THE COMPUTE!'S GAZETTE DISK

OCTOBER 1994

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

G A Z E T T E   O N   D I S K

O C T O B E R   1 9 9 4

Disk Magazine

C-64 Users, Type: Load "Menu",8,1 and press <Return>.

C-128 Users ----: Enter 128 Mode; then type RUN "128 Menu" and <Return>

****

GAZETTE DISK *** October 1994

Feature:

COMMODORE GOES TO SCHOOL
By Steve Vander Ark

When Steve Vander Ark isn't writing computer articles, he's teaching elementary school. Read how he puts 8-bit Commodores to work in his classroom.

Review:

Big Blue Reader versions 4.0 and 4.1
Reviewed by John Elliott

See what's new with the utility that lets you read and write to IBM PC files, using your 64 or 128 and a 1571 or 1581 drive.

Columns:

64/128 VIEW by Tom Netsel
The results of Gazette's readership survey are in.

FEEDBACK
Comments, questions, and answers.

D'IVERSIONS by Fred D'Ignazio
Information Dirt Roads, Bike Paths, and Hiking Trails

MACHINE LANGUAGE by Jim Butterfield
Strings and Structures

BEGINNER BASIC by Larry Cotton
Conversions

PROGRAMMER'S PAGE by David Pankhurst
Logical Ramblings

GEOS by Steve Vander Ark
Who Are You Calling Mature?

PD PICKS by Steve Vander Ark
Sortanos and Space One

128 Programs:

Hurricane Tracker by Hugh McMenamin
Plot the course of hurricanes as they pass through the Caribbean and

up the Atlantic coast.

Map of Australia by Spiro Tanti
Run this graphical geography program and learn something about Australia and its state and territorial capitals.

64 Programs:

Capturing and Restoring Screens by James T. Jones
A screen capture and display utility for the 64.

Bassem + by Farid Ahmad
Gazette's popular 6502 assembler program has some brand-new features. The new material is emphasized in the main article, but the original Bassem documentation is also included.

Spots by T. L. Flynn
Here's a two-player arcade-style game that'll have you seeing spots.

Sprite Editor 64 by Andrew Martin
Design and edit up to 64 sprites at one time with this powerful sprite editor.

One-on-One Basketball by David Garner
Play a little backyard basketball with this two-player game for the 64. Great graphics.

Sortanos (PD)
This one-player game is sort of like dominoes.

Space One (PD)
Swarms of missiles, spaceships, and bombs hurl themselves at you over an impressive scrolling background in this arcade-style game for one player.

Gazette, October 1994

CAPTURING AND RESTORING SCREENS

By James T. Jones

A screen capture and display utility for the 64.

In writing a program for displaying a text file that uses a method different from the usual methods, I encountered the need to "capture" a number of the screens and restore them at a later time. I should point out that not only must the contents of the 1000 addresses (1024-2023 or $0400-$07E7) of the video matrix (25 lines X 40 columns) be stored but also the corresponding values of the color RAM (55296-56295 or $D800-$DBE7).

The machine language (ML) file, SCRN CAP RESTR.O, will store the data of up to four screens and later process the data to reconstitute the screens with appropriate SYS calls. To use this routine in your own programs, simply include the following line in your BASIC program.

```
LN POKE 147,0: SYS 57812"SCRN CAP RESTOR.O",DV,1: SYS 62631
```

LN is the particular line number in your program and DV is the disk drive device number. This loads the file with the ,DV,1 extension into memory from 52736 ($CE00) to 53021 ($CF1D).

The SYS calls to save and restore the four text screens are as follows.

```
SYS 52797 -- Save screen 1
SYS 52825 -- Restore screen 1

SYS 52853 -- Save screen 2
SYS 52881 -- Restore screen 2

SYS 52909 -- Save screen 3
SYS 52937 -- Restore screen 3

SYS 52965 -- Save screen 4
SYS 52993 -- Restore screen 4
```

The data corresponding to the captured screens is stored under BASIC ROM (40960-49151 or $A000-$BFFF). The following table represents a memory map of the addresses used for the characters and corresponding colors of each screen.

```
$A000 - $A3FF   SCREEN 1 CHAR. MEMORY
$A400 - $A7FF   SCREEN 1 COLOR MEMORY

$A800 - $ABFF   SCREEN 2 CHAR. MEMORY
$AC00 - $AFFF   SCREEN 2 COLOR MEMORY

$B000 - $B3FF   SCREEN 3 CHAR. MEMORY
```

$B400 - $B7FF	SCREEN 3 COLOR MEMORY
$B800 - $BBFF	SCREEN 4 CHAR. MEMORY
$BC00 - $BFFF	SCREEN 4 COLOR MEMORY

By using the addresses under BASIC ROM, the entire memory range $C000-$CFFF is left free for other ML programs or data.

This program can be used for a number of purposes, including changing from the currently displayed text screen to a screen or screens with instructions and then returning to the original screen. It can be used for paging back to a preceding screen or to an initial screen, as in the program for displaying text files that I mentioned above. You could also use it to create special effects by rapidly changing screens with different fonts and so on.

Remember that, ordinarily for the address range of the BASIC ROM (40960-49151), the command PEEK(address) yields the contents of "address" for the BASIC ROM. If bit 0 of address 1 is changed to 0 and the remaining bits left unchanged, then underlying RAM corresponding to BASIC ROM can be read. POKE Address,Values always falls through to the underlying RAM in this range of addresses. As indicated in the corresponding source file, SCRN CAP RESTR.S, found on this disk in Merlin format, it is more efficient to use indirect indexed addressing for such a large range.

The program is completely relocatable to any other starting address where 286 bytes of free, undisturbed memory are available. Unused addresses 823-827 ($0337-$033B), preceding the cassette buffer, and 251-254 ($FB-$FE) are employed for various purposes. The SYS calls listed above will have to be changed, of course; however, an examination of the source file and a listing of the corresponding, assembled object file will readily indicate the new values.

When you run the program from the Gazette menu, you will see a demonstration of the program that utilizes several different files and fonts on the flip side. Remember that the file SCRN CAP RESTR.O is the one that you'll have to put on a disk for use with your own programs.

Gazette, October 1994

BASSEM +

By Farid Ahmad

A familiar assembler for 6502 code that's packed with additional utilities

I have used Gazette's Bassem to write many programs, and I have always admired its speed and convenience. There are some other utilities that I am equally fond of, but unfortunately many machine language programs don't work very well in combination with Bassem. I have gathered all of my favorite utilities into an integrated unit and then added some new features to produce a very comfortable working environment. The result is Bassem+.

Bassem+ combines the following programs that have all been published in Gazette: Bassem, DOS Plus, Quick!, Turbosave, and Tag It. In addition, it offers 16 function keys yet takes away very little extra memory from the program area.

To use Bassem+, simply load and run the file B+. It's a BASIC loader that will load and run BESSEM+. It's a good idea to always load and run Bassem+ with this loader file. More about that later.

Almost all features of the above mentioned programs are available in Bassem+. Below is a brief recap of these original features, plus a rundown of the enhancements.

BASSEM
Bassem is a powerful assembler for producing 6502 machine language programs. It supports all the usual assembler features and also allows BASIC language to be used within the source code. (For readers not already familiar with Bassem, see the accompanying documentation on this disk.)

All the features of this program are available in Bassem+. The one change is that you can activate Bassem+ by typing SYS 34782 (as before) or SYS 32768 (which I find easier to remember and type).

Bassem uses an area of memory to store label names. This can be set to any part of the memory, including RAM under the BASIC and Kernel ROMs. The memory segment $A000 - $BFFF is used by default. Bassem+ changes this to $B000 - $BFFF. This can be modified in the usual way to any other unused area. (See memory map below.)

DOS PLUS
DOS Plus is a DOS wedge with many extra features. All the usual DOS wedge commands are available, except that each command must be preceded by the @ symbol. Following is a summary of the commands.

@	Read error channel
@text	Send command to drive

@$text	Display directory
@/name	Load BASIC program
@↑	Load and run BASIC program
@(back arrow)	Save BASIC program
@?name	Verify BASIC program
@%name	Load ML program
@!address1 address2 name	Save ML program. This command has been modified so that it can save the RAM under BASIC ROM. For example, to save a copy of Bassem+ use @!$8000 $AA00 BASSEM+.
@:name	Verify ML program. Compares a disk file with memory, starting at the file's load address.
@*address name	Load at address
@Sname	Scratch file
@£name	Display load address
@#number	Set device number
@Q	Deactivate DOS Plus
@=value	Perform value conversion. This function will convert a hexadecimal value to decimal and a decimal or binary value to hexadecimal.
@X	Turns off the Bassem+ function keys. To reactivate function keys, reactivate Bassem+ with SYS 32768 or SYS 34782.

In the above summary, the text parameter is any string of characters. The name parameter represents the name for a file on disk. Any text found within quotation marks is considered a filename. If no quotation marks are found, all text following the command is considered to be a filename. Note that quotation marks take precedence over other text. Thus @/GAME" would result in a missing filename error, since DOS Plus tries to find a name following the quotation mark but doesn't find one. Finally, trailing quotation marks are not necessary except to exclude unwanted text.

The address, number, and value parameters represent numeric values. DOS Plus accepts values in decimal, hex, and binary. For example, 49152, $C000, and %1100000000000000 are equally valid. Values must not have leading spaces. If, however, another parameter is to follow a value, one space must separate the two.

The interrupt-driven features, @F, of DOS Plus are not available.

TAG IT

Tag It is a utility that makes it easy to find your place in a program listing. All features of Tag It are available except the function keys, which are redundant in Bassem+.

Briefly, you set up to 26 identifying tags in your program. Each tag has the form 10 REM-A, where A can be replaced by any letter of the alphabet and 10 by any line number. Whenever you want to list your program from line 10, hold down the Commodore key and press A or any other letter that you used.

Renumbering your program will move the tag to a different line number, but it will remain in the same place in your program. This ensures that you get the same listing when you press Commodore-A.

Tag lets you cancel quote or insert mode by pressing the Ctrl and Commodore keys simultaneously. You can hold down the Commodore key and press <, >, /, or Crsr down to move the cursor to the start, end, or middle of a line or to the bottom of the screen, respectively.

Tag It was disabled when the Run/Stop-Restore combination was pressed. This does not happen in Bassem+.

TURBOSAVE AND QUICK!

These are a couple of programs which speed up the load, save, and verify operations of the 1541 drive. Turbosave was originally written to be stored under BASIC ROM, but it used a few bytes in low memory below the cassette buffer. These memory locations are no longer used. Quick! has also been moved under BASIC ROM. Together, these two programs are labeled Quickdrive. You can activate or deactivate Quickdrive by using the command SYS 33568 or by pressing Commodore-f4 if the default function keys are active.

One problem will occur when Quickdrive is active. DOS Plus reports the ending address of a program after a load operation. Quickdrive interferes with this operation with the result that the wrong address is displayed. If you want to find the ending address of a program, turn off Quickdrive before loading it.

When performing a verify operation, Quickdrive will report the memory byte at which a verify error occurs. If Quickdrive is turned off, DOS Plus will report the ending address of the disk file if a verify error occurs.

FUNCTION KEYS

Bassem+ features 16 function keys. The first 8 keys are accessed in the usual way. The next 4 are accessed by pressing them while holding down the Ctrl key, and the last 4 by pressing them with the Commodore key.

