on the disks that all I had to do was pick any one of these programs, and I would have a winner. There was so much variety, and the files were such good, solid programs, that I couldn't lose. Since my deadline was all but come and gone, I was grateful for Jim's help.

So this month I offer you a couple of programs from one of Jim Green's disks. Each PD program demonstrates the superb graphics capabilities of the Commodore 64. You can control the screen images in various ways in both programs; there is a documentation file included that tells you everything you need to know.

GLOBE
By John Crider

Besides writing for this and other computer magazines, I am a third-grade teacher. When visitors stop by my classroom on parent night, I always like to have something running on each of the computers which looks interesting and exciting. It's a bonus if it has some educational value as well.

Well, I think I've found the perfect program for that. With this program the graphics are beautiful, and there's definitely educational value since the subject is an animated, interactive globe.

The controls are simple: You use a joystick and the plus (+) and minus (-) keys. The joystick in port 2 controls the angle at which you view the globe, which spins in the center of the screen. The plus and minus keys speed up or slow the rotation, respectively. The globe is detailed with colors to show the earth's topography. It's too small to show much detail, but that doesn't matter. The colors are beautiful, and the animation is surprisingly smooth until you crank up the speed.

WARP FIELD
By Amenophis
Music by Matt Gray

I'm not sure who or what Amenophis is, and the credits list Matt Gray's name with a question mark after it. So I have no idea who to thank for this fabulous graphics display program. (We also try to contact the authors so we can pay them an honorarium for using their programs on Gazette Disk.)

I suppose this one is educational, too, since the values for the eight variables controlling the animated mathematical plot on the screen and can be adjusted to make the image change. It's interesting, but I'm sure the math involved is beyond my third graders. Heck, it's way beyond me.

But kids will love this one. Using the joystick, they'll be able to create a wide variety of patterns on the screen, and there's nothing wrong with letting them see the relationship between the changing values on the table and the scintillating graphics. Besides that, the sheer beauty of the flowing pattern would delight anyone.

These two programs are just a little sample of the treasures on Jim Green's disks. Believe me: You'll see more in the next few months; I already have a great game and a programming utility lined up for next time. See you then!

If you have found a great PD or shareware game or utility, I hope you'll share it with our readers. Send your submissions to Gazette PD Picks, COMPUTE Publications, 324 West Wendover Avenue, Suite 200, Greensboro, North Carolina 27408.

Gaz=tte, July 1994

# ASSEMBLY LANGUAGE

By David Pankhurst

Assembly language won't help you in a foreign country, but it can help you get more fun out of your Commodore.

You can add years of enjoyment to your Commodore and stay at the forefront of programming by learning a new programming language. In this article, we're going to look at assembly language.

Assembly language lets you explore areas of the Commodore you probably didn't even suspect existed. Additionally, the knowledge and discipline gained in assembly language programming gives you an edge in learning other microprocessor languages and systems. Like a human language, learning a new computer tongue can provide years of enjoyable study and makes learning your third (and fourth) language that much easier.

The simplest way to introduce yourself to assembly language is with a monitor. Your computer has one built-in if you own a 128, but 64 monitor programs are easily obtained. (SuperMon, a public domain monitor by Jim Butterfield, was on Gazette Disk last month.)

A monitor allows you to do tasks commonly required in assembly language work, such as viewing and changing memory locations, creating and testing programs, and moving code to and from disk. These are the minimum functions; most monitors have additional options, such as the ability to step through code one instruction at a time or convert between ASCII and various number systems.

When trying to learn any new language, you need to know the vocabulary. The language of the 6502 central processing unit is no different. The vocabulary of the 6502, as well as its siblings the 6510 and 8502, consists of 56 words plus several suffixes, called operands. An operand modifies an instruction, indicating to the assembler how memory is to be accessed. Some instructions (such as LDA) have a wide variety of operands for many ways to use memory; other instructions have no operand, since memory isn't used by their processing.

For example, the instruction LDA #$F0 consists of the instruction LDA and the operand #$F0. The instruction part is a three-character mnemonic (pronounced "NEW-monic") memory aid. LDA is short for LoaD register A. (For a complete listing, see "6502 Mnemonics" elsewhere on this disk.)

The # character lets the assembler know how to assemble LDA, placing the data immediately after the instruction. And the $ indicates the number is in hex format rather than decimal.

The subject of hexadecimal numbers is an important one; if you aren't

familiar with them, a good introductory computer book can help you. Hexadecimal uses 16 numerals and letters instead of decimal's 10 numerals. Hex first counts from 0 to 9 and then continues from A to F. All math works around 16 as the base instead of 10, making for some strange-looking results. Hex's main advantage is that it's useful for computer work, since computers do their processing in powers of 2. Sixteen divides evenly into these powers, while 10 does not, so hex makes for straightforward communication between you and the computer. Being comfortable with hexadecimal is a must for serious assembly programming.

