COMPUTE!'s FIRST BOOK OF COMMODORE 64 SOUND AND GRAPHICS

Tutorials, Utilities, Programs and Other Helpful Information for the Owners and Users of the Commodore 64™ Personal Computer.

A **COMPUTE! Books** Publication

$12.95

COMPUTE!'s FIRST BOOK OF COMMODORE 64 SOUND AND GRAPHICS

COMPUTE!™ Publications, Inc. abc
One of the ABC Publishing Companies
Greensboro, North Carolina

The following article was originally published in *COMPUTE!* Magazine, copyright 1982, Small System Services, Inc.: "Sprite Editor" (December)
The following article was originally published in *COMPUTE!* Magazine, copyright 1983, Small System Services, Inc.: "Instant Art" (May)
The following articles were originally published in *COMPUTE!* Magazine, copyright 1983, COMPUTE! Publications, Inc.:
"MusicMaster" (June)
"Mixing Graphics Modes" (August, September)
"Moving Maze" (October)
The following articles were originally published in *COMPUTE!'s Gazette*, copyright 1983, COMPUTE! Publications, Inc.:
"Enlivening Programs with Sound" (July)
"Hi-Res Graphics Made Simple" (August)
"POKEing Graphics" (September)
"Understanding Sound Part 1" (October)
"Understanding Sound Part 2" (November)
The following article was originally published in *COMPUTE!'s First Book of Commodore 64*, copyright 1983, COMPUTE! Publications, Inc.: "Machine Language Editor"

Printed in the United States of America

ISBN 0-942386-21-3

10 9 8 7 6 5 4 3 2 1

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is a subsidiary of American Broadcasting Companies, Inc., and is not associated with any manufacturer of personal computers. Commodore 64 is a trademark of Commodore Electronics Ltd.

Contents

Foreword

The Commodore 64 can produce some of the best sound and graphics you can get on a home computer. Some of these very fine features, though, can be hard to learn to use. Even if you are not an experienced programmer, *COMPUTE!'s First Book of Commodore 64 Sound and Graphics* will help you learn techniques that will let you use your computer to its fullest. Many of the programs have appeared in *COMPUTE!* Magazine and *COMPUTE!'s Gazette*; many are printed here for the first time anywhere.

As with all COMPUTE! publications, you'll find a range of articles to teach you and utilities to help you. Both the beginning and advanced programmer will find many things they can use at once. As always, all programs are ready to type in and run.

You might take special note of "MusicMaster" and "High-Resolution Sketchpad." Even if you have never programmed in your life, you can use these programs. MusicMaster lets you create tunes on the Commodore 64; High-Resolution Sketchpad lets you draw two-color and four-color pictures, and save them to disk or tape so you can look at them as often as you like.

Regular readers of *COMPUTE!* Magazine and *COMPUTE!'s Gazette* know how useful COMPUTE!'s programs and articles can be; we hope you will find this book just as valuable. And if this is your first COMPUTE! publication, you are in for some pleasant surprises.

Chapter One

Getting Started

Chapter One

The Graphics Machine

Gregg Keizer

"The Graphics Machine" will introduce you to the fundamentals of Commodore 64 graphics.

You've unpacked your Commodore 64, connected it to your television, checked the connections, and opened the *User's Guide*. You're ready to begin programming, you think. For some reason, there's an empty feeling in the pit of your stomach, a moment of anxiety at the prospect of facing the machine. As you turn on the computer, the READY message appears and the cursor flashes. Now what? Remembering the elaborate demos the salespeople showed you, the fascinating arcade games they tried to get you to buy, you decide you want to try your hand at programming graphics. After all, the Commodore 64 has advanced color graphics. It's printed right on the box, you see, and the salespeople told you more than once that this computer has terrific graphics capabilities. An expensive sketchpad, you think. You'll be able to draw pictures and create scenes that will be as impressive as those games and demos.

You read through the *User's Guide*, and soon discover that it won't be that simple. It seems that even the most elementary picture takes line after line of numbers and symbols that are undecipherable. You must learn a new language, BASIC, you're told, before you can create those graphics your imagination was soon filling the blank screen with. Frustration begins, then impatience, then a feeling of hopelessness. You'll never be able to make the computer do what you want, you think.

Wrong. There is hope. Once you've overcome the initial shock that you have to learn new things to do new things, you'll find it's not really that hard. Graphics such as those in *Joust* or *Donkey Kong Jr.* may not come soon, but the basics of computer graphics are quite easy to grasp if you have some knowledge of BASIC programming. You don't need a degree in mathematics or computer science, either. Already you have the necessary attributes for programming graphics. A willingness to learn, to experiment, and to be creative. If you have these, you'll soon enjoy creating graphics on the 64.

Commodore 64 Graphics

The Commodore 64 *is* a graphics machine. It *does* have terrific graphics. All it needs is someone to tell it what to do. That's you, the progammer. Even though you may not think of yourself as a computer programmer (the image of someone sitting before a keyboard at 3 a.m. comes to you), that is what you are. Without a programmer, your computer would flash the cursor on and off, patiently waiting for a command. It will sit there forever, doing nothing, unless you *program* it to do something.

When you program an instruction into the computer, *then* it will begin working. And its graphics abilities will work hard for you.

The Commodore 64 uses its 6567 Video Interface Chip (known as the VIC-II chip) to create these graphics you'll program. There are a variety of graphics modes that this chip will produce, including a 40-column by 25-line text display, a 320 by 200 dot high-resolution display, and sprites, the small movable objects which you design and use in games. Not only does the 64 have several graphic modes, but these modes can be mixed. You can combine the text mode with the high-resolution mode to create a detailed picture at the top of the screen and words at the bottom. Sprites can be mixed with anything, making game-writing simple.

The simplest graphics mode, and the one you'll undoubtedly start with, is the text mode. Don't let the name fool you; you can create impressive and complex graphics patterns on the screen in the text mode. Unlike other computers, you do not have to set the 64 to this mode by a command. When you turn on your computer, you enter this mode automatically. Creating graphics in the high-resolution and sprite modes is more difficult and takes more time, although the rewards may be greater. These modes are covered fully in later chapters of this book, and since you're just beginning with graphics, we'll stick with the text mode to start with.

Coloring Text

As you turn on the Commodore 64, the screen display is in a single-color combination. Light blue text is shown on a dark blue background, with a light blue border. This is the default color, or the color the computer will use until you tell it to do otherwise.

Changing the color of text is quite simple. In fact, there is more than one way to do it. The easiest way, and one you may already know, is to use the color keys on the keyboard.

The 64 has 16 text colors you can work with. The first eight are available by using the CTRL key and a number key, while the second group of eight is available by using the Commodore key with a number key. Table 1 shows the possible colors for text and the key combinations.

Table 1. Commodore 64 Colors

KEYS	COLOR
CONTROL 1	BLACK
CONTROL 2	WHITE
CONTROL 3	RED
CONTROL 4	CYAN
CONTROL 5	PURPLE
CONTROL 6	GREEN
CONTROL 7	BLUE
CONTROL 8	YELLOW
COMMODORE KEY 1	ORANGE
COMMODORE KEY 2	BROWN
COMMODORE KEY 3	LIGHT RED
COMMODORE KEY 4	GRAY 1
COMMODORE KEY 5	GRAY 2
COMMODORE KEY 6	LIGHT GREEN
COMMODORE KEY 7	LIGHT BLUE
COMMODORE KEY 8	GRAY 3

Pressing CTRL and the 1, for example, changes the cursor to black. Any text you type in will appear in black. But when you hit the RETURN key, an error message appears. SYNTAX ERROR, the screen reads. What's going on?

In order to change the color of text, you must use the PRINT command. Only then will your computer understand that you're instructing it to alter text color. As long as you use the PRINT command and enclose the instructions in quotation marks, the 64 will follow directions.

```
PRINT "{BLK}PRINTING IN BLACK"
```

As you press the CTRL and 1 keys, you notice that an inverse video symbol is printed. This is the symbol the computer uses to keep track of which color is to be displayed. Don't worry about remembering which symbol goes with what color; the computer does that for you.

The line just typed will remain in the default color of light blue until the RETURN is pressed. Only then will the cursor and any additional entered text display in black. What you just did was to tell the computer to change the text color. Unless it's changed again, the 64 will continue to use this color.

Changing back to the original color can be done by either pressing the RUN/STOP and RESTORE keys together or typing

```
PRINT "{7}"
```

and hitting RETURN. Now the text is again displayed in light blue.

Any of the characters that can be entered from the keyboard, including the standard graphic characters shown when the SHIFT or Commodore key is used along with another key, can be printed in a different color. But you've probably noticed a problem. As soon as the RETURN is pressed, the text within the quotation marks prints, but you cannot display it again unless you retype the entire line.

So although you've told the computer to do something, you haven't actually programmed it. Without additional instructions, the 64 will execute your command only once, and then forget it. To tell it to remember, the instructions must be written in program form. In other words, line numbers have to be assigned and the computer told to RUN that program. The change is minor, and appears like this:

```
10 PRINT "{BLK}PRINTING IN BLACK"
```

A one-line program, but it will work again and again as long as you type RUN each time. The effect is similar, for additional text will display in black until another change is made or RUN/STOP—RESTORE is used. But the computer remembered the command. In fact, it will not forget it until you, the programmer, tell it to, or until the computer is shut off.

Chapter One

The following short program demonstrates that color can be changed as often as desired, and that the color will continue until it is altered again. Notice that in lines 60-90 the Commodore key is used to select colors from the second group of eight.

Program 1. Textchange

```
10 PRINT"{BLK}A DEMONSTRATION"
20 PRINT"{WHT}OF THE COLORS"
30 PRINT"{RED}THAT ARE AVAILABLE"
40 PRINT"{CYN}ON THE COMMODORE 64"
50 PRINT"{PUR}IS QUITE EASY TO DISPLAY."
60 PRINT"{1}CHANGING BACK"
70 PRINT"{2}TO THE ORIGINAL"
80 PRINT"{3}COLOR IS NOT THAT"
90 PRINT"{7}DIFFICULT"
100 GOTO 100
```

Although each line prints in a different color, the text would have remained in light red (color selected in line 80) unless the text was reset to the default color of light blue in line 90.

Graphic Characters in Color

Just as text can be displayed in various colors, so can the graphic characters. These are the characters shown on the faces of the keys, which are printed by pressing either SHIFT or Commodore key, then the appropriate key. Pressing the SHIFT and a key prints the symbol on the right side of the face, while using the Commodore key and a key prints the symbol on the left side of the face.

These graphics characters are part of the Commodore's standard set, and make the 64 a powerful graphics tool. You don't have to design your own characters if you choose not to, plus you have more available than most other computers. Using these, you can draw shapes, create game characters, and invent new figures. Many games, for instance, are created on the 64 using only the standard graphic characters. As you draw and create your own pictures on the screen, you'll use these characters more often than any other.

A Sketchpad

The Commodore 64 may not be as simple to use as an electronic sketchpad, but it can fill the same role, and it can do it in color. Think of the screen as a piece of graph paper that is 40 columns wide by 25 lines high. In fact, having a piece of graph paper with this rectangle outlined will help.

Each box on the graph paper represents one character on the 64's screen. You can fill each box with any character on the keyboard, ranging from text to graphic characters. Sketching your own figure on paper, deciding which graphics characters to use, and even coloring the figure with pencils will give you an idea of its final appearance.

You can turn the screen and the computer into a sketchpad. Using the space bar and cursor controls, you can place text or characters anywhere on the screen. When you end a line, do not type RETURN; instead, use SHIFT-RETURN, which will move the cursor down a line, but will not print the READY prompt. With this, you can move around the screen at will, inserting new graphics characters, removing others, until the figure is to your liking. You know exactly how the figure will appear on the screen when you're finished.

As when you entered a PRINT statement without a line number, this figure will be lost once the RUN/STOP–RESTORE keys are pressed. To force the computer to remember your drawing, it will have to be written in program form. This means, unfortunately, duplicating the drawing you just finished on the screen, but this time adding line numbers, the PRINT command, and quotation marks. The following program is an example of a completed sketch showing a top view of a pool table with a player ready to strike the cue ball.

Program 2. Pool Table

```
10 PRINT"{CLR}"
20 PRINTTAB(16)"{BLK}{4 DOWN}O⁅7 U⁆P"
30 PRINTTAB(16)"⁅J⁆{7 RIGHT}⁅L⁆"
40 PRINTTAB(16)"⁅J⁆{7 RIGHT}⁅L⁆"
50 PRINTTAB(16)"⁅J⁆{RIGHT}QQQQQ{RIGHT}⁅L⁆"
60 PRINTTAB(16)"⁅J⁆{2 RIGHT}QQQ{2 RIGHT}⁅L⁆"
70 PRINTTAB(16)"⁅J⁆{3 RIGHT}Q{3 RIGHT}⁅L⁆"
80 PRINT
90 PRINTTAB(16)"⁅J⁆{7 RIGHT}⁅L⁆"
100 PRINTTAB(16)"⁅J⁆{7 RIGHT}⁅L⁆"
110 PRINTTAB(16)"⁅J⁆{2 RIGHT}{WHT}{RIGHT}W
    {3 RIGHT}{BLK}⁅L⁆"
120 PRINTTAB(16)"⁅J⁆{3 RIGHT}G{3 RIGHT}⁅L⁆"
130 PRINTTAB(16)"L⁅3 O⁆G⁅3 O⁆@"
140 PRINTTAB(16)"{4 RIGHT}G"
150 PRINTTAB(16)"{4 RIGHT}G"
160 PRINTTAB(16)"{4 RIGHT}G⁅2⁆I"
```

```
170 PRINTTAB(16)"{5 RIGHT}G"
180 PRINTTAB(16)"{3 RIGHT}UEI"
190 PRINTTAB(16)"{3 RIGHT}JFK"
200 GOTO 200
```

The TAB(16) used in each line makes sure the figure has a straight edge. The ■ marks are right cursor moves, to create spaces when needed. Notice that the color is changed in line 20 to black; to white, then again to black in line 110; and finally to brown in line 160. Even though the drawing does not look correct in the program as you type it in, when you type RUN, it will appear as you wanted. The distortion appears because of the color commands in some of the lines, as well as the three-digit line numbers halfway through the program.

It may seem like a lot of work to draw using this method, but once you've typed and SAVEd this, you'll be able to RUN it as many times as you wish. If you SAVE the program to tape or disk, it will not be lost once the power is turned off, or the screen reset for a new program. Experimenting with your own drawings will show you the Commodore's graphic abilities using PRINT statements.

CHR$ Codes

Another way to display graphic characters, text, and colors on the screen with the 64 is by using the CHR$ function. CHR$ (pronounced "character string") gives you a character based on a code ranging from 0 to 255. Every character and color that the Commodore 64 can print is encoded this way. Most reference books, including the *Commodore 64 User's Guide*, the manual that came with your computer, include a table of CHR$ values. (See Appendix F.) To print any character, all you need do is type:

```
PRINT CHR$(N)
```

where N is a number between 0 and 255. For instance, try entering this:

```
PRINT CHR$(65)
```

You should see the A character displayed on the screen.

If you don't have a reference available which includes the CHR$ code values, you can find them yourself by using the function:

```
PRINT ASC("X")
```

where X is any key pressed. Enter:

```
PRINT ASC("A")
```

and you should see the CHR$ value, 65, displayed. This comes in handy when you are looking for the CHR$ values of the second group of eight colors. Many reference books do not include the CHR$ values for these colors, or do not list them separately. For example, if you enter:

```
PRINT ASC("{2}")
```

the number 149, the CHR$ value for the color brown, will be displayed.

Using CHR$, you can duplicate any command that could be entered from the keyboard. A display of text in varying colors, for example, would look like this, using the CHR$ function instead of the keystrokes within quotation marks:

Program 3. Textchange CHR$

```
5 PRINTCHR$(147)
10 PRINTCHR$(5)"A DEMONSTRATION"
20 PRINTCHR$(28)"OF THE COLORS"
30 PRINTCHR$(30)"THAT ARE AVAILABLE"
40 PRINTCHR$(144)"ON THE COMMODORE 64"
50 PRINTCHR$(156)"IS QUITE EASY TO DISPLAY."
60 PRINTCHR$(149)"CHANGING BACK"
70 PRINTCHR$(150)"TO THE ORIGINAL"
80 PRINTCHR$(151)"COLOR IS NOT THAT"
90 PRINTCHR$(154)"DIFFICULT"
100 GOTO 100
```

This is almost identical to Program 1, but the CHR$ code values have been used instead. CHR$(147) in line 5 is the value for clear screen. Notice that in lines 60-90, the CHR$ values are for the second group of eight colors, the ones normally printed when the Commodore key is used with a number key. Line 90 returns the color to light blue, the default color, and finally, line 100 holds the program so the READY prompt doesn't spoil the display.

Compare the two methods of changing colors demonstrated in Programs 1 and 3. Entering the CHR$ code values takes more time, more keystrokes, but produces the same result. For that reason, the use of CHR$ codes in graphics creation *is* somewhat limited. There are often easier ways of accomplishing the same thing. However, at times you'll find applications where the

CHR$ function is more useful, especially if you're experimenting with rapidly changing colors or characters.

If you want to fill the screen with characters, as well as display them in varying colors, for instance, the CHR$ function works well. Since you can assign a variable as the CHR$ value, you can create a random display much easier with this method. Program 4 is one example.

Program 4. Random CHR$

```
10 PRINT CHR$(147)
20 A=(191*(RND(9)))+34
30 IF A>129 AND A<149 THEN 20
40 PRINT CHR$(A);
50 GOTO 20
```

As this program runs, it fills the screen with random characters, as well as altering the colors of these characters. It does this by choosing a random number from 34 to 191 in line 20, which becomes the variable A. Line 40 then PRINTS CHR$(A) and the program repeats. The only exceptions are the CHR$ values between 130 and 148. Leaving these values in does strange things to the screen, which you can see by simply eliminating line 30.

Accomplishing the same thing with simple PRINT statements would take many more lines, more memory in your computer, and would run slower.

Filling the screen with random characters and colors may look interesting, but practical applications may be hard to find. Something more useful, and still operating with the CHR$ function, could be similar to the following program.

Program 5. Checkerboard

```
10 CL=158
20 PRINT CHR$(147);CHR$(CL)
30 FOR A=1 TO 11
40 FOR X= 1 TO 19
50 PRINT CHR$(18)" "CHR$(146)" ";
60 NEXT X:PRINT
70 FOR X=1 TO 19
80 PRINT CHR$(146)" "CHR$(18)" ";
90 NEXT X:PRINT
100 NEXT A
110 PRINT CHR$(154)
120 GOTO 120
```

Here's how the program works.

Line	Function
10	The variable CL is the color value used in the program. Changing this will alter the color of the checkerboard pattern.
20	The screen clears and the color is changed.
30	If you haven't used a FOR-NEXT loop before, this may look confusing. All it does is repeat something; in this case, lines 40-90 are repeated 11 times before the program ends.
40	This loop makes the next line repeat 19 times to produce one line 19 characters long.
50-60	PRINT the two characters: CHR$(18), the reverse on command, then a space; and CHR$(147), the reverse off command, along with a space.
70-90	PRINT another line, switching the order of the characters, so that a true checkerboard pattern is displayed.
110	Change color back to light blue.
120	Hold the pattern on the screen without the READY prompt.

By changing the value of CL, you can alter the color of the pattern. As in all programming, especially with graphics, the thing to remember is to experiment. The more you change things, the more you play with a method or command, the more discoveries you'll make.

POKEs

Even though the PRINT statement can be used to create a variety of graphics on the Commodore 64, there is another method that is much more versatile, and often simpler to use. That is the POKE statement.

The VIC-II chip of the Commodore 64 updates the screen display 60 times a second. You don't have to worry about it—it's done automatically. The important thing to remember is that the VIC-II chip looks at certain memory locations in order to find out what the TV or monitor display should look like. That's how your text and graphic characters are displayed on the screen when you press keys or instruct a program to run. Changing the value in a particular memory location, say the one which determines background colors, tells the VIC-II chip which colors you want. The memory location checked for the background color is 53281, while the border color is set at location 53280.

Table 2. Color POKE Values

COLOR	POKE VALUE
BLACK	0
WHITE	1
RED	2
CYAN	3
PURPLE	4
GREEN	5
BLUE	6
YELLOW	7
ORANGE	8
BROWN	9
LIGHT RED	10
GRAY 1	11
GRAY 2	12
LIGHT GREEN	13
LIGHT BLUE	14
GRAY 3	15

Besides looking at the locations for background and border color, the VIC-II chip looks at other memory locations to find out what the screen should look like. It scans an area called *screen memory* to determine which characters to display on the screen, another set of locations called *color memory*, to find the color of the characters, and yet another area, the *character set*, to see what each character should look like. It checks other locations, too, but these are the ones most important to creating graphics on the 64.

By changing what the computer finds in these locations, using the POKE statement, you can control what the screen displays.

A POKE command puts a new value into a memory location with two numbers, separated by a comma. The first number is the memory location you want to change. The second number is the *new* value you want to be stored there. Although you can POKE to any memory location between 0 and 65535, and POKE in a value from 0 to 255, there are only a few POKE commands you'll use frequently in creating graphics.

POKE 53281,0

This POKE will change the background color of the screen to black, for example. To change the screen colors, you must POKE in a value between 0 and 15. Just as when you used the SHIFT and Commodore keys to create color changes within PRINT statements, so these values change the color with a POKE statement. See Table 2 for a chart of the color values used in POKE statements.

Here's a short program which will POKE in all the combinations of background and border colors, as well as display the numbers you would enter to make that particular change. Note that the background and border colors are set by separate memory locations, unlike some other computers. The background color is found at location 53281, while the border color is at location 53280.

Program 6. Background and Border POKEs

```
10 PRINT"{CLR}"
20 FOR BR=0 TO 15
30 FOR BG=0 TO 15
40 POKE 53280,BR
50 POKE 53281,BG
60 PRINT "{HOME}{2 DOWN}{RIGHT}BORDER COLOR="BR;"
   {LEFT} {2 RIGHT}BACKGROUND COLOR="BG"{LEFT} "
70 FOR T=0 TO 1000:NEXT
80 NEXT:NEXT
```

As this program runs, you'll see the POKE values displayed. Some of the color combinations are not attractive, others are not useful to display text, but some will look good to you. If you see a particular combination you like, just hit the RUN/STOP key and check the values on the screen. If you cannot make them out, you can press RUN/STOP–RESTORE keys and then type:

PRINT BR <RETURN>
and/or
PRINT BG <RETURN>

and the last values used will be shown. (BR is the border color and BG is the background.)

When the background value is 14, or light blue, it seems as if the text has disappeared. The words and numbers are still there, but they're invisible because they are the same color as the screen. This is one way game programmers make objects appear and disappear from the screen. If you ever PRINT or POKE a character onto the screen and it doesn't show up, the first thing to do is POKE a different value into 53281 — perhaps the character is invisible because it's the same color as the screen.

POKEing onto the Screen

So far, you've created graphics using the PRINT statement, which handles data in a sequential fashion. One character is printed after the next, starting from a known place on the screen. Each PRINT statement has the proper number of cursor controls to arrange the characters on the screen, just as you saw with the pool table display earlier in this chapter. But this method takes programming time and often many steps.

An easier way to do this is to use the POKE statement to directly control each location on the screen. This is the most-often-used method of creating graphics on the Commodore 64.

Memory locations are the key to using POKEs when you create graphics on the screen. The 64's memory is a long string of addresses, one after another. One section of this is used for *screen memory*. Since the screen is able to display 1000 characters in a grid 40 columns wide by 25 rows high, there are 1000 memory locations reserved to handle what appears on the screen.

Each memory location can hold a number between 0 and 255. In other words, there are 256 possible values for each memory location. By changing the value, you change what appears on the screen. You can thus select what to display, and also where it will be displayed, on the monitor or television screen.

The VIC-II chip reads screen memory one character at a time, starting with the upper-left-hand corner, moves across the top row from left to right, and then jumps down to the leftmost character of the next row. When it reaches the last character, the bottom-right-hand corner, it returns to the top-left corner and begins again.

Screen memory on the 64 normally starts at location 1024 and ends at 2023. (See Appendix C.) The upper-left-hand corner is the *lowest* address, while the lower-right-hand corner is the *highest*. The 64 reads from left to right, top to bottom, just as you do. If you remember that, it shouldn't be too confusing.

Let's say you want to place a character in the center of the screen. The middle of the screen is column 20, row 12. To find the exact address in screen memory for this spot, multiply the row number (12) by 40, the total number of locations per row. The answer is 480. Then add 20, since you want the twenty-first character (the first character in each row is numbered 0). The total is 500, which you add to the memory address 1024, for the exact memory location of 1524. A simple formula for calculating this is:

```
SCREEN MEMORY LOCATION=1Ø24 + 4Ø*ROW + COLUMN
```

Using this, you can find the address of any of the 1000 memory locations on the screen. To place a character there, all you need to do is something like this:

```
POKE 1524,81
```

As with all POKE statements, the first number is the memory location, and the second is the new value you want placed in that location. You can place any character in a particular location by using the screen code value as the second number. Refer to the screen code table in your *Commodore 64 User's Guide* for these values. For instance, in the example above, the graphics ball character • will be displayed in the center of the screen because its screen code value is 81. To draw another character, such as the letter A, all you need do is change that value to 1. (See Appendix G for a list of screen POKE codes.) If you type this example in and run it, however, you may not see anything on the screen. For every screen memory location, there is a corresponding address in *color memory*. Instead of seeing the numbers stored there as characters, the VIC-II chip interprets the numbers as *color codes*. This means that color memory is a perfect shadow of screen memory. You can individually control the color of a character by setting the appropriate color memory location.

However, most recent 64s automatically fill color memory with the value for the background color when turned on or reset. Thus, unless you change the value in corresponding color memory when you POKE to the screen, the characters you POKE will be invisible. (This isn't a problem when using PRINT because PRINTing automatically takes care of changing the color memory.)

The color addresses begin at 55296 and continue 1000 locations to 56295, just as the screen memory ran for 1000 addresses. The VIC-II chip reads color memory in the same way it reads screen memory, from the top-left-hand corner to the bottom-right-hand corner. The only difference is the number of the memory location. To calculate color memory, a different formula is used.

```
COLOR MEMORY LOCATION=55296 + 40*ROW + COLUMN
```

Color location 55796 is the spot in the center of the screen, matching the location of the character at screen memory location 1524. To change its color, all you need do is POKE a value from 0 to 15 (the same values you used to change background and border colors) into that location. You could do it this way:

```
10 POKE 1524,81
20 POKE 55796,0
```

This will display the ball character in black at the center of the screen.

Using this method of POKEing characters and colors directly to the screen, you can create almost any graphic design you'd like. Although it may seem like a lot of typing, it is shorter than using cursor controls and several keystrokes with the PRINT statement. Most programmers use the POKE method when they create graphics on the 64.

A demonstration of the use of POKE can range from something simple to something quite elaborate. Creating a border around the screen display, for instance, is quite easy. The following program does this.

Program 7. Border

```
10 SC=1024:CL=55296:PRINT"{CLR}"
20 POKE 53281,1
30 ROW=0:FOR COLUMN=0 TO 39:GOSUB 80:NEXT
40 COLUMN=0:FOR ROW=0 TO 24:GOSUB 80:NEXT
50 ROW=24:FOR COLUMN=0 TO 39:GOSUB 80:NEXT
60 COLUMN=39:FOR ROW=0 TO 24:GOSUB 80:NEXT
70 GOTO 70
80 POKE CL+COLUMN+ROW*40,0:POKE SC+COLUMN+ROW*40,1
   02:RETURN
```

Line	Function
10	Set the values for SC and CL, the memory locations for screen and color memory respectively.
20	Change the color of the background to white.
30-60	Set the borders. The top border is set first in line 30, then the left-hand border with line 40, followed by the bottom and right-hand borders in lines 50 and 60.
70	Hold the program so that the READY prompt doesn't ruin the display.

80 POKE in the color and character value for each location around the screen.

This is only a short graphics progam, but its effect is quite dramatic. You can change the color of the border, and the character that is used for that border, simply by changing the values POKEd in line 80. Experiment with your own changes to see the differences.

Beginning Graphics

You now have an idea, though a relatively simple one, of the graphics abilities of the Commodore 64. The choices as you create graphics are numerous. You can use PRINT statements, or you can use POKE to create these graphics. You can even use the CHR$ code values to display text and characters onto the screen.

But you're still not creating those arcade game displays. Other articles in this book show you how to do that. You can see how to create your own graphics characters in Chapter 3, for instance, or how to design and use the 64's sprites in Chapter 4.

Remember that you're learning a new language, BASIC, and like any other language, it takes practice and time to become fluent. You *will* become fluent if you take that time. Your first reaction of anxiety and shock will disappear as you experiment with the computer, as you try out new ideas in your programming.

The Commodore 64 *is* a graphics machine. It only needs you.

Chapter One

Character Graphics

C. Regena

One way to put graphics on the screen is to use the built-in character set. This article will illustrate this technique. Also included here is a typing practice program.

Graphics (pictures) can be drawn with symbols found right on the keyboard of the 64. Notice that each of the keys has a symbol on the top of the key where you press. This is the symbol that is printed when you press the key. Now look at the front of the keys. Many of the keys have two symbols in squares. These are used for graphics.

Press SHIFT and a key simultaneously, and you'll get the symbol on the right side of the key. At the far left of the keyboard on the bottom row of keys is a key with the Commodore symbol, C=, called the Commodore key. Try pressing the Commodore key and a key with symbols on the front. On the screen will be the symbol at the left. For example, look at the key marked S. If you press the key, S will appear on the screen. If you press SHIFT and the S key, a heart will appear. If you press the Commodore key and the S key, ┐ will appear.

Moving the Cursor

To draw a picture on the screen, you don't even have to know how to program. First press SHIFT and CLR/HOME to clear the screen. The cursor (the blinking square that shows you where you're typing) will be in the upper-left corner of the screen. Now you can just start drawing a picture or making a design by using SHIFT or the Commodore key plus the other keys to draw the symbols you want.

The cursor naturally goes from left to right across the screen. When the cursor reaches the end of a line, it moves to the beginning of the next line. To move the cursor in a different sequence, use the cursor control keys. These two keys are at the far right of the main section of the keyboard and on the bottom row. They are marked CRSR with some arrows.

It is possible to move the cursor to the right by using either the space bar or the right CRSR key. The right CRSR key has arrows going left and right. The difference between these methods is that the space bar puts spaces as it moves and will erase anything already there, whereas the CRSR key moves without changing what is already on the screen. To move to the left press SHIFT and the same CRSR key.

Now try the key with the up and down arrows. If you press just the CRSR key, your cursor will go down. If you want to move up, press SHIFT and the CRSR key. Again, you will not erase anything you pass over.

Drawing a Picture

To put a drawing into a BASIC program, you can use PRINT statements and copy what you've done on the screen. You may prefer to sketch out your picture first on graph paper. The screen is 40 columns wide and 25 rows.

To start with a clear screen, use a statement such as

```
10 PRINT "{CLR}"
```

Type a line number, the word PRINT, the quotation mark, then simultaneously the SHIFT key and the CLR/HOME key (a symbol will be printed which means CLEAR and looks like an inverse video heart), and then the closing quotation mark.

Continue using PRINT statements with your desired symbols within the quotation marks. You may also use the CRSR control keys within the quotation marks to tell the computer to move the drawing cursor to a different position. Following is an example (refer to the listing conventions in Appendix B).

```
10 PRINT "{CLR}"
20 PRINT "A"
30 PRINT "{3 RIGHT}S"
40 PRINT "{3 RIGHT}{3 DOWN}X"
50 PRINT "Z"
60 PRINT
70 PRINT "O{3 Y}P{DOWN}{LEFT}N{DOWN}{2 LEFT}N
   {DOWN}{2 LEFT}N"
```

Line 40 indicates to press the right CRSR key three times, then the down CRSR key three times, then SHIFT and the letter X. Line 70 indicates to press SHIFT and the letter O, then the

Commodore key and the letter Y three times, the down CRSR key, the left CRSR key (which is SHIFT and the proper CRSR key), SHIFT and the letter N, down CRSR, and so forth.

If you use the CLR/HOME key without pressing SHIFT, the cursor will return "home," to the top left of the screen, but the screen will not be cleared. If you want to keep your picture on the screen without the word READY appearing, use a line such as 80 GOTO 80 to keep the program running. To stop the program, press the RUN/STOP key.

Adding Color

Now let's add some color to your graphics. Press CTRL and one of the numbers on the top row of the keyboard, then start typing. You now have a new color. The Commodore 64 has eight additional colors. To obtain each color, press the Commodore key with a numbered color key. You can use these color keys in PRINT statements in your program. As soon as you use a color key, everything printed will be that color until you change again.

Two more keys that are useful in screen graphics are the RVS keys. For RVS ON, which means any letters or graphics characters will be reversed, press CTRL and 9. For example, press SHIFT and Q. You will see a colored-in circle. Now press CTRL and 9 for RVS ON, then press SHIFT and Q. The circle is now the background color of blue, and around the circle is the printing color of light blue. To change back to normal, press CTRL and the 0 (zero) for RVS OFF. The listing conventions in PRINT statements are {RVS} and {OFF}.

To get colored bars you can use the RVS ON and then the space bar. Take a look at the asterisk key. The symbol on the left is obtained by pressing the Commodore key and the *. Suppose you want the bottom triangle printed instead of the top triangle, yet in that position and not the position on the British pound key. Press RVS ON then the Commodore key and *.

Program 1 will show how a graph can be drawn from data to make statistics look more interesting. This program illustrates the use of the color keys, the RVS ON and RVS OFF keys.

The TAB function is used with a PRINT statement to start printing at a certain column number. This function is similar to pressing the right CRSR several times. PRINT TAB(10); "X" would mean to print the letter X in column 10.

Chapter One

POKEing Graphics

Besides PRINTing graphics on the screen, you can use the POKE command to put graphics on the screen. Use the "Screen Location Table" (Appendix C) to POKE a certain screen location with a character number from the "Screen Codes" (Appendix G).

Notice that the Screen Location Table contains numbers from 1024 to 2023. Let's say you want to put an asterisk, *, in column 10 and the third row down. According to the table, the row starts with the number 1104. Add 10 to get to the right column, and the location is number 1114. Now looking at the Screen Codes chart, the asterisk in the Set 1 column corresponds to the number 42 in the POKE column. The BASIC command would be POKE 1114,42. Now to add a color for that character, you may either use the Screen Color Memory Table (Appendix D) or simply add 54272 to the Screen Location Table number. Choose a color number from 0 to 15. The command for a red asterisk would be POKE 55386,2. (For a more complete explanation see "POKEing Graphics," the next article in this book.)

To see how fast a circle can zip across the screen using the POKE method, try this program:

```
10 FOR L=1824 TO 1903
20 POKE L,87:POKE L+54272,7
30 POKE L,32
40 NEXT L
```

The FOR-NEXT loop changes L from 1824 to 1903 and is the screen memory location. Line 20 POKEs a circle in the location, then sets the color of the circle to yellow. Line 30 erases the circle by putting a space (character 32) in the location. As the loop index increments, the location changes by one square.

An advantage to POKEing graphics is that you can specify the exact location. When PRINTing graphics you need to know where the previous PRINT statement left the cursor or where the next PRINT statement will be. When you are drawing graphics in a certain order, you may want to PRINT part of the picture and POKE the graphics among locations within the printed picture.