The keys are predefined, and there is no direct command to change the definitions. However, the definitions can be changed by using the Keydef program, which is also on the flip side of this disk. First activate Bassem+ and then load Keydef and list it.

You will see the function key definitions in the lines 130 to 430. Change these as required and then run the program. To add a carriage return to any definition, use +RT$. (See listing for examples.)

The total length of the text, for all 16 keys, must not exceed 274 characters. All 16 keys can be modified. It is suggested, however, that the following definitions not be changed.

f8: The default definition is "SYS start,." If you put a label called "(back arrow)start" at the starting address of your source code, you can press f8, add any parameters that may be required, and press Return to run the machine language program.

Ctrl f1: By default, pressing this key calls a short ML routine which will reset the environment and then perform an OLD command on any source code in memory. This same routine is called when you first activate Bassem+. If you have a hardware reset button installed, you can recover from a computer lockup by pressing the reset button and then entering SYS 32768. If your machine language routine writes over a portion of the source code, however, you will not be able to recover your program properly. That means it's still a good idea to save your work frequently.

A word of caution here. Since Bassem+ performs an OLD on being activated, it will appear to lock up if there is no valid program in memory. If this happens, press Run/Stop-Restore to gain control. This will never happen if you load and activate Bassem+ with the BASIC loader B+, since there will always be a valid program in memory.

Memory Map:

AA00 - AB00	Buffer used by Quick!
A100 - A9FF	Quickdrive
A000 - A0FF	Buffer used by Turbosave
87DE - 9FFF	Bassem
83DC - 87DD	DOS Plus
8326 - 83DB	Control routines for Quickdrive
81E8 - 8324	Tag It
80C4 - 81E7	Function keys definitions
8040 - 80C3	Function keys routine
8000 - 803F	Startup and other short routines

Gazette, October 1994

SPOTS

By T. L. Flynn

A two-player arcade-style game for the 64. Joysticks required.

I once watched some friends play a game in which the players took turns trying to control certain screen positions; the results of a player's move depended on the other screen positions immediately surrounding the position moved to. I thought the game looked simple and figured it would hardly be any challenge at all to play.

As I watched, I realized that I was both right and wrong. Yes, it was simple, but the challenge was created as the game progressed. I found myself glued to the screen, my mind trying desperately to work out the strategies of each possible move.

I had to leave before I got the chance to try the game myself, but I swore that someday I'd put together a game like it. Spots is my best attempt at re-creating that game. It is a two-player game using both joystick ports; player 1 (blue) uses port 1and player 2 (red) uses port 2.

When run, the program will ask you if you want instructions; pressing Y for yes will get you a quick rundown on how to use the joysticks to play. Once you are into the game, it is easy to see how strategies are developed and used.

GAMEPLAY
There are six play screens available; the first one is a solid pattern. Pressing P will let you page through the remaining five patterns, eventually returning you to the solid pattern. When you've decided which pattern to play on, press the port 1 fire button, and the game will begin.

Player 1 always moves first and always starts in the upper left corner of the screen; player 2 starts in the lower right. The idea is to pick up your "spot" and move closer to your opponent. You can move up to two positions at a time. Moving one position leaves the old position in your color while moving two positions turns the original position black, making it available to your opponent.

When you get close enough to move next to any of your opponent's positions, they change color and become your positions! When all the positions are filled, the player who has the most spots is the winner.

STRATEGIES
One of the many strategies of play is to build a wall of spots around a large number of unclaimed positions. This will prevent your opponent from claiming them. This is a good strategy even if it is a little difficult to achieve. It usually ends with your opponent being unable

to move.

When either you or your opponent run out of moves, you must press X. This will allow the other player to continue claiming available positions.

The program automatically counts the number of spots for each player and displays them on the screen after every turn. When all positions have been claimed and a winner has been declared, the program will offer you a new game and/or a new pattern.

Spots is written entirely in machine language but loads and runs just like a BASIC program. I hope you enjoy playing it as much as I have!

Gazette, October 1994

SPRITE EDITOR 64

By Andrew Martin

A sprite design an editor program for the 64.

Designing sprites can be a mundane chore, requiring graph paper and a calculator. Now, you can throw away your graph paper, save your calculator batteries, and let your 64 do the work for you.

Sprite Editor 64 lets you design sprites with your joystick plugged into port 2, and you can work with up to 64 sprites at a time. The sprites are divided into eight sets with eight sprites in each set.

Sprite Editor 64 consists of two programs: a boot program which moves the start of BASIC and the main program. You may want to copy both of these to a new disk. A blank disk can hold up to 620 sprites saved individually or up to 2350 saved sequentially, 64 at a time.

The program's editor consists of five menus, with eight function per menu, all accessed via the function keys. These menus along with other special keys provide you with more than 40 controls for editing and manipulating sprite data. Here is a list of each menu, its function, and directions for use.

MAIN MENU

MOVE
This function lets you move a sprite within a specific area.

1. Select Move function.
2. Select each sprite as desired.
3. Use joystick to move sprites.
4. Press Return to exit.

OVERLAY
This function lets you overlay sprites to create a high resolution object.
1. Select Overlay function.
2. Select each desired sprite.
3. Press Return to exit.
4. Select Overlay function again to restore sprites to their original screen positions.

PLACE
This function lets you place sprites within four quadrants.
1. Select Place function.
2. Select each sprite as desired. To skip a quadrant, press any key other than 1-8.
3. Select Place function again to restore sprites to their original screen positions.

ANIMATE
This function lets you animate a sprite sequence.
1. Select beginning sprite for a sequence.
2. Select Animation function.
3. Select ending sprite.
4. Select Animation function again.
5. Press joystick to the right to animate forwards.
6. Press joystick to the left to animate backwards.
7. Press joystick up to increase speed.
8. Press joystick down to slow speed.
9. Hold fire button down to continue sequence.
10. Hit Return to exit.

DATA
This function allows you to enter sprite data.
1. Select sprite.
2. Select Data function.
3. Enter three digits for every byte of date. For example, 001, 020, 255, and so on.
4. Use joystick to move cursor. Cursor will move automatically to the next byte of data.
5. Hit Return to exit.

DIRECTORY
This function loads the disk directory and lists sequential files.

SAVE *
This function lets you save form one up to 64 sprites.
1. Select beginning sprite for sequence.
2. Select Save function.
3. Type in filename and press Return.
4. Select ending sprite in sequence.
5. Select Save function again.

LOAD
This function lets you load previously saved sprites.
1. Select the beginning sprite to which the Load sequence is to start.
2. Select Load function.
3. Type in filename and press Return. (Note: several sprites have been included on this disk for examples. Do not type the .SPR extension when entering the filenames.)

A EDIT MENU

WORK
This function lets you edit a sprite using the joystick. It is the only way to access the special Work menu.
1. Select sprite.
2. Select Work function.

INVERT
This function lets you turn a sprite upside down.
1. Select sprite.
2. Select Flip function.

COPY *
This function lets you copy a sprite to another location.
1. Select sprite to be copied.
2. Select Copy function.
3. Select sprite to be copied to.
4. Select Copy function again.

90 NORM *
This function rotates a sprite 90 degrees in a clockwise direction without adjusting the 24 x 21 pixel differentiation. The last three columns of the spite will be truncated.
1. Select sprite.
2. Select 90 Norm function.

90 ADJT *
This function rotates but adjusts for the pixel differentiation.
1. Select sprite.
2. Select 90 ADJT function.

X SYM *
This function mirrors the top half of the sprite to the bottom, creating symmetry on the x axis.
1. Select sprite.
2. Select X SYM function.

Y SYM *
This function mirrors the left half of the sprite to the right, creating symmetry on the y axis.
1. Select sprite.
2. Select Y SYM function.

B EDIT MENU

REVERSE
This function turns off bits that were on and vice versa.
1. Select sprite.
2. Select reverse function.

SCROLL
This function scrolls a sprite in any of four directions.
1. Select sprite.
2. Use joystick to select direction.
3. Hit Return when finished.

EXP X
This function expands a sprite horizontally if the sprite is unexpanded. It shrinks the sprite if it has already been expanded.

EXP Y
This function expands a sprite vertically if the sprite is unexpanded. It shrinks the sprite if it has already been expanded.

CUT *
This function is used in conjunction with Reverse. The excess of a reversed image is eliminated.
1. Select sprite.
2. Select Cut function.

FILL *
This function turns on every bit within a sprite.
1. Select sprite.
2. Select Fill function.

ALT MULTI
This function turns all sprites to multicolor mode.

TSF SET
This function transfers an entire sprite set to another location.
1. Select sprite set.
2. Select TSF Set Function.
3. Select set to receive transfer.
4. Select TSF Set again.

COLOR MENU

SPRITE
This function lets you change the color of a sprite. Use color guide at bottom of screen.
1. Select sprite.
2. Select Color function.
3. Select color (A-P).
4. Select another sprite or press return.

AUX 1
This function lets you change auxiliary color 1.
1. Select AUX 1 function.
2. Select color (A-P).
3. Press Return.

AUX 2
This function lets you change auxiliary color 2.
1. Select AUX 2 function.
2. Select color (A-P).
3. Press Return.

TEXT
This function lets you change text color.
1. Select Text function.
2. Select color (A-P).

3. Press Return.

SP/A1, SP/A2, A1/A2
These functions swap the two-bit values that make up multicolor sprites.

WORK MENU

GRID
This function prints the work grid on the screen.

DISPLAY
This function prints a sprite on the grid.

TOGGLE
This function toggles sprite colors using the fire button.

SELECT
This function lets you select the sprite drawing color. Hit Select until you reach the desired color.

GRID+DISPLAY
This function does the same as selecting Grid and Display in succession.

NRM/MLT
This function toggles between normal and multicolor edit.

CIRCLE *
This function puts a circle inside a sprite with a selected center and radius. *
1. Move cursor to center.
2. Select Circle function.
3. Select radius.

EXIT
This function takes you out of the work menu.

* The functions marked with an asterisk (*) require a second keystroke to execute. They can be exited by pressing the back arrow key on the top left of the keyboard.

Special Keys

1-8	Select a sprite
Clr	Clears a sprite
M	Toggles multicolor/ normal mode
Crsr Up/Down	Selects sprite set
Crsr Left/Right	Selects menu
Shift-Q	Exits program

USING SPRITES
After you've created a sprite file, you'll probably want to use it in your own program. After you've saved your sprite file to a program disk you may find it convenient to use one of the sprite loader programs included on this disk. SPRITE LOADER.BAS is a BASIC loader and SPRITE LOADER.ML is a machine language loader that is relocatable. Use your own code to display and move the sprites as you desire.

Gazette, October 1994

ONE-ON-ONE BACKYARD BASKETBALL

By David Garner

A two-player basketball simulation.

Do you remember when you were a little kid playing basketball out in the backyard? Your best friend would come over and the two of you would shoot a few hoops.

Now you can have that fun on your 64 with this two-player basketball simulation. It's a simple little graphics program that hs some swishing sounds to make the game seem more realistic.

Control the black player with thejoystick in port 1; control the white player with the joystick in port 2.

OFFENSE
Dribble the ball by moving the joystick in the desired direction. To shoot, press the firebutton and release it at the top of your jump.