REGISTERS

Most of the words in the 6502 vocabulary make use of six registers. These six registers aren't on any memory map but are on the CPU chip itself. They're labeled A, X, Y, PC, ST, and SP. Without these registers all processing would grind to a halt.

The first register to consider is the PC, or Program Counter. This register holds a 16-bit value, unlike the 8-bit registers. Its value is a pointer to memory showing where the next instruction is to come from. The PC is constantly incremented to point to higher memory as one instruction is executed and a new one is fetched.

JUMP

Only three types of instructions can interrupt this fetch and execute cycle: jumps, branches, and subroutines. A jump (JMP) replaces the old PC value with a new one, transferring control to the new location, much like BASIC's GOTO.

BRANCH

A branch is similar, but instead of replacing the PC's value, it adds (or subtracts) a number. This offset value is a signed 8-bit number, which means a program jump of 127 bytes forward or backward is the maximum allowed. Anything further requires the JMP instruction.

SUBROUTINE

The third group, subroutines, are like BASIC's GOSUB. The address that follows the instruction is used to replace the PC's value, as with JMP. But first the old PC value is saved for later retrieval in a special place called the computer stack. This stack is a special area of memory that gets preferential treatment from the CPU. Whereas an address in memory usually has to be explicitly named in order to access it, the stack can be called by special shorter instructions that know exactly where to go. The advantages are smaller code size and greater speed.

The 6502 stack resembles a stack of dishes in that data can be placed and taken off only from the top. The Jump to Subroutine (JSR) instruction saves (pushes) the old PC value onto this stack. Because it is the topmost value, it can be retrieved (pulled) right away by the code Return from Subroutine (RTS), which takes the value off the stack and places it back into the PC. This pushing/pulling sequence allows JSR/RTS to execute subroutines in the same way as

GOSUB/RETURN.

Although the stack is primarily used by subroutines, it can also be a handy place to store data. There are four instructions that copy the A register or the status register to and from the stack. Care must be used in working with the stack, since a subroutine expects its return address to be on top when it's done. If you push or pull too much, the address on top won't be valid, and the computer is likely to fly off into unexplored territory.

STACK POINTER
In real life, no one has to be told where the top of a stack of plates is, but computers aren't that smart. So the SP or Stack Pointer is needed as an index to the stack. Its value indicates the top of the stack and is moved up or down automatically by pushes and pulls. Although you aren't likely to need to, you can manipulated the stack via instructions that copy it to and from the X register.

X AND Y
The X and Y registers are often used for convenient storage or as pointers to memory arrays. Some addressing modes take an address and add the contents of Y or X to it, giving a second address. This indexing mode is quite flexible, figuring prominently in many routines. The Y register can also be used in a second, more powerful, form of indexing, which we'll discuss later.

ACCUMULATOR
The A (Accumulator) register is the computer's workhorse. Most data has to pass through the A register at one time or another. All addressing methods work with register A instructions, making it the most versatile in working with memory. Many math operations involve the accumulator, such as adding and subtracting. It can also be used with logical operations like AND and OR, and bit shifting. (Bit shifting moves the bits in a binary number one place up or down, in effect, multiplying or dividing the number by 2.)

STATUS
The last register to consider is the ST or Status Register, which contains the status or results of the last operation on a register. It's a collection of seven valid bits (and one not in use), each of which can be 1 or 0 for a True/False or On/Off flag. One of these, the C or Carry bit, is used for holding the carryover value before and after an addition or subtraction, just as in decimal arithmetic. It's also used in comparisons and bit rotations.

FLAGS
Several other flags record the status of the last operation. Whenever a register is changed and only then, these bits reflect details about the new value. The exceptions to this rule include BIT, CMP, CPX, and CPY, which are specifically designed to affect the flags without affecting any register, and the bit shift instructions, which can set these flags by altering memory instead of registers.

The first of these to consider is the N flag, which records whether or not the register change resulted in a negative number. To the computer, a negative number is one with the high bit set, so this flag actually notes if the high bit is set or not. A result of $00 minus $7F would clear N, and $80 minus $FF would set it.

Of the others, the Z flag shows if the last result was 0 or non-0. The V flag indicates if the second highest bit was set or cleared and is also used in signed addition and subtraction, indicating a carry from the second highest bit to the highest.

Another flag is the Break flag, set by the BRK instruction. Breaking transfers control to a monitor or other routine designed to handle this interruption. However, aside from calling up the monitor, it isn't used in assembly programming.

MODES

The remaining two bits in the ST, the D and I flags, called the Decimal mode and Interrupt mode flags, set the method the CPU uses to handle certain operations. Generally, they are only used in advanced programming. Changing to decimal mode makes the computer try to add and subtract numbers in binary coded decimal format. Since the computer isn't ready for this, strange things can happen. Likewise, changing the Interrupt flag can cause normal interruptions, such as checking the keyboard for a keypress, to be ignored. Unless it is carefully programmed, this change can result in a computer lockup.