A Real Example

Program 2, which teaches the home position of touch-typing, illustrates how graphics can enhance an educational program. The hands are drawn using PRINT statements and the graphic symbols on the keys. The letters above the fingers are POKEd into the locations. Several of the PRINT statements illustrate the use of the CRSR control keys to position the words. The TAB function is used in several places rather than using the right CRSR key to move over several columns.

Typing Program Explanation

Line Numbers	
2	Change screen color to white.
3	POKE commands initialize music registers. Variables F1, F2, and W are defined for use later in music commands.
4	Define string variables for printing graphics.
6-8	Read from DATA statements the following subscripted variables: P(I), screen location for POKEing letter above correct finger; P$(I), letter name; L(I), code number to POKE letter or symbol on screen; S(I) and T(I), numbers for sound statements.
9	Branch to main program past subroutines.
10-150	Subroutine to clear screen and draw hands.
200-220	Subroutine to detect which key is pressed and to see if it is the correct key. If the right letter or symbol has been pressed, it is replaced on the screen by a space (erased); otherwise, the computer waits for the right key to be pressed.
400-480	Print title screen and wait for user to press a key.
500	Call subroutine to draw hands.
510-560	Play a tone and print a letter above each finger.
570-610	Print instructions for first drill.
620-640	Erase letters above fingers.
650-710	Present drill to type letters. Three times the letters are presented in order from left to right. A tone sounds, and the letter or symbol is printed above the corresponding finger. Line 690 calls the subroutine to detect when a key is pressed. The right key must be pressed to continue.
720-780	Choose letters randomly.
790-820	Print option to repeat drill or continue program and branch accordingly.
830	Clear screen.
840	Restore data in case the drill is being repeated.

850	Read the first 40 items which have previously been used and are not used for this drill.
860-880	Read from data nine words and phrases in the A$ array for use in the drill.
890-1220	Perform the drill until five phrases are typed correctly.
900-910	Print instructions.
920	Randomly choose a phrase. If the phrase has been typed correctly, it is set to "" (null) and another phrase must be chosen.
930	Initialize the B$ string variable and print the phrase to be copied.
940	Position printing for user's typing.
950-1000	Print the key pressed or branch out of the loop (if RETURN is pressed). B$ contains what the user has typed.
1010	Compare the typed phrase with the given phrase.
1020-1100	If the answer is incorrect, play uh-oh and print WRONG, then wait for user to press RETURN for another phrase.
1110	If the answer is correct, print a red heart. The number of red hearts is the number of correct phrases.
1120-1200	Play arpeggio for correct answer.
1210-1220	Set A$ phrase to "" (null) so it won't be chosen again; return for next phrase.
1230-1280	Print option to repeat the letters drill or the phrases drill or to end the program and branch appropriately.
1290-1300	Clear screen and end.

Program 1. Graph

```
10 PRINT"{CLR}{WHT}"
20 PRINT TAB(15);"POPULATION"
30 PRINT TAB(16);"{DOWN}{RVS}{YEL}{2 SPACES}{OFF}
   {WHT} 1970"
40 PRINT TAB(16);"{RVS}{RED}{2 SPACES}{OFF}{WHT} 1
   980{DOWN}"
50 FOR C=1 TO 5
60 READ S$,P1,P2
70 PRINT"{DOWN}";S$;TAB(10);
80 FOR I=1 TO INT(P1/75000+.5)
90 PRINT "{RVS}{YEL} ";
100 NEXT I
110 PRINT "{OFF}{WHT}";TAB(38-LEN(STR$(P1)));P1
```

```
120 PRINT TAB(10);
130 FOR I=1 TO INT(P2/75000+.5)
140 PRINT "{RVS}{RED} ";
150 NEXT I
160 PRINT "{OFF}{WHT}";
170 PRINT TAB(38-LEN(STR$(P2)));P2
180 NEXT C
190 DATA NEVADA,488738,799184,UTAH,1059273,1461037
    ,WYOMING,332416,470816
200 DATA IDAHO,713015,943935,MONTANA,694409,786690
210 GOTO 210
220 END
```

Program 2. Typing

```
2 POKE 53281,1
3 POKE 54296,15:POKE 54277,8:POKE 54278,8:F1=54273
  :F2=54272:W=54276
4 F$="U*I":G$="B -":H$=G$+G$+G$
6 FOR I=1 TO 8:READ P(I),P$(I),L(I),S(I),T(I):NEXT
7 DATA 1467,A,1,34,75,1390,S,19,38,126,1353,D,4,43
  ,52,1396,F,6,45,198
8 DATA 1411,J,10,51,97,1374,K,11,57,172,1417,L,12,
  64,188,1500,":",58,68,149
9 GOTO 400
10 PRINT"{CLR}{10 DOWN}{RED}"
20 PRINTTAB(8);F$;TAB(29);F$
30 PRINTTAB(5);F$;G$;F$;TAB(26);F$;G$;F$
40 PRINTTAB(5);H$;TAB(26);H$
50 PRINTTAB(5);H$;TAB(26);H$
60 PRINT"{2 SPACES}";F$;H$;TAB(26);H$;F$
70 FOR I=1 TO 3
80 PRINT"{2 SPACES}";G$;H$;TAB(26);H$;G$
90 NEXT I
100 PRINT"{2 SPACES}B JK JK JK - {RVS}{BLK}SPACE B
    AR {OFF}{RED} B JK JK JK -"
110 PRINT"{2 SPACES}B";TAB(13);"-";TAB(26);"B";TAB
    (37);"-"
120 PRINT"{2 SPACES}B";TAB(13);"- N{Y}P
    {4 SPACES}O{Y}M B";TAB(37);"-"
130 PRINT"{2 SPACES}B";TAB(13);"-N{2 SPACES}{M}
    {4 SPACES}{G}{2 SPACES}MB";TAB(37);"-"
140 PRINT"{2 SPACES}B";TAB(17);"N{4 SPACES}M";TAB(
    37);"-"
150 RETURN
200 GET E$:IF E$<>P$(J) THEN 200
210 POKE P(J),32
220 RETURN
400 PRINT "{CLR}"
```

```
410 PRINT TAB(14);"{3 DOWN}T Y P I N G"
420 PRINT TAB(14);"{2 DOWN}U N I T{3 SPACES}1"
430 PRINT TAB(13);"{2 DOWN}HOME POSITION"
440 PRINT "{6 DOWN}YOU WILL SEE A DIAGRAM OF THE H
    ANDS."
450 PRINT "PLACE YOUR FINGERS ON THE KEYS AS SHOWN
    ."
460 PRINT "{2 DOWN}PRESS <RETURN> TO START."
470 GET E$:IF E$=""THEN 470
480 IF ASC(E$)<>13 THEN 470
500 GOSUB 10
510 FOR I=1 TO 8
520 POKE F1,S(I):POKE F2,T(I):POKE W,17
530 POKE P(I),L(I):POKE P(I)+54272,6
540 FOR D=1 TO 100:NEXT
550 POKE F1,0:POKE F2,0:POKE W,0
560 NEXT I
570 PRINT "{HOME}PLACE YOUR FINGERS IN POSITION."
580 PRINT "{DOWN}PRESS ANY KEY TO CONTINUE."
590 GET E$:IF E$=""THEN 590
600 PRINT "{HOME}TYPE EACH LETTER AS IT APPEARS."
610 PRINT "{DOWN}{26 SPACES}"
620 FOR I=1 TO 8
630 POKE P(I),32
640 NEXT I
650 FOR I=1 TO 3
660 FOR J=1 TO 8
670 POKE F1,S(J):POKE F2,T(J):POKE W,17
680 POKE P(J),L(J)
690 GOSUB 200
700 POKE W,0
710 NEXT J,I
720 FOR I=1 TO 30
730 J=INT(RND(0)*8)+1:IF J=K THEN 730
740 K=J:POKE F1,S(J):POKE F2,T(J):POKE W,17
750 POKE P(J),L(J)
760 GOSUB 200
770 POKE W,0
780 NEXT I
790 PRINT "{HOME}CHOOSE:{2 SPACES}1 TRY AGAIN
    {12 SPACES}"
800 PRINT TAB(9);"2 CONTINUE PROGRAM"
810 GET E$:IF E$="1" THEN 500
820 IF E$<>"2" THEN 810
830 PRINT "{CLR}"
840 RESTORE
850 FOR I=1 TO 40:READ E$:NEXT
860 DATA "A SAD LAD:","A FAD:","ASK A LAD:",A SAD
    {SPACE}FAD,A LAD ASKS DAD
```

```
870 DATA "ALFALFA:",ALAS A SAD DAD,"DAD ASKS A LAD
    :","ASK DAD:"
880 FOR I=1 TO 9:READ A$(I):NEXT
890 FOR I=1 TO 5
900 PRINT "{CLR}TYPE THE PHRASE SHOWN"
910 PRINT "THEN PRESS <RETURN>.{8 DOWN}"
920 J=INT(9*RND(0))+1:IF A$(J)=""THEN 920
930 B$="":PRINT TAB(14);A$(J)
940 PRINT TAB(14);
950 FOR K=1 TO 20
960 GET E$:IF E$=""THEN 960
970 IF ASC(E$)=13 THEN 1010
980 PRINT E$;
990 B$=B$+E$
1000 NEXT K
1010 IF B$=A$(J) THEN 1110
1020 POKE F1,43:POKE F2,52:POKE W,17
1030 FOR D=1 TO 100:NEXT
1040 POKE F1,34:POKE F2,75:POKE W,17
1050 FOR D=1 TO 100:NEXT:POKE W,0
1060 PRINT:PRINT"{3 DOWN}{4 RIGHT}WRONG"
1070 PRINT"{DOWN}{4 RIGHT}PRESS <RETURN>"
1080 GET E$:IF E$=""THEN 1080
1090 IF ASC(E$)<>13 THEN 1080
1100 GOTO 900
1110 FOR D=1 TO I:POKE 1600+D,83:POKE 1600+D+54272
     ,2:NEXT
1120 POKE F1,34:POKE F2,75:POKE W,17
1130 FOR D=1 TO 100:NEXT:POKE W,0
1140 POKE F1,43:POKE F2,52:POKE W,17
1150 FOR D=1 TO 100:NEXT:POKE W,0
1160 POKE F1,51:POKE F2,97:POKE W,17
1170 FOR D=1 TO 100:NEXT:POKE W,0
1180 POKE F1,68:POKE F2,149:POKE W,17
1190 FOR D=1 TO 300:NEXT
1200 POKE W,0
1210 A$(J)=""
1220 NEXT I
1230 PRINT:PRINT "{5 DOWN}CHOOSE:{2 SPACES}1 PRACT
     ICE LETTERS"
1240 PRINT TAB(9);"2 PRACTICE WORDS"
1250 PRINT TAB(9);"3 END PROGRAM"
1260 GET E$:IF E$="1" THEN 500
1270 IF E$="2" THEN 830
1280 IF E$<>"3" THEN 1260
1290 PRINT "{CLR}"
1300 END
```

Chapter One

POKEing Graphics

C. Regena

Graphics can be POKEd to the screen as well as PRINTed. The POKE method is especially useful for animation.

The format for the POKE command is POKE n1,n2 where n1 is a memory address and n2 is a numeric value. Try POKE 53280,n2 to change the border color, and POKE 53281,n2 to change the screen color, where n2 is any number from 0 to 15.

Let's try a few:

POKE 53281,12
POKE 53280,1

To get back to normal, just press RUN/STOP and RESTORE, or type POKE 53280,14 and POKE 53281,6.

Here is a program to see all the combinations:

```
10 FOR I=0 TO 15
15 POKE 53281,I: REM SET SCREEN COLOR
20 FOR J=0 TO 15
30 POKE 53280,J: REM SET BORDER COLOR
40 FOR D=1 TO 200: NEXT D
50 NEXT J,I
```

Simple Graphics

Now let's put some graphics on the screen. Turn to Appendix C.

The block represents the screen of 25 rows by 40 columns. Each location number is obtained by adding the row and column numbers. This is the n1 number you need for the POKE location. For example, to POKE to row 10, column 4, we would use an n1 of 1384 + 4 = 1388.

Refer to Appendix G for a chart of character codes for the n2 number in the POKE command. Look under the SET1 column heading for a symbol you want to print. Find the corresponding number in the POKE column. For example, to draw a spade, the number is 65.

You now have the parameters for a POKE command in graphics. Let's put a spade in row 10, column 4. We know that the command is POKE 1388,65.

The only problem is that when you draw graphics this way, you won't be able to see them (except on early model 64s). This is because the graphics character you POKEd in is the same color as the screen background, which makes the character impossible to see. One solution is to change the screen color after POKEing in the graphics.

For example:

```
10 PRINT"{CLR}"
20 POKE 1388,65  : REM DRAWS WHITE SPADE
30 POKE 53281,2  : REM CHANGES SCREEN COLO
   R TO RED
40 GOTO 40
```

Press the RUN/STOP key to stop the program. Press RUN/STOP and RESTORE at the same time to return to the "normal" screen colors.

Changing Colors

Suppose you like your regular colored screen and want to draw graphics. You can change the color of your character by POKEing a memory location with a color code. Refer to Appendix D this time. You will find a color codes memory map. Each screen location has a number (obtained by adding the row and column numbers shown) for keeping track of color; this will be our n1 number for our color POKE. The color codes are listed in Appendix E. This color code will be our n2 number for our color POKE.

For example, let's use our same spade on row 10, column 4. Find the color memory number corresponding with screen location 1388. Counting ten rows down, you should see a 55656. Adding 4 we get 55660. Note that the difference between corresponding screen and color locations will always be 54272.

So, to put a red spade on the screen, we could use this program:

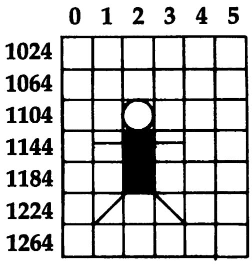
```
10 PRINT"{CLR}"
20 POKE 1388,65
30 POKE 55660,2
```

You can flash an object by changing the color codes. Try the following program:

```
10 PRINT"{CLR}"
20 POKE 1388,65
25 FOR C=1 TO 20
26 POKE 55660,6
27 FOR D=1 TO 100:NEXT D
28 POKE 55660,1
29 FOR D=1 TO 100:NEXT D
35 NEXT C
```

You are now ready to sketch a design of your own and then POKE values to draw your picture. Here is a sample program:

```
5 POKE 53281,1: REM WHITE SCREEN
10 PRINT"{CLR}"
12 L=54272
14 POKE 1106,87:POKE 1106+L,2
16 POKE 1146,102:POKE 1146+L,6
18 POKE 1186,102:POKE 1186+L,6
20 POKE 1145,64:POKE 1145+L,6
22 POKE 1147,64:POKE 1147+L,6
24 POKE 1225,78:POKE 1225+L,6
26 POKE 1227,77:POKE 1227+L,6
28 GOTO 28
```



To try animation, change the graphics by POKEing different characters or by drawing and erasing characters to move the graphics. Change the above program by adding the following lines — can our guy fly?

```
28 FORI=1 TO 50
30 POKE 1145,99
32 POKE 1147,99
34 POKE 1145,64
36 POKE 1147,64
38 NEXTI
40 GOTO40
```

The Character Sets

Two character sets are available for graphics, but only one set can be on the screen at a time. You probably have discovered that if you have some printing on the screen and you press the Commodore key and the SHIFT key at the same time, all capital letters change to lowercase letters. The first condition is Character Set 1, and the second condition is Character Set 2.

Before you start drawing your graphics, POKE 53272,23 will put you in Set 2, and POKE 53272,21 will put you back in Set 1. Note that the values to do this that were listed on page 132 of the original versions of 64 manual were not correct.

Reverse characters are also available. The reverse of any character on the chart is calculated by adding 128 to the number in the chart.

You can use the PEEK command to see what character is in a particular location or what the color is. You can use the PEEK command to detect a barrier or to detect a crash in a game. PEEK(n) will return the value in memory location n. Some valid commands are:

```
PRINT PEEK(7911)
200 IF PEEK(A)=32 THEN 350
```

At first, PEEK doesn't seem to work with color memory, since when you PEEK it, you get a different number than you POKEd in. To fix this just use:

```
X=PEEK(n) AND 15
```

instead of:

```
X=PEEK(n)
```

You only have to do this when n is in color memory.

To further demonstrate POKEing graphics, let's look at a couple of sample programs. In Program 1, I and J are coordinates to determine the location of the ball. The ball bounces within the boundaries.

Graphics in a Game

Program 2 illustrates how you can POKE graphics and create moving graphics for a game. "Defend" is a shooting game for one person. You are positioned on the left of the screen and need to defend your territory — don't let the invader coming from the right of the screen get to your border.

Line up horizontally with an invader by pressing ↑ to move up and CRSR ↓ to go down, then shoot by pressing either the space bar or the f7 key. You score ten points for each invader you successfully shoot, but you lose five points if you miss.

After you have played this game once or twice, change it into your own game. Use different graphics and colors. Change the motion to vertical instead of horizontal. Change the scoring. After you reach certain scores, perhaps you could change the shapes of the invaders and vary their speed.

Chapter One

Program Description

Lines	Explanation
1	Initialize TS for the top score and O for color memory offset.
2	Define function R(X) to calculate the location number for a random row; branch to line 200.
10	Clear screen; set screen and border color. Initialize variables. N is the location of your ship, SC is the score, and D is difficulty level.
20	Place defending ship on screen.
22-25	Randomly place invaders, making sure invaders are not on the same row as the player.
30	Detect which key is pressed. If it is one of the firing keys, branch to line 60.
32-34	If arrow keys are pressed, move up or down.
35	Increment L to determine speed of invaders.
36	Increment invaders' positions; move one spot to the left.
37-42	If an invader reaches left side of screen, branch to line 100 to end game.
44-50	Move invaders; branch back to receive next key press.
62-68	Check positions of invaders to see if one was shot.
70	Decrease score by five if shot missed.
72-78	Procedure if invader is shot; choose new invader position.
80	Increase score by ten; clear invader.
82-84	Print score and branch back for next key press.
90-94	Check boundary position of defender, then draw defender on screen in new position.
100-110	Procedure at game's end.
120-160	Print ending message, score, and high score.
170-190	Print option to try again and branch appropriately.
200-280	Print instruction screen.
290	END.

Program 1. Bouncing Ball

```
5 POKE 53281,1:POKE 53280,12
10 PRINT"{CLR}{BLU}"
20 PRINT "PRESS {GRN}RETURN{BLU} TO STOP
   {2 SPACES}THE BOUNCING BALL"
```

```
30 PRINT "{3 DOWN}{GRN}[40 +]"
40 I=1:J=14:DI=1:DJ=1
50 POKE 1024+I+40*J,81
60 POKE 55296+I+40*J,2
70 POKE 1024+I+40*J,32
80 I=I+DI:IF I=0 OR I=39 THEN DI=-DI
90 J=J+DJ:IF J=6 OR J=24 THEN DJ=-DJ
110 GET A$:IF A$=""THEN 50
120 IF ASC(A$)<>13 THEN 50
130 PRINT "{CLR}{BLU}"
140 END
```

Program 2. Defend

```
1 TS=0:O=54272
2 DEF FNR(X)=1144+40*(INT(RND(0)*20)):GOTO200
3 IFA$=CHR$(17)THENPOKEN,32:N=N+40
10 PRINT"{CLR}":POKE53281,12:N=1464:SC=0:D=5
15 PRINT"{HOME}[5]{RVS}{40 SPACES}{OFF}":PRINT"
   {HOME}{WHT}SCORE =";SC
20 POKEN,90
22 I=FNR(X):J=FNR(X):K=FNR(X):H=FNR(X)
24 IFH=IORH=JORH=KORI=JORJ=K THEN 22
25 POKEH,42:POKEI,42:POKEJ,42:POKEK,42
30 GETA$:IF A$=CHR$(136)OR A$=CHR$(32) THEN 60
32 IFA$=CHR$(94)THEN POKEN,32:N=N-40:GOTO90
34 IFA$=CHR$(17)THENPOKEN,32:N=N+40:GOTO90
35 L=L+1:IFL<D THEN30
36 H=H-1:I=I-1:J=J-1:K=K-1:L=0
37 IF(H-1024)/40=INT((H-1024)/40)THEN100
38 IF(I-1024)/40=INT((I-1024)/40)THEN100
40 IF(J-1024)/40=INT((J-1024)/40)THEN100
42 IF(K-1024)/40=INT((K-1024)/40)THEN100
44 POKE H+1,32:POKEI+1,32:POKEJ+1,32:POKEK+1,32:PO
   KEH,42:POKEI,42:POKEJ,42
45 POKEK,42:POKEH+O,2:POKEI+O,2:POKEJ+O,2:POKEK+O,
   2
50 GOTO30
60 FORM=200TO220:POKEN+O,1:POKEN+O,2:NEXT
62 IFH>N AND H<N+40 THEN 72
64 IFI>N AND I<N+40 THEN 74
66 IFJ>N AND J<N+40 THEN 76
68 IFK>N AND K<N+40 THEN 78
70 SC=SC-5:GOTO82
72 POKEH,102:B=H:H=FNR(X):GOTO80
74 POKEI,102:B=I:I=FNR(X):GOTO80
76 POKEJ,102:B=J:J=FNR(X):GOTO80
78 POKEK,102:B=K:K=FNR(X)
80 SC=SC+10:POKEB,32
```

```
82 PRINT"{HOME}E5E{RVS}{40 SPACES}{OFF}":PRINT"
   {HOME}{WHT}SCORE =";SC
83 IF SC>500 THEN D=0
84 GOTO30
90 IF N<1104 THEN N=1104
92 IF N>1984 THEN N=1984
94 POKEN,90:POKEN+O,0:GOTO30
100 FORC=55377 TO 56257STEP40:POKEC,2:NEXTC:FORC=1
     TO 100:NEXTC
110 FORC=55377 TO 55327STEP40:POKEC,1:NEXTC
120 PRINT"{WHT}GAME OVER"
130 FORC=1 TO 1000:NEXT:POKE53281,0:POKE53280,14
140 PRINT"{CLR}{YEL}{2 DOWN}YOUR FINAL SCORE WAS
    {3 SPACES}":PRINT"{CYN}";SC:PRINT"{YEL}
    {2 DOWN}"
150 IF SC>TS THEN TS=SC
160 PRINT"HIGH SCORE = ";TS
170 PRINT"{WHT}{3 DOWN}TRY AGAIN? (Y/N)"
180 GETA$:IF A$="Y" THEN 10
185 IF A$="N" THEN END
190 GOTO 180
200 POKE53281,12:PRINT"{CLR}{BLK}":PRINTTAB(5);"**
     DEFEND **{2 DOWN}"
210 PRINTTAB(6);"BY REGENA"
220 PRINT"{2 DOWN}PRESS ↑ TO MOVE UP":PRINT"PRESS
    {SPACE}CRSR DOWN TO GO DOWN"
230 PRINT"{DOWN}PRESS F7 OR SPACE":PRINT"TO FIRE.
    {3 DOWN}"
240 PRINT"KEEP THE INVADERS FROM"
250 PRINT"{2 DOWN}{WHT}PRESS RETURN TO START";
260 GETA$:IF A$="" THEN 260
270 IF ASC(A$)=13 THEN 10
280 GOTO260
290 END
```

Chapter One

Hi-Res Graphics Made Simple

Paul F. Schatz

One of the Commodore 64's intriguing features is a high-resolution graphics mode, *which divides the screen into 64,000 dots, or* pixels. *By turning these pixels on and off, you can create finely detailed pictures and charts. But because BASIC lacks special graphics commands, only more advanced programmers could use this mode — until now. This article is a breakthrough in that it shows how to add simple graphics commands to BASIC which anyone can use.*

Although the high-resolution graphics potential of the Commodore 64 is outstanding, accessing and plotting on the hi-res bitmap (320- by 200-pixel resolution) is inefficient and cumbersome from BASIC.

First, BASIC subroutines for calculating and turning on a specific bit can be confusing and intimidating, especially to novice programmers, since the routines require PEEKs, POKEs, ANDs, and ORs. Second, the routines are slow; many BASIC commands need to be interpreted and executed to plot one point. Third, the bitmap has to be located in memory otherwise used by BASIC. The BASIC program space is limited since it is chopped up and some areas are unusable for BASIC programs.

One solution to all of the above shortcomings is to add some new commands to BASIC which drive the high-resolution graphics. This article will describe a method for adding four commands.

Modifying BASIC

Since there is Random Access Memory (RAM) under the BASIC Read Only Memory (ROM), we can copy an image of BASIC into RAM and then modify it to suit our needs. I have modified BASIC by substituting four new commands, HUE, PLOT, WIPE, and SCREEN, in place of four seldom-used commands, LET, WAIT, CONT, and VERIFY.

Briefly, here's how the new commands were added to BASIC. First, notice that the new keywords are the same length as the keywords they replace. A new keyword has to be mapped exactly into an old keyword's spot in the keyword lookup table. Next, the pointers to the old BASIC routines are changed to point to the routines for the new keywords. Finally, the error message routine is modified so the computer switches to the normal character display if an error is encountered during execution of a program.

A Note to Programmers

The graphing routines were developed with an eye to giving up as little of the BASIC program memory as possible. Not a byte has been lost. This was accomplished by using the RAM memory under the Kernal ROM for the bitmap. Bitmap plotting at this location can only be done properly using machine language routines, since the interrupts have to be turned off and the Kernal ROM switched out to PEEK at the RAM memory. The video matrix, used for the background and foreground color nybbles, is located at $C000 and the machine language graphing routines extend from $C400 to $C545.

The New Commands

The four new commands, SCREEN, HUE, WIPE, and PLOT, are explained below.

• **SCREEN <number>**

This statement turns on and off the high-resolution bitmap. If the number is 1, the bitmap is displayed. If the number is 0, the normal character screen is displayed. Any value other than 1 or 0 will give an ILLEGAL QUANTITY ERROR.

• **HUE <number>, <number>**

This statement determines the colors displayed on the bitmap. The first number defines the foreground color (color displayed for bits set to 1). The second number defines the background color. A number 16 or greater will give an ILLEGAL QUANTITY ERROR. The color codes are:

0 Black	**4 Purple**	**8 Orange**	**12 Gray2**
1 White	**5 Green**	**9 Brown**	**13 Light Green**
2 Red	**6 Blue**	**10 Light Red**	**14 Light Blue**
3 Cyan	**7 Yellow**	**11 Gray1**	**15 Gray3**

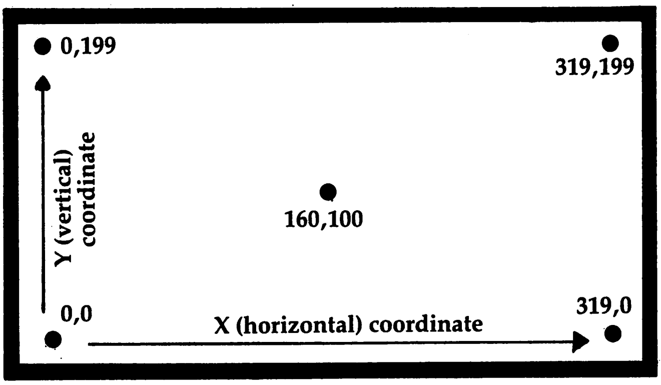
• **WIPE**

This statement causes a high-speed clear of the bitmap. All the bits are set to zero and the screen is cleared.

• **PLOT <number>, < number >**

This statement sets a bit on the bitmap, causing the corresponding pixel on the screen to be displayed in the foreground color. A coordinate system with an origin (0,0) at the lower-left corner is used (see the figure). The first number is the horizontal position relative to the origin, and the second number is the vertical position relative to the origin. The first number can have values from 0 to 319, and the second number can have values from 0 to 199. Numbers outside these ranges give an ILLEGAL QUANTITY ERROR.

Coordinates for PLOT



PLOT X,Y

Loading in the New BASIC

The new BASIC is loaded by entering and running Program 1. When entering the program, be accurate, since an incorrect number may cause the computer to crash (forcing you to switch it off and on to clear it). To be safe, SAVE the program before running it for the first time. A checksum is included to warn if there is a mistake somewhere in the DATA statements. It will take the computer a minute or two to run the program. To enable the new BASIC, enter:

POKE 1,54

The new BASIC can be disabled by pressing the RUN/STOP and RESTORE keys simultaneously, by loading a program, or by entering:

POKE 1,55

When entering programs using the new graphics commands, the new BASIC must be enabled so the tokenizing routine will recognize them. The commands they replaced will no longer work unless the new BASIC is disabled.

Some Simple Programs

We are now ready to enter and run a couple of simple programs using the new BASIC. First, a simple sine wave. LOAD and RUN the new BASIC, type NEW, switch on the new BASIC, and enter Program 2.

Now type RUN and watch the sine wave appear. Wasn't that easy? Compare this program with the one in the *Commodore 64 Programmer's Reference Guide* (pp. 122-26) for ease of programming and speed of execution.

Now, how about a joystick-driven doodle pad? Be sure Program 2 is saved. Then type NEW and enter Program 3. Plug a joystick into port two and use it to draw on the screen. Hit SHIFT-CLR/HOME to clear the screen or f7 to exit the program.

Only the Beginning

Programs written with the new BASIC can be loaded and saved in the normal fashion (but remember, we did away with VERIFY). My purpose was to provide a useful rudimentary graphing tool and to demonstrate the ease with which BASIC can be modified to include new commands. There are numerous extensions of both aspects which could be implemented. For example, a high-speed line drawing command, LINE; or a new command similar to the ON-GOTO statement but with the branching determined by the joystick position, that is, JOYGOTO, or JOYGOSUB. . . .

Program 1. New BASIC

```
Ø REM BASIC HI-RES
1Ø A=Ø:REM INTIALIZE CHECKSUM
2Ø REM MOVE BASIC ROM TO RAM
3Ø FORI=4Ø96ØTO49151:POKEI,PEEK(I):NEXTI
4Ø REM CHANGE LET TO HUE
5Ø FORI=4115ØTO41152:READN:POKEI,N:A=A+N:NEXTI
```

```
60 READL,H:POKE40988,L:POKE40989,H:A=A+L+H
70 DATA 72, 85, 197, 75, 196
80 REM CHANGE WAIT TO PLOT
90 FOR I=41189TO41192:READN:POKEI,N:A=A+N:NEXTI
100 READL,H:POKE41008,L:POKE41009,H:A=A+L+H
110 DATA 80, 76, 79, 212, 130, 196
120 REM CHANGE CONT TO WIPE
130 FORI=41225TO41228:READN:POKEI,N:A=A+N:NEXTI
140 READL,H:POKE41024,L:POKE41025,H:A=A+L+H
150 DATA 87, 73, 80, 197, 53, 196
160 REM CHANGE VERIFY TO SCREEN
170 FORI=41201TO41206:READN:POKEI,N:A=A+N:NEXTI
180 READL,H:POKE41014,L:POKE41015,H:A=A+L+H
190 DATA 83,67,82,69,69,206,11,196
200 REM CHANGE ERROR MESSAGE ROUTINE
210 FORI=42042TO42044:READN:POKEI,N:A=A+N:NEXTI
220 DATA 76, 0, 196
230 REM READ IN NEW ROUTINES
240 FORI=50176TO50480:READN:POKEI,N:A=A+N:NEXTI
250 IFA<>39040THENPRINT"ERROR IN DATA STATEMENTS"
260 END
300 DATA 32, 24,196,138, 10,170, 76, 61,164, 80, 7
    0, 83, 32,158,183,224, 1
310 DATA144, 5,240, 19, 76, 72,178,169, 27,141, 17
    ,208,169, 21,141, 24,208
320 DATA169,151,141, 0,221, 96,169, 59,141, 17,208
    ,169, 8,141, 24,208,169
330 DATA148,208,238,162, 32,169,224,133,252,160, 0
    ,132,251,152,145,251,200
340 DATA208,251,230,252,202,208,246, 96, 32,123,19
    6,138, 10, 10, 10, 10,133
350 DATA 2, 32,253,174, 32,123,196,138, 5, 2,160,1
    92,132,252,160, 0,132
360 DATA251,162, 2,145,251,200,208,251,230,252,202
    , 16,246,145,251,200,192
370 DATA232,144,249, 96, 32,158,183,224, 16,176, 1
    7, 96, 32,235,183,134, 2
380 DATA169,199, 56,229, 2,133, 2,201,200,144, 3,
    {SPACE}76, 72,178,165, 21,240
390 DATA 10,201, 1,208,245,165, 20,201, 64,176,239
    ,169, 0,133,251,169,224
400 DATA133,252,165, 20, 41,248, 24,101,251,133,25
    1,165, 21,101,252,133,252
410 DATA165, 2, 41, 7, 24,101,251,133,251,144, 2,2
    30,252,165, 2, 74, 74
420 DATA 74, 10,170,189,247,196, 24,101,251,133,25
    1,189,248,196,101,252,133
430 DATA252,165, 20, 41, 7,170,160, 0,120,169, 52,
    133, 1,177,251, 29, 41
```

```
440 DATA197,145,251,169, 54,133, 1, 88, 96, Ø, Ø,
    {SPACE}64, 1,128, 2,192, 3
450 DATA Ø, 5, 64, 6,128, 7,192, 8, Ø, 1Ø, 64, 11,
    128, 12,192, 13, Ø
460 DATA 15, 64, 16,128, 17,192, 18, Ø, 2Ø, 64, 21
    ,128, 22,192, 23, Ø, 25
470 DATA 64, 26,128, 27,192, 28, Ø, 3Ø,128, 64, 32
    , 16, 8, 4, 2, 1
```

Program 2. A Simple Sine Wave

```
1Ø SCREEN 1: REM TURN ON BITMAP
2Ø WIPE: REM CLEAR BITMAP
3Ø HUE Ø,1: REM BLACK DOTS, WHITE SCREEN
4Ø FOR X=Ø TO 319 STEP .5
5Ø Y=INT(9Ø+8Ø*SIN(X/1Ø))
6Ø PLOT X,Y: REM PLOT POINT
7Ø NEXT X
8Ø GET A$: IF A$="" THEN 8Ø: REM WAIT FOR KEYSTROK
   E
9Ø SCREEN Ø: REM NORMAL SCREEN
```

Program 3. Doodle Pad

```
1Ø SCREEN 1: WIPE: HUE Ø,1
2Ø X=159: Y=99: PLOT X,Y
3Ø GOSUB 1ØØ: IF J=15 THEN 3Ø
4Ø PLOT X,Y: GOTO 3Ø
5Ø SCREEN Ø: END: REM GRACEFUL EXIT
1ØØ REM READ JOYSTICK
11Ø J=PEEK(5632Ø) AND 15: REM PORT 2
12Ø IF (J AND 8)=Ø THEN X=X+1: REM MOVE RIGHT
13Ø IF (J AND 4)=Ø THEN X=X-1: REM MOVE LEFT
14Ø IF (J AND 2)=Ø THEN Y=Y-1: REM MOVE DOWN
15Ø IF (J AND 1)=Ø THEN Y=Y+1: REM MOVE UP
16Ø IF Y<Ø THEN Y=Ø: REM STAY IN RANGE
17Ø IF Y>199 THEN Y=199
18Ø IF X>319 THEN X=319
19Ø IF X<Ø THEN X=Ø
2ØØ GET A$:IF A$=CHR$(147) THEN WIPE: REM CLEAR SC
    REEN
21Ø IF A$=CHR$(136) THEN 5Ø: REM F7 KEY TO EXIT
22Ø RETURN
```

Chapter Two

Graphic Modes

Graphics Memory

Sheldon Leemon

Understanding how the Commodore 64 memory is organized and used is essential to understanding the best place to locate graphics data.