DEFENSE
Stand just to the right of your opponent and press the fire button to steal the ball. To block a shot, stand just in front of the player as he attempts to shoot. After your opponent presses the button to shoot, press your button to jump and block the shot.

If a player misses a basket, the ball automatically goes to the other player. The first player to score nine baskets wins.

Gazette, October 1994

SORTANOS
You and the computer each have a set of tiles. You can play a tile whenever you can match a number with the tile in play. If you can't make a match, then you have to take a new tile from the boneyard. The game ends when one player has no more tiles.

If you think Sortanos is sort of like dominoes, you're right. Complete instructions are built into the game as an option.

SPACE ONE
When you load this sprogram from the Gazette menu, it loads via an instruction program called Space One Inst. This program gives you all the information you need to play this exciting one-player space game.

After you've read the instructions, the main program loads automatically. When you no longer need to see the instructions, you can bypass them and load the game directly by typing LOAD"SPACE ONE",8.

These public domain programs are discussed in detail in Steve Vander Ark's "PD Picks" column, found elsewhere on this disk.

Gazette, October 1994

BASSEM

A machine language assembler
for the 64

By Fernando Buelna Sanchez

(Editor's note: This article was originally published in the April and May 1990 issues of Gazette. It is reprinted here to help new readers who may not be familiar with Bassem.)

Push your computer to the limit with this outstanding, full-featured assembler for the 64. Now you'll be able to write machine language programs more quickly and easily than ever before.

Bassem is a two-pass assembler that contains many features and commands normally found only on commercial assemblers like the Commodore Macro Assembler, Buddy 64, or Merlin64. This versatile assembler can assemble to memory, to disk, or to both. And, if you want to check the syntax of your source program, Bassem can assemble without creating any code at all. In addition, Bassem has commands that make programming quite convenient. For example, FLP gives you easy access to floating-point values, and OPZ gives you precise control over zero-page addressing.

Although it has a wealth of commands, Bassem's most powerful feature is its ability to work as an extension of the 64's operating system. To the more than 150 commands supported by BASIC 2.0, Bassem adds assembly control commands, disk commands, editing commands, and 6502 machine language instructions. And, because it runs within the BASIC environment, you can use the built-in screen editor to enter and edit your programs, and you can use BASIC's commands to control how your programs assemble. For example, you can use IF-THEN statements for conditional assembly or FOR-NEXT loops to generate tables.

To use Bassem+, load and run the BASIC boot program B+. It automatically loads Bessem+, performs a NEW command, and then enters the proper SYS call to activate it. Bassem+ installs itself and then displays a startup message.

USING THE ASSEMBLER

The first step in creating a machine language program is entering the source code. With Bassem, this is done using the familiar BASIC screen editor. You simply enter each line of code with a line number as you would a BASIC program. For example, the source code for a simple program to change the screen border color to cyan might look like this:

```
10 WRT 1: SET $A000, $B000: BAS $C000

20 PASS 1:' BEGIN ASSEMBLY
```

```
30 BORDER = $D020

40 COLOR = 3:' THE VALUE FOR CYAN

50 START LDA #COLOR: STA BORDER: RTS

60 PASS 2:' END ASSEMBLY
```

In line 10, the WRT command tells Bassem to write the machine language to memory, the SET command establishes the label buffer, and the BAS command sets the starting address for the program. (If you don't understand what's going on, don't worry; we'll discuss each of these in more detail later.) The PASS 1 command in line 20 tells the assembler that the following lines should be assembled. Bassem continues assembling commands until it encounters a PASS 2 command (line 60). As you may have guessed, the text immediately following the PASS 1 command is a comment; Bassem treats the ' character as a REM statement.

Lines 30 and 40 assign values to the labels BORDER and COLOR. Bassem labels are always preceded by a (left-arrow character) and can be up to 40 characters long. They can contain letters of the alphabet, numerical digits, and the decimal point. They may also contain BASIC keywords and reserved variables. Some examples of valid labels are THIS.IS.A.LABEL, PRINTL OUT, and 3RD.JMP.

There are two ways to assign a value to a label. As you can see in lines 30 and 40, you can assign a value to a label using the assignment (=) command. When defining labels this way, you can use hexadecimal (base 16), octal (base 8), binary (base 2), or decimal (base 10) constants or expressions. Hexadecimal values must be preceded by a $ character; octal values, by an &; and binary values, by a % character. Decimal values are the default and require no prefix.

When you use an expression to define a label, you must abide by a few rules. First, with one exception, the expression must be a valid BASIC expression. The exception is that you can use hexadecimal, octal, and binary constants in the expression. Second, Bassem must be able to evaluate the expression during assembly. For example, the expression can't be based on the contents of the accumulator, because Bassem has no way of knowing what will be in the accumulator when the program is run.

The other way to give a label a value is to place it in front of a 6502 mnemonic. Line 50 contains an example of this method. Labels used this way take on the value of the program counter. This value corresponds to the address of the instruction. For example, in line 50, the LDA instruction is at location $C000 (49152), so the label START has a value of 49152. Placing a label on a line by itself immediately before a line containing an opcode produces the same result.

Line 50 demonstrates one more feature of Bassem--you're not limited to one instruction per line. You can fill an entire logical line (two screen lines) with instructions and labels. Simply separate the instructions with colons, just as you would in BASIC.

After you've entered the source code for your program, be sure to save it before you continue. Since Bassem operates in the BASIC environment, you can save your source files just as you would a BASIC program. The next step is to assemble your program. Assembling with Bassem is very easy; you simply load your source code and type RUN. Bassem will assemble your program and save it to memory or to disk, depending upon the destination you've indicated.

MNEMONICS AND PSEUDO-OPS

Bassem supports all 6502 addressing modes and instructions as shown in the Commodore 64 Programmer's Reference Guide. In addition, it supports several pseudo-ops which instruct the assembler on how to generate code. The following paragraphs summarize the pseudo-ops that you'll need to know to start using Bassem, including the ones demonstrated in the example above. In each description, optional parameters are indicated by square brackets, and repeating parameters are represented by ellipses. When only one of several choices is allowed, the parameters are surrounded by parentheses and separated by vertical bars.

BAS address
Sets the program counter (PC) to the specified address. The BAS command is usually used to define the starting address for the program. The address parameter must be a value between 0 and 65535. If you don't set the starting address, Bassem assumes a default value of $C000 (49152).

BUF number of bytes[,byte]
Reserves space for the specified number of bytes. The first parameter of the BUF command is required and tells Bassem how many bytes of memory to reserve within the object code. Legal values range from 1 to 65535. Optionally, BUF may be followed by byte values which determine how the reserved space is filled.

If BUF is followed by only one value, the number of memory locations specified by the value is filled with 0s. Otherwise, it's filled with the pattern established by the given values. For example, if you enter the command BUF 8, 2, 2, 3, Bassem will write 2, 2, 3, 2, 2, 3, 2, 2 to the object file. Legal values for the fill-byte parameters are between 0 and 255.

BYT (numberstring)[,(number string)]
Places the specified byte(s) or string(s) into the object file. If you specify a number or numerical expression, Bassem places that value into the object file. Legal values range from 0 to 255. If you specify a string, Bassem places each character of the string into a byte. You can specify multiple byte values or strings with one BYT command by

separating each with a comma (see the example programs).

PASS (1 or 2)
Tells the assembler where to begin and end assembly. The PASS 1 command lets the assembler know where to begin assembling code. You must place the PASS 1 command just before the first label definition or machine language instruction to be assembled. The PASS 2 command indicates the end of the program and must be placed just after the last label definition or machine language instruction.

SET starting label address,ending label address
Specifies the location of the label buffer. The SET command is used to define the buffer where Bassem stores labels as it assembles. The first argument sets the beginning of the L buffer, and the second argument sets the end. If you don't specify a location for the label buffer, Bassem places it under BASIC ROM ($A000-$BFFF). When defining the label buffer, be sure to use an area of memory that won't conflict with BASIC, Bassem, or your object code (if you are writing it to memory). In general, it's best to use areas above $A000.

WRT (0 or 1)
Specifies whether or not the object code should be written to memory. If the parameter following WRT is 0, the assembler won't write the code to memory. If the parameter is 1, Bassem writes the code to memory. The WRT command is useful when you don't want to place the code in memory but you want to check the syntax of your program or to assemble it to disk.

See Bassem (Part 2) for additional information and commands.

Gazette, October 1994

BASSEM (Part 2)

Because Bassem is an extension to BASIC, it's source files are very similar to BASIC program files. Each line of Bassem code must be preceded by a line number in the range 0-63999, can contain up to 80 characters, and have multiple commands separated by colons. Bassem's commands and 6502 mnemonics are tokenized, and, like standard BASIC commands, they can be presented using abbreviations.

Here are the Bassem commands, abbreviations, and tokens:

Command	Abbr.	Token Value
AFFIX	A Shift-F	$D8 (216)
AUTO	A Shift-U	$D2 (210)
DEL	none	$D3 (211)
DIR	none	$D6 (214)
DISK	DI Shift-S	$CC (204)
DLIST	D Shift-L	$DB (219)
FIND	F Shift-I	$CD (205)
HELP	H Shift-E	$D9 (217)
LABEL	L Shift-A	$CE (206)
LFT	L Shift-F	$DA (218)
MERGE	M Shift-E	$DC (220)
OLD	O Shift-L	$D4 (212)
PUT	P Shift-U	$D7 (215)
RENUM	RE Shift-N	$D1 (209)
WRITE	W Shift-R	$d5 (213)

In part 1 we discussed how to get Bassem up and running and introduced a few of its commands. Now, we'll examine the rest of Bassem's commands and explain how to use them.

MORE COMMANDS

WOR number[,number]...
Places the specified number(s) into the object file in low-byte, high-byte format. Legal values for number range from 0 to 65535. You can specify multiple values with one WOR command by separating them with commas.

FLP number[,number]...
FLP writes the number(s) into the object file in five-byte floating point format (five bytes). Legal values for number range from -1E38 to 1E38.

As with BYT and WOR, FLP follows the same syntax rules. You can specify multiple values by separating them with commas.

OPZ [0 or 1][,number]
OPZ directs the assembler on how to assemble the zero-page addressing modes for those instructions which support it. Setting the first parameter to 1 tells Bassem to use zero-page addressing whenever

possible. (This is the way most assemblers handle zero-page addressing.) Setting it to 0 tells Bassem to use absolute addressing mode.

If for example you enter the command

```
250 OPZ 1:LDA $C6
```

in your source file, the assembler generates the values A5 C6. If you change the OPZ 1 command to OPZ 0, it generates AD C6 00.

As you can see, the first example is in zero-page mode, and the second in absolute addressing mode; though the two examples do the same thing: load the accumulator with the value in address $C6.

You must be careful because OPZ turns on or off the zero-page mode in all the following instructions that require parameters. There are instructions like STX that have Y-indexed, zero-page addressing mode, but not Y-indexed, absolute addressing mode. If you try to assemble an instruction like STX $61,Y with zero-page mode turned off, it will give you a SYNTAX ERROR. The assembler takes the value $61 and since zero-page mode is not available, the computer thinks that it is absolute mode and then expects to find a colon (:) or an end-of-line marker; not a comma.

OPZ is also useful if the timing in your program is critical. In the above examples of LDA, the first requires three clock cycles, while the second uses four.

PROGRAMMING AIDS
In addition to commands which affect how your programs assemble, Bassem also has commands that aid program development. Although most of these commands are intended to be used in immediate mode, some can be included in your source code.