The C, N, V, and Z flags are used by decision instructions to conditionally branch, like IF/THEN. For instance, BEQ will branch to a new address only if the last register change had a 0 result. Otherwise, it will continue as normal. The opposite can be done with the instruction BNE, or Branch if Not Equal. Similar instructions exist for the C, V, and N flags, checking to see if they are cleared or set.

ADDRESSING MODES

Registers require data to process, and ways that they obtain data are called addressing modes. The easiest to consider is called immediate. It gets its name from where the data is stored: immediately after the instruction.

If we assembled the instruction LDA#$00 and then looked at it in memory, we would see $A0 $00, with $A0 being the instruction for LDA (immediate mode) and $00 as the value. The # sign serves as an indicator to the assembler that you want an immediate mode instruction, and not some other mode of addressing data.

Immediate mode is used similarly to the use of constants in BASIC. Just as BASIC does, assembly language uses variables more often than constants. To use memory variables, another mode is available called absolute. An address in memory can be written by a four-digit hexadecimal number (5 on the 128), ranging from $0000 to $FFFF. Memory can then be written to or read from using this address; for instance,

STA $FFFF stores the value of the accumulator into memory location $FFFF.

PAGES

Normally, an absolute instruction is three bytes long--one for the instruction and two for the 16-bit address. However, like the stack, the CPU gives privileged status to another area of memory, 0 page. A page refers to a 256-byte location, with the last two hex digits ranging from $00 to $FF and the upper digits giving the page number. Page 2 runs from $0200-$02FF, page 1 (where the stack resides) is from $0100-$01FF, and $0000-$00FF is page 0. With page 0 instructions, the page is implied, saving a byte in the instruction and speeding up processing. For example, STX $00FF (absolute addressing) is machine code $8E $FF $00, but STX $FF (page 0) is $86 $FF.

INDEXING

Assembly language on the 6502 provides us with a method of accessing larger amounts of memory in the form of arrays of up to 256 bytes. Indexed addressing is a mode that uses the X or Y register as an index to the array. The address in the instruction is added to the value of the register, and this new address is the source or destination.

If we have a table of data starting at location $CD00, we could use LDA $CD00,X to copy it into A (the ,X indicates X indexing; ,Y is used for Y indexing). If X contained $00, the actual address for the load would be $CD00 plus $00 which equals $CD00. Changing X to 1 would load from $CD00 plus $01 which equals $CD01, making array accesses simply a matter of changing X's or Y's value. One limit to this mode is the register size; since X and Y are limited to eight bits, the longest array can be no more than 256 bytes.

DIRECT OFFSET

Indexing does reuse the same instructions, but it has its limits. What if we wanted to perform the above operation additionally on a table at $CA00? Then we would have to make a duplicate of all the code and assemble it. Fortunately, another addressing mode, called direct offset indexing, lets us avoid this unpleasant prospect.

In this mode, the table's start is not assembled into the instruction but is placed in page 0. The 16-bit address has to be split into two consecutive locations, with the lower two hex digits first, followed by the upper two digits. What is assembled is an instruction pointing to these page 0 locations. When executed, the address is plucked from 0 page and added to the Y index value to get the actual address. Only the Y register is used for this memory addressing. (The X register has a special indexing mode called indexed indirect, but it's not particularly useful and so rarely used.)

An example will help illustrate. We'll use the table at $CD00, but now we assemble LDA ($FE),Y. Note the syntax indicating this mode. The table address is now stored at $FE and $FF and looks like this in memory: $00 $CD. As the instruction executes, it looks at the value located at $FE/$FF and adds it to the offset in Y. The result is the

address from which A is loaded.

The advantage in this addressing mode now becomes apparent. To use the same program for our second data table at $CA00, we merely change the value of $FE/$FF from $CD00 to $CA00. No reassembling of the program is required. What seems at first to be more effort to access memory ends up being simpler, since only one routine needs to be coded, tested, and assembled, instead of many.

AN EXAMPLE

Let's put all this knowledge of assembly language to work by focusing on a small routine that sets all of color memory to the same value, doing it at machine language speeds. On your monitor, enter the following lines of code.

```
.A C000 LDA #$05
.A C002 LDX #$00
.A C004 STA $D800,X
.A C007 STA $D900,X
.A C00A STA $DA00,X
.A C00D STA $DB00,X
.A C010 INX
.A C011 BNE $C004
.A C013 RTS
```

After assembling, this code could be saved and then reloaded by a BASIC program. Use SYS 49152 to call it, and poke 49153 with the desired color code to be in A.