Commodore computers have come a long way from the days of the PET, when the subject of graphics memory could be completely covered by saying that screen memory was located at 32768. The Commodore 64 features bitmap graphics, character graphics, and sprite graphics, thanks to the VIC-II chip, a sophisticated graphics display device which takes care of all the details of arranging the screen display. In order to display any of these types of graphics, however, the VIC-II chip must look to data in memory to tell it what to display. Therefore, to the user who wants to get the most out of the 64's graphics capabilities, the question of where in memory to place this data is an important one.

You would think that with 64K of RAM, there would be no problem finding adequate space for the placement of graphics memory. But the VIC-II can only address 16K of memory at a time. Within this area, sprite graphics data may be placed in any of 256 groups of 64 bytes each. Character data can be stored in any of eight 2K blocks. Text screen memory may be in any of 16 1K areas, and bitmap screen memory may be in either of two 8K sections.

When you turn the power on, the VIC-II uses the bottom 16K of memory for graphics. Unfortunately, this block of memory is also used extensively for other important purposes. The first 1024 locations are reserved for use as RAM workspace for the operating system. The second 1024 locations are taken up by screen memory. BASIC program text starts right above that. Needless to say, there isn't a whole lot of room left over for sprites, characters, and 8K bitmap screens. Though there are ways to eliminate some of these conflicts, as we will see below, these solutions are far from complete. In many situations more flexiblility would be helpful.

Flexibility

Fortunately, the 64 has that kind of flexibility. Even though the VIC-II chip can only address 16K of memory at a time, you can control which 16K you wish it to use. This bank select feature is used by manipulating bits 0 and 1 of Port A of the second CIA chip. That sounds complicated, but all it really involves is a simple POKE. These bits must be set as outputs to change banks (this is the default condition on powering-up). The technique for making this change from BASIC is discussed below. But before we go ahead and start changing banks, let's examine each one to see what areas are available for special graphics.

Bank 0 (0-16383) [$0-$3FFF]

This area is normally used for system variables and BASIC program text. Locations 1024-2048 ($400-$800) are reserved for the default position of screen memory.

There is an additional limitation on memory usage that applies to this block and to block 2. All of the data that the VIC-II chip sees must be within the same 16K block, including the data within the character generator ROM that tells the chip how to draw the shape of each letter on the screen. Since this ROM could not be stuck right in the middle of the BASIC program text area, an addressing trick is used. As a result of this trick, the VIC-II chip sees the character generator ROM at 4096-8191 ($1000-$1FFF), even though the 6510 microprocessor addresses this ROM at 53248 ($D000). So while the 6510 uses the RAM at these locations for program text, the VIC-II sees only the ROM and pays no attention to what is in RAM at these locations. This portion of memory is therefore unavailable for sprite patterns, user-defined characters, or screen memory, whether hi-res or text.

As pointed out above, there is little free space here for graphics display data. Locations 679-767 are unused, and could hold one sprite shape (number 11) or data for 11 characters. The area from 820-1023 ($334-$3FF), which includes the cassette buffer, is available for graphics memory, and is large enough to hold 3 sprite shapes (numbers 13, 14 and 15), or data for 25 characters. But something like bitmap graphics, which requires 8K of memory for the screen display, is a little trickier.

One solution is to use some of the area normally taken up by BASIC program text. This can be accomplished either by lowering the top of the BASIC text area, thereby protecting higher memory from a collision with BASIC, or by raising the beginning of BASIC text, thus protecting the memory below that point. To lower the top of BASIC memory, you need only change the system pointer to top of BASIC memory. For example, you can set aside memory from 8192 on with the statement POKE 56,32:CLR. This changes the top of BASIC to 32*256, or 8192, by POKEing that value into the high byte of the pointer. The space from 8192 to 16384 can now be used for a hi-res screen, new character sets, sprite shapes, or alternate text screens. Of course, such a solution sharply limits the amount of space left for a BASIC program.

The other alternative is to raise the start of BASIC text. For example, if you wanted to place an 8K bitmapped screen at 8192, you could move the start of BASIC to 16384 to protect that memory, leaving you with 24K for a BASIC program. Typing in the immediate mode, enter the following line:

```
POKE 44,64:POKE 16384,Ø:NEW
```

This solution has the drawback of being somewhat messy to implement without changing the pointer from the immediate mode before entering and running the program.

Bank 1 (16384-32767) [$4000-$7FFF]

This section is normally used for BASIC program storage. When using this bank, the VIC-II chip does not have access to the character generator ROM.

Providing that you lower the top of memory so that BASIC programs do not interfere, this area is wide open for sprite shapes, character graphics, and bitmap graphics. The drawbacks to using this bank are the unavailability of the character ROM and the limitation on BASIC program space (as little as 14K). The absence of the character ROM is a relatively minor nuisance, because you can always switch in the ROM and copy any or all of the characters to RAM. While the size problem may be eased somewhat by sticking to the upper portion of this bank, it still leaves this bank a less desirable choice for all but bitmap graphics.

Because this 16K block is the only one comprised totally of free RAM, it is a relatively good choice for bitmap graphics. Using the top 9K for the bitmap screen and color map, you will still be left with 21K of program space. The lack of the character ROM is not important in bitmap mode, and is actually an advantage, because it allows you to use either 8K section.

Bank 2 (32768-49151) [$8000-$BFFF]

This block consists of 8K RAM, half of which is seen by the VIC-II chip as character ROM, and the 8K BASIC interpreter ROM.

The BASIC ROM area is not, as you might think, totally unavailable for graphics. Because of its special addressing, aside from the character ROM, the VIC-II chip reads only from RAM. And even though the 6510 microprocessor chip cannot read RAM here as long as the BASIC ROM is switched in (a PEEK will only show the ROM value), it can write to it (with a POKE, for example). Whatever is written to the RAM underlying the BASIC ROM is displayed normally by the VIC-II chip. This opens up an extra 8K area for sprites and character data under the BASIC ROM.

You should keep in mind that while you can write to this area, you cannot read it from BASIC. This may not be a serious problem when it comes to character sets and sprite data, but it's more of a drawback if you want to use this RAM for screen memory. For example, the operating system has to read the text screen to move the cursor properly, and if it reads the ROM value instead of the RAM screen data, it gets hopelessly confused, making it impossible to type in any commands. Likewise, you would not be able to read the hi-res screen if placed here, without some machine language trickery. With locations 36864-40959 ousted by the character ROM, only 4K of true RAM remains for use as screen memory, not enough for a complete hi-res screen. Therefore, this block is not recommended for use in bitmap mode if your program needs to check the screen. Otherwise, this is a pretty good place for graphics memory, particularly if you need to emulate the screen configuration of the PET.

Bank 3 (49152-65535) [$C000-$FFFF]

This block normally contains 4K of RAM that is completely unused by the system, 4K of I/O registers, and the 8K Operating System Kernal ROM. It is very convenient to use when you need a lot of memory space for graphics. First, it is well above the BASIC program storage area, so you don't have to change pointers to protect your graphics from BASIC, and you don't have to limit your program space. As a matter of fact, since you won't need the area of 1024-2048 for screen memory if you use this block, you can lower the BASIC text pointer and get another 1K of BASIC program space if necessary. Second, it has enough free RAM for four text screens, while the ROM area can be used to store two character sets and 64 sprite shapes simultaneously. Although the character ROM is not available, it can be copied very quickly to the last 4K under the Kernal ROM with the following machine language program:

```
10 FOR I=1 TO 33:READ A: POKE 49151+I,A:N
   EXT: REM SET UP ML ROUTINE
20 POKE 56334,PEEK(56334) AND 254: REM DI
   SABLE INTERRUPTS
30 POKE 1,PEEK(1) AND 251: REM SWITCH CHA
   RACTER ROM INTO 6510 MEMORY
40 SYS 49152: REM COPY ROM CHARACTER SET
   {SPACE}TO RAM AT 61440
50 POKE 1,PEEK(1) OR 4: REM SWITCH CHARAC
   TER ROM OUT OF 6510 MEMORY
60 POKE 56334,PEEK(56334) OR 1: REM ENABL
   E INTERRUPTS
70 DATA169,0:    REM         LDA #00
80 DATA 133,251: REM         STA $FB
90 DATA 133,253: REM         STA $FD
100 DATA169,208: REM         LDA #$D0
110 DATA 133,252:REM         STA $FB+1
120 DATA 169,240:REM         LDA #$F0
130 DATA 133,254:REM         STA $FD+1
140 DATA 162,16: REM         LDX #16
150 DATA160,0:   REM  LOOP   LDY #00
160 DATA 177,251:REM  LOOP1  LDA ($FB),Y
170 DATA 145,253:REM         STA ($FD),Y
180 DATA 136:    REM         DEY
190 DATA 208,249:REM         BNE LOOP1
200 DATA 230,252:REM         INC $FB+1
210 DATA 230,254:REM         INC $FD+1
220 DATA 202:    REM         DEX
230 DATA 208,240:REM         BNE LOOP
240 DATA 96:     REM         RTS
```

Although this example transfers the ROM character set to RAM at 61440, you can change the destination to any even page by altering the DATA statement on line 120. You simply substitute your new destination address divided by 256 for the number 240 (which is 61440/256) given in the example.

While there is no RAM area available here for a hi-res screen, it is possible to use the area under the Kernal ROM for this purpose. Though the contents of this RAM cannot be read from BASIC, a short machine language routine could be used to momentarily turn off the interrupts and switch out the ROM so that the RAM could be used. It is likely that most plotting in bitmap mode will be done in machine language anyway, since BASIC is too slow to be very useful for this purpose.

One possible conflict that you should be aware of is that the current version of the DOS support program is written to reside at 52224($CC00). It would be safest to avoid using 52224-53247 for graphics if you plan to use DOS support.

Making the Change

Now that we have examined the possible banks to use for graphics memory, let's review the steps for making such a change. They are:

1. Select a bank. Banks 0-3 can be chosen by entering the following lines:

```
POKE 56578,PEEK(56578)OR 3:REM SET FOR OUTPUT IF N
  OT ALREADY
POKE 56576,(PEEK(56576)AND 252)OR BANK:REM BANK IS
   BANK #,MUST BE Ø-3
```

2. Set the VIC-II register for character memory. Since the chip can use any 2K segment within the bank for character memory, we must set this register to tell the chip where the character shape data is located. The formula for this is:

```
POKE 53272,PEEK(53272)OR TK:REM TK IS 2 KBYTE OFFS
  ET FROM BEGINNING OF BLOCK
```

For example, the ROM character set appears in banks 0 and 2 offset from the beginning of the bank by 4096 bytes (4K). Therefore, to point the chip to this ROM set, you would POKE 53272, PEEK (53272) OR 4.

Remember, in banks 1 and 3 the character ROM is not available, so you will need to move the set from ROM to RAM as shown in the sample program above.

3. Set the VIC-II register for display memory. Since the chip can use any 1K segment within the block for screen memory, we must set this register to tell the chip where the character shape data is located. The formula for this is:

```
POKE 53272,PEEK(53272)OR K*16:REM K IS KBYTE OFFSE
  T FROM BEGINNING OF BLOCK
```

In bank 0, for instance, the default screen area is set at 1024, at a 1K offset from the beginning of the block. To set the register to point to this location, you would POKE 53272, PEEK (53272) OR 16.

Since steps 2 and 3 operate on the same register, you could combine these steps and just POKE 53272, (16*K + TK). Using the default values of the two examples above, you would POKE 53272, 20.

4. Set the operating system pointer for display memory. Even though you have just told the VIC-II chip where to display memory for the screen, the operating system (OS) does not yet know where to write its text characters. Let it know with this statement:

```
POKE 648,AD/256:REM AD IS THE ACTUAL ADDRESS OF SC
  REEN MEMORY
```

You will notice that this pointer does not use a relative offset from the start of VIC-II memory, but rather the actual address of screen memory. To calculate this address, you will have to add the base address to the offset. For example, if the screen is offset 1K from bank 3, its location would be 1024 + 49152, or 50176. If you divide this number by 256, you find that the value to POKE is 196.

When you have done all of this, there will be no perceptible change, except perhaps for some garbage on the screen. But if you try to POKE to screen memory using the 1024 default starting location, nothing will appear. You will really be able to tell that something has happened if you hit the STOP and RESTORE keys. This sequence changes the screen display default to location 1024 in bank 0, but the OS pointer is not changed (at least not in the machines with early versions of the Kernal). As a result, what you are typing will not be displayed on the screen. If you enter POKE 648,4, things should get back to normal. There are two ways to avoid this problem. The simplest way is to disable the RESTORE key entirely. With the current version of the Operating System Kernal ROM, you just have to POKE 792,193 (POKE 792,71 returns normal function). But if you want the RESTORE key to really reset the default display parameters, you must route the Non-Maskable Interrupt (NMI) which is caused by the RESTORE key through a machine language routine that changes the OS pointer back to the default value of 4. An example of this technique is given in the sample Program 2.

Putting It All Together

To tie things together, I will close with a couple of examples of changing banks of screen memory. The first shows you how to configure your Commodore 64 so that its screen memory and BASIC program text start in the same places that they do on the PET. The second is a more elaborate demonstration of using bank 3 that includes the machine language transfer routine to move the ROM character set to RAM, and a short interrupt routine to correct the RESTORE key problem. After the switch is made, a loop is used to POKE characters to the new screen memory area. Next, the character data is slowly erased, to show that the character set is now in RAM. Then, a loop is used to read the locations of the character set and write to the same locations. This demonstrates that the 6510 reads the Kernal ROM when you PEEK those locations, but POKEs to the RAM which is being displayed. Finally, the machine language move is used again to show how quickly the set is restored.

Program 1. Configure the Commodore 64 Like a PET

```
10 REM EXAMPLE 1--CONFIGURE 64 LIKE PET
20 POKE 56576,PEEK(56576) AND 253: REM  S
   TEP 1, ENABLE BANK 2
30 POKE 53272,4: REM STEPS 2-3, POINT  VI
   C-II TO SCREEN AND CHARACTER MEMORY
40 REM SCREEN OFFSET IS 0*16, CHARACTER
   {SPACE}OFFSET IS 4
50 POKE 648,128: REM STEP 4, POINT OS TO
   {SPACE}SCREEN AT 32768 (128*256)
60 POKE 44,4:POKE 1024,0: REM MOVE START
   {SPACE}OF BASIC TO 1024 (4*256)
70 POKE 56,128: CLR: REM LOWER TOP OF MEM
   ORY TO 32768
80 POKE 792,193: REM DISABLE RESTORE KEY
90 PRINT CHR$(147): REM CLEAR SCREEN
```

Program 2. Using Bank 3

```
10 REM EXAMPLE 2, DEMONSTRATES USE OF BAN
   K 3
20 FOR I=1 TO 33:READ A:POKE 49151+I,A:NE
   XT: REM SET UP ML ROUTINE
30 GOSUB 200: REM ML COPY OF ROM CHARACTE
   R SET TO RAM
40 POKE 56576,PEEK(56576) AND 252: REM  S
   TEP 1, ENABLE BANK 3
50 POKE 53272,44: REM STEPS 2-3, POINT  V
   IC-II TO SCREEN AND CHARACTER MEMORY
```

```
60 REM SCREEN OFFSET IS 2*16, CHARACTER
   {SPACE}OFFSET IS 1212
70 POKE 648,200: REM STEP 4, POINT OS TO
   {SPACE}SCREEN AT 51200 (200*256)
80 PRINT CHR$(147): REM CLEAR SCREEN
90 FOR I=53236 TO 53245: READ A: POKE I,A
   : NEXT: REM NEW INTERRUPT ROUTINE
100 POKE 53246,PEEK(792):POKE 53247,PEEK(
    793): REM SAVE OLD NMI VECTOR
110 POKE 792,244: POKE 793,207: REM ROUTE
     THE INTERRUPT THROUGH THE NEW ROUTIN
    E
120 FOR I=0 TO 255: POKE 51400+I,I:POKE 5
    5496+I,1:NEXT
125 REM POKE CHARACTERS TO SCREEN
130 FOR J=1 TO 8: FOR I=61439+J TO I+2048
     STEP 8
140 POKE I,0:NEXT I,J: REM ERASE CHARACTE
    R SET
150 FOR I=61440 TO I+2048:POKE I,PEEK(I):
    NEXT: REM POKE ROM TO RAM
160 GOSUB 200:END: REM RESTORE CHARACTER
    {SPACE}SET
200 POKE 56334,PEEK(56334) AND 254: REM D
    ISABLE INTERRUPTS
210 POKE 1,PEEK(1) AND 251: REM SWITCH CH
    ARACTER ROM INTO 6510 MEMORY
220 SYS 49152: REM COPY ROM CHARACTER SET
     TO RAM AT 61440
230 POKE 1,PEEK(1) OR 4: REM SWITCH CHARA
    CTER ROM OUT OF 6510 MEMORY
240 POKE 56334,PEEK(56334) OR 1: REM ENAB
    LE INTERRUPTS
250 RETURN
300 REM DATA FOR ML PROGRAM TO COPY CHARA
    CTER SET TO RAM
310 DATA169,0,133,251,133,253,169,208,133
    ,252,169,240,133,254,162,16
320 DATA160,0,177,251,145,253,136,208,249
    ,230,252,230,254,202,208,240,96
330 REM NEXT IS ML PROGRAM TO MAKE THE RE
    STORE KY RESET OS POINTER TO SCREEN
340 DATA 72,169,4,141,136,02,104,108,254,
    207
```

Chapter Two

Understanding Bitmapped Graphics

Michael Tinglof

How to find your way through the bits and bytes of the high-resolution graphics screen, with a machine language subroutine you can use to plot or erase points in your own programs.

The high-resolution graphics screen is made up of little dots — 64,000 of them. Each of them is either on or off. Since each dot, or pixel, can be individually controlled, your computer must have an on-off instruction for every one. If you used one byte for each pixel, it would take almost every byte of RAM. There'd be no room for BASIC or the Kernal or the operating system or anything else.

But it doesn't take one byte for each dot on the screen. Instead, eight pixels can be controlled by a single byte through a technique called bitmapping.

It Looks Like Math, But It Isn't

If you don't already know binary mathematics, it doesn't matter. You can use bitmapping without understanding twos complement, arithmetic shift left, and logical shift right. All you have to know is how to turn on and off the dots on the screen.

Each byte consists of eight bits. They're like eight light switches, all in a row. The switches are either off or on.

The eight bits in a byte are either 1, which means *on*, or 0, which means *off*. The VIC-II video chip scans through screen memory reading each bit in each byte. If the bit is *on*, or 1, the VIC chip will light up a dot on the screen. If the bit is *off*, or 0, the VIC chip will leave that dot the background color.

Figure 1 is a bitmap for a very small screen. This screen is exactly 32 pixels wide and 8 pixels high. Each 1 represents a lit-up dot, and each 0 represents a dot that is the background color.

Chapter Two

Figure 1. A 32-by-8 Bitmap

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
```

Figure 2. The *On* Bits

```
                  1 1 1 1 1 1 1 1 1 1 1 1 1 1
                1                           1
      1 1 1 1 1                               1 1 1 1 1 1
    1                                                     1
  1 1                                                     1 1
      1 1 1 1       1 1 1 1 1 1 1 1 1 1 1       1 1 1 1 1
              1 1 1                       1 1 1
```

Figure 3. The Bytes in the Bitmap

```
0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0
(0)               (0)               (0)               (0)

0 0 0 0 0 0 0 0   0 1 1 1 1 1 1 1   1 1 1 1 1 1 0 0   0 0 0 0 0 0 0 0
(0)               (127)             (252)             (0)

0 0 0 0 0 0 0 0   1 0 0 0 0 0 0 0   0 0 0 0 0 0 1 0   0 0 0 0 0 0 0 0
(0)               (128)             (2)               (0)

0 0 0 1 1 1 1 1   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 1   1 1 1 1 1 0 0 0
(31)              (0)               (1)               (248)

0 0 1 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 1 0 0
(32)              (0)               (0)               (4)

0 1 1 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 1 1 0
(96)              (0)               (0)               (6)

0 0 0 1 1 1 1 0   0 0 1 1 1 1 1 1   1 1 1 1 1 0 0 0   1 1 1 1 1 0 0 0
(30)              (63)              (248)             (248)

0 0 0 0 0 0 0 1   1 1 0 0 0 0 0 0   0 0 0 0 0 1 1 1   0 0 0 0 0 0 0 0
(1)               (192)             (7)               (0)
```

In Figure 2, only the *on* bits are shown so you can see that this miniscreen contains a very simple drawing of the outline of a car.

Each group of eight dots is controlled by a single byte. Each bit in a byte controls one dot. Figure 3 shows this bitmap divided into its bytes. The decimal representation of each byte appears below the binary representation.

Now let's look at this mini-bitmap the way it is set up in memory. Memory is set up as one long sequence of bytes, from location 0 to location 65535. But the VIC-II chip reads bitmap memory as if it were divided up like a huge character set. That is, it reads memory as if it were divided into *cells* eight bits wide and eight bits high. There are 1000 such cells, 40 across by 25 down. Figure 4 is a map of the screen cells. There are exactly as many *cells* in the bitmap as there are pixels in regular screen memory.

Each cell consists of eight bytes. This gives a pattern eight bits wide by eight bits high. The VIC-II chip reads each byte in the cell in order from top to bottom before going on to read the next cell, as shown in Figure 5.

The overall pattern the VIC chip follows, then, is to start reading screen memory in cell 0, which is in the upper-left-hand corner of the screen. The eight bytes of that cell are read in order from top to bottom. Then the VIC-II reads cell 1, which is on the top row, just to the right of cell 0. The VIC-II continues until it reaches the last cell of the first row, 39. When it reads cell 40, it begins a new row.

This means that, following this pattern, our tiny bitmap from Figures 1-3 would appear *in memory* as shown in Figure 6. If the bitmap started at address 16384, you would find the bytes in the order shown. The first eight bytes are cell 0; the next eight bytes are cell 1; and so on.

Binary Operations

How does the computer actually change which dot is on or off? You can't PEEK or POKE one bit at a time in screen memory, after all — if you want to change one dot on the screen, you have to POKE the whole byte, controlling eight pixels, not just one.

The 64 provides some commands that let you take a byte of screen memory, change one pixel — or more — individually, and then put the byte back into place.

Before we set up a program that plots an individual dot, let's set up a subroutine that pulls a number out of screen memory and then puts it back when we're through with our operation.

Figure 4. Cell Map

```
        +0        +10       +20       +30
0 +     0123456789012345678901234567890123456789
40 +    0123456789012345678901234567890123456789
80 +    0123456789012345678901234567890123456789
120 +   0123456789012345678901234567890123456789
160 +   0123456789012345678901234567890123456789
200 +   0123456789012345678901234567890123456789
240 +   0123456789012345678901234567890123456789
280 +   0123456789012345678901234567890123456789
320 +   0123456789012345678901234567890123456789
360 +   0123456789012345678901234567890123456789
400 +   0123456789012345678901234567890123456789
440 +   0123456789012345678901234567890123456789
480 +   0123456789012345678901234567890123456789
520 +   0123456789012345678901234567890123456789
560 +   0123456789012345678901234567890123456789
600 +   0123456789012345678901234567890123456789
640 +   0123456789012345678901234567890123456789
680 +   0123456789012345678901234567890123456789
720 +   0123456789012345678901234567890123456789
760 +   0123456789012345678901234567890123456789
800 +   0123456789012345678901234567890123456789
840 +   0123456789012345678901234567890123456789
880 +   0123456789012345678901234567890123456789
920 +   0123456789012345678901234567890123456789
960 +   0123456789012345678901234567890123456789
```

Figure 5. The Eight-Byte Cell

byte	bit pattern
0	0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 0
3	0 0 0 0 0 0 0 0
4	0 0 0 0 0 0 0 0
5	0 0 0 0 0 0 0 0
6	0 0 0 0 0 0 0 0
7	0 0 0 0 0 0 0 0

Figure 6. The Bitmap in Memory

Screen Arrangement of Bytes

Cell 0	Cell 1	Cell 2	Cell 3
0	0	0	0
0	127	252	0
0	128	2	0
31	0	1	248
32	0	0	4
96	0	0	6
30	63	248	248
1	192	7	0

Bytes in Memory

Address	Byte	
16384	0	cell 0 start
16385	0	
16386	0	
16387	31	
16388	32	
16389	96	
16390	30	
16391	1	
16392	0	cell 1 start
16393	127	
16394	128	
16395	0	
16396	0	
16397	0	
16398	63	
16399	192	
16400	0	cell 2 start
16401	252	
16402	2	
16403	1	
16404	0	
16405	0	
16406	248	
16407	7	
16408	0	cell 3 start
16409	0	
16410	0	
16411	248	
16412	4	
16413	6	
16414	248	
16415	0	

We'll assume that bitmap memory starts at address MM. When this subroutine is accessed, the variable CW will say which cell, from 0 to 999, we want to change, and BW will say which byte within the cell, from 0 to 7, we want to change.

```
500 W=MM+CW*8+BW
510 XB=PEEK(W)
599 POKE W,NB:RETURN
```

The variable XB holds the old value of the byte, and NB holds the changed value. W is set to the absolute address of the byte we are changing: the start of bitmap memory plus the cell (multiplied by 8) plus the byte within the cell. Later, between lines 510 and 599, we'll insert the lines that perform the actual changes on the byte.

Now that you have the byte, what do you do with it?

Bitwise AND. When you use an expression like A = 5 AND 3, the word AND causes a *binary operation* to take place. The two numbers are compared, bit by bit. Let's stack them on top of each other to see the comparison more easily:

bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
5	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
3	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**

Notice that the bits are numbered from right to left, from 0 to 7. It looks odd, but it really makes sense. Bit 0 is the bit with the least value — that is, a 1 in that position is only worth 1. Bit 1 has twice the value of bit 0 — a 1 in that position has a value of 2. Bit 2 has a value of 4, bit 3 a value of 8, and so on. Here's a listing of what a 1 is worth in each bit position:

bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Now, when we perform an AND operation on the numbers 5 and 3, the computer compares each bit in the first number with the corresponding bit in the second number. For instance, bit 7 of the number 5 is a 0; bit 7 of the number 3 is a 0.

When AND compares the two numbers, it is looking for a 1 in the same bit in both numbers. Whenever it finds matching 1's, it puts a 1 in the result of the operation; the rest of the time, it puts a 0 in the result:

	bit:	7 6 5 4 3 2 1 0
AND	5	0 0 0 0 0 1 0 1
	3	0 0 0 0 0 0 1 1
result	1	0 0 0 0 0 0 0 1

In bit 2, the AND operation found a 1 in the number 5, but there was no matching 1 in the number 3. Therefore, a 0 was put into the result. In bit 1, the AND operation found a 1 in the number 3, but there was no matching 1 in the number 5 in that position. Result? Another 0. Only in bit 0, where both numbers have a 1, is the result 1. Therefore, 3 AND 5 = 1.

Bitwise OR. The OR operation pairs up numbers just like the AND operation, only now it isn't looking for a match. If it finds a 1 in *either* number, it will put a 1 into the result. Here's what 5 OR 3 looks like:

	bit:	7 6 5 4 3 2 1 0
OR	5	0 0 0 0 0 1 0 1
	3	0 0 0 0 0 0 1 1
result	7	0 0 0 0 0 1 1 1

Since the number 5 has a 1 in bit 2, there'll be a 1 in bit 2 of the result, regardless of what the number 3 has in that position.

The rule is, then:
AND results in a 1 wherever *both* numbers have a 1.
OR results in a 1 wherever *either* number has a 1.

Using AND and OR with a bitmap. You probably already see how this lets you turn on or off one pixel. Let's say you want to turn on bit 3 in the byte, regardless of what is already there. However, you don't want to change any of the rest of the bits in that byte. Here is what our subroutine would do:

```
500 W=MM+CW*8+BW
510 XB=PEEK(W)
520 NB=XB OR 8
599 POKE W,NB:RETURN
```

What is happening in line 510? In binary notation the number 8 looks like this: 00001000. Bit 3 (the fourth bit from the right) is a 1. All the rest are zeros. When you OR 8 with any number, the resulting number will always have a 1 in bit 3. OR 8, then, switches on bit 3.

To switch on any bit, you use the same procedure. OR 2 turns on bit 1. OR 128 turns on bit 7.

To turn on two bits, just add the numbers together before ORing them with the screen memory byte. For instance, to turn on bits 0 and 1, NB = XB OR (1 + 2). To turn on bits 6 and 7, NB = XB OR (128 + 64). Of course, you don't have to show the addition in your actual program. You'd write that last statement like this:

NB = XB OR 192

How do you switch on every dot in a byte? OR the byte with 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128, which adds up to 255. NB = XB OR 255. Of course, if you're switching on *every* byte, you might as well just say NB = 255. You need OR when you want to change only a few bits in a byte, and leave the others unchanged.

OR is used for switching *on* bits. AND is used for switching *off* bits. To turn off a dot, remember, you need to have a 0 in that position. With the AND operation, *both* numbers have to have a 1 in a certain position for the result to have a 1 in that position. Therefore, you can put a 0 in a particular bit position by ANDing the screen memory byte with a byte that has a 0 in that position.

Any number AND 0 will result in 0, since there can't possibly be a match. Therefore, to turn off all the bits in a byte, you just have to AND it with 0. (However, if you just want to erase a whole byte, you don't need AND — just POKE the location with 0.)

But let's say we want to erase only bit 7. We want to leave all the other bits unchanged. Since putting a 0 in a bit position will always leave a 0 in that same position in the result, you must put a 1 in every position that you want to leave unchanged. Here's what the program would look like:

```
500 W=MM+CW*8+BW
510 XB=PEEK(W)
520 NB=XB AND 127
599 POKE W,NB:RETURN
```

Why 127? Because 127 = 255 – 128. Let's look at the binary number:

```
01111111
(127)
```

Notice that 127 has all its bits *on* except for bit 7. If you AND any number with 127, all the bits from bit 0 to bit 6 that were on in the original number will still be on in the result, since there is a 1 in

127 to match them. All the bits that were off in the original number would remain off. With bit 7, however, there cannot possibly be a match since there is a 0 in that position in the number 127. There can never be a match there, and the result will always be 0.

So to draw dots, start with 0, put a 1 in every position you want to turn on, and then OR that number with the number already in screen memory. To erase dots, start with 255, put a 0 in every position you want to turn off, and then AND that number with the number already in screen memory.

To switch on bit 7, start with 0 and add 128, which is the value of bit 7 when it is on. The OR 128 with the byte in screen memory, and bit 7 will be switched on.

To switch off bit 7, start with 255 and subtract 128, which puts a 0 in bit 7, for a result of 127. Then OR 127 with the byte in screen memory, and bit 7 will be switched off.

Locating the Bitmap in Memory

Now that we've seen how the bitmap works, it's time to decide where in memory it should be. To do that, we need to understand how the VIC-II chip sees memory.

Screen memory, color memory, and the bitmap. If you have worked with graphics in the character mode (as opposed to bitmap mode), you're probably used to using both screen memory, which consists of the screen code values for the characters to be displayed on the screen, and color memory, which consists of the color code values for each character on the screen.

With bitmap mode, the color memory area at 55296 is ignored. However, the 1000 bytes of screen memory are now used as color memory for the bitmap. Each byte of screen memory contains the color code for the corresponding *cell* in the bitmap. So from now on, when we talk about screen memory, we're talking about the area in memory where *color* is controlled, and when we talk about the bitmap, we'll be talking about the area in memory where the individual dots are turned on and off.

The graphics base address. The VIC-II can't handle 64K. It can only control a maximum of 16K of memory at a time. So, unlike your 6510 CPU, the VIC-II uses memory as if it were cut into four banks of 16K each, like this.

bank 0	**addresses**	**0-16383**
bank 1	**addresses**	**16384-32767**

bank 2	**addresses**	**32768-49151**
bank 3	**addresses**	**49152-65535**

The VIC-II can read any one of those four banks, but only one at a time. That means that if you put the bitmap in bank 3, then screen memory must also be in bank 3.

How do you tell the VIC-II which bank to use? Bits 0 and 1 of location 56576 control the bank selection in this fashion:

Bank	**Bits**	**Decimal Value to POKE 56576**
0	11	3
1	10	2
2	01	1
3	00	0

Thus, POKE 56576,0 will tell the VIC-II to use bank 3, starting at 49152.

Which block should you use for bitmapped graphics? The best is bank 1, from 16384 to 32767. Why? Because the other blocks are too busy. Bank 0, the block that the VIC-II normally selects, is also used for your BASIC program and contains many vital operating system functions. Banks 2 and 3 lose a lot of space to ROM. So you'll probably want to POKE 56576,2.

The first address in each bank is the graphics base address. In calculating other addresses, you will use the base address as a starting point, and calculate the other locations by adding numbers to the base address. (If you use variables for these addresses in your programs, then you can later switch from one bank to another simply by changing the values stored in the variables, instead of having to find every occurrence of those numbers in the program.)

The screen memory block. Screen memory (which controls color) uses almost 1K, and the bitmap uses nearly 8K. Both must be located within the 16K graphics bank. Screen memory must begin on a 1K boundary — that is, its starting address must be evenly divisible by 1024.

Therefore, there are 16 possible locations for screen memory within the block. Here are the starting addresses of each possible screen memory block, expressed as an offset from the graphics base address.

To POKE a number into the upper-left-hand corner of screen memory, you would POKE into the graphics base address *plus* the offset to the screen memory block.

Figure 7. Possible Screen Memory Offsets

offset	screen memory block	POKE value	offset	screen memory block	POKE value
0	0	0	8192	8	128
1024	1	16	9216	9	144
2048	2	32	10240	10	160
3072	3	48	11264	11	176
4096	4	64	12288	12	192
5102	5	80	13312	13	208
6144	6	96	14366	14	224
7168	7	112	15360	15	240

To tell the VIC-II which 1K block you are using for screen memory, you must POKE the block number *times 16* into location 53272.

Why multiply it by 16?

Location 53272 is bitmapped, too! The leftmost four bits (bits 4-7) of 53272 control the color memory block. If bits 4-7 are 0000, then color memory block 0 is selected; if they are 0001, then block 1; if they are 0010, then block 2, and so on.

However, when you are POKEing values into the byte at 53272, you can't just POKE the four high bits — you have to POKE the whole byte. If you want to select block 7, for instance (binary number 0111), you couldn't do it with the command POKE 53272,7. That would put the binary number 7, or 00000111, into that location. Bits 4-7 are all zeros!

But if you multiply the block number by 16, it has the effect of moving all the *on* bits four positions to the left. Instead of POKEing 7, we'll POKE 7*16, or 112. This puts the binary number 01110000 into location 53272 — which is exactly what we want.

The bitmap block. Since the bitmap uses 8K, there are only two possible bitmap blocks within the 16K graphics bank, one starting at an offset of 0 and one starting at an offset of 8192, or 8K. In other words, the bitmap block must take up either the first half or the second half of the graphics bank.

To tell the VIC-II whether you have selected block 0 or 1 for the bitmap, you again POKE a number into location 53272, the same location where you POKE the information about the screen memory block. This time, though, it's bit 3 that selects the block, so to get the right number you must multiply by 8.

Since the same location, 53272, controls both the color

Figure 8. Possible Bitmap Offsets

offset	bitmap memory block	POKE value
0	0	0
8192	1	8

memory and bitmap block, you must add the two numbers together before POKEing them in. (This is because POKEing in either number alone will cause the other number to be 0.) If the variable SB holds the color block number (0-15) and the variable MB holds the bitmap block number (0 or 1), you would POKE 53272,SB*16 + MB*8.

So if you want graphics bank 1, and within that block you want bitmap block 1 and screen memory block 7, this is what your program should do:

```
10 GB=2:POKE 56576,GB:REM SELECT VIC-II BANK 1
20 SB=7:MB=1:POKE 53272,SB*16+MB*8:REM SELECT SCRE
   EN BLOCK 7 AND BITMAP BLOCK 1
30 GM=49152-GB*16384:SM=GM+SB*1024:MM=GM+MB*8192:R
   EM SET ADDRESS VARIABLES
```

In line 30, this program sets the variable GM to equal the graphics base address. Then it sets SM to the screen memory starting address and MM to the bitmap memory starting address.

Switching on bitmapped graphics. Once you have the pointers set, you have to tell the VIC-II chip to switch from character graphics to bitmapped graphics. You do that by switching on bit 5 of the byte at location 53265. This POKE command will do the job:

POKE 53265,PEEK(53265) OR 32

Plotting Points on the Bitmapped Screen

How do you translate all this into standard X-Y coordinate plotting? You know how to use AND and OR to plot within a byte; you know how to tell the computer to use bitmapped graphics and where to find the bitmap; but how do you tell the computer exactly which bit on the whole screen to switch on or off?

To find a particular pixel, think of the screen as one large crisscrossed field of squares, 320 vertical columns by 250 horizontal rows. You want to fill in a square on the screen at position

X,Y, where X is the column number (from 0 to 319) and Y is the row number (from 0 to 249). If position 0,0 is the upper-left-hand corner, this X-Y coordinate grid would look like Figure 9. In Figure 9, the square at 3,1 is filled in.

Figure 9. The X,Y Coordinate Grid

```
        column
row   0    1    2    3    4    .    .    .    319
 0
 1
 2
 3
 4
 .
 .
 .
249
```

Remember, the point we are plotting is one pixel on the screen, which is represented by one single bit somewhere in the bitmap. Point 3,1 would be easy to find, since it would be the third bit from the left (bit 5) in the second byte (byte 1) of the first cell (cell 0) of the bitmap. It won't always be that easy.

For instance, point 299,144 is far into the bitmap. How can we find which byte that bit is in, so we can plot it? The routine we worked out before won't do the job — it assumes that the cell and byte have already been found. We need a program that can start from the coordinates of a pixel and find the bit in the bitmap from that information alone.

Here's a small program that will do it. Before this program, the variable MM has been set to the absolute address of the start of the bitmap:

```
1ØØ X=299:Y=144
11Ø XC=INT(X/8)*8:YC=INT(Y/8)*8
12Ø XB=2↑(X-XC):YB=Y-YC
13Ø PT=MM+YC*32Ø+XC+YB
14Ø POKE PT,PEEK(PT) OR XB
```

How does this program work?

Line	**Function**
100	Assign X and Y coordinates.
110	Set XC to the column number times 8; set YC to the row number times 8.

120	Set YB to the number of the byte within the cell. Set XB to the decimal value of the bit to be turned on within the byte.
130	Set PT to the absolute address of the byte to be plotted (the start of the bitmap *plus* the offset to the cell row *plus* the offset to the cell column *plus* the offset to the byte within the cell).
140	PEEK the byte currently at location PT, bitwise OR it with XB (the bit to turn on within the byte), and POKE it back into location PT.

Controlling Color

In bitmapped graphics, the VIC-II chip uses screen memory to determine the colors of the *on* and *off* pixels in the bitmap.

Each cell in the bitmap is color-controlled by one byte in screen memory. The left four bits (bits 4-7) of each screen memory byte control the color that will be displayed by every *on* (1) bit in the bitmap cell. The right four bits (bits 0-3) of each screen memory byte control the color that will be displayed by every *off* (0) bit in the bitmap cell, as shown in Figure 10.

Figure 10. The Color Control Byte in Screen Memory

```
color of on bits          color of off bits
   7  6  5  4           3  2  1  0
```

This is one of the most powerful features of Commodore 64 graphics. You can display up to 16 different colors on the screen at the same time. There is a drawback, however. Changing the colors for one bit will change the colors for every other bit in the same cell. Still, by careful planning you can make very effective high-resolution drawings with many different colors on the screen.

What numbers do you POKE into screen memory? The color codes are numbers 0 through 15. For the background color—the color to display for every 0 bit in the bitmap cell — you merely have to POKE the color code into screen memory. For the foreground color — the color to display for 1 bits in the bitmap cell — you have to multiply the color code by 16 to move it four bits over to the left. If the variable C1 represents the foreground color and the variable C0 is the background color, this statement will get the right color into screen memory location SM:

```
POKE SM, C0 + 16*C1
```

If you want to change the background color already at location SM without disturbing the foreground color, you would use this statement:

```
POKE SM, (PEEK (SM) AND 240) OR C0
```

To change the foreground color without changing the background color, use this statement:

```
POKE SM, (PEEK (SM) AND 15) OR 16* C1
```

Multicolor Mode

There is another bitmapped graphics mode that we haven't looked at yet: multicolor bitmap mode. This mode allows you to get around the limitation that only two colors can be displayed in any one cell. In multicolor bitmap mode, up to four colors can be displayed — the background color and three foreground colors. To tell the VIC-II to enter multicolor mode, after you are in bitmapped graphics mode, POKE 53270, PEEK(53270) OR 16.

How the bitmap codes the colors. Since each bit in the bitmap is either on or off, how can we code *four* colors? The Commodore 64 does this by linking every two bits together in bit-pairs, which act together. One bit can offer only two choices, on or off. Two bits acting together, however, can offer four choices:

00	0
01	1
10	2
11	3

Each bit-pair, then, can specify either the background color (0) or one of the three foreground colors (1-3).

This means that it takes *two* bits to control each dot. That would take up 16K, the entire graphics bank. To get around this problem, the pixels on the screen are also paired. Thus, each bit-pair controls one pixel-pair. That allows you to hold the multicolor screen in the same 8K as the regular bitmap mode.

There is one disadvantage. Since both pixels in a pixel-pair are controlled by the same bit-pair, they must always have the same color. In effect, all dots on the screen will be two high-resolution pixels wide. Your resolution will only be 160 by 250 instead of 320 by 250. However, the added possibilities of multicolor drawings often make up for the loss in fine-line resolution.

AND and OR with multicolor bytes. Each byte in multicolor mode consists of four bit-pairs, like this:

```
00 00 00 00
```

To change one pixel-pair on the screen you have to change two bits at a time, not one.

Often the most convenient way to do this is to set up a color matrix and a bit-pair matrix. You will need four color matrices, one for each color:

Color 0	00 00 00 00 (0)
Color 1	01 01 01 01 (85)
Color 2	10 10 10 10 (170)
Color 3	11 11 11 11 (255)

Notice that each color matrix consists of one byte with every bit-pair set to the same color. If you used the color matrix alone, you could only change whole bytes at a time, not individual bit-pairs.

You will also need four bit-pair matrices, one for each bit-pair:

Bit-pair 0	11 00 00 00
Bit-pair 1	00 11 00 00
Bit-pair 2	00 00 11 00
Bit-pair 3	00 00 00 11

Notice that each bit-pair matrix has one bit-pair on; if this were used directly, it would always set the target bit-pair to color 3.

But in combination, you can use bit-pair and color matrices to set exactly the right bit-pair to exactly the right color, without changing the other bit-pairs in the byte.

First, set up the two sets of matrices in arrays. The color matrices are C(0) through C(3). The bit-pair matrices are BP(0) through BP(3). In this example, let's say we are working on bit-pair 2; we want to change it to color 1. After the operation, we want to make sure that the byte looks like this:

```
?? ?? 01 ??
```

The question marks represent bits that we are not changing. We don't know what they are and don't care, except that we want to leave them unchanged.

For our example, however, we'll say that the byte we're working on was set entirely to color 2. The binary number 10 10 10 10 has a decimal value of 170. Our result should then be the binary number 10 10 01 10, which has a decimal value of 166.

Here's how we make sure we get that result.

First, take the original byte, XB, and create a window for the new color by erasing the current contents of the target bit-pair. You do this by ANDing it with the *inverse* of the bit-pair matrix. The inverse is 255 *minus* the bit-pair matrix. In our case, the bit-pair matrix is BP(2), the binary number 00 00 11 00, or decimal 12. Subtract it from 255 (binary 11 11 11 11) and you get the result 243, or binary 11 11 00 11. When you AND this number with the byte XB, it will turn bit-pair 2 to zeros, and leave the other bit-pairs completely undisturbed.

Here's a program line that does this, and stores the resulting window byte in the variable WB. In our example, WB would be binary 10 10 00 10, or decimal 162.

```
WB = XB AND (255 - BP(2))
```

Now that you have a window byte to receive the new bit-pair, you have to create a bit-pair of the right color in the right position. All you do is AND the color matrix with the bit-pair matrix. The color matrix C(1) is binary 01 01 01 01, and the bit-pair matrix BP(2) is binary 00 00 11 00. The result of the AND operation is the binary number 00 00 01 00 — the target bit-pair is set to the right color, and the other bits are all zeros.

Here is a statement to do this, storing the final bit-pair in the variable FP:

```
FP = C(1) AND BP(2)
```

Now all that remains to do is OR the final bit-pair with the window byte. In our example, the window byte was 10 10 00 10, and the target bit-pair was 00 00 01 00. ORing them results in the binary number 10 10 01 10, which is exactly the result we wanted. In this statement, the result of the operation is stored in the variable NB:

```
NB = FP OR WB
```

All these operations can be put together in a single program line:

```
WB = XB AND (255 - BP(2)):FP = C(1) AND BP(2):NB = FP OR WB
```

Or, even more simply expressed:

```
NB = (XB AND (255 – BP(2)) OR (C(1) AND BP(2))
```

If the bit-pair number and color number were also variable (BN and CN), this line would plot the right bit-pair in any multicolor bitmap program:

```
NB = (XB AND (255 – BP(BN)) OR (C(CN) AND BP(BN))
```

In fact, if the variable MM is set to the absolute address of the byte you want to change, you can eliminate NB and XB, too:

```
POKE MM,(PEEK(MM) AND (255 – BP(BN)) OR (C(CN) AND
BP(BN))
```

That line will execute as quickly as you could hope for.

Changing colors in multicolor mode. Besides handling bit-pairs, there's one more problem with multicolor mode. Screen memory can only hold two color codes per byte, one in the left four bits and the background in the right four bits. Where do you assign the other two colors available within a multicolor bitmap cell?

Since we're using screen memory for color assignments in the bitmapped graphics modes, we still have regular color memory beginning at location 55296. Color memory is arranged in the same order as screen memory and the cells in the bitmap.

However, only the lower four bits (bits 0-3) are meaningful in color memory, so we still have one more color to assign. For that, we use the VIC-II chip's background color register at 53281. Unfortunately, this means that all the cells have to have the same background color, unless you use raster interrupts (see "Mixing Graphics Modes"). However, the other three colors can be individually assigned for each cell, giving you many possibilities for color combinations.

The color called for by the bit-pair 00 (color 0) will be the background color, which is stored in the background color register at 53281.

The color called for by the bit-pair 01 (color 1) will be the color stored in the left four bits (bits 4-7) of the corresponding byte in screen memory.

The color called for by the bit-pair 10 (color 2) will be the color stored in the right four bits (bits 0-3) of the corresponding byte in screen memory.

The color called for by the bit-pair 11 (color 3) will be the color stored in the right four bits (bits 0-3) of the corresponding byte in color memory.

This would all be very confusing if you wanted to change the colors in midprogram, except that the bitmap cells, screen memory bytes, and color memory bytes are laid out in exactly the same order. This means that the same offset number can be used to find the screen memory byte and color memory byte that control a particular cell's colors. Suppose you want to change the colors that affect byte 683 in the bitmap. The cell offset is INT (683/8), or 85. Therefore, to change a color that affects byte number 683 from the start of the bitmap, you would change byte 85 from the start of color memory and/or byte 85 from the start of screen memory.

Bringing Your 64 Back to Normal

To restore your computer to normal operation, use the following commands:

```
POKE 53265,27:POKE 53270,200:POKE 53272,20:POKE 56576,151
```

Protecting Your Picture

When using BASIC and the bitmap together, BASIC may have a tendency to spill over and start using bitmap memory or screen memory to store program lines or variables. To stop this, you must fool BASIC into thinking that the computer's memory ends *before* it reaches your map. This is why you should not use graphics bank 0 for a bitmapped graphics screen — there'll be almost no room for any kind of program if BASIC has to share one 16K block with the bitmap.

To change where BASIC thinks memory stops, you must POKE new values into locations 55 and 56. The top of available memory should be set to the *lowest*-numbered address you use in the graphics block. If you start the bitmap at, say, 16384, then that number will be the number you use as the top of memory. If you start the bitmap in bitmap block 1 and screen memory at screen memory block 7, you can let BASIC use memory up to address 23551; the new end-of-memory address will be 23552, which is the first address in the screen memory block.

The number 23552 is too large to POKE into any one memory location, since no location can hold more than one eight-bit byte. The largest number any location can hold is 255. All but the first 256 addresses in the computer, however, are numbers larger than 255. The computer handles this by breaking the address into two bytes. The lower part of the address you are storing is almost always placed before the higher part.

The address 23552 is the 16-bit binary number 0101110000000000. This number is split into two halves, 01011100 (decimal 92) and 00000000 (decimal 0). The lower part of this address, 0, is POKEd into location 55, and the higher part, 92, into location 56.

You don't have to calculate the binary numbers, however. Instead, this program line will split any integer XX from 0 to 65535 into the low byte (LB) and the high byte (HB):

```
    HB = INT(XX/256):LB = XX-HB*256
```

Then your program only needs to POKE LB into the first memory location and HB into the second memory location. Chances are that you're already familiar with this technique — every computer that uses the 6502 or 6510 CPU uses it frequently.

Special Effects

When you create drawings using bitmaps, much of their effect depends on what you do with them.

You can set the pointers to your bitmap and screen memory *before* your program draws the picture. That way the user can watch the picture being drawn.

You can draw the picture in the bitmap area while the pointers are still indicating the default graphics settings. Then, when you change the pointers, the complete picture suddenly appears on the screen. It gives the effect of lightning speed, even in BASIC.

You can set up two or three bitmaps and switch back and forth between them by changing the pointers. This uses up a lot of memory, but the effect can be dazzling, and the switching is almost instantaneous in BASIC.

Because each cell is exactly the same size as the character patterns in character set memory, you can easily put letters and characters on the high-resolution screen by PEEKing the pattern in character set memory and POKEing it into individual cells in the same order.

You can supplement the colors of the bitmap screen by using sprites, which can have up to three visible colors each. Sprites don't always have to move, either. By combining sprites and high-resolution graphics, you can get very realistic, detailed drawings.

The color codes in screen memory (and color memory, for multicolor mode) consist of one byte each, while the cells in the bitmap are eight bytes. Therefore, you can change screen and color memory much faster than you can change the bitmap. If you're trying to do animation in BASIC, where speed is always a problem, you can get much greater quickness by changing *colors* than by moving *pixels*. This won't be appropriate for most animation, of course, but where it *will* work, the increased speed can be remarkable.

A Machine Language Plotting Routine

Here is a short machine language routine that will execute four commands on the two-color bitmap screen. You can add it to your own BASIC programs, executing it by using the statement SYS AD, where AD is the address where your program POKEd the first byte of the machine language routine into memory.

The SYS call must be followed by a command number, from 0 to 3. In addition, some of the commands require you to include further numbers with the SYS call. An example is shown for each command:

Command 0: Clear Screen. Format:

SYS AD, 0

This command clears the bitmap screen by setting all bytes in the bitmap to 0.

Command 1: Set Colors. Format:

SYS AD, 1, *nn*

This command sets all the bytes in screen memory to the value *nn*. This allows you to set all the colors for every bitmap cell at once, at machine language speed. Remember that bits 4-7 of the number *nn* control the foreground color and bits 0-3 of the number *nn* control the background color.

Command 2: Plot Point. Format:

SYS AD, 2, *xx*, *yy*

This command puts a single dot on the screen at the location marked by the values *xx* and *yy*. The number *xx* represents the column (horizontal position) of the target pixel and must be a number from 0 to 319. The number *yy* represents the row (vertical position) of the target pixel and must be a number from 0 to 249.

Command 3: Erase Point. Format:

SYS AD, 3, *xx*, *yy*

This routine is identical to command 2, except that instead of setting the pixel to 1, it is set to 0.

Before you can use the routine, your BASIC program must tell the computer where screen memory and the bitmap begin. POKE location 680 with the starting address of the bitmap, divided by 256. POKE location 681 with the starting address of screen memory, divided by 256. You don't have to POKE the low byte of these addresses into memory, because the routine "knows" that the low byte will always be zero.

Lines 10-30 READ the DATA statements and POKE the machine language routine into memory. You may POKE the routine somewhere else, but it's a good idea to make sure you put it in a protected area of memory.

Lines 100-300 are the machine language routine in the form of DATA statements. You'll need to be very careful typing these in. It is easy to make typographical errors when typing rows and rows of numbers. If the routine doesn't work, check the DATA statements first for errors.

Lines 500-600 are an example program that uses the machine language routine to plot a sine wave. You would not include these lines in your own program.

Bitmap Utility

```
1 REM *BIT MAP UTILITY*
3 REM{2 SPACES}COMMAND:
4 REM{5 SPACES}XX SYS (BASE),OPTION,DATA
5 REM{4 SPACES}OPTIONS:
6 REM SYS B, Ø{2 SPACES}-{2 SPACES}CLEAR SCREEN
7 REM SYS B, 1, CL - SET COLOR CL
8 REM SYS B, 2, X, Y - SET POINT (X,Y)
9 REM SYS B, 3, X, Y - CLEAR POINT
1Ø AD=32768:REM ** BASE ADDRESS
2Ø READD:CK=CK+D:IFD=-1THEN4Ø
3Ø POKEAD,D:AD=AD+1:GOTO2Ø
4Ø IF CK<>38745 THEN PRINT"ERROR IN DATA STATEMENT
   S":STOP
5Ø GOTO 5ØØ:REM ** JUMP TO USER SUBROUTINE
1ØØ DATA 32, 115, Ø, 32, 158, 173, 32, 247, 183, 1
    4Ø, 17Ø, 2, 192, Ø
11Ø DATA 24Ø, 6, 192, 1, 24Ø, 32, 2Ø8, 77, 173, 16
    8, 2, 133, 252, 24
12Ø DATA 1Ø5, 32, 133, 253, 169, Ø, 133, 251, 168,
     145, 251, 23Ø, 251, 2Ø8
13Ø DATA 2, 23Ø, 252, 166, 252, 228, 253, 144, 242
    , 96, 32, 115, Ø, 32
14Ø DATA 158, 173, 32, 247, 183, 132, 253, 173, 16
    9, 2, 56, 233, 1, 133
15Ø DATA 252, 24, 1Ø5, 4, 133, 254, 169, 8, 133, 2
    51, 16Ø, 247, 165, 253
16Ø DATA 145, 251, 23Ø, 251, 2Ø8, 2, 23Ø, 252, 166
    , 252, 228, 254, 144, 242
```

```
170 DATA 96, 32, 115, 0, 32, 158, 173, 32, 247, 18
    3, 140, 171, 2, 141
180 DATA 172, 2, 32, 115, 0, 32, 158, 173, 32, 247
    , 183, 140, 173, 2
190 DATA 152, 41, 248, 133, 253, 141, 180, 2, 141,
     174, 2, 169, 0, 133
200 DATA 254, 141, 181, 2, 162, 4, 24, 38, 253, 38
    , 254, 202, 16, 248
210 DATA 162, 2, 24, 46, 180, 2, 46, 181, 2, 202,
    {SPACE}16, 246, 24, 165
220 DATA 253, 109, 180, 2, 141, 178, 2, 165, 254,
    {SPACE}109, 181, 2, 141, 179
230 DATA 2, 173, 171, 2, 41, 248, 141, 176, 2, 173
    , 172, 2, 141, 177
240 DATA 2, 56, 173, 173, 2, 237, 174, 2, 24, 109,
     176, 2, 133, 251
250 DATA 173, 177, 2, 109, 168, 2, 133, 252, 24, 1
    73, 178, 2, 101, 251
260 DATA 133, 251, 173, 179, 2, 101, 252, 133, 252
    , 56, 173, 171, 2, 237
270 DATA 176, 2, 133, 253, 56, 162, 255, 169, 0, 1
    06, 232, 228, 253, 208
280 DATA 250, 141, 180, 2, 174, 170, 2, 224, 3, 24
    0, 10, 160, 0, 177
290 DATA 251, 13, 180, 2, 145, 251, 96, 56, 169, 2
    55, 237, 180, 2, 141
300 DATA 180, 2, 160, 0, 177, 251, 45, 180, 2, 145
    , 251, 96,-1
500 REM ** USER ROUTINE **
501 REM GRAPHS SINE CURVE
505 POKE 53265, PEEK(53265)OR2↑5:REM ** SET BIT MA
    P MODE
510 POKE680,96:POKE681,92:REM ** SET POINTERS FOR
    {SPACE}UTILITY
515 POKE 53272, 120:POKE 56576, 2:REM ** SET UP VI
    C II MEMORY
520 POKE 55, 0:POKE 56, 60:CLR:REM ** PROTECTS BIT
     MAP FROM BASIC PROGRAM
530 B=32768:REM ** SET BASE ADDRESS OF UTILITY
540 SYS B,0: SYS B,1,16:REM ** CLEAR SCREEN AND SE
    T COLOR
550 FOR X=0 TO 6 STEP .05 :Y=SIN(X):REM ** GET VAL
    UE FOR SINE CURVE
560 X1=X*50:Y=Y*50:REM ** ENLARGE GRAPH SIZE
570 Y=100-Y:SYS B,2,X1,Y:REM ** GRAPH POINT
580 NEXT X:REM ** GRAPH NEXT
590 GOTO 590
600 REM ** EXIT WITH BREAK/RESTORE
```

Instant Art

Bob Urso

Both of these Commodore 64 graphics programs — one random, the other user-controlled — create impressive, handsome designs.

Anyone seeing your 64 while you're running one of these two programs might think that you've just looted the Museum of Modern Art. Each program lets you create colorful and expressive graphics on your Commodore 64.

Program 1 is a totally random graphics routine. Color, direction, and symbol selection are done in lines 30-89. POKEing in the symbol and updating its position for the next cycle are handled by line 90. Lines 95 and 96 limit the design to the screen area.

The time (line 11) is set at 1000 to clear the screen after it fills up a bit. You can increase T to let your design become more complicated; or you can eliminate lines 11 and 99-120, and the graphics will fill your screen until the next power outage.

The second program is called "Sketch-0"; it lets *you* do the designing. You can change the colors by pressing the color keys without having to press CTRL. The symbol select keys are grouped to the left so that they do not interfere with your direction selection keys.

You can move in eight directions, allowing for diagonal, as well as horizontal and vertical, lines. Once you press a direction key, the design will continue to print in that direction until it reaches the edge of the screen, or until you press any of the other keys to stop it.

It's doubtful that you'll ever make a Rembrandt jealous, but you should be more than rewarded for the short time it takes to type these programs.

Program 1. Random Graphics Routine

```
10 REM RANDOM{2 SPACES}DOODLE
11 T=1000
15 PRINT"{CLR}"
17 POKE53280,0:POKE53281,0
20 P=1024+INT(RND(1)*999)+1:G=P+54272
30 Z=INT(5*RND(1))+1
```

```
40 IFZ=1THENS=81
41 IFZ=2THENS=64
42 IFZ=3THENS=84
43 IFZ=4THENS=102
44 IFZ=5THENS=160
45 K=INT(8*RND(1))+1
50 IFK=1THENC=9
51 IFK=2THENC=1
52 IFK=3THENC=2
53 IFK=4THENC=3
54 IFK=5THENC=4
55 IFK=6THENC=5
56 IFK=7THENC=6
57 IFK=8THENC=7
80 D=INT(8*RND(1))+1
81 IFD=1THENR=-39
82 IFD=2THENR=-40
83 IFD=3THENR=-41
84 IFD=4THENR=-1
85 IFD=5THENR=1
86 IFD=6THENR=39
87 IFD=7THENR=40
88 IFD=8THENR=41
89 M=INT(40*RND(1))+1
90 FORZ=1TOM:POKEP,S:POKEG,C:P=P+R
95 IFP<=1024THENP=P-R
96 IFP>=2023 THEN P=P-R
97 G=P+54272
99 T=T-1
100 IFT=0THENGOTO10
110 PRINT"TIME";T
120 PRINT"{3 UP}"
1101 NEXTZ
1110 GOTO30
```

Program 2. Sketch-0

```
10 REM SKETCH-0
20 P=1524:S=160:C=1
90 POKE53280,0:POKE53281,0
95 GOTO1000
99 PRINT"{CLR}"
100 G=P+54272
200 POKE P,S :POKEG,C
300 GET G$:IFA$<>G$ANDG$<>""THENA$=G$
310 IFA$="I"THENP=P-40
320 IFA$="U"THENP=P-41
330 IFA$="O"THENP=P-39
340 IFA$="J"THENP=P-1
```

```
350 IFA$="K"THENP=P+1
360 IFA$="N"THENP=P+39
365 IFA$="M"THENP=P+40
370 IFA$=","THENP=P+41
380 IFA$="1"THENC=0
390 IFA$="2"THENC=1
400 IFA$="3"THENC=2
410 IFA$="4"THENC=3
420 IFA$="5"THENC=4
430 IFA$="6"THENC=5
440 IFA$="7"THENC=6
450 IFA$="8"THENC=7
460 IFA$="Q"THENS=81
470 IFA$="A"THENS=64
480 IFA$="Z"THENS=66
490 IFA$="W"THENS=102
500 IFA$="S"THENS=160
510 FORZ=1024TO1984STEP40:IFP=ZTHENP=P+1
530 IFP<1024THENP=P+40
540 IFP>2023THENP=P-40
550 GOTO 100
1000 PRINT"{CLR}":PRINT"{2 DOWN}{5 SPACES}DOODLE":
     PRINT"{DOWN}"
1010 PRINT"HERE ARE THE SYMBOLS YOU CAN PRINT"
1020 PRINT"{3 SPACES}PRESS Q FOR Q"
1021 PRINT"{3 SPACES}PRESS A FOR C"
1022 PRINT"{3 SPACES}PRESS Z FOR B"
1023 PRINT"{3 SPACES}PRESS W FOR {+}"
1024 PRINT"{3 SPACES}PRESS S FOR {RVS} {OFF}"
1030 PRINT"{GRN}TO CHANGE COLORS PRESS 1 THRU 8"
1040 PRINT"FOR THE COLOR INDICATED ON THE KEY":PRI
     NT"{DOWN}"
1070 PRINT"{7}TO MOVE YOUR SYMBOL PRESS"
1080 PRINT"{10 SPACES}U{2 SPACES}I{2 SPACES}O"
1090 PRINT"{11 SPACES}M ↑ N"
1100 PRINT"{10 SPACES}J← Q *K"
1110 PRINT"{11 SPACES}N B M"
1120 PRINT"{10 SPACES}N{2 SPACES}M{2 SPACES},"
1130 PRINT"{PUR}TO STOP SYMBOL PRESS ANY COLOR KEY
     "
1150 PRINT"{7}FINISHED WITH INSTRUCTIONS? PRESS
     {SPACE}Y"
1160 INPUTR$:IF R$="Y" GOTO 99
```

Chapter Two

Extended Background Color Mode

Sheldon Leemon

The extended background color mode can be a very useful tool when you want to create colorful displays. As well as discussing how to use this mode, this article also includes a short program to help you select good color combinations.

It is common knowledge that you can individually select the foreground color of each letter on the Commodore 64 text screen. Less well-known is the fact that you can individually select background colors as well. This is made possible by the 64's extended background color mode. Though this display mode is not mentioned at all in the *User's Guide,* and is dealt with very briefly in the *Programmer's Reference Guide,* it is well worth a closer examination. Let's take a look at how it is used and how it differs from the standard text display mode.

Normally, there are 256 character shapes that may be displayed on screen. You can see them either by using the PRINT statement or by POKEing a display code from 0 to 255 into screen memory, and a color code from 0 to 15 into color memory (for example, if you POKE 1024,1 and POKE 55296,1 a white letter A appears in the top-left corner of the screen). The background color of the screen is determined by Background Color Register 0, at 53281. You can change this background color by POKEing a new value to 53281. For example, POKE 53281,0 creates a black background.

When the extended background color mode is activated, however, the number of character shapes that may be displayed is reduced to 64; only the first 64 shapes found in the table of screen display codes (Appendix G) can be displayed on the screen. This group includes the letters of the alphabet, numerals, and punctuation marks.

If you try to display on the screen a character having a higher display code, the shape that will be displayed will be from the first group of 64, but the character's background color will no longer be determined by the register at 53281. Instead, it will be determined by one of the other background color registers. Characters having display codes 64-127, will take their background color from register 1, at location 53282. These characters include shifted alphabetic and other graphics characters. Those with codes 128-191 will have their background color determined by register 2, at 53283. These include the reversed numbers, letters, and punctuation marks. Finally, characters with codes 192-255 will use register 3, at 53284. These are the reversed graphics characters.

Let's try an experiment to see just how this works. First, we will print four letters on the screen:

```
FOR I=Ø TO 3:POKE 123Ø+(I*8),I*64+1:POKE 555Ø2+(I*
  8),1:NEXT
```

Four white letters should appear on the screen, an A, a shifted A, a reversed A, and a reversed, shifted A, all on a blue background. Next, we will put colors in the other background color registers:

```
POKE 53282,Ø:POKE 53282,2:POKE 53284,5
```

This sets these registers to black, red, and green, respectively. Finally, we will activate extended color mode. This is done by setting bit 6 of the VIC-II register at location 53265 to a 1. Therefore, to turn this mode on, we use the statement:

```
POKE 53265,PEEK(53265) OR 64
```

You will notice that two things happened. First, all the letters took the same shape, that of the letter A. Second, each took the background color of a different color register. To get things back to normal, turn off extended color mode with this statement:

```
POKE 53265,PEEK(53265) AND 191
```

Extended color mode can be a very useful enhancement for your text displays. It allows the creation of windows, which, because of their different background colors, make different bodies of text stand out as visually distinct from one another. For example, a text adventure program could have one window to display the player's current location, one to show an inventory of possessions, and one to accept commands for the next move. A window can be flashed to draw attention to a particular message at certain times just by POKEing a new value to the color register. And by varying the foreground color, either the window or the message could be made to vanish and reappear later.

Overcoming the Limitations

There are, however, a couple of problems involved in using these windows. The character shape that you want to use may not have a screen code of less than 64. In that case, the solution would be to define your own character set, in which the shape you want is within the first group of 64.

Another problem is that characters within a PRINT statement in your program listing are not always going to look the same on screen. Having to figure out what letter to print to get the character 4 can be very inconvenient. The easiest solution to this problem may be to have a subroutine do the translation for you. Since letters will appear normally in window 1, and window 3 characters are simply window 1 characters reversed, you will have problems only with characters in windows 2 and 4. To convert these characters, put your message into the string A$, and use the following subroutine:

```
500 B$="":FOR I=1 TO LEN(A$):B=ASC(MID$(A$,I,1))
510 B=B+32:IF B<96 THEN B=B+96
520 B$=B$+CHR$(B):NEXT I:RETURN
```

This subroutine converts each letter to its ASCII equivalent, adds the proper offset, and converts it back to part of the new string, B$. When the conversion is complete, B$ will hold the characters necessary to PRINT that message in window 2. For window 4, PRINT CHR$(18); B$; CHR$(146). This will turn reverse video on before printing the string, and turn it off afterwards.

One other thing you will have to watch is positioning of the cursor prior to using a PRINT statement, to make sure that you print within the window. Horizontal positioning is easy; you can use the TAB statement to move the cursor to the proper column. Vertical positioning is a little trickier, as there is no specific statement to handle it. One solution is to home the cursor and print a number of cursor down characters. An easy way of doing this is to create a string array, with each string containing the cursor home character, and enough cursor down characters to land it on the correct line. The statement:

```
DIM RO$(25):RO$(Ø)=CHR$(19):FOR I=1 TO 24:RO$(I)=R
  O$(I-1)+CHR$(17):NEXT
```

produces such an array. If you want to print a message on row 10, you merely PRINT RO$(10);"HELLO."

Some Practical Examples

A practical demonstration of the technique for setting up windows is given in Program 1. The program sets up three windows and shows them flashing, appearing and disappearing.

Program 2 helps with another practical problem: what colors to select for foreground and background to create the proper contrast for good legibility. This is a much greater problem on the 64 than on the VIC, where the letters are much larger. Commodore includes a chart on page 152 of the *Programmer's Reference Guide* which shows which combinations are best, but such a chart cannot substitute for your own firsthand observation. Therefore, with the help of a little machine language magic, Program 2 sets up the visual equivalent of such a chart on screen. It displays all 256 combinations of background and foreground colors simultaneously. Background colors run from 0 on the top line to 15 on the bottom, and foreground colors go from 0 at the left to 15 at the right. This is accomplished by the use of the raster register, which tells the machine language program what line is currently being scanned, so it knows when to change the background color in middisplay. Because the program loops continuously, the only way to break out of it is to hit the STOP and RESTORE keys together. Be sure to SAVE this program before you run it.

Program 1. Windows

```
1 REM{3 SPACES}*** WINDOWS{3 SPACES}****
5 DIM RO$(25):RO$(Ø)=CHR$(19):FOR I=1 TO 24:RO$(I)
  =RO$(I-1)+CHR$(17):NEXT
1Ø POKE 53265,PEEK(53265) OR 64
2Ø POKE 5328Ø,Ø: POKE 53281,Ø:POKE 53282,1:POKE 53
   283,2:POKE 53284,13
25 OP$=CHR$(16Ø):FOR I=1 TO 4:OP$=OP$+OP$:NEXTI:PR
   INTCHR$(147);RO$(3);
3Ø FOR I=1 TO1Ø:PRINTTAB(1);CHR$(18);"{15 SPACES}"
   ;TAB(23);OP$:NEXT
4Ø PRINT CHR$(146):PRINT:PRINT:FOR I=1 TO 4:PRINTO
   P$;OP$;OP$;OP$;OP$;:NEXTI
5Ø PRINT RO$(5);CHR$(5);CHR$(18);TAB(2);"A RED WIN
   DOW"
6Ø PRINT CHR$(18);TAB(2);"COULD BE USED"
```