LABEL [((number1 label1 string1)[,[(number2 label2 string2)]]CLR)]
This comamands displays the labels that have been already defined. LABEL typed alone prints the message "LABEL FILE:" in reverse video, and lists all the labels defined in the label memory area, followed by their values in hex. Press the Ctrl key to slow the listing, the Shift key to freeze it, or the Run/Stop key to halt it.

When you type a string in front of LABEL, the computer searches for labels whose names are equal to the one specified in the string. As an added facility in searching label names, you can use the question mark (?), and the asterisk (*) as wildcards inside the string, just like when you use them in DOS commands. Example:

```
LABEL "?E*"

HERE = $C000      RESET = $FCE2
```

2 LABELS DEFINED.

LABEL typed with a number or a pair of numbers separated by a comma ---or even a single comma--- works like the LIST command, but instead of listing lines, it lists the labels whose values are in the range between number1 and number2.

Examples:

You type:	You get:
LABEL 34	Labels $22 (hex).
LABEL $801,END	Labels 2049 and the value of the label END.
LABEL $C000,	Labels from $C000 to $FFFF
LABEL ,255	Labels with values from 0 to 255.

LABEL CLR, clears or erases the memory area defined by the SET command, but doesn't affect the settings. In fact, LABEL CLR only puts a zero in the first byte of the label memory area (LMA), and resets to the beginning of LMA the vector that points to the following available byte for storing labels. If you just used LABEL CLR, and want to view the old list, simply poke to the first byte of the LMA the ASCII value of any legal label character if the first label was two or more characters long. Poke the ASCII value plus 128 if the first label contained only one character. But be careful, this method of rescuing an old label list doesn't permit you to define new labels without really affecting the old labels.

FIND char1 string char2
This command lets you search through the text of your program (source code) for a given string. The computer lists all the lines where string appears.

The string parameter is the string you want to search for and the char parameters are delimiting characters. You can use any character that's not found in you search string as as your delimiter. If you use the quote marks (") as the delimiter, the search string will will be used as is; othrwise the string will be tokenized before the search begins.

If the computer lists a lot of lines, you can stop it by pressing the Run/Stop key or slow it with Ctrl.

Char1 and char2 can be any character but they both must be the same. Here are some examples of valid search commands.

FIND /PRINT/	Searchs for the PRINT command
FIND "PRINT"	Searches for the word PRINT in a string

FIND ZXYZ	Searches for the variable XY
FIND @"@	Searches for a quotation mark.

AUTO [increment]
This eases the job of typing programs by providing the line numbers automatically. Typed alone, AUTO disables the automatic line numbering, but if you specify an increment (0-65535, the numbering is enabled. When you add a line to a program and the numbering is enabled, the computer prints for you the next line number and places the cursor one space to the right. For example, if you enter the command AUTO 10 and then type

```
100'World's Greatest Program
```

Bassem will print 110 as the next line number.

If the new line number is equal to one of an existing line, the computer prints an apostrophe (') prior to the line number. If this happens, you can press Return, and the old line is not affected; the numbering is stopped but not disabled.

Another way to stop the numbering without turning it off, is by pressing Return on an empty new line, or by moving the cursor off the line by pressing Shift-Return or the cursor keys. Also, the numbering is stopped if the new line number is higher than 63999.

RENUM [starting line number[,increment]]
This command changes the line numbers of your programs. If you type it without parameters, it renumbers the program in memory using its default values of 10 for the starting line number and 10 for the increment. If you only want to specify the increment, you must set the starting line number too.

For example, if you type RENUM 100,5, it will renumber the program so that the first line is line 100 and each line after that is incremented by 5. This instruction renumbers the whole program, and only the line numbers. Numbers in front of GOTO, GOSUB and so on, are not recalculated; you must change them manually.

DEL (line number [-[line number]])
This command deletes a block of lines from source code in memory. It works like the LIST command. Here are some examples:

DEL 100	Deletes line 100
DEL 25-230	Deletes lines 25 through 230, inclusive.
DEL -85	Deletes all lines from the beginning of the program through line 85
DEL 40-	Deletes all lines from 40 to the end of the program.

Be careful. If you type DEL by itself, you'll erase all code in memory and you can't recover it.

OLD
Rescues a program erased by BASIC's NEW command. It cannot rescue a program erased with DEL.

HELP
This instruction lists the line where the last error ocurred. If a portion of the of the listed lin is in reverse video, the error occured just before the reversed section. If you modify the program, HELP still lists the line, but the spot where the error was may not be indicated properly. If in direct mode you get an error, HELP doesn't list any line. When running a program and an error occurs, there's no need to type HELP; the computer lists the line. HELP is used to list the line again.

LFT [SET/CLR]
Sets or clears the formatted listing flag. LFT SET causes the LIST command to format the source lines as it displays them. LFT CLR returns LIST to normal mode of operations. When you use LFT SET the listings are formatted in a single line in the form listed below.

If the instruction is the first one on a source line, then the line number is printed in the first column.

If the instruction is preceded by a label or a label is defined by the assignment operator, the label is printed starting in the sixth column.

The instruction itself or the assignment operator for a label definition is printed beginning in the 15th column.

New BASSEM instructions and mnemonics are printed in the 20th column.

Comments are listed right after the line number if they are at the start of the line; or if they are after a colon, they will be listed on the 40th column, the start of the next line on the screen.

This format is used unless the column is not available. For example, if in a line, there is a label containing 30 characters and a instruction after it, the instruction will be listed like in a normal listing.

This kind of format is useful when you use HELP because the error is identified more easily. When the new list format is set, you can still enter new lines to the program as you normally do, but never edit a line in the new format unless it occupies only one screen line. Be sure the listing is not formatted before editing.

As an added bonus, you can use the apostrophe (') to indicate a comment. You can still use REM, but when using the new list format, a comment defined by an apostrophe is more notable than the ones in REM.

The keyword INPUT can be abbreviated by typing the exclamation point (!). This is like using the question mark to mean PRINT in BASIC.

DISK COMMANDS

BASSEM also adds seven commands for use with your disk drive. If you do not specify a device number, the command will operate with the default device number 8. Commands that require strings or filenames as parameters accept up to 41 characters.

DISK string[,dev.] default device number

DISK sends a string to the disk drive command channel and displays the drive status; or, sets the default device. Typed alone, it prints the disk drive status on the screen. If you specify a string, it is sent to the drive, and the computer waits until the drive responds with its status.

You can change the default device by typing a number in the range 8-11.

DIR [string[,dev.]]

DIR lists to the screen the current disk directory without disturbing the program in memory. After the directory has been listed, the number of files in the disk is printed in reverse video. You can stop the listing by pressing the Run/Stop key.

You can specify which files DIR displays by including string parameters. The string can contain a valid filename or Commodore wildcards.

WRITE filename [,dev.]

With this instruction you can write your object code generated by the assembler to disk. You have to add the extension ",P,W" to the filename to prevent a disk error. You can specify the drive to write to by including the drive number. For example, WRITE "CL.ML,P,W",9 saves the object file CL.ML on th disk in drive 9.

For Bassem to write the object code properly, you must place the WRITE command before the BAS and PASS1 commands. The following program fragment shows the order that's required.

```
10 WRITE "CODE,P,W"

20 BAS $C000
```

```
30 PASS 1

your code

1000 PASS 2
```

Before using the write instruction, you may want to try to assembly the code without writing it to memory or disk (using WRT 0). When writing to disk, the PASS 2 instruction at the end of your source code is required because it closes the file. Without it, you'll get a splat file.

PUT (line number[-[line number]]), filename [,device number]
Saves portions of the source program to disk. The line number parameters are used to specify which lines of the code are saved. All the line number options of BASIC's LIST command are available.

AFFIX filename[,dev.]
Appends a source file to the end of a source file in memory. AFFIX works like LOAD except that it doesn't overwrite any file in memory. If memory is empty, the program is LOADed normally.

DLIST filename[,dev.]
Lists a file from disk without disturbing the file in memory. DLIST works like BASIC's LOAD command, but the file is only displayed onscreen. The file in memory is not disturbed. The type of listing you get, depends on the list mode you are working with, normal or formatted.

MERGE filename[,dev.]
This command reads a program from disk, adds the line to the program in memory, and then lists the line to the screen in the mode in which you are using. If for example, both programs have equal line numbers, the lines from disk will replace the ones in memory.

ERROR MESSAGES
To help you debug your programs, Bassem adds several error and control messages to BASIC.

OUT OF LABEL MEMORY
This error means that there's no more room in memory to store label definitions. See the SET instruction above.

LABEL ALREADY EXISTS
This error appears whenever you try to define a label with a name that is equal to one already defined. Change, add, or delete one or more characters to make it different. Also use FIND to search for the twin label.

LABEL NOT FOUND

If you attempt to use the value of an undefined label, this error message will appear.

BRANCH TOO LONG
In relative addresing mode, you can jump up to 127 bytes forward or 128 bytes backward; if you exceed these limits, you'll get this error.

DIRECT MODE ONLY
This appears when a direct mode command is reached inside a program.

?ZERO VALUE IN xxxxx and ?ZERO ADDRESS IN xxxxx
These two control messages are warnings that indicate an instruction had a zero value for its argument. You can turn off these warnings with the OPZ command.

NOTES AND PROGRAMMING TIPS
Bassem uses several memory locations that you must be aware of, and do not try to modify them until it's absolutely necesary. Here are the locations and their uses.

Locations:	Use:
$0002 (2)	Identify a PASS 1 or PASS 2 during the assembly process.
$00FB-$00FE (251-254)	Current PC address pointer, and pointer to the next available byte of the label memory area.
$02A7-$02FF (679-767)	Storage area for the current label name;
$0334-$033B (820-827)	Temporary buffer for preserving several three-page pointers.

Gazette, October 1994

64/128 VIEW: Survey Results

By Tom Netsel

The results of the Gazette readership survey that we published on the July disk are in and tabulated. I'd like to thank all of you who sent in your replies. While most came by mail, a good number arrived via QuantumLink and the Internet.

Tabulating the answers to our questions was the easy part, but many of you also sent suggestions about the disk. It'll take us a little longer to read through and absorb all of your comments, but you can be sure that we'll read every one. Whatever changes we make in the disk will be directly related to the survey. With your input, we hope that any changes will be viewed as improvements rather than irksome changes.

It's interesting to note that over the years the number of 64 owners who have completed our surveys has remained around 75 percent. This year the number is 74 percent, with 56 percent of you owning 128s. The 128 figure is up more than 10 percent over previous years, but hang on! Those figures add up to way more than 100 percent. That's because many of you own both computers. Breaking down the figures a bit more reveals that 44 percent of you own only a 64, 26 percent own only a 128, but 30 percent of you own both computers. It seems that 90 percent of you who own 128s also use an 80-column monitor with your system.

Before Gazette switched to its present disk format, 61 percent of you had at least used the companion Gazette Disk, but 39 percent had not. I was curious about this because a number of people seemed to have no idea how to use the disk, despite my answering a number of questions by mail and in this column, I gather a small percentage of you are still confused about a few things. When you load a program's documentation, a help/menu screen appears. This screen tells you how to change the colors of the background and text and it also explains how you can print the documentation with either the default settings or your own.