Reviewing the code, we see that the first two instructions are in immediate mode, setting A and X to $05 and $00. Then come the STA instructions, storing the $05 from A into memory. The STA is repeated four times because we need to fill four separate pages of color memory, for a total of 1024 bytes.

Following the stores is INX, which increments X to 1. All X indexing instructions will now point one higher in memory. It also has the side effect of setting the Z flag, depending on whether X is 0 or not. This flag is then tested by the instruction BNE, and since X is not 0, the branch to $C004 is performed. At this point, the four store instructions are executed again. Note however that X now is 1, so the instructions are effectively STA $D801, STA $D901, STA $DA01, and STA $DB01.

After that, X is incremented to 2. Since 2 is non-0, the BNE is performed again. And so the program goes, filling ever higher locations of memory with $05. The only way it would stop would be for X to become 0. But since X is only getting higher (via INX) how does it end?

The registers in the 6502 behave like a car odometer, in that when they reach their limit, they cycle over to 0. For an 8-bit register, this means that the next number after $FF is not $100 (a 9-bit number)

but $00. Looking at the code again, we see that X starts at 0 and is incremented by 1 up to $FF, filling each page of memory with $05. Following that, it is incremented once more to $00, whereupon the BNE fails, and the computer falls through to the next instruction, the RTS, which ends the routine.

A small program like this shows the real power of assembly language programming. Calling this routine changes screen colors instantly, yet it's only 12 bytes long. Nothing in BASIC could compare to this in size and speed.

It's this combination of blazing speed and compact size that makes assembly language ideal for working with the Commodore computers, where memory is at a premium and every speed increase is a blessing.

No matter what the future holds for the 64/128, knowledge of assembly language remains useful. Once you learn to think on the machine's level, each subsequent assembly language becomes easier. And for the beginner, the 6502 can't be beat. The Intel 8086, the grandfather of today's 80386 and 80486 chips, has more instructions, memory modes, and registers than the 6502. Yet a knowledge of the 6502 gives a formidable start on the road to any machine code, no matter how complicated.

Commodore computers are still thriving, due in no small part to programming that wrings every ounce of power out of them. Assembly language is the key to getting the most out of your computer and will continue to be a major contributor to the Commodore's continued popularity.

Gazette, July 1994

7@7@7@7@7@7d7@7777e7dee7e7e7e7e7e7e7e7e7e7e7e7a7e7e7e7e7e7e7e7e7e7e7

# 6502 MNEMONICS

By David Pankhurst

Here's a list of the three-letter commands or nmemonics used in assembly language.

Program Control
BCC Branch if Carry Cleared to 0
BCS Branch if Carry Set to 1
BEQ Branch if Equal to 0 (Z=1)
BNE Branch if Not Equal to 0 (Z=0)
BMI Branch if MInus (N=1)
BPL Branch if PLus (N=0)
BVC Branch if V flag Clear (V=0)
BVS Branch if V flag Set (V=1)
JMP JuMP to address
JSR Jump to SubRoutine
RTS ReTurn from Subroutine
RTI ReTurn from Interruption
BRK BReaK to sp=cial routine

Data Movement
LDA LoaD register A from memory
LDX LoaD register X from memory
LDY LoaD register Y from memory
STA STore A to memory
STX STore X to memory
STY STore Y to memory
TAX Transfer a copy of A to X
TAY Transfer A to Y
TXA Transfer X to A
TYA Transfer Y to A
TXS Transfer X to the Stack pointer register
TSX Transfer the Stack pointer to X
PHA PusH A onto the stack's top
PHP PusH the Processor status register ST onto the stack's top
PLA PuLl A from the stack's top
PLP PuLl the Processor status register ST from the stack's top

Processor Status Flag Instructions
CLC CLear Carry flag to 0
SEC SEt Carry flag to 1
CLD CLear the Decimal mode flag (disabling decimal mode)
SED SEt the Decimal mode flag (enabling decimal mode)
CLV CLear the V flag to 0
CLI CLear the Interrupt mode flag (allowing interruptions)
SEI SEt the Interrupt mode flag (ignoring interruptions)

Math Instructions
ADC ADd memory to A with Carry
SBC SuBtract memory from A with Carry

AND logically AND memory with A
ORA logically OR memory with A
EOR logically Exclusive OR memory with A
ROL ROtate memory or A one bit Left
ROR ROtate memory or A one bit Right
ASL Arithmetic Shift memory or A one bit Left
LSR Logically Shift memory or A one bit Right
DEX DEcrement X by 1
DEY DEcrement Y by 1
DEC DECrement memory by 1
INX INcrement X by 1
INY INcrement Y by 1
INC INCrement memory by 1
BIT test BITs of memory with A
CMP CoMPare memory to A
CPX ComPare memory to X
CPY ComPare memory to Y

Gazette, July 1994