```
70 PRINT CHR$(18);TAB(2);"FOR ERROR"
80 PRINT CHR$(18);TAB(2);"MESSAGES"
100 A$="A GREEN WINDOW":GOSUB 300:PRINT RO$(5);CHR
    $(144);CHR$(18);TAB(24);B$
110 A$="COULD BE USED":GOSUB 300:PRINTTAB(24);CHR$
    (18);B$
120 A$="TO GIVE":GOSUB 300:PRINTTAB(24);CHR$(18);B
    $
130 A$="INSTRUCTIONS":GOSUB 300:PRINTTAB(24);CHR$(
    18);B$
140 PRINT CHR$(31);RO$(19);
150 A$="{2 SPACES}WHILE THE MAIN WINDOW COULD BE U
    SED":GOSUB300:PRINT B$
160 A$="{2 SPACES}FOR ACCEPTING COMMANDS.":GOSUB30
    0:PRINT B$
170 FOR I=1 TO 5000:NEXT I: POKE 53284,0
180 FOR I=1 TO 5:FOR J=1 TO 300:NEXT J:POKE 53282,
    15
190 FOR J=1 TO 300:NEXT J:POKE 53282,1
200 NEXT I: POKE 53283,-2*(PEEK(53283)=240):POKE 5
    3284,-13*(PEEK(53284)=240)
210 GOTO 180
300 B$="":FOR I=1TOLEN(A$):B=ASC(MID$(A$,I,1))
310 B=B+32:IFB<96THENB=B+96
320 B$=B$+CHR$(B):NEXTI:RETURN
```

Program 2. Color Chart

```
20 REM ***     COLOR CHART     ***
30 REM
40 FOR I=49152 TO 49188: READ A: POKE I,A
   : NEXT
50 PRINT CHR$(147):FOR I=1024 TO I+1000:
   {SPACE}POKE I,160: POKE I+54272,11:NEX
   TI
60 FOR I=0 TO 15: FOR J=0 TO 15
70 P=1196+(40*I)+J: POKE P,J+1: POKE P+54
   272,J: NEXT J,I
80 SYS 12*4096
100 DATA 169,90,133,251,169,0,141,33,208,
    162,15,120,173,17,208,48,251,173,18,2
    08
110 DATA 197,251,208,249,238,33,208,24,10
    5,8,133,251,202,16,233,48,219
```

Chapter Two

Mixing Graphics Modes

Sheldon Leemon

It's possible to have several different graphics modes simultaneously on the 64 screen. Program 1 shows you how to divide the display into three zones: high resolution, regular text, and multicolor bitmap mode. Program 2 uses the same utility program, but creates entirely different effects. The screen displays all three text modes: regular, extended background color, and multicolor.

This graphics technique provides you with significant control over what appears on your screen. For example, you can switch modes with simple POKEs. Although there's plenty of technical information here for advanced programmers, the author has provided instructions and example programs which beginners can follow. Everyone can take advantage of these important techniques.

The *Commodore 64 Programmer's Reference Guide* hints that more than one graphics mode may be displayed on the screen at once. When it comes time to explain how it can be done, however, the *Guide* states only that you must set a raster interrupt for the screen line where you want a different type of display to start, set the VIC-II chip for the new mode during that interrupt, and then set up another interrupt to change the mode back a little farther down the display. This explanation might be clear to advanced machine language programmers, but it leaves a lot of others in the dark.

In this tutorial, we'll look at some examples of raster interrupts that can be easily used by BASIC programmers to create split-screen displays and other effects. We'll also discuss, in more detail, how machine language programmers can use the raster interrupt capability.

The Interrupt

The most obvious way to start our discussion is by explaining what an interrupt is. An interrupt is a signal given to the microprocessor (the brains of the computer) that tells it to stop executing its machine language program (for example, BASIC itself is a machine language program) and to work on another program for a short time, perhaps only a fraction of a second. After finishing the interrupt program, the computer goes back to executing the main program, just as if there had never been a detour.

There are several ways to cause such an interrupt on the 64. Pressing the RESTORE key causes an interrupt, and if the STOP key is also pressed, the interrupt routine clears the screen and restores the computer to its normal state. There are internal timers on the CIA Input/Output chips that can each generate interrupts. One of these timers is set by the operating system to interrupt every 1/60 second, and the interrupt routine that is called is used to check the keyboard and to update the jiffy clock which is used by TI and TI$. In addition, the VIC-II chip can also interrupt normal program execution when one of a number of events related to the graphics display occurs. One of these is called a raster interrupt.

On a normal TV display, a beam of electrons (raster) scans the screen, starting in the top left-hand corner and moving in a straight line to the right, lighting up appropriate parts of the screen line on the way. When it comes to the right edge, the beam moves down a line and starts again from the left. There are 263 such lines that are scanned by the Commodore 64 display, 200 of which form the visible screen area. This scan updates the complete screen display 60 times every second.

The VIC-II chip has memory registers that keep track of the line that the raster is scanning at any given moment. Since the line number can be greater than 255, one register is not enough to do the job. Therefore, the part of the number that is less than 256 is kept in location 53266 ($D012), and if bit 7 of location 53265 ($D011) is set to 1, 256 is added to that number to arrive at the correct scan line. Of course, since these numbers change 15,780 times per second, a BASIC program executes far too slowly to read the registers and take effective action based on their contents. Only a machine language program has the speed to accomplish something with a particular raster scan line, and even it may not be quick enough to change the display without some slight, but visible, disruption.

The raster registers have two functions. When read, they tell what line is presently being scanned. But when written to, they designate a particular scan line as the place where a raster interrupt will occur. If the raster interrupt is enabled, the interrupt program will be executed at the exact moment that the raster beam reaches that line. This allows the user to reset any of the VIC-II registers at any point in the display and thus change character sets, background color, or graphics mode for only a part of the screen display.

Setting up a raster interrupt program is admittedly not a job for a beginning programmer, but with the following step-by-step explanation, most machine language programmers should be able to write such a routine. Those with no machine language experience should read the explanation in order to get a general idea of what is taking place. Afterwards, we'll see how to use the example interrupt routine even if you don't know anything about machine language programming.

Writing a Raster Interrupt

When you have finished writing the machine language routine that you want the interrupt to execute, the steps required to set up the raster interrupt are:

1. Set the interrupt disable flag in the microprocessor's status register with an SEI instruction. This will disable all interrupts and prevent the system from crashing while you are changing the interrupt vectors.

2. Enable the raster interrupt. This is done by setting bit 0 of the VIC-II chip interrupt enable register at location 53274 ($D01A) to 1.

3. Indicate the scan line on which you want the interrupt to occur by writing to the raster registers. Don't forget that this is a nine-bit value, and you must set both the low byte (in location 53266) and the high bit (in the register at 53265) in order to insure that the interrupt will start at the scan line you want it to, and not 256 lines earlier or later.

4. Let the computer know where the machine language routine that you want the interrupt to execute starts. This is done by placing the address in the interrupt vector at locations 788-789 ($314-$315). This address is split into two parts, a low byte and a high byte, with the low byte stored at 788. To calculate the two values for a given address AD, you may use the formula HIBYTE = INT(AD/256) and LOWBYTE = AD-(HIBYTE*256). The value LOWBYTE would go into location 788, and the value HIBYTE would go into location 789.

5. Reenable interrupts with a CLI instruction, which clears the interrupt disable flag on the status register.

When the computer is first turned on, the interrupt vector is set to point to the normal hardware timer interrupt routine, the one that advances the jiffy clock and reads the keyboard. Since this interrupt routine uses the same vector as the raster interrupt routine, it is best to turn off the hardware timer interrupt by putting a value of 127 in location 56333. If you want the keyboard and jiffy clock to function normally while your interrupt is enabled, you must preserve the contents of locations 788 and 789 before you change them to point to your new routine. Then you must have your interrupt routine jump to the old interrupt routine exactly once per screen refresh (every 1/60 second).

Another thing that you should keep in mind is that at least two raster interrupts are required if you want to change only a part of the screen. The interrupt routine must not only change the display, but it must also set up another raster interrupt that will change it back.

Program 1 is a BASIC program that uses a raster-scan interrupt to divide the display into three sections. The first 80 scan lines are in high-resolution bitmap mode, the next 40 are regular text, and the last 80 are in multicolor bitmap mode. The screen will split this way as soon as a SYS to the routine that turns on the interrupt occurs, and the display will stay split even after the program ends. Only if you hit the STOP and RESTORE keys together will the display return to normal.

Program 2 shows how a completely different split screen can be set up using the same machine language program. The DATA statements for the interrupt routine are the same as for Program 1, except for the tables starting at line 49264. By changing these tables, we now have a display that shows all three text modes: regular, extended background color, and multicolor. Upper- and lowercase text are mixed, and each area has a different background color. This program also shows that you can change the table values during a program by POKEing the new value into the memory location where those table values are stored. In that way, you can, for example, change the background color of any of the screen parts while the program is running.

Once you know how to use all the graphics features that the VIC-II chip makes available, the sample interrupt program should enable you to combine several different display modes on a single screen, so that you can take maximum advantage of the 64's graphics power.

Control Registers

The interrupt uses a table of values that are POKEd into four key locations during each of the three interrupts, as well as values to determine at what scan lines the interrupts will occur. The locations affected are Control Register 1, Control Register 2, the Memory Control Register, and Background Color 0.

Control Register 1 (at location 53265) allows the selection of extended background color text mode, bitmap mode, screen blanking, and 24 or 25 rows of text. Control Register 2, at 53270, controls the selection of multicolor mode, and of a 38- or 40-column display. The Memory Control Register (53272) allows you to select which portion of memory will be used for the video display (screen memory), and which for the data that defines the shape of text characters. Background Color Register 0 (53281) controls the background color in text mode. More detailed information about the bit assignments of these locations can be found in Appendix O of the *Commodore 64 User's Guide* and in the *Programmer's Reference Guide.*

The data for the interrupt routine is contained in lines 49152-49276 of Program 1. Each of these line numbers corresponds to the location where the first data byte in the statement is POKEd into memory. If you look at lines 49264-49276 of the BASIC program, you will see REMark statements that explain which VIC-II registers are affected by the DATA statements in each line. The numbers in these DATA statements appear in the reverse order in which they are put into the VIC register. For example, line 49273 holds the data that will go into Control Register 2. The last number, 8, is the one that will be placed into Control Register 2 while the top part of the screen is displayed. The first number, 24, is placed into Control Register 2 during the bottom part of the screen display and changes that portion of the display to multicolor mode.

The only tricky part in determining which data byte affects which interrupt comes in line 49264, which holds the data that determines the scan line at which each interrupt will occur. Each DATA statement entry reflects the scan line at which the *next* interrupt will occur. The first item in line 49264 is 49. Even though this is the entry for the third interrupt, this number corresponds to the top of the screen (only scan lines 50-249 are visible on the display). That is because after the third interrupt, the next to be generated is the first interrupt, which occurs at the top of the screen. Likewise, the last data item of 129 is used during the first interrupt to start the next one at scan line 129, in the middle of the screen. Try experimenting with these values to see what results you come up with. For example, if you change the number 170 to 210, you will increase the text area by 5 lines (40 scan lines).

Changing Effects

By changing the values in the data tables, you can alter the effect of each interrupt. Change the 20 in line 49276 to 22, for example, and you will get lowercase text in the middle of the screen. Change the first 8 in line 49273 to 24, and you will get multicolor text in the center window. Each of these table items may be used in exactly the same way that you would use the corresponding register, in order to change background color, to obtain text or bitmap graphics, regular or multicolor modes, screen blanking, or extended background color mode.

Program 1. Text with Graphics

```
10 FOR I=49152 TO 49278: READ A:POKE I,A:NEXT:SYS1
   2*4096
20 PRINT CHR$(147):FOR I=0 TO 8:PRINT:NEXT
30 PRINT"THE TOP AREA IS HIGH-RES BIT MAP MODE"
40 PRINT:PRINT"THE MIDDLE AREA IS ORDINARY TEXT "
50 PRINT:PRINT"THE BOTTOM AREA IS MULTI-COLOR BIT
   {SPACE}MAP"
60 FORG=1024 TO 1383:POKEG,114:NEXT:FORG=1384 TO 1
   423:POKE G,6:NEXT
70 FORG=1664 TO 2023:POKEG,234:NEXT
80 FORG=55936TO56295:POKEG,13:NEXT
90 FOR I=8192 TO 11391:POKE I,0:POKE I+4800,0:NEXT
100 BASE=2*4096:BK=49267
110 H=40:C=0:FORX=0TO319:GOSUB150:NEXT
120 H=160:C=0:FORX=0TO319STEP2:GOSUB150:NEXT:C=40:
    FORX=1TO319STEP2:GOSUB150:NEXT
130 C=80:FOR X=0 TO 319 STEP2:W=0:GOSUB150:W=1:GOS
    UB150:NEXT
140 GOTO 140
150 Y=INT(H+20*SIN(X/10+C)):CH=INT(X/8):RO=INT(Y/8
    ):LN=YAND7
160 BY=BASE+RO*320+8*CH+LN:BI=ABS(7-(XAND7)-W)
170 POKEBY,PEEK(BY)OR(2↑BI):RETURN
49152 DATA 120, 169, 127, 141, 13, 220
49158 DATA 169, 1, 141, 26, 208, 169
49164 DATA 3, 133, 251, 173, 112, 192
49170 DATA 141, 18, 208, 169, 24, 141
49176 DATA 17, 208, 173, 20, 3, 141
```

```
49182 DATA 11Ø, 192, 173, 21, 3, 141
49188 DATA 111, 192, 169, 5Ø, 141, 2Ø
49194 DATA 3, 169, 192, 141, 21, 3
492ØØ DATA 88, 96, 173, 25, 2Ø8, 141
492Ø6 DATA 25, 2Ø8, 41, 1, 24Ø, 43
49212 DATA 198, 251, 16, 4, 169, 2
49218 DATA 133, 251, 166, 251, 189, 115
49224 DATA 192, 141, 33, 2Ø8, 189, 118
4923Ø DATA 192, 141, 17, 2Ø8, 189, 121
49236 DATA 192, 141, 22, 2Ø8, 189, 124
49242 DATA 192, 141, 24, 2Ø8, 189, 112
49248 DATA 192, 141, 18, 2Ø8, 138, 24Ø
49254 DATA 6, 1Ø4, 168, 1Ø4, 17Ø, 1Ø4
4926Ø DATA 64, 76, 49, 234
49264 DATA 49, 17Ø, 129 :REM SCAN LINES
49267 DATA Ø, 6, Ø:REM BACKGROUND COLOR
4927Ø DATA 59, 27,59:REM CONTROL REG. 1
49273 DATA 24, 8, 8:REM CONTROL REG. 2
49276 DATA 24, 2Ø, 24:REM MEMORY CONTROL
```

Program 2. The Three Text Modes

```
1Ø FOR I=49152 TO 49278: READ A:POKE I,A:NEXT:SYS1
   2*4Ø96
2Ø PRINTCHR$(147)CHR$(5):POKE 5328Ø,Ø
3Ø POKE 5328Ø,Ø:POKE 53282,6:POKE 53283,5:POKE 532
   84,4
4Ø PRINT:PRINT"THIS IS MULTI-COLOR TEXT MODE"
5Ø PRINT:PRINT"FOUR-COLOR CHARACTERS ARE HARD TO R
   EAD"
6Ø PRINT:PRINT CHR$(15Ø)"ABCDEFGHIJKLMNOPQRSTUVWXY
   Z123456789Ø"
7Ø PRINT:PRINT:PRINT:PRINT CHR$(28)"THIS IS NORMAL
    TEXT MODE..."
8Ø PRINT:PRINT"NOTHING FANCY GOING ON HERE":PRINT:
   PRINT:PRINT
9Ø PRINTCHR$(144)"{6 SPACES}EX{RVS}TE{OFF}ND{RVS}E
   D{OFF} BA{RVS}CK{OFF}GR{RVS}OU{OFF}ND{RVS} C
   {OFF}OL{RVS}OR{OFF} MO{RVS}DE{OFF}{UP}"
1ØØ PRINT:PRINT"LETS YOU USE DIFFERENT BACKGROUND
    {SPACE}COLORS"
11Ø PRINT "{RVS}LETS YOU USE DIFFERENT BACKGROUND
    {SPACE}COLORS"
12Ø PRINT"LETS{SHIFT-SPACE}YOU{SHIFT-SPACE}USE
    {SHIFT-SPACE}DIFFERENT{SHIFT-SPACE}BACKGROUND
    {SHIFT-SPACE}COLORS"
13Ø PRINT "{RVS}LETS{SHIFT-SPACE}YOU{SHIFT-SPACE}U
    SE{SHIFT-SPACE}DIFFERENT{SHIFT-SPACE}BACKGROUN
    D{SHIFT-SPACE}COLORS";
14Ø FORS=ØTO3ØØØ:NEXT
```

```
150 FORS=49267TO49269:POKES,RND(1)*16:FOR I=1 TO 2
    000:NEXT I,S:GOTO 140
49152 DATA 120, 169, 127, 141, 13, 220
49158 DATA 169, 1, 141, 26, 208, 169
49164 DATA 3, 133, 251, 173, 112, 192
49170 DATA 141, 18, 208, 169, 24, 141
49176 DATA 17, 208, 173, 20, 3, 141
49182 DATA 110, 192, 173, 21, 3, 141
49188 DATA 111, 192, 169, 50, 141, 20
49194 DATA 3, 169, 192, 141, 21, 3
49200 DATA 88, 96, 173, 25, 208, 141
49206 DATA 25, 208, 41, 1, 240, 43
49212 DATA 198, 251, 16, 4, 169, 2
49218 DATA 133, 251, 166, 251, 189, 115
49224 DATA 192, 141, 33, 208, 189, 118
49230 DATA 192, 141, 17, 208, 189, 121
49236 DATA 192, 141, 22, 208, 189, 124
49242 DATA 192, 141, 24, 208, 189, 112
49248 DATA 192, 141, 18, 208, 138, 240
49254 DATA 6, 104, 168, 104, 170, 104
49260 DATA 64, 76, 49, 234
49264 DATA 49, 177, 113 :REM SCAN LINES
49267 DATA 2, 7, 6:REM BACKGROUND COLOR
49270 DATA 91, 27,27:REM CONTROL REG. 1
49273 DATA 8, 8, 24:REM CONTROL REG. 2
49276 DATA 20, 22, 20:REM MEMORY CONTROL
```

High-Resolution Sketchpad

Chris Metcalf

High-resolution graphics can be detailed and spectacular. Yet creating them can be difficult. "High-Resolution Sketchpad" makes the task of creating high-resolution graphics easy. Once you create your masterpiece, it's easy to save it to disk or tape for use in your programs.

The magic words *high resolution* were part of what prompted me to buy a Commodore 64. No doubt you too were influenced by the idea of having a 320 x 200 dot map of picture elements on the screen, a total of 16 colors to be spread about on the screen, and the ability to mix up to four colors within each 8 x 8 pixel area.

Unfortunately, it is very difficult to employ these powerful features. The Commodore 64 lacks BASIC commands for high resolution (such as Atari BASIC's PLOT, POSITION, DRAWTO, and LOCATE), but does have a pair of high-resolution bitmapping modes with great potential. The only difficulty is in accessing them from BASIC.

BASIC provides only minimal control over the graphics. A series of POKEs is needed even to bring up the high-resolution graphics screen, then further POKEs are needed to clear the graphics page out for use. Once this has been accomplished, more POKEs are necessary to plot points on the screen and set their colors. This process is slow, tedious, and difficult.

High-Resolution Graphics

Elsewhere in this book can be found detailed descriptions of the high-resolution graphics modes, but a brief overview here might be useful. The actual bitmapping screen can be located at any of eight 8K areas in memory. The "Sketchpad" program uses 40960 to 48959 ($A000-$BF3F) for this screen. The color data is stored elsewhere in memory. In the standard high-resolution mode, color can come from any 1K block in the same 16K area of memory as the bitmap screen. Sketchpad uses the area from 35840 to 36839 ($8C00-$8FE7) for this floating color memory. In multicolor bitmap mode, further memory is needed to support the additional colors, and this color memory is fixed at 55296 to 56296 ($D800-$DB87).

On the 64, the high-resolution screen resembles 1000 programmable characters in its format. The first byte of the screen defines the eight pixels at the beginning of the top line. The following seven bytes define the first eight pixels of each following line. However, the next group of eight bytes is located not below but to the right of the initial eight. After 40 groups of 8 bytes (the equivalent of a line of programmable characters), the sequence repeats for the next 8 pixel lines.

In standard high-resolution mode both background color and pixel color are defined by the selectable 1K of color memory. The most significant nybble (four bits, half a byte) defines the color of all the pixels within one 8 x 8 pixel group (one "character"). The least significant nybble defines the background color in the same area (seen when a bit is 0).

Multicolor mode allows multiple colors within one 8 x 8 pixel area by assigning one of four colors to each possible combination of two bits. However, the result is that it takes both bits to define a single pixel. Each bit-pair takes its color from the corresponding byte on the floating color screen, the fixed screen, or the background color register as follows:

bit-pair	**source of its color code**
00	background color register (53280, $D021)
01	high nybble of floating color memory
10	low nybble of floating color memory
11	fixed color memory

As in standard high-resolution mode, the color memory provides separate color information for each group of 8 x 8 pixels. However, unlike standard high-resolution mode, all 00 bits are set from the register at 53281.

However, High-Resolution Sketchpad allows you to ignore most of these details. You should, however, understand why you cannot plot too many colors together. New colors simply change the color of all the appropriate pixels within each 8 x 8 area.

Typing in Sketchpad

The Sketchpad is designed to be used with BASIC programs in memory at the same time. The program itself starts at 36864 and runs up to 40095 ($9000 to $9C9F); various data tables run from

40192 to 40959 ($9D00-$9FFF) and 51968 to 52223 ($CB00-$CBFF). The floating color screen is at 35840, so the top of BASIC is set to 35839 by the Sketchpad. The bitmap screen area is from 40960 to 48959. Normally, the BASIC interpreter takes up this memory, but a POKE 1,54 instruction leaves this memory available for other uses. However, this POKE cannot be used directly from BASIC (as the interpreter will no longer be present) and can only be done from machine language.

This program is written entirely in machine language, so it is necessary to enter it using the Machine Language Editor (MLX), Appendix I.

Using the MLX program will make entering machine language programs much easier. Please read and understand the directions for using the MLX before attempting to type in the Sketchpad. The information needed to enter High-Resolution Sketchpad with the MLX is:

Starting address: 36864
Ending address: 40095

When you've finished your typing, be sure to use the MLX Save command to make a copy of your program on disk or tape.

Whenever you wish to use the program, enter LOAD "SKETCHPAD",8,1 for disk or LOAD "SKETCHPAD",1,1 for tape. To enter the program, type SYS 36864 and press RETURN. The following message should come up:

```
HIRES SKETCHPAD - BY CHRIS METCALF
MULTICOLOR MODE? N
```

and the cursor will blink on the N. At this point, enter either Y or N to determine whether you will use standard or multicolor bitmapping during the program run. If you enter nothing, the program aborts. Standard mode provides better resolution for more intricate designs, while multicolor is more useful for less detailed or more colorful displays.

Simple Graphics with the Sketchpad

Once you press RETURN, the bitmap screen should come up. A small turtle sprite in the center of the screen indicates where you are plotting. The first time you enter the program after turning on the computer, the display will be covered with random pixels and colors. Press SHIFT-CLR to clear out the screen. At any time you may press CTRL-Left Arrow (←) to leave the program.

The program has been designed so that either joystick or keyboard can control the turtle plotter. Joystick users can move the turtle with the joystick in Control Port 2, and can control various modes with the fire button. However, a number of keys have been defined for moving the turtle as well. The square of keys with Q, E, Z, and C as its corner points will steer the turtle in all eight directions.

	up/left	**up**	**up/right**	
	Q	**W**	**E**	
left	**A**		**D**	**right**
	Z	**X**	**C**	
	down/left	**down**	**down/right**	

The S key at the center of the square is used to return the turtle to its starting position at the center of the screen, and the HOME key puts the turtle at the top-left corner of the screen.

The first thing to experiment with is simple plotting. Press the space bar (or SHIFT-space bar) or the fire button on the joystick to enter plot mode. A dot will appear in the center of the turtle. Now you will draw a line wherever you go. To stop plotting and just move about, hit the space bar or the fire button again.

When you first enter the program, the turtle will be the same color that you were typing in before (the character color). To change this color, use the CTRL or Commodore key with the numbers one through eight. The CTRL colors are shown on the front of the numeric keys. The color you are using is plotted with every point you plot; therefore, if you try to plot in an 8 x 8 pixel area previously in a different color, the color of the pixel area will change to your plot color.

Multicolor Mode

This problem can be reduced by using multicolor mode. When you SYS 36864, enter Y for multicolor. You will see that the pixels you plot are in fact larger. However, you can now intermix colors freely. Each of the three types of plotting (bit-pairs 01, 10, and 11) and the erase mode (00) are represented by the function keys. The f1 key corresponds to 11, f3 to 10, and f5 to 01. When the program begins, you begin in f5. While using any one plot type, you are constrained by the same problem with colors that affects standard bitmap mode. However, the coloring of each of the three types is completely independent, so by changing between f1, f3, and f5, you can plot without affecting the colors in different plot types. The f7 key will put you in erase mode.

In standard bitmap mode, the same function keys can be used. In either mode, the plus and minus keys (and their shifted equivalents), which correspond to f5 and f7 respectively, can also be used. Normal plotting in standard hi-res is in f5 mode, the mode you begin in. The f1 key has been set to yield f5 when pressed in standard bitmap mode. The f7 mode has the same effect as in multicolor — it erases pixels without affecting the color of neighboring pixels. The f3 mode does not plot any pixels — instead, it changes the background color within each 8 x 8 square, without altering the pixel plot colors. Plotting in this mode can be a good way to familiarize yourself with the 8 x 8 pixel color setup.

Special Features

Changing the border and background colors can also be done from within the program. However, if you are in standard bitmap mode, the bitmap background color will not change until you press SHIFT-CLR. Border and background colors are changed with the joystick or the direction keys. To enter the color change mode, press the up-arrow (↑) key. Moving the joystick left and right or using the corresponding keys on the keyboard will change the border color. To change the background color (this will be immediately apparent only in multicolor mode), move the joystick up and down, or use the keys. To break out of this mode, press the fire button or any key other than those in the direction keys, and you will return to the main loop.

Moving by steps is another feature of this program. When you begin the program, you move one pixel at a time. However, whenever you press a number key or its shifted equivalent, you will begin to move that many pixels at a time. For example, if you want to do double-spaced plotting, press the 2 key; to move eight pixels at a time, press the 8 key. The same feature works in multicolor mode, but, because of the double-width pixels, odd numbers give somewhat peculiar results.

More Advanced Graphics Modes

More powerful options are available with the shifted function keys. The first option, known as the draw-from mode, is turned on and off with f2. When you press f2, the start point for the line-draw routine is assigned to your location. Now, as you move around, you will see a line connecting your turtle to the start point you have selected. This rubber-band line does not change the pixels around it. However, it does change the colors if you are in any of the plotting modes. Only in f7 or minus mode will no colors be plotted to the rubber-band line as you move about. Once the line is in a position that suits you, press the SHIFT key or the fire button, and a real line will be drawn in the color and plot mode you are using.

As you continue to move about and draw lines, the start point will remain where you initially assigned it. This allows you to create intricate abstract works simply by setting a spacing of three or four (or whatever you like), and moving around while holding down the SHIFT key or the fire button. The SHIFT-LOCK key can also be used. However, the space bar will still toggle on and off the simple plotting beneath the turtle. To terminate the draw-from mode, hit f2 again. Then, to assign your position as the new start point, press f2 yet again. Note that since the SHIFT key will draw a line, it is often helpful to use the Commodore logo key for normally shifted characters (e.g., SHIFT-CLR, SHIFT-f8), since it yields the same results.

The second mode is selected with the f4 key. This is the draw-to mode, which is very much similar to draw-from. However, in this mode, every time you press the fire button or the SHIFT key, the line is drawn and the line-draw start point automatically assigned to your current position. This provides the same effect as the Atari DRAWTO command. The draw-to mode allows you to draw figures more easily. Note that if you are in f4 mode and select f2, f4 will be cancelled and replaced by f2. The reverse is also true.

The third line-draw mode (f6) is useful primarily for making shaded figures. When you're in this mode, every time you press the SHIFT key or fire button, a line will be drawn to the right in the mode and color you are in until it encounters another pixel or the right-hand edge. This mode has no rubber-band effect.

You can also select where the draw-right will stop. Normally the line will stop when it encounters a pixel in the same mode, so it would erase a filled-in area to the right or draw right in f3 mode until it encountered an f3 bit-pair, and so forth, depending on your mode. However, the asterisk will toggle a variation. If you press the asterisk key once after beginning the program, the draw mode will search for any on pixel. Thus you can draw right in f1 mode and stop at an f3-mode pixel, creating a border of a different color. Pressing this key again will return you to the initial fill-to same mode.

Fill

One of the most powerful features of the program is called when you press the f8 key. This key activates a fill. This function will fill in any area bounded by pixels, or fill to the edges of the screen. This feature is also dependent on the asterisk to know what to fill to. Normally it fills to any pixel of the same type, but it can be toggled to fill to any on pixel, thus allowing differently colored borders in multicolor mode. The fill can and will escape from any shape in which there is a hole in the border, but it does not slip between diagonally separated pixels.

The Status Line

All of these modes are somewhat difficult to hold in mind. What with four plot modes, a plot/no-plot option, three kinds of lines, and a fill type (asterisk) toggle, things can get confused. This is especially true since fill-right has no rubber band, since plot-minus and no-plot appear the same, and since the multicolor plots are indistinguishable when in the same color. To help keep them all straight, a status line can be toggled on at the bottom of the screen by pressing and holding down the RETURN key.

The status line consists of four parts. The first indicates the mode you are in (f1, f3, f5, or f7). The second indicates whether your plotting is on or off (plotting or just moving about). The third displays the type of line-draw mode you are in (OFF, FROM, TO, or LINE), and the fourth tells the status of the asterisk mode (SAME is what you begin in; ANY means stop filling at any on pixel).

Input/Output for Sketchpad

The program is provided with a feature for loading and saving all the data that makes up the hi-res image. To access this feature, press the @ key. The program will ask whether you wish to Load or Save (note that only the first letter is significant). Any other answer will abort the process. Then you must specify the device number. The Datassette is 1, and disk drives can be either 8 (as most are) or 9. (Device 2, the RS-232 channel, can also be used, but modifications to the machine language will be necessary to include sending baud rate and other parameters.) No other devices are permitted. Finally, you will have to provide the name. If no name is given, the process will terminate. Now the turtle sprite will disappear for the duration of the Load or Save. When the process is finished, the turtle will return.

Disk input/output is simple. Specify L or S, 8, and the name. Make sure a disk is in the drive, and, most importantly, turned on and plugged in. *If the drive is not ready, the program will lock up.* In this case, RUN/STOP-RESTORE is all that can recover the program. No suffixes are necessary for disk Saves or Loads, but any prefixes you wish (such as "0:" or "@0:") will have to be included in the name. When the disk drive is finished, the error channel is read and displayed for about two seconds. Normally a "00,OK,00,00" is returned. Some common errors are:

62, FILE NOT FOUND — Loading a nonexistent file
63, FILE EXISTS — Save under another name or with "@0:"
64, FILE TYPE MISMATCH — Saving with "@0:" over a program file
72, DISK FULL — get a new disk or scratch some files
74, DRIVE NOT READY — the drive door is open; Save with "0:"

Any other error (particularly 21) indicates a disk malfunction of some sort. Refer to your 1541 manual.

Tape users don't have to contend with error messages. To Save or Load with tape, enter L or S, 1, and a name. However, it is a good idea to press PLAY or RECORD & PLAY before pressing RETURN for the last question. If you do so, the tape will send no messages. Messages cause unwanted color information to be put on the fixed color screen. Furthermore, if the message causes the display to scroll, the color screen will scroll with it, and throw off all the multicolor f1 color information (11). However, even this is by no means catastrophic. To avoid it, simply clear the screen before typing SYS 36864 to guard against messages. You can press RUN/STOP during the load or save and return directly to the Sketchpad program.

The high-resolution information is saved in a completely unique format. The first two bytes saved are the border and background colors. This is followed by the floating color screen data (1000 bytes), the fixed color screen data (another 1000 bytes), and, finally, the high-resolution screen. The screen is saved by a data-compaction technique. All nonzero bytes are output normally, but a zero flags a special mode: the next two bytes are the address of the following nonzero byte in low-byte, high-byte format. This allows the program to clear the intervening space quickly and load only the relevant picture data.

Chapter Two

Load/Save Subroutine

Program 2 is a subroutine to allow you to integrate Sketchpad designs into your own programs. The subroutine comes in three main parts: the data loader, the subroutine itself (at line 50000), and the machine language data. The data loader goes at the beginning of your program and simply reads the DATA statements into memory from 51676 to 51967 ($C9DC-$CAFF). The subroutine processes your request and calls the machine language.

To use the subroutine, load LS with either load or save (load = 0, save = 1), DV with the device (8 for disk, 1 for tape), and NM$ with the name of the file. Then GOSUB 50000. The BASIC subroutine is not, however, necessary; the machine language can be called on its own. To do so, POKE 2 with 0 for load or 1 for save. Then OPEN the appropriate type of file:

disk load: OPEN 1,8,2, "filename"
disk save: OPEN 1,8,2, "filename, S,W"
tape load: OPEN 1 or OPEN 1,1,0, "filename"
tape save: OPEN 1,1,1, "filename"

Finally, SYS 51676. For example, to load a picture ("DESIGN3") from disk:

```
POKE 2,0: OPEN 1,8,2, "DESIGN3":SYS 51676
```

Machine Language

Program 3 is the source code for the Sketchpad. The program can be entered using an assembler.

The source code is commented and is supplied for those interested in studying how the program works. Below is a list of the starting addresses of the major routines:

$9000	initialize; called only at the beginning of the program
$9167	main loop — keyboard input
$93BC	— joystick input
$945E	— move and plot
$9538	draw line subroutine
$96CE	fill area subroutine
$97C6	miscellaneous subroutines; raster interrupt
$992A	load and save subroutine
$9BA5	data

Program 1. High-Resolution Sketchpad

```
36864 :032,231,255,160,000,185,095
36870 :201,155,240,006,032,210,082
36876 :255,200,208,245,160,000,056
36882 :032,207,255,201,013,208,166
36888 :001,096,201,089,208,001,108
36894 :200,152,072,032,207,255,180
36900 :201,013,208,249,160,045,144
36906 :185,057,000,153,000,203,128
36912 :136,016,247,104,133,060,232
36918 :169,000,133,157,169,140,054
36924 :133,056,133,052,169,000,091
36930 :133,055,133,051,169,128,223
36936 :141,138,002,169,197,141,092
36942 :000,221,169,054,133,001,144
36948 :169,056,133,076,169,059,234
36954 :133,075,169,008,164,060,187
36960 :240,002,169,024,133,077,229
36966 :133,078,169,000,133,064,167
36972 :133,057,133,058,133,079,189
36978 :133,080,133,068,169,160,089
36984 :133,067,169,100,133,069,023
36990 :169,001,133,062,133,059,171
36996 :165,060,010,010,010,133,008
37002 :070,169,007,056,229,060,217
37008 :133,073,169,001,024,101,133
37014 :060,133,074,169,140,141,099
37020 :096,203,169,000,141,064,061
37026 :203,169,160,141,160,203,174
37032 :169,000,141,128,203,170,211
37038 :189,064,203,024,105,040,031
37044 :157,065,203,189,096,203,069
37050 :105,000,157,097,203,189,169
37056 :128,203,024,105,064,157,105
37062 :129,203,189,160,203,105,163
37068 :001,157,161,203,232,224,158
37074 :024,208,217,169,001,160,221
37080 :007,153,192,203,010,136,149
37086 :016,249,169,001,160,006,055
37092 :153,200,203,010,153,208,131
37098 :203,010,136,136,016,244,211
37104 :169,003,160,006,153,216,179
37110 :203,010,010,136,136,016,245
37116 :247,169,254,160,007,153,218
37122 :224,203,056,042,136,016,167
37128 :248,169,252,160,007,153,229
37134 :231,203,153,239,203,153,172
37140 :247,203,056,042,056,042,154
37146 :136,136,016,239,160,040,241
```