Finally, if a program will not load and run from the menu, reboot your computer and try loading and running it as you would any Commodore program. For the most part, documentation is on the front of the disk and the programs are on the back. Our programs are not copy protected, but keep a few things in mind. If a program has a feature in which it writes data to the disk, you'll have to load the program and copy it to a one of your own work disks. Gazette Disks come without notches in them. This means you cannot write or save programs to this original disk--unless you punch your own notches. Once you've seen what a program can do, we recommend that you make a copy of it on your own work disk and put away the original disk for safe keeping.

About 72 percent of you do some programming in BASIC and half of this number consider themselves intermediate, 41 percent beginners, and 11 percent advanced. A smaller number of you (28 percent) program in machine language, with 76 of those replying classifying themselves as beginners, 20 percent as intermediate, and 4 percent as advanced.

The 8-bit Commodore market has often been called "mature." Well, it seems that those of us you use those computers are also mature. The average Gazette subscriber is 48 years old. There's a 94 percent chance that he's male, and he's subscribed to Gazette for about 5 years.

While these are the averages, it's interesting to note that more that 40 percent of you have subscribed to Gazette for more than 10 years and our largest number of users (31 percent) are more than 65 years old.

The teenager blasting aliens with his 64 is no longer the average Commodore computer user! In fact, less than 1 percent of you are teenages and only 4 percent of you want more games in Gazette. The number requesting more utilities is 40 percent, and the number who want a mix of both is 56 percent.

For years our philosophy has been to select programs that will run on the largest number of machines. If a program required special equipment not owned by a majority of readers, we've tended to reject it. We've sometimes published programs of special interest but we've been hesitant to spend resources and time developing programs that couldn't be used by the majority of our readers. In the past, only a small number of you have owned RAM expansion units, for examaple, so we've not featured many programs that require an REU.

Now, it seems that about half of you own an REU of some kind. It looks like Gazette had better start publishing more programs that take advantage of that extra memory. Programmers should take note.

When it comes to disk drives, the 1541 is still the most popular with 47 percent of you owning at least one. The 1571 comes in next with 31 percent, followed by the 1581 with 21 percent.

Once again, I'd like to thank the hundreds of you who returned your surveys. We'll be looking at your comments and suggestions and using this data to help us plot the course of Gazette Disk for the next year or so.

Gazette, October 1994

FEEDBACK

Questions and comments from our readers.

NEW PRINTER BLUES

I recently bought a Star Micronics NX-2450 printer and use it with a Micrographix interface and my 128-D. I am having problems using the formatting commands with SpeedScript 128+. Can you help?
RICHARD W. CONKLIN
BATH, NY

There are two types of formatting comamnds used with SpeedScript, stage 1 and stage 2 commands. Stage 1 commands usually control variables such as margins and page length. These commands are usually a single reversed letter followed by a number with no space between the command and the number. In most cases they are placed at the top of a document and are executed before printing starts.

Stage 2 commands are often used to center or underline text and are usually embedded within a document. These commands are programmable from within SpeedScript and you can define your own. To define a printkey, press Ctrl-3, then the key you want to assign as the printkey, then an equal sign (=), and finally the ASCII value to be substituted for the printkey during printing. The entire upper case alphabet is available for use as printkeys.

For example, many printers use an ASCII value of 18 to print text in reverse video and a value of 146 to turn off reverse video. You could define uppercase R as a printkey and assign it the value of 18 to turn on reverse printing, and then define uppercase O as 146 to turn off reverse printing.

SpeedScript was designed to work with a large number of printers, but it is impossible for us to test it with all the printers and interfaces that are now available. SpeedScript doesn't understand the intent of the printkey, it just sends out that value.

Many printers and interfaces require different codes and settings for different printing effects, and we just don't have the hardware available to find out what works and what doesn't. Perhaps if readers have printers and interfaces like yours, they will write in and share what codes and settings that they've found to work.

CHIPS, PLEASE

I am looking for the 8563 80-column video chip used in the 128. I'm also trying to find a 6560 video chip used in the VIC-20. I would appreciate any help you could give.
ARRON BURNELL
DAYTON, OH

Give The Grapevine Group a call at (800) 292-7445. It stocks many of the chips found in Commodore computers. The address is 3 Chestnut Street, Suffern, New York, New York 10901.

SPRITE COLLISIONS

I am writing a game that has to detect sprite-to-sprite and sprite-to-screen collisions. The program has to decide whether the sprite can pass through an object such as a cloud or whether it's something solid like a wall.
JUSTIN HUSSEL
MELBOURNE BEACH, FL

Many Commodore reference books devote whole chapters to understanding sprites, but here's some information that might help. In games it's necessary to find out when a sprite has collided with another sprite or some other screen data. There's an easy way using PEEK and the AND operator to tell if a sprite has been in a collision.

Location 53278 is a hardware location in the VIC-II chip. The eight bits in this location, one for each sprite, are usually clear (0). When two sprites overlap, a collision is said to occur, and the bits in location 53278 which correspond to the colliding sprites are set to 1. The bits remain set even after the sprites move apart. To determine if a particular sprite has been involved in a collision, peek location 53278 and examine the appropriate bit.

The general procedure is to peek the location, assign the value to a variable, and then use the variable with the AND operator and sprite numbers to check the individual bits. You might use a line of code something like the following.

```
P=PEEK(53278): IF P AND N THEN sprite was in a collision
```

The value of N, the bit value, is determined from the table below.

Sprite	Number
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

For example, the numbers that correspond to sprites 2 and 3 are 4 and 8, respectively.

```
100 P=PEEK(53278)
110 IF P AND 4 THEN spite 2 has collided
120 IF P AND 8 THEN spite 3 has collided
```

Actual program statements would be placed in the code after each THEN. In the example below execution jumps to line 300 if spite 5 is involved in a collision. Line 300 could be the start of a subroutine that activates a collision sequence.

```
100 P=PEEK(53278)
110 IF P AND 32 THEN 300
```

Location 53278 only reports sprite-to-sprite collisions, in which one sprite has hit one or more other sprites. To check if a sprite has overlapped any screen data, such as characters or bitmapped graphics, use the same technique with location 53279.

```
P=PEEK (53279): IF P AND N THEN sprite has hit screen data
```

To check if sprite 6 has hit a character-drawn barrier, for example, use a line like the following.

```
P=PEEK(53279): IF P AND 64 THEN 300
```

For more information abou sprites, you might want to check a reference such as "All About the Commodore 64, Volume Two" by Craig Chamberlain.

OUT OF MEMORY
I was writing a fairly large program that used a lot of memory. When I tried to run one version of it, it loaded fine, but when I tried to run it I got an OUT OF MEMORY error in the line where I had DIM statements for my arrays. Why would this line trigger the error message?
ED GRIMSLEY
RALEIGH, NC

When you create an array with a DIM statement, the computer sets aside some memory for exclusive use by the array. When you get an OUT OF MEMORY message on the line that contains the DIM command, that means that there isn't enough memory remaining in the computer for the arrays that you want to create.

One way to save memory when creating arrays is to use integer arrays instead of numeric arrays. Floating point arrays need five bytes per element, but integer arrays need only two bytes per element. The statement DIM(100) would create an array of 101 floating point numbers from 0-100 that would take up 505 bytes of memory. If you used DIM%(100) instead, it would take up only about 202 bytes of memory.

If you have a question or comment about, or simply want to share some information with other Gazette readers, write to Gazette Feedback, COMPUTE Publications, 324 West Wendover Avenue, Suite 200, Greensboro, North Carolina 27408.

D'IVERSIONS: Information Dirt Roads, Bike Paths, and Hiking Trails

By Fred D'Ignazio

We have all heard the hype about the information superhighway which will soon link people together around the globe. We hear about school children who are corresponding with electronic pen pals in St. Petersburg, Russia, and Okinawa, Japan. Subscribers to national and international computer services are supposed to be able to shop, play games, visit libraries, and have live online chats with people all over the world without ever leaving the comfort of their homes.

Connecting with people and information from around the world is glitzy and thrilling, but it shouldn't distract us from the real benefits we can receive by connecting with people right in our own community.

During the last year I have been involved with a local project known as Multi-Media Detectives. The mid-Michigan cable operator for TCI Cable, a Lansing-area bulletin board for children, TCN (The Children's Network), and my company, Multi-Media Classrooms, partnered with teachers and fifth graders in three school districts. Our motto was Think global but link local. We connected with national services such as America Online and Turner Broadcasting, but we encouraged the children to be research gophers by looking for resources right in their own backyard.

We were all surprised by how rich that backyard turned out to be. Students and their families knew many exciting people who lived right in our small community. They brought these people into our classrooms and shared them electronically over the cable and computer network we had patched together. And they uncovered many artifacts as well--old photos, Civil War souvenirs, family treasures, and so on--which they brought to school and sent across our little network.

HOLDING A TALK SHOW WITH JIM CASH

Linking students in the classroom with famous personalities in their hometown was the most exciting way for us to use the cable TV and computer networks. One of our teachers knew the screenwriter Jim Cash. Mr. Cash has written screenplays for six movies, including "Top Gun," "Dick Tracy," "Legal Eagles," and "Sister Act." His home was only a few blocks from one of the schools on our network, and he was able to visit our classrooms electronically via his computer and modem.

Several times during the year the students scheduled live computer chat sessions with Mr. Cash. They called these chats "talk shows." First Mr. Cash would talk online about his career and give tips to kids about writing, and then students would ask him questions which they had written before logging onto the network. The technology permitted all five classrooms (four fifth-grade classrooms and one third-grade classroom) in the three districts to participate simultaneously in the talk show.

Mr. Cash, who is incredibly busy, would never have had the time to visit each school in person, but in order to "appear" on the talk shows all he had to do was press a button. He never even had to leave his study. He could work on a new movie right up to the time of the talk show, dial the kids, field their questions, and then immediately click a button and return to his latest movie project.

THE ERNEST GREEN STORY

Perhaps our biggest electronic catch of the year was Ernest Green. Mr. Green was one of the African-American students who became nationally famous when they integrated an all-white high school in Little Rock, Arkansas, in 1957. Mr. Green went on to become an advisor to several U.S. presidents. He represented the U.S. at the inauguration of South African president Nelson Mandela, and he recently cotaught a civil rights lesson in a Washington, DC, classroom with President Clinton.

Mr. Green came to our state this spring as a speaker at Michigan State University's spring commencement ceremony. One of the partners in our Multi-Media Detectives project knew a friend of Mr. Green's on the Michigan State faculty. After he heard about our project, Mr. Green agreed to come to the local TCI public-access studio to hold a two-way televised talk show with all the classrooms in our project.

The adults were excited about having Mr. Green talk with the kids in their classrooms because they had been alive during the historic events in Little Rock in the 1950s. The amazing thing was that the kids were excited, too. But why? It wasn't as if they were going to speak with a current media star like Michael Jordan or Shaquille O'Neal.

The students were excited because they had seen a biographical movie about Mr. Green titled "The Ernest Green Story." The movie appeared on the Disney Channel early in the year. And TCI Cable sent out copies of the movie to each of the classrooms a couple of weeks before Mr. Green visited the local studio.

The part of Ernest Green in the movie was played by a talented African-American actor who had starred in the recent movie "Boyz 'N the Hood." The Ernest Green Story was dramatic, tense, and realistic. It made history come to life for students who were born 20 years after the original events in Little Rock. The students identified with the plight of Mr. Green in the movie and his African-American friends. After watching the movie, they felt Mr. Green and his friends had been true heroes in standing up to an entire community just to receive their basic rights.

They couldn't believe that they were going to meet THE REAL ERNEST GREEN!

The biggest surprise to the students during the talk show with Mr. Green was how old he was! The historical events in the movie had seemed so real to them that the students regarded them as current events. Mr. Green had to patiently explain to the fifth graders that the incident at his high school had occurred 37 years ago!

YOUR LOCAL IS EVERYONE ELSE'S GLOBAL

One of our biggest insights in this project was that what was local for us and might be boring and normal was global--faraway, exotic, and exciting--for kids in other cities, countries, and other schools. Local things that kids grew up with their whole lives, which once were invisible because they were so close, all of a sudden came into focus when students realized these same things would be hot items for students in distant schools and remote communities.

These same ho-hum local resources also became more cool when they were broadcast over cable TV or transmitted via the computer bulletin board to other classrooms in our project. For example, a parent of one of the students was an avid Civil War buff and reenactor. He came to class one day dressed in his Civil War uniform and carrying his marching and fighting gear. The students in the fifth-grade classroom shared him with all the classrooms on the network by pointing a video camera at him while he demonstrated all his neat stuff. He was a star! Students and teachers in the other classrooms watched in fascination for almost two hours as he showed how a real Civil War soldier actually lived, fought, and survived on the battlefield.

It was a further incentive to students that they could ask the soldier personal questions over the two-way TV link. For all of us who were there that day it was like talking with a real, live, breathing Civil War soldier. It felt as if we had boarded a time machine and zipped 130 years into the past, right into the middle of the Civil War. The soldier always stayed in character, and pretty soon we all accepted that he was real and somehow we were able to talk to him live!

ONE-WAY TV IS BORING

One of the most important lessons we learned in our project was how much children liked to interact with TV, once given the chance. As adults, we hold the stereotype that children "zone out" and become couch potatoes when the TV comes on. Actually this is not the case. Unless a TV program was extraordinarily interesting (as in the case of "The Ernest Green Story") students in the classrooms in our project quickly lost interest in the TV and began talking, teasing their neighbors, and looking for excuses to leave their chairs and walk around their room. Even live, two-way TV was disappointing when the student groups were too large. If the students weren't given an opportunity to talk directly with their friends in other classrooms or with the experts on TV they showed their boredom after only a few minutes into an event.

A proof of the success of our Think global but link local philosophy was when Turner Adventure Learning held an electronic field trip to the Battle of Gettysburg. The field trip was well done and featured live national television coverage to classrooms broadcast from the historic Gettysburg battlefield in central Pennsylvania. The program featured child actors who portrayed historical characters from the original battle. It also allowed students across the U. S. to ask questions of the CNN NewsRoom commentators onscreen via phone, fax, and America Online.

For our students, however, this wasn't enough! They were used to being able to turn on their televisions and interact with students in other classrooms or famous people like soldiers and civil rights pioneers. They were used to turning on their computers and holding lively "talk shows" with famous screenwriters like Jim Cash. On the national field trip our students submitted dozens of questions via modem and not a single one was answered. On the other hand, when they participated in the local TV and modem talk shows every one of their questions was immediately answered, and each student was featured on camera or identified by name. On the national network they were anonymous, but in their local project the spotlight shone just on each of them alone!

WOULD THE REAL DR. R PLEASE STAND UP?
As part of our project, students frequently asked their parents to help them scout out local experts who would participate in our project. The expert could visit the classrooms via modem like Jim Cash or visit the classrooms via two-way cable TV like Ernest Green. They could even visit via speakerphone or live like Dr. Roger Rosentretor, a Civil War historian, did one morning midway through the project.

That's just what Dr. Rosentretor (Dr. R) did during the three days of the Gettysburg electronic field trip. He came to our classrooms each day after the field trip was broadcast over our classroom TV, and he told the kids colorful stories that made the incidents shown on TV seem more personal and understandable to our fifth graders.

One of our proudest--and funniest--experiences of the entire year occurred during a classroom visit by Dr. R.

The students were online chatting with other students from all over America in the "Studio Room" of Turner Adventure Learning. The subject was the Civil War and the importance of the Battle of Gettysburg. But the conversation was going nowhere, and the students were clowning around and driving the official Turner chat moderators crazy.

That's when Dr. R sat down at the computer in Mrs. Lafkas' fifth-grade classroom and logged on. He introduced himself as an historian, a specialist on the Battle of Gettysburg, and editor of the "Michigan History Review", the largest state history review in the U.S.

The Turner moderators were delighted to see him. "We're so glad you're online, Dr. R!" they wrote. "We would like you to officially lead our discussion on the Battle of Gettysburg."

Dr. R complied, and the discussion took off. All the students and teachers in classrooms across the country immediately began feeding him their personal questions about the Civil War. Dr. R's knowledge proved to be encyclopedic and extraordinarily funny.
He had a witty or poignant story to tell about almost every incident,

event, or personality in question. His answers to the students' questions were brief, factual, and intriguing. The Turner people wrote him while he was online and credited him with saving the discussion!

Then, suddenly, Mrs. Lafkas pulled Dr. R away from the computer. He had to begin meeting privately with the student "detective" teams to help them with their research projects. The teams and Dr. R moved across the hall to an empty classroom, leaving the rest of the class to continue the electronic chat online.

That's when one of the fifth graders still in the room noticed that the other students around the country participating in the America Online chat were still flooding Dr. R with questions. "Hey!" said one of Mrs. Lafkas' students. "They still think Dr. R is online."

What happened next was the greatest proof of success for our project. Without consulting their teacher, Mrs. Lafkas' students grouped around the computer and read the incoming questions aloud.

"Did General Pickett charge along with his men on the third day of the Gettysburg Battle during the famous Pickett's Charge?"

"How many people were killed during the Battle of Gettysburg?"

"Were there any women killed?"

"Were blacks or native Americans killed?"

"Did any children fight at the Battle of Gettysburg?"

On and on, the questions kept coming.

Mrs. Lafkas' students formed ad hoc teams and began researching the questions as they came in. They grabbed their Civil War research books. They discussed the videos they had watched in class. They quizzed Mrs. Lafkas. They even ran across the hall and stuck their head in the room and fed a couple of questions to Dr. R and then dashed back to the classroom and breathlessly dictated his answers to a classmate who was at the computer keyboard.

Mrs. Lafkas' students typed their answers as quickly as they could into the computer and fed them via America Online out into the national chat session.

"Thank you, Dr. R!" other students around the country replied. "You sure know a lot, Dr. R!" they wrote.

And Mrs. Lafkas' students glowed!

Once Mrs. Lafkas discovered that her students were "subbing" for Dr. R, she decided that it would be a good time to conclude the online chat. She was very proud of her students and showered them with praise for their research.

Her one area of concern was that the teachers in classrooms around America might be wondering why the knowledgeable and erudite Dr. R had suddenly begun spelling like a fifth grader!

Gazette, October 1994

BEGINNER BASIC: Conversions

By Larry Cotton

Let's finish our units conversion program that we started last month. I hope that we'll learn a little more about constants and variables.

Incidentally, I'm probably a little more particular about distinguishing between constants and variables than most people. The correct word for letters which represent numbers and other letters is probably "variable," but I like to reserve that word for something that varies while a program is running or that gets a different value from the user each time a program is run. For instance, if "Q" is the letter which will represent a number of units to convert, such as 4, then it's a variable.

Those things which don't change value are what I call constants. In this program, the numeric constants are those numbers which convert one unit to another. For instance, since there are always 25.4 mm in one inch, 25.4 is a conversion constant. The string constants are the abbreviations for units, such as MM (millimeters) and IN (inches).

Where were we? Oh yes, getting ready to convert metric units to English. But first, let me repeat lines 220-250 from last month's program, which gather and check for valid user input and control program flow. And please add another [UP] (cursor up) to lines 230 and 240; this ensures that the cursor will stay on the input line when an out-of-range menu item is entered. Here are those lines again.

```
220 INPUT"[DOWN]WHICH NUMBER";N
230 IFN<1THENPRINT"[UP][UP][UP]": GOTO220
240 IFN>32THENPRINT"[UP][UP][UP]": GOTO220
250 IFN>16THEN290
```

Lines 290-310 (for menu items 17-32) parallel lines 260-280 (for menu items 1-16), which were covered last month.

```
290 PRINT"[DOWN]HOW MANY(spc)"M$(N-X);
300 INPUTQ:A=Q*1/C(N-X)
310 PRINT:PRINTQ;M$(N-X)"(spc)="A; E$(N-X):END
```

Here's where things get a tad tricky. In those three lines we calculate an index to all three arrays. This is a very important and useful concept in BASIC programming. Often, arrays are accessed indirectly--from calculations rather than from exact numbers.

Recall that in line 50 we read 16 three-item groups of data. The first item in each group is a numeric constant, C( ). Thus, C(1) is 0.0000254, and C(16) is 1609344. For the 16 English-to-metric conversions, they're used without change.

But for metric-to-English conversions (N = 17 through 32), the inverses of the constants are used.

An inverse of a number is that number divided into 1. The inverse of 2 is 1/2 or 0.5; the inverse of 25.4 is exactly 0.03937. (Actually, the computer will introduce a slight error, but that relates to how it converts decimal numbers to binary, does its calculations, and then converts them back from binary to decimal so we can read them. But it's close enough.)

By using inverses, we can neatly generate the 16 other constants that we need to handle metric-to-English conversions.

Thus, when menu number N is either 4 or 20, C(4), which is 25.4, is the basic conversion factor. We just have to subtract 16 from any input greater than 16 and use inverses to take care of those conversions.

In line 40 we defined the constant X as 16. Lines 290-310 subtract 16 from N to index the arrays.

An example: The user wants to convert two millimeters to inches. N would be 20 from line 220. Since it's greater than 16, line 250 sends control to line 290. The computer then performs five operations:

1. Moves the cursor down one line
2. Prints "HOW MANY "
3. Subtracts N-X to get 4
4. Looks up the value of M$(4) and finds MM
5. Prints MM on the screen

At this point, the computer screen reads the following.

```
HOW MANY MM?
```

Line 300 waits for the user to input Q. He or she enters "2" and presses Return. Control continues along line 300 to calculate the answer. The computer then performs these next calculations:

1. Calculates N-X again to get 4
2. Finds the value of C(4) to be 25.4
3. Divides 1 by 25.4 to get approximately 0.03937
4. Multiplies 2 (the value of Q) by 0.03937 to get 0.07874

Thus, the value of A (the answer) is 0.07874. Line 310:

1. Prints a blank line
2. Prints Q as "2"
3. Subtracts N-X again
4. Prints "MM" again
5. Prints an equal sign
6. Prints the value of A as 0.07874
7. Calculates N-X again
8. Prints the "value" of E$(4)--"IN"

9. Ends the program

The computer screen now displays the following line.

```
2 MM = 0.07874 IN
```

Try a few conversions. An easy way to test the program for accuracy is to convert from one unit to the other and back again with the same measurement.

In other words, run the program and pick option 4 for inches to millimeters. At the prompt, type in "1" and press Return. The equivalent number of millimeters will be displayed as 25.4, and the program will end.