```
37152 :162,040,165,060,240,002,189
37158 :162,081,189,076,156,153,087
37164 :192,191,202,136,016,246,003
37170 :169,000,160,021,153,233,018
37176 :191,136,016,250,169,255,049
37182 :141,248,143,169,172,024,191
37188 :101,060,141,000,208,169,235
37194 :143,141,001,208,169,000,224
37200 :141,027,208,141,028,208,065
37206 :141,029,208,141,023,208,068
37212 :173,134,002,133,061,141,224
37218 :039,208,032,125,152,169,055
37224 :001,141,139,002,032,228,135
37230 :255,072,165,058,240,040,172
37236 :173,141,002,041,001,208,170
37242 :007,173,000,220,041,016,067
37248 :208,006,032,056,149,076,143
37254 :156,145,165,058,201,003,094
37260 :240,014,169,001,133,079,008
37266 :032,056,149,032,056,149,108
37272 :169,000,133,079,104,208,077
37278 :003,076,188,147,072,032,164
37284 :021,152,104,164,066,240,143
37290 :003,076,217,147,201,032,078
37296 :240,004,201,160,208,009,230
37302 :165,057,073,001,133,057,156
37308 :076,188,147,201,083,208,067
37314 :015,169,000,133,068,169,236
37320 :160,133,067,169,100,133,194
37326 :069,076,188,147,201,019,138
37332 :208,011,169,000,133,067,032
37338 :133,068,133,069,076,188,117
37344 :147,201,043,240,004,201,036
37350 :219,208,011,169,001,133,203
37356 :059,169,008,133,070,076,239
37362 :188,147,201,045,240,004,043
37368 :201,221,208,005,169,000,028
37374 :076,235,145,201,140,208,235
37380 :006,032,206,150,076,188,150
37386 :147,201,137,208,023,165,123
37392 :058,041,001,073,001,133,067
37398 :058,165,067,133,081,165,179
37404 :068,133,082,165,069,133,166
37410 :083,076,094,148,201,138,006
37416 :208,011,165,058,041,002,013
37422 :073,002,133,058,076,023,155
37428 :146,201,139,208,018,165,161
37434 :058,208,007,169,003,133,124
37440 :058,076,188,147,169,000,190
```

```
37446 :133,058,076,188,147,201,105
37452 :042,208,009,165,080,073,141
37458 :001,133,080,076,188,147,195
37464 :201,094,208,010,036,197,066
37470 :080,252,032,198,151,076,115
37476 :103,145,201,013,208,091,093
37482 :032,055,152,160,039,185,217
37488 :000,156,153,192,143,136,124
37494 :016,247,164,059,185,040,061
37500 :156,141,198,143,165,057,216
37506 :010,010,168,162,000,185,153
37512 :044,156,157,200,143,200,012
37518 :232,224,004,208,244,165,195
37524 :058,010,010,168,162,000,044
37530 :185,052,156,157,215,143,038
37536 :200,232,224,004,208,244,248
37542 :165,080,010,010,168,162,249
37548 :000,185,068,156,157,228,198
37554 :143,200,232,224,004,208,165
37560 :244,165,197,201,001,240,208
37566 :250,032,095,152,076,188,215
37572 :147,201,147,208,072,160,107
37578 :000,132,253,152,162,160,037
37584 :134,254,145,253,200,208,122
37590 :251,232,224,191,208,244,028
37596 :134,254,145,253,200,192,118
37602 :064,208,249,160,000,132,015
37608 :253,165,061,010,010,010,229
37614 :010,077,033,208,041,240,079
37620 :077,033,208,162,140,134,230
37626 :254,145,253,200,208,251,025
37632 :232,224,143,208,244,134,161
37638 :254,145,253,200,192,232,002
37644 :208,249,076,188,147,201,057
37650 :018,208,039,160,000,132,063
37656 :251,169,160,133,252,177,142
37662 :251,073,255,145,251,200,181
37668 :208,247,230,252,165,252,110
37674 :201,191,208,239,177,251,029
37680 :073,255,145,251,200,192,140
37686 :064,208,245,076,188,147,214
37692 :201,006,208,049,169,027,208
37698 :141,017,208,169,021,141,251
37704 :024,208,169,008,141,022,132
37710 :208,169,000,141,021,208,057
37716 :169,199,141,000,221,169,215
37722 :055,133,001,032,178,152,129
37728 :160,045,185,000,203,153,074
37734 :057,000,169,000,153,000,225
```

```
37740 :ØØ2,136,Ø16,242,Ø96,2Ø1,Ø33
37746 :Ø64,2Ø8,ØØ3,Ø76,Ø42,153,148
37752 :162,Ø15,221,181,155,2Ø8,Ø38
37758 :ØØ8,134,Ø61,142,Ø39,2Ø8,2Ø6
37764 :Ø76,188,147,2Ø2,Ø16,24Ø,233
37770 :162,ØØ3,221,197,155,2Ø8,Ø6Ø
37776 :Ø27,165,Ø6Ø,2Ø8,ØØ6,224,Ø66
37782 :ØØ3,2Ø8,ØØ2,162,ØØ1,134,148
37788 :Ø59,162,ØØ8,165,Ø59,24Ø,Ø81
37794 :ØØ4,Ø1Ø,Ø1Ø,Ø1Ø,17Ø,134,244
378ØØ :Ø7Ø,Ø76,188,147,2Ø2,Ø16,Ø99
378Ø6 :221,Ø56,Ø41,239,233,Ø32,228
37812 :24Ø,ØØ6,2Ø1,Ø1Ø,176,ØØ2,Ø47
37818 :133,Ø62,165,162,197,Ø65,2Ø2
37824 :2Ø8,ØØ3,Ø76,Ø94,148,165,118
3783Ø :162,133,Ø65,173,ØØØ,22Ø,183
37836 :Ø73,127,133,Ø66,165,197,197
37842 :2Ø1,ØØ5,2Ø8,ØØ3,Ø76,Ø94,Ø29
37848 :148,165,Ø66,Ø41,Ø16,24Ø,124
37854 :Ø25,165,Ø58,24Ø,ØØ6,Ø32,236
3786Ø :Ø56,149,Ø76,248,147,165,Ø45
37866 :Ø64,2Ø8,Ø15,23Ø,Ø64,165,212
37872 :Ø57,Ø73,ØØ1,133,Ø57,Ø76,125
37878 :252,147,169,ØØØ,133,Ø64,243
37884 :165,Ø66,Ø41,ØØ1,24Ø,Ø11,ØØ8
3789Ø :165,Ø69,Ø56,229,Ø62,2Ø1,Ø16
37896 :2ØØ,176,ØØ2,133,Ø69,165,241
379Ø2 :Ø66,Ø41,ØØ2,24Ø,Ø11,165,Ø27
379Ø8 :Ø69,Ø24,1Ø1,Ø62,2Ø1,2ØØ,165
37914 :176,ØØ2,133,Ø69,165,Ø66,125
3792Ø :Ø41,ØØ4,24Ø,Ø23,165,Ø67,Ø6Ø
37926 :Ø56,229,Ø62,133,253,165,168
37932 :Ø68,233,ØØØ,133,254,Ø48,Ø12
37938 :ØØ8,165,253,133,Ø67,165,Ø73
37944 :254,133,Ø68,165,Ø66,Ø41,Ø15
3795Ø :ØØ8,24Ø,Ø29,165,Ø67,Ø24,Ø83
37956 :1Ø1,Ø62,133,253,165,Ø68,Ø82
37962 :1Ø5,ØØØ,133,254,24Ø,ØØ6,Ø44
37968 :165,253,2Ø1,Ø64,176,ØØ8,179
37974 :165,253,133,Ø67,165,254,Ø99
3798Ø :133,Ø68,165,Ø67,166,Ø6Ø,239
37986 :24Ø,ØØ2,Ø41,254,Ø24,1Ø5,252
37992 :Ø13,141,ØØØ,2Ø8,165,Ø68,187
37998 :1Ø5,ØØØ,141,Ø16,2Ø8,165,233
38ØØ4 :Ø69,1Ø5,Ø43,141,ØØ1,2Ø8,171
38Ø1Ø :165,Ø57,2Ø8,ØØ3,Ø76,1Ø3,222
38Ø16 :145,165,Ø6Ø,2Ø8,Ø12,165,115
38Ø22 :Ø59,2Ø1,ØØ2,2Ø8,ØØ6,Ø32,13Ø
38Ø28 :231,148,Ø76,1Ø3,145,Ø32,1Ø7
```

```
38034 :194,148,076,103,145,165,209
38040 :069,074,074,074,170,165,010
38046 :067,069,069,041,248,069,209
38052 :069,024,125,128,203,133,078
38058 :251,165,068,125,160,203,118
38064 :133,252,165,067,041,007,073
38070 :166,060,240,004,041,254,179
38076 :005,070,170,160,000,096,177
38082 :032,151,148,165,079,208,209
38088 :012,165,059,208,016,177,069
38094 :251,061,224,203,076,229,226
38100 :148,177,251,093,192,203,252
38106 :076,229,148,177,251,061,136
38112 :224,203,029,192,203,145,196
38118 :251,165,059,208,001,096,242
38124 :165,069,074,074,074,168,092
38130 :165,068,074,165,067,106,119
38136 :074,074,024,121,064,203,040
38142 :133,253,185,096,203,105,205
38148 :000,133,254,160,000,165,204
38154 :059,201,001,208,017,177,161
38160 :253,041,015,133,251,165,106
38166 :061,010,010,010,010,005,128
38172 :251,145,253,096,201,002,208
38178 :208,009,177,253,041,240,194
38184 :005,061,145,253,096,165,253
38190 :254,073,084,133,254,165,241
38196 :061,145,253,096,165,067,071
38202 :133,084,165,068,133,085,214
38208 :165,069,133,086,032,106,143
38214 :149,165,084,133,067,165,065
38220 :085,133,068,165,086,133,234
38226 :069,165,058,201,002,208,017
38232 :016,165,079,208,012,165,221
38238 :084,133,081,165,085,133,007
38244 :082,165,086,133,083,096,233
38250 :165,058,201,003,208,043,016
38256 :165,060,240,006,165,067,047
38262 :041,254,133,067,165,067,077
38268 :024,101,074,133,067,144,155
38274 :002,230,068,165,068,240,135
38280 :006,165,067,201,064,176,047
38286 :011,032,130,151,240,006,200
38292 :032,194,148,076,122,149,101
38298 :096,165,084,056,229,081,097
38304 :133,087,165,085,229,082,173
38310 :133,088,165,086,056,229,155
38316 :083,133,089,160,001,162,032
38322 :000,165,082,197,085,144,083
```

```
38328 :025,208,006,165,084,197,101
38334 :081,176,017,160,255,162,017
38340 :255,165,081,056,229,084,042
38346 :133,087,165,082,229,085,215
38352 :133,088,132,100,134,101,128
38358 :160,001,165,086,197,083,138
38364 :176,009,160,255,165,083,044
38370 :056,229,086,133,089,132,183
38376 :102,169,000,133,098,133,099
38382 :096,166,087,164,088,208,023
38388 :014,228,089,176,010,166,159
38394 :089,032,011,150,133,096,249
38400 :076,020,150,032,011,150,183
38406 :133,098,076,020,150,132,103
38412 :091,152,074,134,090,138,179
38418 :106,096,169,000,133,094,104
38424 :133,095,133,097,133,099,202
38430 :165,081,133,067,165,082,211
38436 :133,068,165,083,133,069,175
38442 :165,090,024,105,001,133,048
38448 :092,165,091,105,000,133,122
38454 :093,165,060,240,014,165,023
38460 :254,197,069,208,008,165,193
38466 :067,041,254,197,253,240,094
38472 :003,032,194,148,165,067,169
38478 :041,254,133,253,165,069,225
38484 :133,254,165,096,024,101,089
38490 :087,133,096,165,097,101,001
38496 :088,133,097,197,091,240,174
38502 :004,144,033,208,006,165,150
38508 :096,197,090,144,025,165,057
38514 :096,229,090,133,096,165,155
38520 :097,229,091,133,097,165,164
38526 :067,024,101,100,133,067,106
38532 :165,068,101,101,133,068,000
38538 :165,098,024,101,089,133,236
38544 :098,165,099,105,000,133,232
38550 :099,197,091,240,004,144,157
38556 :027,208,006,165,098,197,089
38562 :090,144,019,165,098,229,139
38568 :090,133,098,165,099,229,214
38574 :091,133,099,165,069,024,243
38580 :101,102,133,069,230,094,141
38586 :208,002,230,095,165,095,213
38592 :197,093,144,006,165,094,123
38598 :197,092,176,003,076,055,029
38604 :150,096,169,000,133,063,047
38610 :165,060,240,006,165,067,145
38616 :041,254,133,067,169,000,112
```

```
38622 :133,072,133,071,165,068,096
38628 :208,004,165,067,240,031,175
38634 :165,067,056,229,074,133,190
38640 :067,165,068,233,000,133,138
38646 :068,032,130,151,208,230,041
38652 :165,067,024,101,074,133,048
38658 :067,165,068,105,000,133,028
38664 :068,230,069,032,130,151,176
38670 :240,013,165,072,208,013,213
38676 :032,178,151,169,001,133,172
38682 :072,208,004,169,000,133,100
38688 :072,198,069,198,069,032,158
38694 :130,151,240,013,165,071,040
38700 :208,013,032,178,151,169,027
38706 :001,133,071,208,004,169,124
38712 :000,133,071,230,069,032,079
38718 :194,148,165,067,024,101,249
38724 :074,133,067,165,068,105,168
38730 :000,133,068,165,068,240,236
38736 :006,165,067,201,064,176,247
38742 :005,032,130,151,208,173,017
38748 :164,063,240,101,032,228,152
38754 :255,201,000,208,094,136,224
38760 :185,000,157,133,069,185,065
38766 :000,158,133,068,185,000,142
38772 :159,133,067,132,063,165,067
38778 :069,201,200,176,221,076,041
38784 :220,150,032,151,148,189,250
38790 :224,203,073,255,049,251,165
38796 :072,138,041,007,170,104,160
38802 :228,073,176,006,074,232,167
38808 :228,073,144,250,166,080,069
38814 :208,005,197,059,076,177,112
38820 :151,162,001,008,201,000,175
38826 :240,004,040,162,000,096,200
38832 :040,096,164,063,165,067,003
38838 :153,000,159,165,068,153,112
38844 :000,158,165,069,153,000,221
38850 :157,230,063,096,032,021,025
38856 :152,201,255,208,001,096,089
38862 :201,000,208,007,173,000,027
38868 :220,073,127,133,066,165,228
38874 :066,041,016,208,053,165,255
38880 :066,041,003,240,016,010,088
38886 :056,233,003,073,254,024,105
38892 :109,033,208,141,033,208,200
38898 :076,006,152,165,066,041,236
38904 :012,240,203,074,056,233,042
38910 :003,024,109,032,208,141,003
```

```
38916 :Ø32,2Ø8,162,Ø64,16Ø,255,117
38922 :Ø72,1Ø4,136,2Ø8,251,2Ø2,215
38928 :2Ø8,248,24Ø,178,Ø96,165,127
38934 :197,2Ø1,Ø64,2Ø8,ØØ5,169,Ø98
3894Ø :ØØØ,133,Ø66,Ø96,162,ØØ7,236
38946 :221,165,155,2Ø8,ØØ6,189,21Ø
38952 :173,155,133,Ø66,Ø96,2Ø2,Ø97
38958 :Ø16,242,169,ØØØ,133,Ø66,16Ø
38964 :169,255,Ø96,16Ø,Ø39,185,188
3897Ø :192,143,153,Ø64,191,185,218
38976 :192,219,153,112,191,165,Ø72
38982 :Ø61,153,192,219,169,Ø32,128
38988 :153,192,143,136,Ø16,231,179
38994 :169,Ø27,133,Ø75,169,Ø53,196
39ØØØ :133,Ø76,169,ØØ8,133,Ø77,172
39ØØ6 :Ø96,16Ø,Ø39,185,Ø64,191,Ø61
39Ø12 :153,192,143,185,112,191,Ø52
39Ø18 :153,192,219,136,Ø16,241,Ø39
39Ø24 :169,Ø59,133,Ø75,169,Ø56,ØØ5
39Ø3Ø :133,Ø76,165,Ø78,133,Ø77,Ø12
39Ø36 :Ø96,12Ø,169,127,141,Ø13,Ø22
39Ø42 :22Ø,169,ØØ1,141,Ø26,2Ø8,127
39Ø48 :169,ØØØ,141,Ø18,2Ø8,173,Ø77
39Ø54 :Ø17,2Ø8,Ø41,127,141,Ø17,181
39Ø6Ø :2Ø8,173,Ø2Ø,ØØ3,141,Ø34,215
39Ø66 :153,173,Ø21,ØØ3,141,Ø35,168
39Ø72 :153,169,211,141,Ø2Ø,ØØ3,Ø89
39Ø78 :169,152,141,Ø21,ØØ3,Ø88,228
39Ø84 :169,ØØ1,141,Ø21,2Ø8,Ø96,Ø4Ø
39Ø9Ø :169,ØØØ,141,Ø26,2Ø8,173,127
39Ø96 :Ø13,22Ø,ØØ9,129,141,Ø13,197
391Ø2 :22Ø,12Ø,173,Ø34,153,141,ØØ7
391Ø8 :Ø2Ø,ØØ3,173,Ø35,153,141,2Ø9
39114 :Ø21,ØØ3,Ø88,169,ØØØ,141,112
3912Ø :Ø21,2Ø8,Ø96,173,Ø25,2Ø8,171
39126 :141,Ø25,2Ø8,Ø41,ØØ1,24Ø,1Ø2
39132 :Ø71,165,Ø75,141,Ø17,2Ø8,129
39138 :165,Ø76,141,Ø24,2Ø8,165,237
39144 :Ø77,141,Ø22,2Ø8,162,242,Ø6Ø
3915Ø :16Ø,ØØ1,173,Ø18,2Ø8,Ø16,Ø46
39156 :ØØ4,162,ØØØ,16Ø,ØØØ,142,2ØØ
39162 :Ø18,2Ø8,173,Ø17,2Ø8,Ø41,147
39168 :127,141,Ø17,2Ø8,192,ØØØ,173
39174 :2Ø8,ØØ3,Ø76,Ø26,153,169,129
3918Ø :Ø59,141,Ø17,2Ø8,169,Ø56,15Ø
39186 :141,Ø24,2Ø8,165,Ø78,141,ØØ7
39192 :Ø22,2Ø8,173,Ø13,22Ø,Ø41,189
39198 :ØØ1,24Ø,ØØ3,Ø76,Ø49,234,121
392Ø4 :1Ø4,168,1Ø4,17Ø,1Ø4,Ø64,238
```

```
39210 :032,055,152,162,253,160,088
39216 :153,032,049,154,240,072,236
39222 :169,000,133,002,173,000,019
39228 :002,201,076,240,006,201,018
39234 :083,208,057,230,002,162,040
39240 :011,160,154,032,049,154,120
39246 :201,001,208,044,173,000,193
39252 :002,056,233,048,162,003,076
39258 :221,037,154,240,005,202,181
39264 :016,248,048,026,188,041,151
39270 :154,133,063,170,224,001,079
39276 :208,002,164,002,169,001,142
39282 :032,186,255,162,026,160,167
39288 :154,032,049,154,208,006,211
39294 :032,095,152,076,103,145,217
39300 :165,063,201,008,144,018,219
39306 :165,002,240,014,160,000,207
39312 :185,045,154,157,000,002,175
39318 :232,200,192,004,208,244,206
39324 :138,162,000,160,002,032,138
39330 :189,255,032,095,152,032,149
39336 :178,152,032,149,154,169,234
39342 :001,032,195,255,032,125,046
39348 :152,165,063,201,008,144,145
39354 :060,032,055,152,169,015,157
39360 :168,166,063,032,186,255,038
39366 :169,000,032,189,255,032,107
39372 :192,255,162,015,032,198,034
39378 :255,160,000,032,207,255,095
39384 :201,013,240,011,041,063,017
39390 :153,192,143,200,032,183,101
39396 :255,240,238,169,015,032,153
39402 :195,255,169,150,133,162,018
39408 :165,162,208,252,032,095,130
39414 :152,032,231,255,076,103,071
39420 :145,012,015,001,004,032,205
39426 :015,018,032,019,001,022,109
39432 :005,063,000,004,005,022,107
39438 :009,003,005,032,014,021,098
39444 :013,002,005,018,063,000,121
39450 :006,009,012,005,032,014,104
39456 :001,013,005,058,000,001,110
39462 :002,008,009,001,000,002,060
39468 :002,044,083,044,087,134,182
39474 :253,132,254,160,039,169,033
39480 :032,153,192,143,136,016,216
39486 :250,200,177,253,240,006,164
39492 :153,192,143,200,208,246,186
39498 :200,162,000,169,160,153,150
```

```
39504 :192,143,132,251,134,252,160
39510 :032,228,255,240,251,164,232
39516 :251,166,252,201,013,208,159
39522 :007,169,032,153,192,143,026
39528 :138,096,201,020,208,014,013
39534 :224,000,240,219,169,032,226
39540 :153,192,143,136,202,076,250
39546 :077,154,201,032,144,205,167
39552 :201,096,176,201,192,039,009
39558 :240,197,157,000,002,041,003
39564 :063,153,192,143,200,232,099
39570 :076,077,154,032,192,255,164
39576 :176,016,032,183,255,208,254
39582 :011,162,001,165,002,208,195
39588 :006,032,198,255,144,006,037
39594 :096,032,201,255,176,250,156
39600 :032,183,255,208,245,169,244
39606 :208,133,252,169,032,133,085
39612 :251,032,093,155,230,251,176
39618 :032,093,155,169,000,133,008
39624 :251,169,140,133,252,032,153
39630 :093,155,230,251,208,249,112
39636 :230,252,165,252,201,143,175
39642 :208,241,032,093,155,230,153
39648 :251,165,251,201,232,208,252
39654 :245,169,000,133,251,169,173
39660 :216,133,252,032,093,155,093
39666 :230,251,208,249,230,252,126
39672 :165,252,201,219,208,241,254
39678 :032,093,155,230,251,165,156
39684 :251,201,232,208,245,169,030
39690 :000,133,251,169,160,133,088
39696 :252,032,093,155,160,000,196
39702 :177,251,208,061,165,002,118
39708 :208,034,032,207,255,133,129
39714 :253,032,207,255,133,254,144
39720 :169,000,168,145,251,032,037
39726 :146,155,176,042,165,251,213
39732 :197,253,208,240,165,252,087
39738 :197,254,208,234,240,209,120
39744 :032,146,155,144,006,032,067
39750 :135,155,076,092,155,160,075
39756 :000,177,251,240,239,032,247
39762 :135,155,032,093,155,032,172
39768 :146,155,144,181,096,165,207
39774 :002,208,021,032,207,255,051
39780 :072,176,028,032,183,255,078
39786 :240,004,201,064,208,019,074
39792 :104,160,000,145,251,096,100
```

```
39798 :160,000,177,251,032,210,180
39804 :255,032,183,255,208,002,035
39810 :096,104,104,104,096,165,031
39816 :251,032,210,255,165,252,021
39822 :032,210,255,096,230,251,192
39828 :208,004,230,252,024,096,194
39834 :165,252,201,191,144,004,087
39840 :165,251,201,065,096,018,188
39846 :023,010,009,020,012,062,046
39852 :014,008,002,004,001,010,211
39858 :006,005,009,144,005,028,119
39864 :159,156,030,031,158,129,079
39870 :149,150,151,152,153,154,075
39876 :155,136,135,134,133,072,193
39882 :073,082,069,083,032,083,112
39888 :075,069,084,067,072,080,143
39894 :065,068,032,045,032,066,010
39900 :089,032,067,072,082,073,123
39906 :083,032,077,069,084,067,126
39912 :065,076,070,013,077,085,106
39918 :076,084,073,067,079,076,181
39924 :079,082,032,077,079,068,149
39930 :069,063,032,078,157,000,137
39936 :016,012,015,020,058,006,127
39942 :032,058,032,032,032,032,224
39948 :032,012,009,014,005,032,116
39954 :004,018,001,023,058,032,154
39960 :032,032,032,032,032,006,190
39966 :009,012,012,020,015,058,156
39972 :032,032,032,032,055,053,016
39978 :051,049,015,006,006,032,201
39984 :015,014,032,032,015,006,162
39990 :006,032,006,018,015,013,144
39996 :020,015,032,032,012,009,180
40002 :014,005,019,001,013,005,123
40008 :001,014,025,032,000,056,200
40014 :000,000,068,000,000,068,214
40020 :000,006,254,192,009,001,034
40026 :032,006,000,192,004,000,068
40032 :064,004,000,064,004,000,232
40038 :064,006,000,192,009,001,118
40044 :032,006,254,192,000,056,136
40050 :000,000,012,000,024,000,150
40056 :000,036,000,000,036,000,192
40062 :003,126,192,004,129,032,100
40068 :003,000,192,002,000,064,137
40074 :002,000,064,002,000,064,014
40080 :003,000,192,004,129,032,248
40086 :003,126,192,000,024,000,239
40092 :000,012,000,000,255,255,166
```

Program 2. LOAD and SAVE Subroutine

```
100 REM PASS LOAD OR SAVE TO THIS SUB-
110 REM ROUTINE IN LS (LOAD = Ø, SAVE
120 REM 1). FILE NAME IN NM$ (WITH Ø:
130 REM OR @Ø: IF DESIRED) AND DEVICE
140 REM IN DV. THEN GOSUB 5ØØØØ.
150 REM ROUTINE ABORTS ON ERROR,
160 REM AND YOU READ THE CHANNEL.
170 :
1ØØØ I=51676
1Ø1Ø READA:IFA>=ØTHENPOKEI,A:I=I+1:GOTO1Ø1Ø
1Ø2Ø REM SIMPLE SAMPLE PROGRAM
1Ø3Ø INPUT"LOAD OR SAVE";A$:LS=Ø:IFLEFT$(A$,1)="S"
     THENLS=1
1Ø4Ø INPUT"DEVICE";DV
1Ø5Ø INPUT"NAME";NM$
1Ø6Ø GOSUB 5ØØØØ:END
2ØØØ :
5ØØØØ ZS=2:IF(LS<>1ANDLS<>Ø)ORNM$=""ORLEN(NM$)>16O
      RDV<1ORDV>11THENRETURN
5ØØ1Ø IFDV<3THENZS=Ø:IFDV<2THENZS=LS
5ØØ2Ø IFDV>7ANDLS=1THENNM$=NM$+",S,W"
5ØØ3Ø IFLS=1THENPRINT"{DOWN}SAVING{SHIFT-SPACE}"NM
      $:POKE2,1:GOTO5ØØ5Ø
5ØØ4Ø PRINT"{DOWN}LOADING "NM$:POKE2,Ø:POKE56576,1
      97:POKE53272,56:POKE53265,59
5ØØ5Ø OPEN1,DV,ZS,NM$:SYS(51676):POKE56576,199:POK
      E53272,21:POKE53265,27
5ØØ6Ø RETURN
5ØØ7Ø :
51676 DATA 169,Ø,133,157,32,183,255,2Ø8,7,169,54,1
      33,1,32,249,2Ø1
51692 DATA 169,1,32,195,255,32,231,255,169,55,133,
      1,96,162,1,165
517Ø8 DATA 2,2Ø8,6,32,198,255,144,6,96,32,2Ø1,255,
      176,25Ø,32,183
51724 DATA 255,2Ø8,245,169,2Ø8,133,252,169,32,133,
      251,32,183,2Ø2,23Ø,251
5174Ø DATA 32,183,2Ø2,169,Ø,133,251,169,14Ø,133,25
      2,32,183,2Ø2,23Ø,251
51756 DATA 2Ø8,249,23Ø,252,165,252,2Ø1,143,2Ø8,241
      ,32,183,2Ø2,23Ø,251,165
51772 DATA 251,2Ø1,232,2Ø8,245,169,Ø,133,251,169,2
      16,133,252,32,183,2Ø2
51788 DATA 23Ø,251,2Ø8,249,23Ø,252,165,252,2Ø1,219
      ,2Ø8,241,32,183,2Ø2,23Ø
518Ø4 DATA 251,165,251,2Ø1,232,2Ø8,245,169,Ø,133,2
      51,169,16Ø,133,252,32
```

```
51820 DATA 183,202,160,0,177,251,208,61,165,2,208,
      34,32,228,255,133
51836 DATA 253,32,228,255,133,254,169,0,168,145,25
      1,32,236,202,176,42
51852 DATA 165,251,197,253,208,240,165,252,197,254
      ,208,234,240,209,32,236
51868 DATA 202,144,6,32,225,202,76,182,202,160,0,1
      77,251,240,239,32
51884 DATA 225,202,32,183,202,32,236,202,144,181,9
      6,165,2,208,21,32
51900 DATA 228,255,72,176,28,32,183,255,240,4,201,
      64,208,19,104,160
51916 DATA 0,145,251,96,160,0,177,251,32,210,255,3
      2,183,255,208,2
51932 DATA 96,104,104,104,96,165,251,32,210,255,16
      5,252,32,210,255,96
51948 DATA 230,251,208,4,230,252,24,96,165,252,201
      ,191,144,4,165,251
51964 DATA 201,65,96,255,-1
```

Program 3. Sketchpad Source Code

```
                ;BREAKDOWN OF MEMORY USAGE
9000
                ; $0400-$07E7 TEXT SCREEN (BASIC)
                ; $0801-$8BFF BASIC PROGRAM AREA (UNUSED)
                ; $8C00-$8FE7 TEMPORARY COLOR TEXT SCREEN
                ; $9000-$9C9F 3232 BYTES FOR MAIN PROGRAM
                ; $9D00-$9FFF FILL STACK STORAGE
                ; $A000-$BF3F HI-RES MAP PAGE
                ; $BF40-$BF97 ADDITIONAL STORAGE
                ; $BFC0-$BFFF SPRITE 0 BLOCK
                ; $CB00-$CBFF ADDITIONAL STORAGE
                ; $D800-$DBE7 COLOR SCREEN
                ;
                ;LIST OF PROGRAM VARIABLES
                ;
9000            DRAW    =   57      ;WHETHER TO PLOT WHILE MOVING
9000            LINES   =   58      ;TYPE OF LINES TO DRAW
9000            PLOTMD  =   59      ;TYPE OF PLOTTING
9000            SCRNMD  =   60      ;0 = HIRES, 1 = MULTICOLOR
9000            PLCOL   =   61      ;PLOTTING COLOR
9000            PLINC   =   62      ;PLOT INCREMENT
9000            PNTR    =   63      ;FILL POINTER/DEVICE NUMBER
9000            PRESSD  =   64      ;FIRE BUTTON PRESSED
9000            TIMSTO  =   65      ;LAST JIFFY VALUE
9000            JOY     =   66      ;CURRENT JOYSTICK VALUE
9000            XPOS    =   67      ;(2) X-POSITION OF PLOTTER
9000            YPOS    =   69      ;Y-POSITION OF PLOTTER
9000            AD      =   251     ;(2) WHERE BIT IS PLOTTED
9000            TYPE    =   70      ;TYPE OF PLOT
9000            MISC    =   253     ;(2) MISC COUNTER/POINTER
9000            UPF     =   71      ;FILL FLAG FOR UP DIRECTION
9000            DOWNF   =   72      ;FILL FLAG FOR DOWN
9000            FLSHFT  =   73      ;HOW FAR TO SHIFT A BYTE (7/6)
9000            FLLINC  =   74      ;INCREMENT FOR FILL (1/2)
9000            HIR1    =   75      ;HIRES VIC CHIP SHADOWS (SCREEN BANK)
9000            HIR2    =   76      ; (HIRES/TEXT)
9000            HIR3    =   77      ; (MULTICOLOR)
9000            HIR4    =   78      ; (DISPLAY MULTICOLOR ON/OFF)
9000            XORIT   =   79      ;FLAG FOR RUBBERBAND
9000            FILBOR  =   80      ;FILL TO WHAT
```

```
9000            LDSV     =    2        ;LOAD/SAVE FLAG
                 ;
9000            X1       =    81       ;(2) X START
9000            Y1       =    83       ;Y START
9000            X2       =    84       ;(2) X END
9000            Y2       =    86       ;Y END
9000            XDIFF    =    87       ;(2) ABSOLUTE X DIFFERENCE
9000            YDIFF    =    89       ;Y DIFFERENCE
9000            DIS      =    90       ;(2) GREATER DISTANCE
9000            DIS2     =    92       ;(2) HOLDS DIS+2
9000            CNTR     =    94       ;(2) COUNTER FOR DISTANCE
9000            XCNTR    =    96       ;(2) COUNTER FOR NEXT PLOT
9000            YCNTR    =    98       ;(2) NEXT PLOT COUNTER
9000            XSGN     =    100      ;(2) SIGNUM OF X-DIFFERENCE
9000            YSGN     =    102      ;YDIFF SGN
                 ;
9000            TABLE1   =    $9D00    ;FILL STACKS
9000            TABLE2   =    $9E00
9000            TABLE3   =    $9F00
9000            SCLINE   =    $BF40    ;COPY OF BOTTOM SCREEN LINE
9000            COLINE   =    $BF70    ;BOTTOM COLOR SCREEN LINE
9000            ZERSTO   =    $CB00    ;57-102 ZERO PAGE STORAGE
9000            FORTYL   =    $CB40    ;TABLE OF ADDRESSES
9000            FORTYH   =    $CB60    ;  OF SCREEN LINES
9000            LINEL    =    $CB80    ;TABLE OF ADDRESSES
9000            LINEH    =    $CBA0    ;  OF LINES ON HIRES PAGE
9000            PIXEL    =    $CBC0    ;HIRES/MULTI PIXELS
9000            MASK     =    $CBE0    ;PIXEL MASKS
                 ;
                 ;LIST OF SYSTEM LOCATIONS
                 ;
9000            R6510    =    $01      ;ON-CHIP MEMERY CONTROL REGISTER
9000            MEMSIZ   =    $37      ;BASIC TOP OF MEMORY
9000            MSGFLG   =    $9D      ;KERNAL MESSAGES ON/OFF
9000            TIME     =    $A0      ;3-BYTE TIMER (HML)
9000            LSTX     =    $C5      ;CURRENT KEY PRESSED (64=NONE)
9000            INBUFF   =    $0200    ;LINE INPUT BUFFER
9000            COLOR    =    $0286    ;CURRENT CHARACTER COLOR
9000            RPTFLG   =    $028A    ;KEY REPEAT ON/OFF
9000            KOUNT    =    $028B    ;KEY REPEAT SPEED
9000            SHFLAG   =    $028D    ;SHIFT/CTRL/C= FLAG
9000            INTPNT   =    $0314    ;IRQ POINTER
9000            CRT      =    $8C00    ;TEMPORARY TEXT SCREEN
9000            S0PNTR   =    CRT+1016  ;SPRITE 0 POINTER
9000            HIPAGE   =    $A000    ;START OF HIRES PAGE
9000            SBLOCK   =    $BFC0    ;SPRITE BLOCK ADDRESS
9000            VIC      =    $D000    ;START OF VIC CHIP
9000            JSTICK   =    $DC00    ;JOYSTICK #2 STATUS
9000            CIAICR   =    $DC0D    ;CIA INTERRUPT CONTROL REG.
9000            COLCRT   =    $D800    ;START OF COLOR SCREEN
9000            VBANK    =    $DD00    ;16K BANK SELECTER FOR VIC
                 ;
9000            INTRPT   =    $EA31    ;NORMAL IRQ VECTOR
9000            CHKIN    =    $FFC6    ;OPEN INPUT CHANNEL
9000            CHKOUT   =    $FFC9    ;OPEN OUTPUT CHANNEL
9000            CHRIN    =    $FFCF    ;INPUT A BYTE
9000            CHROUT   =    $FFD2    ;PRINT CHR$(.A)
9000            CLALL    =    $FFE7    ;CLOSE ALL
9000            CLOSE    =    $FFC3    ;CLOSE A FILE
9000            GETIN    =    $FFE4    ;GET CHARACTER
9000            OPEN     =    $FFC0    ;OPEN THE FILE
9000            READST   =    $FFB7    ;CHECK STATUS WORD
9000            SETLFS   =    $FFBA    ;SET FILE PARAMETERS
9000            SETNAM   =    $FFBD    ;SET FILE NAME
                 ;
                 ;INITIALIZE USER INFORMATION
                 ;
9000 20 E7 FF PROGRM    JSR  CLALL    ;CLOSE ALL OPEN FILES
9003 A0 00              LDY  #0       ;DISPLAY "MULTICOLOR MODE? N"
9005 B9 C9 9B QLOOP     LDA  QUESTN,Y  ;GET A CHARACTER
9008 F0 06              BEQ  QEND     ;ZERO BYTE FLAGS THE END
900A 20 D2 FF           JSR  CHROUT   ;PRINT THE CHARACTER
```