Now run the program again and select option 20, which converts millimeters to inches. At the prompt, enter "25.4," and the answer 1 (inch) should appear. If it doesn't, go back and check your typing carefully, especially the DATA statements.

Gazette, October 1994

MACHINE LANGUAGE: Strings and Structures

By Jim Butterfield

Strings are not hard to handle once you get used to them. "Canned" messages can be defined in your program; input string data can be brought into a buffer area. To reference a string, all you need is two items of information: where in memory the string is located and how long it is.

A "structure" is a collection of data of various sorts, grouped together. Structures are popular in many high-level languages. BASIC doesn't use structures. In machine language, we could use structures or we could use other ways of organizing data. This time, we'll use them to get a feel of how they work.

DEFINING STRINGS

It's not hard to identify where a string starts, so just make note of its address. The length of the string is usually defined in one of two ways: explicit length information or a special terminating character.

If you know that all the strings you are interested in will be shorter than 256 characters, you could define each length as a single byte of data. These shorter strings are easier to handle since indexing will reach across the entire string.

If your string lengths may exceed this limit, you'll need to define the length with a two-byte word; alternatively, you could supply the string's ending address.

Special terminating characters are quite popular. The C language, for example, puts a binary 0 at the end of strings. We can do that, too, or we might find it convenient to use some other character such as a Return. The disadvantage of this method is that our terminating character can't be used in the middle of a string. If you are dealing with simple text, you can be confident that strings will never contain such a thing as, say, binary 0. But strings can be used for data other than text.

Our sample program, Months, will use a mixture of these methods to define the lengths of various strings.

STRUCTURES

It's often convenient to group various types of data as a unit, or structure. A BASIC array, for example, consists of the same kind of data throughout, either all strings or all numbers. But a structure can be a collection of various kinds of data.

If you look at code in higher level languages, you might find a structure such as this.

```
    name   char[10];

    abbrev char[3];

    numday int;
```

This might signal you to set aside ten bytes to hold the name of this unit, then three bytes to hold an abbreviation, and, finally, set aside enough space for an integer value. The number of bytes needed to hold an integer might vary from one machine to another.

OUR PROGRAM
Our simple program will use a structure similar to that described above. It will ask the user for the name of a month and then report how many days are in that month.

The name of a month may be up to nine characters in length. We'll use a binary 0 to terminate the name, giving a total possible size of ten bytes for this field.

The abbreviation holds the first three characters of the month name in uppercase. That will make it easier for the program to check what is typed in. The user's input will be converted to uppercase, and only the first three characters will be checked. We don't need a terminating 0 byte, since this string length is fixed at three characters.

The number of days in each month will obviously be less than 255, so a single byte will hold this value. The total length of this structure is 14 (10 + 3 + 1) bytes.

The length of each field is fixed, so a short month name such as March must be padded out with binary zeros to fill the available space. The exact method may depend on the assembler you use. For example, here's the structure for March written in PAL/BUDDY format.

```
     .asc   "March"

     .byte 0,0,0,0,0

     .asc   "MAR"

     .byte $31
```

Other assemblers might accept a simpler definition such as the following.

```
     .byte "March",0,0,0,0,0,"MAR"

     .byte $31
```

Note, by the way, that the number of days is held in BCD, not binary. Hex 31 would have a binary value of 49, and we obviously mean March to have 31 days. The BCD format will make it easier for us to output the value.

PROGRAM COMMENTS
The program's source listings are available on this disk. I'll make a few notes on the coding here.

The prompt message is defined with a binary 0 at the end. So, rather than counting characters, our program will print until it sees this end signal.

```
120  ldx #0
130 prlp lda prompt,x
140  beq inpt       ; out we go!
150  jsr $ffd2      ; else print it
160  inx
170  bne prlp
```

The program calls the Kernal's INPUT routine at $FFCF, rather than using the more usual GET at $FFE4. This gives the user more freedom to type, but the program is suspended until the Return key is pressed. We take only the first three characters of what the user has typed.

```
190  ldy #0
200 inlp sty ysave  ;$FFCF might hit Y
210  jsr $ffcf
220  ldy ysave
```

If we see a Return before three characters have been received, we don't have a valid data entry.

```
230  cmp #$0d
240  beq exit  ; less than 3 saves
250  sta inbuff,y
260  iny
270  cpy #3
280  bcc inlp
```

Sending a Return to the output cancels the rest of the keyboard input line. It also, of course, starts a fresh line on the screen.

```
300 scan lda #$0d
310  jsr $ffd2
```

Since we're going to hop from structure to structure (month to month) as we check the input, we must put the address of the structure into an indirect address.

```
330  lda #<table
340  sta $fd
350  lda #>table
360  sta $fe
```

Our plan is to check the user's input against the abbreviation field of the structure. That's at position 10, so we put that value into the Y register.

```
380 scnlp ldy #10
```

To allow for the value of Y starting at 10, we offset the input buffer downward by the same amount. After we load the character, we convert it to uppercase with an OR command.

```
390 scchr lda inbuff-10,y
400  ora #$80 ; change to uppercase
```

Since Y started at 10 and we want to compare three characters, we know that we are complete when Y reaches 13.

```
440  cpy #13  ; test 3 saves matched
```

If we didn't find a match on the last structure, we must move along to the next one. We do that by bumping the indirect address by the number of bytes in each structure, 14.

```
480  clc
490  lda $fd
500  adc #14     ; size of structure
510  sta $fd
520  bcc ninc
530  inc $fe
540 ninc ...
```

When we do find a match, we print the full name of the month; that's at position 0 in the structure. Since the length of this name is not fixed, look for that binary 0 at the end.

```
590  ldy #0  ; position in structure
600  mlp lda ($fd),y
610  beq hprn      ; binary zero - exit!
620  jsr $ffd2
630  iny
640  bne mlp
```

The program counts out the number of characters in each of the fixed strings. The value printed between the two is taken from the structure. This time, we look at position 13 for the field we want.

```
730  ldy #13   ; position in structure
740  lda ($fd),y
```

The two digits of the BCD number are split apart and printed in the usual way. Then the whole program repeats, asking for another month to be named.

An invalid name causes the program to exit. That includes no name--which is what you get by just pressing Return.

SUMMARY
We've handled several different kinds of strings and played a little with a simple structure. You may find this useful in understanding how some higher-level languages go about handling data.

This program doesn't care if the user type in "april," "APRIL," "aPriL," "Apr," or, for that matter, "Apron." It's good to aim for this kind of user friendliness in your programs.

Incidentally, Months, which is on the flip side of this disk runs on virtually any Commodore 8-bit computer, comes in two pieces. Months is the BASIC boot program that brings in the machine language module, called MONTHS.ML. If you haven't seen this type of boot program before, it's worth taking a look at how it is constructed.

Gazette, October 1994

PROGRAMMER'S PAGE: Logical Ramblings

By David Pankhurst

Is 1+1=1 valid? Yes, if you're dealing with Boolean Algebra. Although you may not have heard of it, you've seen it in action; every IF statement is comprised of Boolean statements, called propositions. The plus sign was the original symbol for OR, making 1+1=1 equivalent to today's 1OR1=1.

Boolean logic gets its name from George Boole, a British mathematician of the nineteenth century. He envisioned an algebra of logical statements or propositions. A proposition is as simple as A>B, or as complex as A*B+D>=X-3. The key is that each proposition has one of two values: true or false, yes or no, or (in the case of computers) zero and nonzero.

This month we're going to look into the logical aspect of things and see what we can do on the Commodore with George's discovery.

USING BOOLEANS IN MATH
Commodore BASIC uses several operators in evaluating logical or Boolean expressions. We use >, <, and = to create propositions, and AND, OR, and NOT to combine the individual statements into one. Although these propositions are used mainly with IF statements, they can be used in any math expression, since they have a value. In Commodore BASIC, it's 0 for an untrue statement and -1 for a true statement. For example, X=A>B would assign -1 to X if the value of A were greater than B and 0 otherwise.

A practical example of how we can alter these values for our benefit can be found in the following.

```
X=-5*(A<B)
```

Here the statement (A<B) is evaluated, with two possible outcomes: -1 (true) or 0 (false). The result is multiplied by -5, giving 0 (false) or 5 (true). Use any number instead of 5, and you change the true value placed in X. What if you want to change the false value from 0? Make the expression two terms, one for each possibility, like in the following.

```
X=-5*(A<B)-9*(A>=B)
```

Here the two expressions A<B and A>=B are exact opposites, meaning only one is true at a time. (Note that the opposite of A<B is A>=B, not A>B.)

Since a false expression is 0 and multiplying it by any other number gives 0, only one of the two expressions is -1 (true) at a time. Result: X is assigned 5 if A<B and 9 otherwise.

MANY THINGS ARE TRUE
Strictly speaking, IF recognizes only two possibilities: false, which is 0, and true, which is nonzero. As far as IF is concerned, all nonzero results are considered true. This simplifies a statement like IFA<>0THEN to IFATHEN, saving time and space.

SIMPLIFIED BRANCHING
Knowing that any logical expression has only one of two values provides a way to do special branching. Adding 2 to a proposition gives 2 for false, and 1 for true. This is especially useful with the ON/GOTO command, giving you a simple kind of IF/THEN/ELSE statement.

```
100 ON (A<B)+2 GOTO 110:PRINT"HERE":END
110 PRINT"THERE":END
```

There's only one branch on this ON/GOTO for a reason. If A<B is true, then (A<B)+2 equals 1, meaning the GOTO 110 command is executed. But the convenient thing about BASIC's ON/GOTO (and ON/GOSUB) is that, if any out-of-range numbers are found, processing continues with the next statement, not the next line. The result is that a false condition (resulting in a value of 2) causes the ON/GOTO to continue with the rest of line 100, the PRINT command. Just remember that a false statement stays on the same line and a true one branches.

Another method of simplifying branching occurs when using the AND operator. A statement such as IF(A>B)AND(C<D)THEN is slower than it need be. When the program executes, both of the expressions A>B and C<D have to be evaluated. Time is saved by breaking them into two nested IF statements.

```
IF(A>B)THENIF(C<D)THEN
```

First A>B is checked. If it fails, the second IF (with C<D in it) need never be checked, which saves time. For even quicker running, place the expression that is least often true first. The fewer number of times that the first IF is true, then the less often the second IF is checked.

EXCLUSIVE, OR
When I say I'm going to the bank or the post office, people naturally expect me to go to one or the other, but not both. In English, the word "or" is an exclusive form, but in Boolean algebra, OR is inclusive; one statement, OR the other, OR both, may be true. This bears keeping in mind when you're writing IF statements.

The exclusive form of OR, called exclusive OR, (abbreviated XOR or EOR) works the way we're most familiar with. The result is true only if one or the other proposition, but not both, is true. If you have propositions A and B (each being a true/false expression), Exclusive OR them with the following.

```
A XOR B=(NOT(A AND B))AND(A OR B)
```

YOU CAN COUNT ON IT
When you do counting, usually you need two statements, one to count and another to test for overflow or total. Typically, you have to write code something like the following line.

```
X=X+1:IFX>MAXTHENX=0
```

Another way to implement it is like this.

```
X=((X+1) AND 3)
```

This statement will cycle from 0 to 3 and repeat without overflow. This only works with specific values, namely the powers of 2 minus 1: 4-1, 8-1, 16-1, 32-1, and so one. If that's what you need, it certainly is faster--and neater, too.

A FEW CONCLUDING BITS
We've focused on the Boolean or logical uses of the AND, OR, and NOT operators, but they have another use, as anyone who has set up a screen display or programmed the Commodore SID chip has found. In these cases you are concerned with setting and resetting individual bits of a variable or a memory location. For anything more than the simplest programming, you're going to have to twiddle bits, so here's a listing of how to perform the four basic actions involving individual bits.

1. Set a Bit: X=(2↑B)ORX
2. Clear a Bit: X=(NOT(2↑B))ANDX
3. Get a Bit's Value: Y=(2↑B)ANDX
4. Reverse a Bit: X=((2↑B)ORX)ANDNOT ((2↑B)ANDX)

In these equations X is the byte value, and B is the bit position: 0 for the lowest bit position and 14 for the highest (anything higher than 14 will give you an ILLEGAL QUANTITY error). All other bits, except the one designated by B, remain untouched. Normally you would replace the 2↑B calculation with a lookup table to speed things up.

Gazette, October 1994

GEOS: Who Are You Calling Mature?

By Steve Vander Ark

What's new? Years ago, there used to be a column with a name something like that in every one of the Commodore magazines on the newsstand. Nowadays there doesn't seem to be very much new to report. The reason, we're told, is that this is a mature market, which means that Commodore users have already spent all the money they're likely to spend on Commodore products. In a mature market there are no products released; there is nothing new.

Now I'd call the Atari videogame market a mature market, at least by that definition, or maybe the market for the Timex Sinclair computer. But the Commodore market? I think not. After all, there are always new products coming out for the Commodore and especially for GEOS. Don't believe me? Read on.

Here are a few goodies available from Parsec (P.O. Box 111, Salem, Massachusetts 01970-0111). These programs were originally included on some of the Twin Cities 128-64 Magazine disks. They have now been released on a single disk called Parsec's Tools #01, available for $19.95 plus $4.25 shipping and handling (prices somewhat higher for foreign orders). These programs are not shareware, so don't expect to find them on your local bulletin board. They are well worth the price of the disk, however.

geoPager
By Rob Knop, Jr.

Yet another minor inconvenience of the GEOS system has been conquered by a resourceful GEOS programmer. Rob has made it possible to batch-print a whole bunch of geoPaint documents without any further input from you. This leaves you free to go off and do something else while geoPager and your printer grind away unattended. Until now, if you wanted several copies of a geoPaint file or had several geoPaint files to print, you had to feed them into the printer one file at a time. GeoPager saves a lot of time spent babysitting your printer by letting you create a list of the documents you want printed and then taking over the work for you.

There are actually two versions of geoPager. The version called geoPager 128 is 80 columns only, while geoPager 64 is usable on either the 64 or the 128, but only in 40 columns.

An impressive list of features is the same for either version of the program. The main screen includes the option of previewing any geoPaint page before you add it to the list to be printed. GeoPager will print pages from more than one drive during the same print job. The print control dialog box lets you print more than one copy of each page. You can also tell the program to print the list more than once, which means that you could print more than one set of several pages.

Let's say you had created a three-page newsletter and you wanted two copies printed. You could tell geoPager first to print the three pages in order and then go back and print them again in order. You'd come back to find two sets of three pages waiting for you. Any kind of printing task you can imagine, geoPager can probably do for you.

Parsec's Tools also includes three geoPaint documents, two of which are excellent scanned clip art images, as well as a wonderful font called Zingies that comes in regular and megafont versions. This font is essentially a clip art collection, with beautifully drawn flowers, portraits, and symbols. Because it's a font, this clip art collection is accessible quickly from the font menu.

Font Paint 64-128
By Kent Smotherman

If you collect fonts, you need this program that's also part of Parsec's Tools collection. Font Paint is a font printing utility that's done right. The program creates one or more geoPaint pages showing all the fonts on a disk, each displaying the character set and listing the font ID numbers. The result is a ready-to-print font guide, complete with font information and examples. Once you've created a series of pages showing all your fonts, you could even use geoPager to print them with a minimum of fuss. This method of creating font examples has long been needed in the GEOS community, where the tongue-in-cheek philosophy seems to be Whoever dies with the most fonts wins!

Font Paint is not a perfect program, however. The interface is confusing, even more so because of the lack of documentation. For most GEOS programs this isn't necessarily a problem, since things tend to work in certain accepted ways. Font Paint fails to follow many of those conventions, however. For example, when you want to select a disk from which to print fonts or on which to create geoPaint documents, the program presents you with file selector dialog boxes, even though you aren't selecting files. There are a few bugs as well, one of the most obvious being that the program doesn't seem to properly log in a new disk when you click on the Disk button.

Since the author has left GEOS and started programming for the IBM, bug fixes seem unlikely. If you are careful to have the disks you want already in the drives when you start the program, however, you should have no trouble. The wonderful pages this program creates are worth the confusion.

Parsec isn't the only place to get new and exciting GEOS products, and all the crackerjack programmers haven't defected to the IBM world. Here's an example of a new upgrade to an existing product that really pushes the limits of Commodore applications.

GeoCanvas 3.0
By Nate Fiedler

5711 Mt. Pleasant Rd.
Bernville, PA 19506

The 64 (40-column) version and 128 (80-column) version are $28 each. You can order both for $43. Upgrades from earlier versions cost $13 plus trade-in disk.

Version 3.0 of geoCanvas is more than just an upgrade to 80 columns. Nate has reworked the way the program uses tools, adding functions and making it easier to upgrade or modify the system. The result is an even more powerful program that offers full GEOS 128 support as well.

As with earlier versions, 3.0 gives a very large work window and multiple windows to view documents. The tool icons are invisible until called for from the menus, which helps to keep the screen open for drawing. You can even view a full screen's worth of your document, which in 80 columns is quite a good-sized chunk. Having this much screen space given to the drawing window, not to visible tools and controls, makes it necessary to do a lot of menu-hopping as you work, but plenty of keyboard shortcuts are included to help things move along.

The new version is being sold directly by Nate. The disk includes several other utilities, most importantly Scrap Can, a useful tool to cut and paste large photo scraps. These utilities are upgraded to 80-column use with the 128 version as well. Look for a full review of geoCanvas 3.0 soon.

Next month, I'll be starting a series of GEOS columns in which I'll talk about programming GEOS applications using geoBASIC. I'll demonstrate some of what makes programming in GEOS unique. If you don't have a copy of this program but would like to give GEOS programming a try, contact Creative Micro Designs for a copy (P.O. Box 646, East Longmeadow, Massachusetts, 01028; order number 800-638-3263). The current price is $20. The manual for geoBASIC assumes that you already know how to program in BASIC, and my columns will be written with that same assumption.

Gazette, October 1994

PD Picks: Sortanos and Space One

By Steve Vander Ark

I have a heck of a lot of respect for programmers. I mean real programmers, the kind who turn out useful utilities or exciting, playable games. Two things happened over the past few weeks that convinced me that these people are unsung heroes. First of all, I began working with a fellow who programs for the Newton, a hand-held computer I have for school; he's going to write a utility for teachers to my specifications. Second, I tried my hand at programming again after being away from it for a few years.

The programmer I'm working with is a friend of mine on GEnie. I have traded a few messages with him via E-mail, discussing the possibilities for a set of portable teacher's programs. As we talked, I started to realize just what an art programming really is. In order to create the kind of program I had in mind, he had to help me transform my vague ideas into a real workable plan for a program. He has the vision and the creative thinking to turn my ramblings and half-realized ideas into a fully dimensioned application. He can see the result as an interconnected set of steps: input routines, screen displays, and so on. He's an artist; there's no doubt about it.

That impression was reinforced when I started my own little programming project, a very small GEOS program using geoBASIC. I realized that, while I understand how to program in BASIC, I am not a programmer. I am no artist either. I slogged my way through, trying to keep from creating a mess of spaghetti code, and thought how lost I'd be if I were trying to write something that actually did something useful. It made me again appreciate what a lot of time, effort, and inspiration go into a good program.

The reason I bring this up at all is that each of the programs in this month's column is the work of someone like that, someone who worked very hard to create a great, exciting utility or game. (In fact, every one of these @@PD Picks'' programs are like that.) The names of these programmers aren't always listed in my columns only because they don't leave their names on the work. But I want to take this opportunity to thank them all for the time and effort they've put into these programs and for all the pleasure I've derived from using them.

Some programs in this column are shareware, too, and that's where all this appreciation has to turn into some cold hard cash. If a program in this column is a shareware program and you choose to keep and use it, you owe that programmer the shareware fee. The fee requested for most Commodore programs is always reasonable. Send the creator his or her due, straightaway. And while you're at it, send a word of thanks, too. He or she deserves it.

SORTANOS

Author unknown.

Oh, I'm getting downright sedate these days. I can almost see myself hunkered down on a park bench somewhere playing some peaceful strategy game that requires heavy thinking, such as chess or dominoes or, oh, maybe Sortanos. Yes, sirree, here's another calm, quiet thinking game. Sortanos got its name from the fact that it's "sort of like dominoes." As a matter of fact, it is. It's also a lot of fun.

This is one of those games that you play against the computer. I usually don't like those kinds of games because I think computers cheat, but in this case the computer doesn't really have an advantage. You have a set of tiles to play in your hand, as does the computer. One by one you place them in a row, matching up like numbers. If you can't play, you have to draw an extra tile from the "boneyard."

The rules aren't much more complicated than that. You don't need to be the next Bobby Fischer to win. All you need is a little time, a little careful thinking, and a little strategy. It's not always easy to win, but it's always worth trying. And I promise you nothing will blow up.

Now that I have all these great disks from Jim Green, I'm going to start putting a third file in this column every so often. Jim includes a lot of graphics on his disks, including some animated cartoons he's created himself. These are done on the text screen using character graphics, and they feature sound effects and movement and show off a little of Jim's quirky sense of humor. The gags are usually pretty lame, but the programming is a delight to watch. There are much fancier graphic and sound presentations available for the Commodore, of course, but Jim's little cartoons have a charm of their own. This month I'm including one called Weird Marriage.

SPACE ONE
Author unknown.
So much for strategy and thinking. So much for utilities, and so much for the nice graphics displays. It's time to pop the top on another can of Mountain Dew and test the continuous fire button on my joystick. It's time to fry something.

When I first started this game, I was very impressed with the scrolling background that features nicely detailed gray craters. I also liked the swarms of enemy missiles, spaceships, and bombs that hurled themselves at my little round spaceship.

After six or seven tries at the game, however, my high score was edging only a little higher each time, and I started getting bored with things. I then figured out how to beat the missiles that scooted in on an angle and sucked toward me. At this point I wondered if I had found a dud game.

Then I hit level 2.

Space One is one of those games that doesn't make a game harder simply by making the bad guys move a little more quickly. No, as the levels go by in Space One, these bad guys get smarter and more powerful as well as faster. It wasn't long before I was careening all over the screen trying to avoid the hosts of nasty odds and ends. Some ships need multiple hits to take them out. Others fly in and make a sneaky little turn just when you think you've given them the slip. There's enough enemy firepower here to keep any space jockey busy for hours.

(Editor's note: Bruce Thomas of Edmonton, Alberta, Canada, wrote in to say that Tony Bratner is the author of Tenpins, a program featured in the August 1994 "PD Picks.")

Gazette, October 1994