```
900D C8                  INY                ;NEXT BYTE
900E D0 F5               BNE   QLOOP        ;BRANCH-ALWAYS
9010 A0 00      QEND     LDY   #0           ;PREPARE FOR A 'NO'
9012 20 CF FF            JSR   CHRIN        ;GET OUR INPUT
9015 C9 0D               CMP   #13          ;JUST A RETURN
9017 D0 01               BNE   NYCK         ;NO
9019 60                  RTS                ;YES, SO ABORT
901A C9 59      NYCK     CMP   #"Y"         ;CHECK IF "YES"
901C D0 01               BNE   HIPICK       ;NO
901E C8                  INY                ;SET .Y TO 1
901F 98         HIPICK   TYA
9020 48                  PHA                ;SAVE .Y ON STACK
9021 20 CF FF   CLRIN    JSR   CHRIN        ;PULL THE REST OF THE INPUT
9024 C9 0D               CMP   #13
9026 D0 F9               BNE   CLRIN        ;UNTIL A RETURN COMES UP
                ;
9028 A0 2D               LDY   #45
902A B9 39 00   ZEROFF   LDA   57,Y         ;STORE 57-102 ZERO PAGE
902D 99 00 CB            STA   ZERSTO,Y     ;... AT $CB00
9030 88                  DEY
9031 10 F7               BPL   ZEROFF       ;DO 45 TO 0
9033 68                  PLA
9034 85 3C               STA   SCRNMD       ;PUT OLD .Y IN SCRNMD
                ;
                ;SET UP SYSTEM FOR HIGH RESOLUTION
                ;
9036 A9 00      SETUP    LDA   #0           ;SET PROGRAM MODE
9038 85 9D               STA   MSGFLG       ;$00=PROGRAM,$80=IMMEDIATE
903A A9 8C               LDA   #>CRT        ;SET TOP OF MEMORY AND
903C 85 38               STA   MEMSIZ+1     ; BASIC STRING POINTER
903E 85 34               STA   MEMSIZ-3     ; TO START OF TEMPORARY
9040 A9 00               LDA   #<CRT        ; SCREEN MEMORY
9042 85 37               STA   MEMSIZ
9044 85 33               STA   MEMSIZ-4
9046 A9 80               LDA   #$80         ;SET ALL-KEY REPEAT MODE
9048 8D 8A 02            STA   RPTFLG       ;$0=CURSOR,$40=NONE,$80=ALL REPEAT
904B A9 C5               LDA   #196+1       ;1 INDICATES BANK 2
904D 8D 00 DD            STA   VBANK        ;VIC SEES $8000
9050 A9 36               LDA   #55-1        ;FLIP OUT BASIC
9052 85 01               STA   R6510
9054 A9 38               LDA   #8+48        ;HIRES $A000, TEXT $8C00
9056 85 4C               STA   HIR2         ;VIC+24 SHADOW
9058 A9 3B               LDA   #27+32       ;32 SETS HIRES GRAPHICS
905A 85 4B               STA   HIR1         ;VIC+17 SHADOW
905C A9 08               LDA   #8           ;ASSUME NO MULTICOLOR ...
905E A4 3C               LDY   SCRNMD       ;CHECK IT
9060 F0 02               BEQ   SETHIR       ;NONE
9062 A9 18               LDA   #8+16        ;16 SETS MULTICOLOR
9064 85 4D      SETHIR   STA   HIR3
9066 85 4E               STA   HIR4         ;VIC+22 SHADOWS
                ;
                ;INITIALIZE PROGRAM VARIABLES
                ;
9068 A9 00      SETVAR   LDA   #0
906A 85 40               STA   PRESSD       ;JOYSTICK NOT PRESSED
906C 85 39               STA   DRAW         ;BEGIN WITH PEN UP
906E 85 3A               STA   LINES        ;LINE DRAW OFF
9070 85 4F               STA   XORIT        ;NO RUBBERBAND LINE
9072 85 50               STA   FILBOR       ;FILL TO SAME PIXEL PATTERN
9074 85 44               STA   XPOS+1       ;SET XPOS TO 160
9076 A9 A0               LDA   #160
9078 85 43               STA   XPOS
907A A9 64               LDA   #100         ;SET YPOS TO 100
907C 85 45               STA   YPOS
907E A9 01               LDA   #1
9080 85 3E               STA   PLINC        ;INITIAL PLOT INCREMENT 1
9082 85 3B               STA   PLOTMD       ;F5 PLOT MODE
9084 A5 3C               LDA   SCRNMD
9086 0A                  ASL   A            ;MULTIPLY SCRNMD BY 8
9087 0A                  ASL   A
9088 0A                  ASL   A
9089 85 46               STA   TYPE         ;SET PIXEL TABLE POINTER
```

```
908B A9 07              LDA  #7       ;SET FILL SHIFT BY SCRNMD (7 OR 6)
908D 38                 SEC
908E E5 3C              SBC  SCRNMD
9090 85 49              STA  FLSHFT
9092 A9 01              LDA  #1       ;SET FILL INCREMENT (0 OR 1)
9094 18                 CLC
9095 65 3C              ADC  SCRNMD
9097 85 4A              STA  FLLINC
                  ;
                  ;SET UP TABLES (ADDRESSES,PIXELS,MASKS)
                  ;
9099 A9 8C              LDA  #>CRT    ;SET COLOR SCREEN ADDRESSES FROM $8C00
909B 8D 60 CB           STA  FORTYH
909E A9 00              LDA  #<CRT
90A0 8D 40 CB           STA  FORTYL
90A3 A9 A0              LDA  #>HIPAGE  ;SET HIRES LINE ADDRESSES FROM $A000
90A5 8D A0 CB           STA  LINEH
90A8 A9 00              LDA  #<HIPAGE
90AA 8D 80 CB           STA  LINEL
90AD AA                 TAX           ;<HIPAGE = 0 THEREFORE LDX #0
90AE BD 40 CB FLOOP     LDA  FORTYL,X  ;GET PREVIOUS ENTRY
90B1 18                 CLC
90B2 69 28              ADC  #40      ;ADD 40
90B4 9D 41 CB           STA  FORTYL+1,X  ;STORE AT NEXT ENTRY
90B7 BD 60 CB           LDA  FORTYH,X  ;GET HIGH BYTE (DIFFERENT TABLE)
90BA 69 00              ADC  #0       ;INCREMENT IF NECESSARY
90BC 9D 61 CB           STA  FORTYH+1,X
90BF BD 80 CB           LDA  LINEL,X  ;GET PREVIOUS
90C2 18                 CLC
90C3 69 40              ADC  #64      ;ADD 64 TO LOW BYTE
90C5 9D 81 CB           STA  LINEL+1,X
90C8 BD A0 CB           LDA  LINEH,X
90CB 69 01              ADC  #1       ;AND 1 OR 2 TO HIGH BYTE
90CD 9D A1 CB           STA  LINEH+1,X
90D0 E8                 INX           ;NEXT ROW
90D1 E0 18              CPX  #24      ;ASSIGNED ALL 25 ROWS
90D3 D0 D9              BNE  FLOOP
                  ;
90D5 A9 01              LDA  #%00000001
90D7 A0 07              LDY  #7       ;SET UP FOR HIRES PIXELS
90D9 99 C0 CB PXLP1     STA  PIXEL,Y
90DC 0A                 ASL  A        ;SHIFT IT
90DD 88                 DEY           ; NEXT ENTRY
90DE 10 F9              BPL  PXLP1
90E0 A9 01              LDA  #%00000001
90E2 A0 06              LDY  #6       ;SET UP FOR F3/F5 MULTICOLOR
90E4 99 C8 CB PXLP2     STA  PIXEL+8,Y
90E7 0A                 ASL  A        ;SHIFT FOR F3
90E8 99 D0 CB           STA  PIXEL+16,Y
90EB 0A                 ASL  A        ;SHIFT FOR F5
90EC 88                 DEY           ;SKIP AN ENTRY
90ED 88                 DEY
90EE 10 F4              BPL  PXLP2
90F0 A9 03              LDA  #%00000011
90F2 A0 06              LDY  #6       ;SET UP FOR MULTICOLOR F1
90F4 99 D8 CB PXLP3     STA  PIXEL+24,Y
90F7 0A                 ASL  A        ;SHIFT TWICE
90F8 0A                 ASL  A
90F9 88                 DEY           ;SKIP AN ENTRY
90FA 88                 DEY
90FB 10 F7              BPL  PXLP3
                  ;
90FD A9 FE              LDA  #$FE
90FF A0 07              LDY  #7       ;SET UP MASK TABLES
9101 99 E0 CB MASK1     STA  MASK,Y
9104 38                 SEC
9105 2A                 ROL  A        ;ROLL ON A BIT
9106 88                 DEY
9107 10 F8              BPL  MASK1    ;DO 7 THROUGH 0
9109 A9 FC              LDA  #%11111100
910B A0 07              LDY  #7       ;SET UP FOR MULTI MASKS
910D 99 E7 CB MASK2     STA  MASK+7,Y
```

```
9110 99 EF CB           STA  MASK+15,Y
9113 99 F7 CB           STA  MASK+23,Y
9116 38                 SEC           ;ROLL ON TWO BITS
9117 2A                 ROL  A
9118 38                 SEC
9119 2A                 ROL  A
911A 88                 DEY
911B 88                 DEY
911C 10 EF              BPL  MASK2
                  ;
                  ;ASSIGN SPRITE VARIABLES
                  ;
911E A0 28              LDY  #40      ;COUNTER FOR BYTES LOADED
9120 A2 28              LDX  #40      ;TOP OF HIRES SPRITE
9122 A5 3C              LDA  SCRNMD
9124 F0 02              BEQ  SLOOP    ;HIRES
9126 A2 51              LDX  #81      ;TOP OF MULTICOLOR SPRITE
9128 BD 4C 9C SLOOP     LDA  SPRITE,X  ;FROM APPROPRIATE SPRITE
912B 99 C0 BF           STA  SBLOCK,Y  ;STORE IN TABLE
912E CA                 DEX           ;NEXT LOOP
912F 88                 DEY           ;COUNT ONE BYTE
9130 10 F6              BPL  SLOOP
9132 A9 00              LDA  #0       ;CLEAR REMAINING BYTES
9134 A0 15              LDY  #21      ;22 BYTES REMAIN
9136 99 E9 BF LOOP0     STA  SBLOCK+41,Y  ;CLEAR THEM
9139 88                 DEY
913A 10 FA              BPL  LOOP0
913C A9 FF              LDA  #255     ;SPRITE BLOCK (AT 49088)
913E 8D F8 8F           STA  S0PNTR   ;SPRITE 0 POINTER
9141 A9 AC              LDA  #172     ;160 + X-OFFSET (12)
9143 18                 CLC
9144 65 3C              ADC  SCRNMD   ;MULTICOLOR OFFSET
9146 8D 00 D0           STA  VIC      ;HARDWARE SPRITE 0 X-REGISTER
9149 A9 8F              LDA  #143     ;100 + Y-OFFSET (43)
914B 8D 01 D0           STA  VIC+1    ;SPRITE 0 Y-REGISTER
914E A9 00              LDA  #0       ;TURN OFF ...
9150 8D 1B D0           STA  VIC+27   ;... BACKGROUND PRIORITY
9153 8D 1C D0           STA  VIC+28   ;... SPRITE MULTICOLOR
9156 8D 1D D0           STA  VIC+29   ;... SPRITE EXPAND X
9159 8D 17 D0           STA  VIC+23   ;... SPRITE EXPAND Y
915C AD 86 02           LDA  COLOR    ;CHARACTER COLOR
915F 85 3D              STA  PLCOL    ;PLOTTING COLOR SHADOW
9161 8D 27 D0           STA  VIC+39   ;SPRITE 0 COLOR
9164 20 7D 98           JSR  RSTON    ;TURN ON RASTER INTERRUPT
                  ;
                  ;BEGINNING OF MAIN LOOP
                  ;
9167 A9 01    BEGIN     LDA  #1
9169 8D 8B 02           STA  KOUNT    ;SET MINIMUM REPEAT
916C 20 E4 FF           JSR  GETIN    ;PULL CHARACTER FROM KEYBOARD
916F 48                 PHA           ;SAVE ON STACK
9170 A5 3A              LDA  LINES    ;CHECK FOR LINE MODE
9172 F0 28              BEQ  ANALZE   ;NO
9174 AD 8D 02           LDA  SHFLAG   ;SHIFT/CTRL/C= FLAG
9177 29 01              AND  #1       ;TEST SHIFT KEY ONLY
9179 D0 07              BNE  YESLIN   ;PUSHED DOWN
917B AD 00 DC           LDA  JSTICK   ;GET PORT 2 INPUT
917E 29 10              AND  #16      ;CHECK FIRE BUTTON
9180 D0 06              BNE  XORLIN   ;NOT PRESSED
9182 20 38 95 YESLIN    JSR  LINE     ;PLOT A LINE
9185 4C 9C 91           JMP  ANALZE   ;SKIP RUBBERBAND
9188 A5 3A    XORLIN    LDA  LINES    ;LINE MODE FLAG
918A C9 03              CMP  #3       ;CHECK FOR RIGHT FILL MODE
918C F0 0E              BEQ  ANALZE   ;IF YES, SKIP
918E A9 01              LDA  #1       ;SET RUBBERBAND FLAG
9190 85 4F              STA  XORIT
9192 20 38 95           JSR  LINE     ;DRAW IT ON
9195 20 38 95           JSR  LINE     ;PLOT IT OFF
9198 A9 00              LDA  #0       ;UNSET RUBBERBAND MODE
919A 85 4F              STA  XORIT
919C 68       ANALZE    PLA           ;RESTORE GETIN CHARACTER
919D D0 03              BNE  DOKEYS   ;IF A KEY PRESSED
```

```
919F 4C BC 93           JMP   JOYSTK   ;ELSE SKIP KEY CHECKS
91A2 48        DOKEYS   PHA            ;RESAVE .A ON STACK
91A3 20 15 98           JSR   KEYS     ;SET JOY IF A KEYBOARD DIRECTION
91A6 68                 PLA            ;RESTORE CHARACTER
91A7 A4 42              LDY   JOY      ;TEST FOR A DIRECTION
91A9 F0 03              BEQ   SPACE    ;IF NOT, CHECK OTHERS
91AB 4C D9 93           JMP   FIRECK   ;YES, SO EXECUTE
               ;
               ;CHECK THE FUNCTIONS
               ;
91AE C9 20     SPACE    CMP   #32      ;SPACE
91B0 F0 04              BEQ   SPACDO   ;YES
91B2 C9 A0              CMP   #160     ;SHIFT SPACE
91B4 D0 09              BNE   HOME     ;IF NOT, TRY ANOTHER
91B6 A5 39     SPACDO   LDA   DRAW     ;MOVE/PLOT FLAG
91B8 49 01              EOR   #1       ;TOGGLE ON/OFF
91BA 85 39              STA   DRAW     ;STORE IT BACK
91BC 4C BC 93           JMP   JOYSTK   ;GO AND CHECK JOYSTICK
91BF C9 53     HOME     CMP   #"S"     ;HOME THE PLOTTER
91C1 D0 0F              BNE   CORNER
91C3 A9 00              LDA   #0       ;CENTER THE PLOTTER
91C5 85 44              STA   XPOS+1
91C7 A9 A0              LDA   #160     ;X-CENTER
91C9 85 43              STA   XPOS
91CB A9 64              LDA   #100     ;Y-CENTER
91CD 85 45              STA   YPOS
91CF 4C BC 93           JMP   JOYSTK   ;GO TO JOYSTICK
91D2 C9 13     CORNER   CMP   #19      ;CONTROL-S (HOME)
91D4 D0 0B              BNE   PLUS
91D6 A9 00              LDA   #0       ;ZERO X AND Y COORDS
91D8 85 43              STA   XPOS
91DA 85 44              STA   XPOS+1
91DC 85 45              STA   YPOS
91DE 4C BC 93           JMP   JOYSTK
91E1 C9 2B     PLUS     CMP   #"+"     ;PLOT ON (F1)
91E3 F0 04              BEQ   PLUSDO
91E5 C9 DB              CMP   #219     ;CHECK SHIFT-PLUS
91E7 D0 0B              BNE   MINUS
91E9 A9 01     PLUSDO   LDA   #1       ;F5 = 1
91EB 85 3B     SET      STA   PLOTMD   ;STORE .A
91ED A9 08              LDA   #8       ;SET PIXEL/MASK POINTER
91EF 85 46              STA   TYPE
91F1 4C BC 93           JMP   JOYSTK
91F4 C9 2D     MINUS    CMP   #"-"     ;PLOT OFF (F7)
91F6 F0 04              BEQ   MINDO
91F8 C9 DD              CMP   #221     ;SHIFT-MINUS
91FA D0 05              BNE   FILLCK
91FC A9 00     MINDO    LDA   #0       ;F7 = 0
91FE 4C EB 91           JMP   SET      ;SET TYPE ETC.
9201 C9 8C     FILLCK   CMP   #140     ;F8
9203 D0 06              BNE   LINE1
9205 20 CE 96           JSR   FILL     ;EXECUTE A FILL
9208 4C BC 93           JMP   JOYSTK
920B C9 89     LINE1    CMP   #137     ;F2 = LINE DRAW FROM
920D D0 17              BNE   LINE2
920F A5 3A              LDA   LINES    ;0=OFF,1=FROM,2=TO,3=RIGHT
9211 29 01              AND   #1       ;MASK OFF ALL BUT FROM
9213 49 01              EOR   #1       ;TOGGLE THAT
9215 85 3A              STA   LINES    ;AND RESAVE
9217 A5 43     PTINIT   LDA   XPOS
9219 85 51              STA   X1       ;SET UP INITIAL POINTS
921B A5 44              LDA   XPOS+1
921D 85 52              STA   X1+1
921F A5 45              LDA   YPOS
9221 85 53              STA   Y1
9223 4C 5E 94           JMP   PLOT     ;SKIP OVER JOYSTICK
9226 C9 8A     LINE2    CMP   #138     ;F4 = DRAWTO
9228 D0 0B              BNE   LINE3
922A A5 3A              LDA   LINES    ;GET LINE MODE
922C 29 02              AND   #2       ;MASK OFF ALL BUT DRAWTO
922E 49 02              EOR   #2       ;TOGGLE
9230 85 3A              STA   LINES    ;RESAVE
```

```
9232 4C 17 92           JMP  PTINIT  ;GO BACK AND SET THE INITIAL POINT
9235 C9 8B    LINE3     CMP  #139    ;F6 = RIGHT DRAW
9237 D0 12              BNE  ASTCK
9239 A5 3A              LDA  LINES
923B D0 07              BNE  OFF     ;IF ANYTHING IS ON
923D A9 03              LDA  #3      ;TURN ON RIGHT DRAW
923F 85 3A              STA  LINES
9241 4C BC 93           JMP  JOYSTK
9244 A9 00    OFF       LDA  #0      ;TURN OFF RIGHT DRAW
9246 85 3A              STA  LINES
9248 4C BC 93           JMP  JOYSTK
924B C9 2A    ASTCK     CMP  #"*"    ;FILL TO ANY/SAME
924D D0 09              BNE  ARRCK
924F A5 50              LDA  FILBOR  ;GET SAME/ANY FILL MODE FLAG
9251 49 01              EOR  #1      ;TOGGLE IT
9253 85 50              STA  FILBOR  ;STORE
9255 4C BC 93           JMP  JOYSTK
9258 C9 5E    ARRCK     CMP  #"^"    ;UP-ARROW (BORDER/BACKGROUND)
925A D0 0A              BNE  RETCK
925C 24 C5    WAITCH    BIT  LSTX    ;OVFLOW SET BY BIT 6
925E 50 FC              BVC  WAITCH  ;WAIT UNTIL BIT 6 SET (NO KEY PRESSED)
9260 20 C6 97           JSR  BORBAK  ;ADJUST BORDER/BACKGROUND COLORS
9263 4C 67 91           JMP  BEGIN
              ;
9266 C9 0D    RETCK     CMP  #13
9268 D0 5B              BNE  CLEAR
926A 20 37 98           JSR  OUT     ;ENABLE THE STATUS LINE
926D A0 27              LDY  #39
926F B9 00 9C MESSGE    LDA  STLINE,Y  ;GET A BYTE OF STATUS LINE
9272 99 C0 8F           STA  CRT+960,Y  ;PUT IT ON BOTTOM LINE
9275 88                 DEY
9276 10 F7              BPL  MESSGE  ;DISPLAY 39 TO 0
9278 A4 3B              LDY  PLOTMD  ;CHECK PLOT TYPE
927A B9 28 9C           LDA  FNUM,Y  ;LOAD (F)7,5,3,1
927D 8D C6 8F           STA  CRT+966  ;DISPLAY IT
9280 A5 39              LDA  DRAW    ;GET 0 OR 1 FOR MOVE/PLOT
9282 0A                 ASL  A
9283 0A                 ASL  A       ;SHIFT IT TWICE (* 4)
9284 A8                 TAY          ;TRANSFER TO Y FOR INDIRECT ACCESS
9285 A2 00              LDX  #0      ;ZERO X FOR THE DISPLAY LOOP
9287 B9 2C 9C ONOFF1    LDA  DRMD,Y  ;GET AN "ON  " OR "OFF " BYTE
928A 9D C8 8F           STA  CRT+968,X  ;DISPLAY IT
928D C8                 INY          ;NEXT BYTE OF MESSAGE
?928E E8                 INX          ;NEXT SCREEN BYTE
928F E0 04              CPX  #4      ;LAST BYTE DISPLAYED
9291 D0 F4              BNE  ONOFF1  ;NO
9293 A5 3A              LDA  LINES   ;GET LINE DRAW MODE (0-3)
9295 0A                 ASL  A
9296 0A                 ASL  A       ;MULTIPLY BY 4
9297 A8                 TAY
9298 A2 00              LDX  #0      ;SET UP INDEXERS
929A B9 34 9C ONOFF2    LDA  LNMD,Y  ;GET A BYTE
929D 9D D7 8F           STA  CRT+983,X  ;DISPLAY
92A0 C8                 INY          ;CONTINUE TO NEXT BYTE
92A1 E8                 INX
92A2 E0 04              CPX  #4
92A4 D0 F4              BNE  ONOFF2
92A6 A5 50              LDA  FILBOR  ;SET UP FILL TYPE (ANY/SAME)
92A8 0A                 ASL  A
92A9 0A                 ASL  A
92AA A8                 TAY
92AB A2 00              LDX  #0
92AD B9 44 9C ONOFF3    LDA  FLMD,Y
92B0 9D E4 8F           STA  CRT+996,X  ;DISPLAY
92B3 C8                 INY
92B4 E8                 INX
92B5 E0 04              CPX  #4
92B7 D0 F4              BNE  ONOFF3
92B9 A5 C5    RETWT     LDA  LSTX    ;GET KEY PRESSED
92BB C9 01              CMP  #1      ;RETURN KEY YIELDS 1
92BD F0 FA              BEQ  RETWT   ;WAIT UNTIL RELEASED
92BF 20 5F 98           JSR  IN      ;BRING BACK ALL-HIRES
```

```
92C2 4C BC 93           JMP  JOYSTK
                   ;
92C5 C9 93    CLEAR     CMP  #147     ;CLR KEY
92C7 DØ 48              BNE  RVSCK
92C9 AØ ØØ              LDY  #Ø       ;ZERO LO-BYTE ADDRESS
92CB 84 FD              STY  MISC
92CD 98                 TYA           ;ZERO .A
92CE A2 AØ              LDX  #>HIPAGE  ;X USED AS MISC+1
92DØ 86 FE    LOOP1     STX  MISC+1
92D2 91 FD    LOOP2     STA  (MISC),Y  ;.A = Ø
92D4 C8                 INY           ;LOW BYTE
92D5 DØ FB              BNE  LOOP2    ;WAIT ONE PAGE
92D7 E8                 INX           ;INCREMENT PAGE COUNT
92D8 EØ BF              CPX  #>31*256+HIPAGE  ;LAST PAGE
92DA DØ F4              BNE  LOOP1    ;NOT YET
92DC 86 FE              STX  MISC+1   ;SAVE IT FOR LAST PAGE
92DE 91 FD    LOOP3     STA  (MISC),Y  ;CLEAR 64 BYTES
92EØ C8                 INY           ;NEXT BYTE
92E1 CØ 4Ø              CPY  #64
92E3 DØ F9              BNE  LOOP3    ;NOT FINISHED
92E5 AØ ØØ              LDY  #Ø       ;CLEAR COLOR (TEXT) SCREEN
92E7 84 FD              STY  MISC     ;ZERO HIGH BYTE
92E9 A5 3D              LDA  PLCOL    ;GET CURRENT PLOT COLOR
92EB ØA                 ASL  A        ;SHIFT OVER 4 TIMES (* 16)
92EC ØA                 ASL  A
92ED ØA                 ASL  A
92EE ØA                 ASL  A
92EF 4D 21 DØ           EOR  VIC+33   ;ADD IN CURRENT BACKGROUND COLOR
92F2 29 FØ              AND  #%11110000
92F4 4D 21 DØ           EOR  VIC+33   ;LO = VIC+33, HI = PLCOL
92F7 A2 8C              LDX  #>CRT    ;HIGH BYTE
92F9 86 FE    COL1      STX  MISC+1
92FB 91 FD    COLCLR    STA  (MISC),Y  ;STORE COLOR ON SCREEN
92FD C8                 INY           ;NEXT BYTE
92FE DØ FB              BNE  COLCLR
93ØØ E8                 INX           ;NEXT PAGE
93Ø1 EØ 8F              CPX  #>3*256+CRT  ;3 PAGES ONLY
93Ø3 DØ F4              BNE  COL1     ;NOT LAST PAGE
93Ø5 86 FE              STX  MISC+1
93Ø7 91 FD    COL2      STA  (MISC),Y  ;ON LAST PAGE
93Ø9 C8                 INY
93ØA CØ E8              CPY  #232     ;232 BYTES ON LAST PAGE
93ØC DØ F9              BNE  COL2
93ØE 4C BC 93           JMP  JOYSTK
                   ;
9311 C9 12    RVSCK     CMP  #18      ;CONTROL-R (RVS ON)
9313 DØ 27              BNE  ENDCK
9315 AØ ØØ              LDY  #Ø
9317 84 FB              STY  AD       ;ZERO LOW BYTE
9319 A9 AØ              LDA  #>HIPAGE
931B 85 FC              STA  AD+1     ;SET HIGH BYTE
931D B1 FB    RVSLP1    LDA  (AD),Y
931F 49 FF              EOR  #$FF     ;FLIP ALL BITS
9321 91 FB              STA  (AD),Y   ;RETURN IT TO PAGE
9323 C8                 INY
9324 DØ F7              BNE  RVSLP1
9326 E6 FC              INC  AD+1
9328 A5 FC              LDA  AD+1     ;CHECK LAST PAGE
932A C9 BF              CMP  #>31*256+HIPAGE
932C DØ EF              BNE  RVSLP1   ;NOT YET
932E B1 FB    RVSLP2    LDA  (AD),Y
933Ø 49 FF              EOR  #$FF     ;FLIP LAST 64 BYTES
9332 91 FB              STA  (AD),Y
9334 C8                 INY
9335 CØ 4Ø              CPY  #64
9337 DØ F5              BNE  RVSLP2
9339 4C BC 93           JMP  JOYSTK
                   ;
933C C9 Ø6    ENDCK     CMP  #6       ;CONTROL-BACK ARROW
933E DØ 31              BNE  LSCK
934Ø A9 1B              LDA  #27      ;TURN OFF HIRES
9342 8D 11 DØ           STA  VIC+17
```

```
9345 A9 15              LDA   #21       ;RESTORE SCREEN/CHARACTER SET
9347 8D 18 DØ           STA   VIC+24
934A A9 Ø8              LDA   #8        ;TURN OFF MULTICOLOR (IF ON)
934C 8D 16 DØ           STA   VIC+22
934F A9 ØØ              LDA   #Ø
9351 8D 15 DØ           STA   VIC+21    ;TURN OFF SPRITES
9354 A9 C7              LDA   #196+3    ;VIC BANK Ø
9356 8D ØØ DD           STA   VBANK
9359 A9 37              LDA   #54+1     ;FLIP BASIC BACK IN
935B 85 Ø1              STA   R651Ø
935D 2Ø B2 98           JSR   RSTOFF    ;DISABLE RASTER INTERRUPTS
936Ø AØ 2D              LDY   #45       ;READ ZERO PAGE BACK IN
9362 B9 ØØ CB OFFZER    LDA   ZERSTO,Y   ;READ FROM $CBØØ
9365 99 39 ØØ           STA   57,Y
9368 A9 ØØ              LDA   #Ø
936A 99 ØØ Ø2           STA   INBUFF,Y   ;SIMULTANEOUSLY CLEAR LINE BUFFER
936D 88                 DEY
936E 1Ø F2              BPL   OFFZER
937Ø 6Ø                 RTS             ;RETURN TO BASIC
                  ;
9371 C9 4Ø    LSCK      CMP   #"@ ;LOAD OR SAVE
9373 DØ Ø3              BNE   COLRCK
9375 4C 2A 99           JMP   LS        ;GO DO INPUT/OUTPUT
                  ;
                  ;CHECK MULTIPLE-KEY FUNCTIONS
                  ;
9378 A2 ØF    COLRCK    LDX   #15       ;TOP OF TABLE
937A DD B5 9B LOOPC     CMP   COLORS,X   ;CHECK IF A COLOR
937D DØ Ø8              BNE   NEXTC     ;NO
937F 86 3D              STX   PLCOL     ;IN TABLE SO SAVE IT
9381 8E 27 DØ           STX   VIC+39    ;CHANGE SPRITE COLOR
9384 4C BC 93           JMP   JOYSTK
9387 CA       NEXTC     DEX             ;CHECK NEXT COLOR
9388 1Ø FØ              BPL   LOOPC     ;IF NOT FINISHED
                  ;
938A A2 Ø3              LDX   #3        ;TABLE TOP FOR FUNCTION KEYS
938C DD C5 9B LOOPF     CMP   FNCTNS,X   ;CHECK IF A FUNCTION KEY
938F DØ 1B              BNE   NEXTF
9391 A5 3C              LDA   SCRNMD    ;CHECK HIRES/MULTICOLOR
9393 DØ Ø6              BNE   SETX      ;IF MULTICOLOR
9395 EØ Ø3              CPX   #3        ;CHECK FOR F1
9397 DØ Ø2              BNE   SETX      ;NO
9399 A2 Ø1              LDX   #1        ;YES, SO SET IT F5
939B 86 3B    SETX      STX   PLOTMD    ;SET PLOT MODE
939D A2 Ø8              LDX   #8        ;DEFAULT TYPE
939F A5 3B              LDA   PLOTMD
93A1 FØ Ø4              BEQ   SETT      ;IF ERASE
93A3 ØA                 ASL   A         ;SHIFT OVER 3 TIMES ...
93A4 ØA                 ASL   A
93A5 ØA                 ASL   A
93A6 AA                 TAX             ;... AND STORE IN .X
93A7 86 46    SETT      STX   TYPE      ;PUT IN TYPE
93A9 4C BC 93           JMP   JOYSTK
93AC CA       NEXTF     DEX             ;NEXT FUNCTION KEY
93AD 1Ø DD              BPL   LOOPF
                  ;
93AF 38                 SEC             ;CHECK IF A NUMBER 1 TO 9
93BØ 29 EF              AND   #239      ;TO INCLUDE SHIFTED NUMBERS
93B2 E9 2Ø              SBC   #32       ;TO MAKE IT HEX Ø-9
93B4 FØ Ø6              BEQ   JOYSTK    ;A "Ø" NO GOOD
93B6 C9 ØA              CMP   #1Ø       ;AN ASCII ":" OR GREATER
93B8 BØ Ø2              BCS   JOYSTK
93BA 85 3E              STA   PLINC     ;NEW PLOT INCREMENT
                  ;
                  ;CHECK THE JOYSTICK
                  ;
93BC A5 A2    JOYSTK    LDA   TIME+2    ;CURRENT JIFFY
93BE C5 41              CMP   TIMSTO    ;LAST RECORDED VALUE
93CØ DØ Ø3              BNE   GETJOY    ;ONLY ONCE/JIFFY
93C2 4C 5E 94           JMP   PLOT
93C5 A5 A2    GETJOY    LDA   TIME+2    ;JIFFY COUNTER
93C7 85 41              STA   TIMSTO    ;RESET WAIT LOOP
```

```
93C9 AD 00 DC           LDA  JSTICK
93CC 49 7F              EOR  #%01111111  ;SET PUSHED = 1
93CE 85 42              STA  JOY       ;SAVE FOR FUTURE REFERENCE
93D0 A5 C5              LDA  LSTX      ;CHECK KEY PUSHED
93D2 C9 05              CMP  #5        ;IF NOT F3
93D4 D0 03              BNE  FIRECK    ;THEN CHECK JOYSTICK
93D6 4C 5E 94           JMP  PLOT      ;ELSE SKIP
93D9 A5 42     FIRECK   LDA  JOY
93DB 29 10              AND  #%00010000  ;CHECK FOR FIRE BUTTON
93DD F0 19              BEQ  SETPR     ;NOT PRESSED
93DF A5 3A              LDA  LINES
93E1 F0 06              BEQ  BNCECK    ;NO LINE MODE
93E3 20 38 95           JSR  LINE      ;DRAW  A LINE
93E6 4C F8 93           JMP  SETPR
93E9 A5 40     BNCECK   LDA  PRESSD    ;IF JUST PRESSED
93EB D0 0F              BNE  CHECKU
93ED E6 40              INC  PRESSD    ;SET AS JUST PRESSED
93EF A5 39              LDA  DRAW
93F1 49 01              EOR  #%00000001  ;TOGGLE
93F3 85 39              STA  DRAW
93F5 4C FC 93           JMP  CHECKU    ;SKIP RESET
93F8 A9 00     SETPR    LDA  #0
93FA 85 40              STA  PRESSD    ;NOT JUST PRESSED
               ;
93FC A5 42     CHECKU   LDA  JOY
93FE 29 01              AND  #%00000001  ;UP
9400 F0 0B              BEQ  CHECKD    ;NOT PRESSED
9402 A5 45              LDA  YPOS      ;DECREMENT YPOS
9404 38                 SEC
9405 E5 3E              SBC  PLINC
9407 C9 C8              CMP  #200      ;WRAP-AROUND ILLEGAL
9409 B0 02              BCS  CHECKD
940B 85 45              STA  YPOS      ;LEGAL, SO STORE IT
               ;
940D A5 42     CHECKD   LDA  JOY
940F 29 02              AND  #%00000010  ;DOWN
9411 F0 0B              BEQ  CHECKL    ;NOT PRESSED
9413 A5 45              LDA  YPOS      ;INCREMENT YPOS
9415 18                 CLC
9416 65 3E              ADC  PLINC
9418 C9 C8              CMP  #200      ;CHECK IF YPOS>=200
941A B0 02              BCS  CHECKL    ;IT IS
941C 85 45              STA  YPOS      ;NO, SO STORE IT
               ;
941E A5 42     CHECKL   LDA  JOY
9420 29 04              AND  #%00000100  ;LEFT
9422 F0 17              BEQ  CHECKR    ;PRESSED
9424 A5 43              LDA  XPOS      ;DECREMENT LOW BYTE
9426 38                 SEC
9427 E5 3E              SBC  PLINC
9429 85 FD              STA  MISC      ;TEMP SAVE
942B A5 44              LDA  XPOS+1    ;DECREMENT HIGH IF NECESSARY
942D E9 00              SBC  #0
942F 85 FE              STA  MISC+1
9431 30 08              BMI  CHECKR    ;IF HIGH BYTE NEGATIVE, ILLEGAL
9433 A5 FD              LDA  MISC      ;ELSE STORE MISCS
9435 85 43              STA  XPOS
9437 A5 FE              LDA  MISC+1
9439 85 44              STA  XPOS+1
               ;
943B A5 42     CHECKR   LDA  JOY
943D 29 08              AND  #%00001000  ;RIGHT
943F F0 1D              BEQ  PLOT      ;NOT PRESSED
9441 A5 43              LDA  XPOS      ;INCREMENT LOW BYTE
9443 18                 CLC
9444 65 3E              ADC  PLINC
9446 85 FD              STA  MISC      ;TEMPORARY STORAGE
9448 A5 44              LDA  XPOS+1    ;INCREMENT HIGH IF NECESSARY
944A 69 00              ADC  #0
944C 85 FE              STA  MISC+1
944E F0 06              BEQ  DOR       ;LEGAL
9450 A5 FD              LDA  MISC
```

```
9452 C9 40             CMP  #64      ;FAR RIGHT EDGE
9454 B0 08             BCS  PLOT     ;ILLEGAL
9456 A5 FD    DOR      LDA  MISC     ;LEGAL, SO STORE
9458 85 43             STA  XPOS
945A A5 FE             LDA  MISC+1
945C 85 44             STA  XPOS+1
              ;
              ;MAIN PLOTTING ROUTINE
              ;
945E A5 43    PLOT     LDA  XPOS     ;POSITION SPRITE
9460 A6 3C             LDX  SCRNMD
9462 F0 02             BEQ  OFFSET   ;IF NOT MULTICOLOR
9464 29 FE             AND  #254     ;HALF HORIZONTAL RESOLUTION
9466 18       OFFSET   CLC
9467 69 0D             ADC  #13      ;HORIZONTAL OFFSET
9469 8D 00 D0          STA  VIC      ;SPRITE 0 X REGISTER
946C A5 44             LDA  XPOS+1   ;HIGH BYTE
946E 69 00             ADC  #0
9470 8D 10 D0          STA  VIC+16   ;SPRITE 0'S BIT 8
9473 A5 45             LDA  YPOS
9475 69 2B             ADC  #43      ;VERTICAL OFFSET
9477 8D 01 D0          STA  VIC+1    ;SPRITE 0 Y REGISTER
947A A5 39             LDA  DRAW     ;CHECK IF JUST MOVING
947C D0 03             BNE  DOPL     ;NO
947E 4C 67 91          JMP  BEGIN    ;IF DRAW = 0
9481 A5 3C    DOPL     LDA  SCRNMD
9483 D0 0C             BNE  DOPL1    ;IF IN MULTICOLOR
9485 A5 3B             LDA  PLOTMD
9487 C9 02             CMP  #2
9489 D0 06             BNE  DOPL1    ;IF NOT IN F3-MODE
948B 20 E7 94          JSR  COLPUT   ;SKIP BIT-PLOTTING
948E 4C 67 91          JMP  BEGIN    ;RETURN TO LOOP
9491 20 C2 94 DOPL1    JSR  PLOTIT
9494 4C 67 91          JMP  BEGIN
              ;
9497 A5 45    LOC      LDA  YPOS     ;FIND ADDRESS IN AD
9499 4A                LSR  A
949A 4A                LSR  A
949B 4A                LSR  A        ;DIVIDE BY 8
949C AA                TAX           ;SHIFT TO .X FOR LATER USE
949D A5 43             LDA  XPOS     ;SET UP LOW YPOS HIGH XPOS
949F 45 45             EOR  YPOS
94A1 29 F8             AND  #%11111000
94A3 45 45             EOR  YPOS
94A5 18                CLC
94A6 7D 80 CB          ADC  LINEL,X   ;ADD LO-BYTE LINE ADDRESSES
94A9 85 FB             STA  AD       ;STORE IT
94AB A5 44             LDA  XPOS+1   ;GET HIGH BYTE
94AD 7D A0 CB          ADC  LINEH,X   ;ADD HIGH-BYTE ADDRESSES
94B0 85 FC             STA  AD+1     ;STORE IT
94B2 A5 43             LDA  XPOS
94B4 29 07             AND  #7       ;GET LOW 3 BITS
94B6 A6 3C             LDX  SCRNMD
94B8 F0 04             BEQ  TRANS    ;IF HIRES
94BA 29 FE             AND  #254
94BC 05 46             ORA  TYPE     ;ADD ON TABLE OFFSET FOR PIXEL/MASK
94BE AA       TRANS    TAX           ;LEAVE TABLE POINTER IN .X
94BF A0 00             LDY  #0       ;ZERO .Y FOR USE
94C1 60                RTS           ;END OF SUBROUTINE
              ;
94C2 20 97 94 PLOTIT   JSR  LOC      ;USE ABOVE ROUTINE FOR AD
94C5 A5 4F             LDA  XORIT
94C7 D0 0C             BNE  XOR      ;IF RUBBERBAND
94C9 A5 3B             LDA  PLOTMD
94CB D0 10             BNE  ADD      ;IF NOT F7
94CD B1 FB             LDA  (AD),Y   ;GET FROM HIRES SCREEN
94CF 3D E0 CB          AND  MASK,X   ;AND OFF ALL BUT PIXEL
94D2 4C E5 94          JMP  POKE     ;GO AND LOAD IT BACK
94D5 B1 FB    XOR      LDA  (AD),Y
94D7 5D C0 CB          EOR  PIXEL,X   ;FLIP THAT PIXEL
94DA 4C E5 94          JMP  POKE
94DD B1 FB    ADD      LDA  (AD),Y
```

```
94DF 3D E0 CB           AND  MASK,X   ;CLEAR THE SPACE
94E2 1D C0 CB           ORA  PIXEL,X  ;LOAD IN THE CORRECT PIXEL
94E5 91 FB     POKE     STA  (AD),Y   ;BACK ON THE SCREEN
               ;
               ;POKE CORRECT COLOR VALUE
               ;
94E7 A5 3B     COLPUT   LDA  PLOTMD
94E9 D0 01              BNE  DOCOL    ;IF NOT F7
94EB 60                 RTS           ;NO, DON'T RESET COLORS
94EC A5 45     DOCOL    LDA  YPOS
94EE 4A                 LSR  A        ;GET YPOS/8
94EF 4A                 LSR  A
94F0 4A                 LSR  A
94F1 A8                 TAY           ;USE AS POINTER TO 40'S TABLE
94F2 A5 44              LDA  XPOS+1
94F4 4A                 LSR  A        ;GET BIT FROM HIGH BYTE
94F5 A5 43              LDA  XPOS     ;LOAD LOW BYTE
94F7 6A                 ROR  A        ;GET THE HIGH BIT IN
94F8 4A                 LSR  A        ;SHIFT TWICE
94F9 4A                 LSR  A
94FA 18                 CLC
94FB 79 40 CB           ADC  FORTYL,Y  ;ADD ON LOW BYTE
94FE 85 FD              STA  MISC     ;MISC HOLDS COLOR SCREEN ADDRESS
9500 B9 60 CB           LDA  FORTYH,Y  ;AND HIGH BYTE
9503 69 00              ADC  #0       ;IF SPILLOVER
9505 85 FE              STA  MISC+1
9507 A0 00              LDY  #0       ;FOR LATER USE
               ;
9509 A5 3B     MODE1    LDA  PLOTMD
950B C9 01              CMP  #%01     ;F5
950D D0 11              BNE  MODE2
950F B1 FD     DO1      LDA  (MISC),Y
9511 29 0F              AND  #%00001111  ;TAKE LOW NYBBLE
9513 85 FB              STA  AD       ;TEMP
9515 A5 3D              LDA  PLCOL
9517 0A                 ASL  A        ;PUSH COLOR INTO HIGH NYBBLE
9518 0A                 ASL  A
9519 0A                 ASL  A
951A 0A                 ASL  A
951B 05 FB              ORA  AD       ;PUT TOGETHER ..
951D 91 FD              STA  (MISC),Y  ;AND STORE AT COLOR SCREEN
951F 60                 RTS
               ;
9520 C9 02     MODE2    CMP  #%10     ;IF F3
9522 D0 09              BNE  MODE3
9524 B1 FD              LDA  (MISC),Y
9526 29 F0              AND  #%11110000  ;CLEAR LOW NYBBLE
9528 05 3D              ORA  PLCOL    ;ADD COLOR IN
952A 91 FD              STA  (MISC),Y  ;STORE
952C 60                 RTS
               ;
952D A5 FE     MODE3    LDA  MISC+1   ;TO COLOR SCREEN
952F 49 54              EOR  #%01010100  ;CHANGE HIGH BYTE TO $D800
9531 85 FE              STA  MISC+1
9533 A5 3D              LDA  PLCOL
9535 91 FD              STA  (MISC),Y  ;ON COLOR SCREEN
9537 60                 RTS
               ;
               ;SUBROUTINE TO DRAW A LINE
               ;
9538 A5 43     LINE     LDA  XPOS     ;TRANSFER CURRENT TO END
953A 85 54              STA  X2
953C A5 44              LDA  XPOS+1
953E 85 55              STA  X2+1
9540 A5 45              LDA  YPOS
9542 85 56              STA  Y2
9544 20 6A 95           JSR  LINEDO   ;EXECUTE THE LINE
9547 A5 54              LDA  X2
9549 85 43              STA  XPOS     ;TRANSFER IT BACK
954B A5 55              LDA  X2+1
954D 85 44              STA  XPOS+1
954F A5 56              LDA  Y2
```

```
9551 85 45              STA  YPOS
9553 A5 3A              LDA  LINES
9555 C9 Ø2              CMP  #2
9557 DØ 1Ø              BNE  FINLIN    ;IF NOT A DRAWTO
9559 A5 4F              LDA  XORIT
955B DØ ØC              BNE  FINLIN    ;IF A RUBBERBAND
955D A5 54              LDA  X2
955F 85 51              STA  X1        ;OTHERWISE, SET A NEW BEGINNING
9561 A5 55              LDA  X2+1
9563 85 52              STA  X1+1
9565 A5 56              LDA  Y2
9567 85 53              STA  Y1
9569 6Ø        FINLIN   RTS            ;END OF CONTROL LOOP
               ;
956A A5 3A     LINEDO   LDA  LINES
956C C9 Ø3              CMP  #3
956E DØ 2B              BNE  LINER     ;IF NOT A RIGHT-DRAW
957Ø A5 3C              LDA  SCRNMD
9572 FØ Ø6              BEQ  RGT       ;IF HIRES
9574 A5 43              LDA  XPOS
9576 29 FE              AND  #254      ;ONLY EVEN POSITIONS
9578 85 43              STA  XPOS
957A A5 43     RGT      LDA  XPOS      ;INCREMENT OUR X-POSITION
957C 18                 CLC
957D 65 4A              ADC  FLLINC
957F 85 43              STA  XPOS
9581 9Ø Ø2              BCC  RGT2      ;CARRY CLEAR IF NO OVERFLOW
9583 E6 44              INC  XPOS+1    ;OTHERWISE INCREMENT HIGH BYTE
9585 A5 44     RGT2     LDA  XPOS+1
9587 FØ Ø6              BEQ  SIDE      ;NOT TO RIGHT SIDE YET
9589 A5 43              LDA  XPOS
958B C9 4Ø              CMP  #64
958D BØ ØB              BCS  ENDIT     ;IF AT RIGHT EDGE
958F 2Ø 82 97  SIDE     JSR  PEEK      ;CHECK X,Y POSITION
9592 FØ Ø6              BEQ  ENDIT     ;IF A STOP PATTERN
9594 2Ø C2 94           JSR  PLOTIT    ;PLOT THE POINT
9597 4C 7A 95           JMP  RGT       ;REPEAT THE LOOP
959A 6Ø        ENDIT    RTS
               ;
959B A5 54     LINER    LDA  X2        ;DEFAULT XDIFF,YDIFF
959D 38                 SEC            ;SET XDIFF = X2 - X1
959E E5 51              SBC  X1
95AØ 85 57              STA  XDIFF
95A2 A5 55              LDA  X2+1      ;HIGH BYTES
95A4 E5 52              SBC  X1+1
95A6 85 58              STA  XDIFF+1
95A8 A5 56              LDA  Y2        ;AND SET YDIFF = Y2 - Y1
95AA 38                 SEC
95AB E5 53              SBC  Y1
95AD 85 59              STA  YDIFF

95AF AØ Ø1              LDY  #1        ;FIND SIGNUM OF X'S
95B1 A2 ØØ              LDX  #Ø        ;DEFAULT TO $ØØØ1 (IN .X,.Y)
95B3 A5 52              LDA  X1+1
95B5 C5 55              CMP  X2+1
95B7 9Ø 19              BCC  SGNY      ;IF HIGH BYTE X2<X1 THEN OK
95B9 DØ Ø6              BNE  CHX       ;IF NOT EQUAL, CHANGE SIGNUM
95BB A5 54              LDA  X2        ;CHECK LOW BYTES
95BD C5 51              CMP  X1
95BF BØ 11              BCS  SGNY      ;IF X1>=X2 THEN OK
95C1 AØ FF     CHX      LDY  #$FF      ;CHANGE SIGNUM TO $FFFF
95C3 A2 FF              LDX  #$FF
95C5 A5 51              LDA  X1        ;AND MAKE XDIFF = X1 - X2
95C7 38                 SEC
95C8 E5 54              SBC  X2
95CA 85 57              STA  XDIFF
95CC A5 52              LDA  X1+1      ;HIGH BYTES
95CE E5 55              SBC  X2+1
95DØ 85 58              STA  XDIFF+1
95D2 84 64     SGNY     STY  XSGN      ;STORE NEW SIGNUM
95D4 86 65              STX  XSGN+1
```

```
95D6 A0 01               LDY  #1        ;FIND SIGNUM OF Y'S
95D8 A5 56               LDA  Y2
95DA C5 53               CMP  Y1
95DC B0 09               BCS  ABSX      ;IF Y1>=Y2 THEN OK
95DE A0 FF               LDY  #$FF      ;ELSE CHANGE SIGNS
95E0 A5 53               LDA  Y1        ;AND MAKE YDIFF = Y1 - Y2
95E2 38                  SEC
95E3 E5 56               SBC  Y2
95E5 85 59               STA  YDIFF
95E7 84 66      ABSX     STY  YSGN      ;STORE Y SIGNUM
95E9 A9 00               LDA  #0        ;ZERO X & Y COUNTER
95EB 85 62               STA  YCNTR
95ED 85 60               STA  XCNTR
                ;
95EF A6 57      DISFND   LDX  XDIFF     ;FIND GREATER DIS
95F1 A4 58               LDY  XDIFF+1
95F3 D0 0E               BNE  XSET      ;IF HIGH BYTE X SET, GREATER THAN Y
95F5 E4 59               CPX  YDIFF     ;ELSE COMPARE LOW BYTES
95F7 B0 0A               BCS  XSET      ;IF YDIFF>=XDIFF, SET YDIFF GREATER
95F9 A6 59               LDX  YDIFF
95FB 20 0B 96            JSR  HAFDIS    ;PUT .Y,.X IN DIS
95FE 85 60               STA  XCNTR     ;.A HOLDS HALF DISTANCE
9600 4C 14 96            JMP  INIT
9603 20 0B 96 XSET       JSR  HAFDIS
9606 85 62               STA  YCNTR     ;STORE DIS/2 IN YCNTR
9608 4C 14 96            JMP  INIT
960B 84 5B      HAFDIS   STY  DIS+1     ;HIGH BYTE
960D 98                  TYA            ;GET THE CARRY BIT
960E 4A                  LSR  A
960F 86 5A               STX  DIS       ;LOW BYTE
9611 8A                  TXA
9612 6A                  ROR  A         ;CARRY BYTE IN AT LEFT
9613 60                  RTS            ;RETURN 1/2 DIS IN .A
                ;
9614 A9 00      INIT     LDA  #0        ;INITIALIZE VARIABLES
9616 85 5E               STA  CNTR      ;ZERO DISTANCE COUNTER
9618 85 5F               STA  CNTR+1
961A 85 61               STA  XCNTR+1    ;ZERO HIGH BYTES X,Y COUNTERS
961C 85 63               STA  YCNTR+1
961E A5 51               LDA  X1        ;INITIALIZE XPOS,YPOS
9620 85 43               STA  XPOS
9622 A5 52               LDA  X1+1
9624 85 44               STA  XPOS+1
9626 A5 53               LDA  Y1
9628 85 45               STA  YPOS
962A A5 5A               LDA  DIS       ;SET UP DIS2
962C 18                  CLC
962D 69 01               ADC  #1
962F 85 5C               STA  DIS2
9631 A5 5B               LDA  DIS+1     ;HIGH BYTES
9633 69 00               ADC  #0
9635 85 5D               STA  DIS2+1
                ;
9637 A5 3C      LOOP     LDA  SCRNMD
9639 F0 0E               BEQ  LOOPDO    ;IF IN HIRES MODE
963B A5 FE               LDA  MISC+1
963D C5 45               CMP  YPOS
963F D0 08               BNE  LOOPDO    ;MISC+1 HOLDS OLD YPOS
9641 A5 43               LDA  XPOS
9643 29 FE               AND  #$FE      ;EVEN COLUMNS ONLY
9645 C5 FD               CMP  MISC
9647 F0 03               BEQ  XINC      ;MISC HOLDS OLD XPOS
9649 20 C2 94 LOOPDO     JSR  PLOTIT    ;PLOT POINT
                ;
964C A5 43      XINC     LDA  XPOS
964E 29 FE               AND  #$FE
9650 85 FD               STA  MISC      ;SAVE MISC
9652 A5 45               LDA  YPOS
9654 85 FE               STA  MISC+1    ;SAVE YPOS
9656 A5 60               LDA  XCNTR     ;ADD XDIFF TO XCNTR
9658 18                  CLC
9659 65 57               ADC  XDIFF
```

```
965B 85 60          STA  XCNTR
965D A5 61          LDA  XCNTR+1  ;HIGH BYTES
965F 65 58          ADC  XDIFF+1
9661 85 61          STA  XCNTR+1
9663 C5 5B          CMP  DIS+1
9665 F0 04          BEQ  LOXCK   ;CHECK LOW BYTES
9667 90 21          BCC  YGREAT  ;NO INCREMENT TO XPOS
9669 D0 06          BNE  XDO     ;GO INCREMENT XPOS
966B A5 60   LOXCK  LDA  XCNTR
966D C5 5A          CMP  DIS
966F 90 19          BCC  YGREAT  ;IGNORE IF DIS<XCNTR
9671 A5 60   XDO    LDA  XCNTR   ;PULL DIS OFF XCNTR
9673 E5 5A          SBC  DIS
9675 85 60          STA  XCNTR
9677 A5 61          LDA  XCNTR+1  ;HIGH BYTES
9679 E5 5B          SBC  DIS+1
967B 85 61          STA  XCNTR+1
967D A5 43          LDA  XPOS    ;ADD XSGN (1 OR -1) TO YPOS
967F 18             CLC
9680 65 64          ADC  XSGN
9682 85 43          STA  XPOS
9684 A5 44          LDA  XPOS+1
9686 65 65          ADC  XSGN+1
9688 85 44          STA  XPOS+1
             ;
968A A5 62   YGREAT LDA  YCNTR   ;ADD YDIFF
968C 18             CLC
968D 65 59          ADC  YDIFF
968F 85 62          STA  YCNTR
9691 A5 63          LDA  YCNTR+1  ;HIGH BYTES
9693 69 00          ADC  #0
9695 85 63          STA  YCNTR+1
9697 C5 5B          CMP  DIS+1
9699 F0 04          BEQ  LOYCK   ;CHECK LOW BYTES
969B 90 1B          BCC  CINC    ;DON'T CHANGE YPOS
969D D0 06          BNE  YDEC    ;CHANGE YPOS
969F A5 62   LOYCK  LDA  YCNTR
96A1 C5 5A          CMP  DIS
96A3 90 13          BCC  CINC    ;IGNORE IF LESS
96A5 A5 62   YDEC   LDA  YCNTR   ;PULL DIS FROM YCNTR
96A7 E5 5A          SBC  DIS     ;CARRY SET
96A9 85 62          STA  YCNTR
96AB A5 63          LDA  YCNTR+1  ;HIGH BYTES
96AD E5 5B          SBC  DIS+1
96AF 85 63          STA  YCNTR+1
96B1 A5 45          LDA  YPOS    ;ADD YSGN (1 OR -1) TO YPOS
96B3 18             CLC
96B4 65 66          ADC  YSGN
96B6 85 45          STA  YPOS
             ;
96B8 E6 5E   CINC   INC  CNTR    ;INCREMENT COUNTER
96BA D0 02          BNE  CINC2   ;IF NO OVERFLOW
96BC E6 5F          INC  CNTR+1  ;INCREMENT HIGH BYTE
96BE A5 5F   CINC2  LDA  CNTR+1
96C0 C5 5D          CMP  DIS2+1
96C2 90 06          BCC  CONT    ;NOT FINISHED YET
96C4 A5 5E          LDA  CNTR
96C6 C5 5C          CMP  DIS2
96C8 B0 03          BCS  RETURN  ;FINISHED
96CA 4C 37 96 CONT  JMP  LOOP    ;BACK TO LOOP
96CD 60      RETURN RTS
             ;
             ;SUBROUTINE TO FILL AN AREA
             ;
96CE A9 00   FILL   LDA  #0
96D0 85 3F          STA  PNTR    ;BOTTOM OF STACK
96D2 A5 3C          LDA  SCRNMD
96D4 F0 06          BEQ  BTFIND  ;IF HIRES
96D6 A5 43          LDA  XPOS
96D8 29 FE          AND  #$FE
96DA 85 43          STA  XPOS    ;SET TO AN EVEN COLUMN
96DC A9 00   BTFIND LDA  #0      ;CLEAR UP/DOWN FLAGS
```

```
96DE 85 48              STA  DOWNF
96E0 85 47              STA  UPF
96E2 A5 44     FIND     LDA  XPOS+1
96E4 D0 04              BNE  DOFIND   ;STILL ON 256-319
96E6 A5 43              LDA  XPOS
96E8 F0 1F              BEQ  FILLBT   ;AT LEFT EDGE
96EA A5 43     DOFIND   LDA  XPOS     ;DECREMENT XPOS
96EC 38                 SEC
96ED E5 4A              SBC  FLLINC
96EF 85 43              STA  XPOS
96F1 A5 44              LDA  XPOS+1   ;HIGH BYTES
96F3 E9 00              SBC  #0
96F5 85 44              STA  XPOS+1
96F7 20 82 97           JSR  PEEK
96FA D0 E6              BNE  FIND     ;KEEP SCANNING LEFT
96FC A5 43              LDA  XPOS     ;BACK RIGHT ONE PIXEL
96FE 18                 CLC
96FF 65 4A              ADC  FLLINC
9701 85 43              STA  XPOS
9703 A5 44              LDA  XPOS+1
9705 69 00              ADC  #0
9707 85 44              STA  XPOS+1
               ;
9709 E6 45     FILLBT   INC  YPOS     ;CHECK BELOW
970B 20 82 97           JSR  PEEK     ;GET A VALUE
970E F0 0D              BEQ  DZER     ;EQUAL TO PLOT, SO ...
9710 A5 48              LDA  DOWNF
9712 D0 0D              BNE  UCK      ;FLAG SET
9714 20 B2 97           JSR  PUSH     ;STORE THIS LOCATION
9717 A9 01              LDA  #1
9719 85 48              STA  DOWNF    ;SET THE FLAG
971B D0 04              BNE  UCK      ;BRANCH-ALWAYS
971D A9 00     DZER     LDA  #0
971F 85 48              STA  DOWNF    ;... RESET DOWN FLAG
9721 C6 45     UCK      DEC  YPOS
9723 C6 45              DEC  YPOS     ;LOOK ABOVE THE PLOT
9725 20 82 97           JSR  PEEK
9728 F0 0D              BEQ  UZER     ;IF THE SAME AS PLOT ...
972A A5 47              LDA  UPF
972C D0 0D              BNE  DOPOKE   ;FLAG SET
972E 20 B2 97           JSR  PUSH     ;SAVE THE LOCATION
9731 A9 01              LDA  #1
9733 85 47              STA  UPF      ;SET THE UP FLAG
9735 D0 04              BNE  DOPOKE   ;BRANCH-ALWAYS
9737 A9 00     UZER     LDA  #0
9739 85 47              STA  UPF      ;... CLEAR THE FLAG
973B E6 45     DOPOKE   INC  YPOS     ;RESET YPOS
973D 20 C2 94           JSR  PLOTIT   ;PLOT THIS POINT
9740 A5 43              LDA  XPOS     ;GO A PIXEL RIGHT
9742 18                 CLC
9743 65 4A              ADC  FLLINC
9745 85 43              STA  XPOS
9747 A5 44              LDA  XPOS+1   ;HIGH BYTES
9749 69 00              ADC  #0
974B 85 44              STA  XPOS+1
974D A5 44     PXCK     LDA  XPOS+1
974F F0 06              BEQ  ENDTST   ;NOT AT RIGHT EDGE
9751 A5 43              LDA  XPOS
9753 C9 40              CMP  #64
9755 B0 05              BCS  PULL     ;AT RIGHT EDGE
9757 20 82 97  ENDTST   JSR  PEEK
975A D0 AD              BNE  FILLBT   ;NOT TO A SAME PIXEL
975C A4 3F     PULL     LDY  PNTR
975E F0 65              BEQ  ENDPSH   ;NOTHING LEFT TO PULL
9760 20 E4 FF           JSR  GETIN
9763 C9 00              CMP  #0
9765 D0 5E              BNE  ENDPSH   ;KEY PRESSED = ABORT
9767 88        DOPULL   DEY
9768 B9 00 9D           LDA  TABLE1,Y   ;PULL Y,X,X+1 OFF TABLES
976B 85 45              STA  YPOS
976D B9 00 9E           LDA  TABLE2,Y
9770 85 44              STA  XPOS+1
```

```
9772 B9 00 9F           LDA  TABLE3,Y
9775 85 43              STA  XPOS
9777 84 3F              STY  PNTR      ;STORE RESET POINTER
9779 A5 45              LDA  YPOS
977B C9 C8              CMP  #200
977D B0 DD              BCS  PULL      ;IF OUT OF RANGE
977F 4C DC 96           JMP  BTFIND    ;RE-ENTER LOOP
                 ;
9782 20 97 94 PEEK      JSR  LOC       ;GET ADDRESS AND .X
9785 BD E0 CB           LDA  MASK,X
9788 49 FF              EOR  #$FF
978A 31 FB              AND  (AD),Y    ;MASK OFF ALL BUT PIXELS
978C 48                 PHA            ;PUSH ONTO STACK
978D 8A                 TXA            ;AND X WITH #7
978E 29 07              AND  #7
9790 AA                 TAX
9791 68                 PLA            ;RECALL THE PIXEL PATTERN
9792 E4 49              CPX  FLSHFT
9794 B0 06              BCS  ENDPK     ;DON'T SHIFT
9796 4A          PEEKLP LSR  A         ;SHIFT TOWARDS BOTTOM BITS
9797 E8                 INX            ;INCREMENT SHIFT COUNTER
9798 E4 49              CPX  FLSHFT
979A 90 FA              BCC  PEEKLP    ;NOT FINISHED
979C A6 50       ENDPK  LDX  FILBOR
979E D0 05              BNE  ANY       ;ANY PIXEL WILL STOP THE FILL
97A0 C5 3B              CMP  PLOTMD    ;ZERO SET IF END OF LINE
97A2 4C B1 97           JMP  ENDPK2
97A5 A2 01       ANY    LDX  #1        ;CLEAR ZERO FLAG
97A7 08                 PHP            ;SAVE THIS
97A8 C9 00              CMP  #0        ;CHECK FOR SOMETHING ON SCREEN
97AA F0 04              BEQ  NOTH      ;NOTHING
97AC 28                 PLP            ;SOMETHING, SO PULL ...
97AD A2 00              LDX  #0        ;SET ZERO ...
97AF 60                 RTS            ;AND RETURN
97B0 28          NOTH   PLP            ;PULL THE ZERO FLAG
97B1 60          ENDPK2 RTS            ;RETURN
                 ;
97B2 A4 3F       PUSH   LDY  PNTR      ;FILL STACK POINTER
97B4 A5 43              LDA  XPOS
97B6 99 00 9F           STA  TABLE3,Y  ;PUSH XPOS
97B9 A5 44              LDA  XPOS+1
97BB 99 00 9E           STA  TABLE2,Y  ;PUSH HIGH BYTE XPOS
97BE A5 45              LDA  YPOS
97C0 99 00 9D           STA  TABLE1,Y  ;PUSH YPOS
97C3 E6 3F              INC  PNTR      ;MOVE UP POINTER
97C5 60          ENDPSH RTS
                 ;
                 ;CHANGE BORDER BACKGROUND
                 ;
97C6 20 15 98 BORBAK    JSR  KEYS      ;GET A DIRECTION
97C9 C9 FF              CMP  #$FF
97CB D0 01              BNE  CHECK     ;A DIRECTION OR NOTHING
97CD 60                 RTS            ;NOT RECOGNIZED BY KEYS
97CE C9 00       CHECK  CMP  #0
97D0 D0 07              BNE  EXEC      ;SOME DIRECTION WAS PRESSED
97D2 AD 00 DC           LDA  JSTICK
97D5 49 7F              EOR  #127
97D7 85 42              STA  JOY       ;GET A DIRECTION FROM JOYSTICK
97D9 A5 42       EXEC   LDA  JOY
97DB 29 10              AND  #16
97DD D0 35              BNE  ENDBB     ;FIRE BUTTON PRESSED
97DF A5 42              LDA  JOY
97E1 29 03              AND  #3
97E3 F0 10              BEQ  NEXT1     ;NO VERTICAL MOTION
97E5 0A                 ASL  A         ;DOUBLE
97E6 38                 SEC            ;SUBTRACT 3 (-1 OR 1 RESULTS)
97E7 E9 03              SBC  #3
97E9 49 FE              EOR  #$FE      ;TO GET 1 OR -1
97EB 18                 CLC            ;ADD BACKGROUND TO IT
97EC 6D 21 D0           ADC  VIC+33
97EF 8D 21 D0           STA  VIC+33
97F2 4C 06 98           JMP  DELIT
```

```
97F5 A5 42     NEXT1     LDA  JOY
97F7 29 ØC               AND  #12
97F9 FØ CB               BEQ  BORBAK   ;NO HORIZONTAL MOTION
97FB 4A                  LSR  A        ;MAKE IT 2 OR 4
97FC 38                  SEC
97FD E9 Ø3               SBC  #3       ;$FF OR $Ø1
97FF 18                  CLC           ;ADD BORDER COLOR
98ØØ 6D 2Ø DØ            ADC  VIC+32
98Ø3 8D 2Ø DØ            STA  VIC+32   ;STORE IT
98Ø6 A2 4Ø     DELIT     LDX  #64      ;64 LONG LOOPS
98Ø8 AØ FF               LDY  #$FF     ;ONE LONG LOOP
98ØA 48        DELIT1    PHA           ;DELAY
98ØB 68                  PLA
98ØC 88                  DEY
98ØD DØ FB               BNE  DELIT1
98ØF CA                  DEX
981Ø DØ F8               BNE  DELIT1
9812 FØ B2               BEQ  BORBAK   ;BRANCH-ALWAYS
9814 6Ø        ENDBB     RTS           ;RETURN TO MAIN LOOP
               ;
               ;GET DIR FROM KEYBOARD
               ;
9815 A5 C5     KEYS      LDA  LSTX
9817 C9 4Ø               CMP  #64
9819 DØ Ø5               BNE  CKKEY    ;SOMETHING IS BEING PRESSED
981B A9 ØØ               LDA  #Ø       ;SET ZERO IN .A TO FLAG NO KEY
981D 85 42               STA  JOY
981F 6Ø                  RTS
982Ø A2 Ø7     CKKEY     LDX  #7       ;DIRECTION-KEY TABLE
9822 DD A5 9B  KLOOP     CMP  DIRKEY,X  ;IN TABLE
9825 DØ Ø6               BNE  KEND     ;GO TO LOOP END
9827 BD AD 9B            LDA  DIRECT,X  ;GET DIRECTION
982A 85 42               STA  JOY      ;STORE IT IN JOYSTICK
982C 6Ø                  RTS
982D CA        KEND      DEX
982E 1Ø F2               BPL  KLOOP    ;CHECK 8 KEYS
983Ø A9 ØØ               LDA  #Ø       ;ZERO JOY
9832 85 42               STA  JOY
9834 A9 FF               LDA  #$FF     ;CLEAR ZERO FLAG
9836 6Ø                  RTS           ;RETURN
               ;
               ;TOGGLE MESSAGE LINE
               ;
9837 AØ 27     OUT       LDY  #39      ;COPY 4Ø BYTES
9839 B9 CØ 8F  XFER1     LDA  CRT+96Ø,Y  ;FROM SCREEN
983C 99 4Ø BF            STA  SCLINE,Y  ;TO SCLINE TABLE
983F B9 CØ DB            LDA  COLCRT+96Ø,Y  ;FROM COLOR SCREEN
9842 99 7Ø BF            STA  COLINE,Y  ;TO COLINE TABLE
9845 A5 3D               LDA  PLCOL    ;SET COLORS ON SCREEN
9847 99 CØ DB            STA  COLCRT+96Ø,Y
984A A9 2Ø               LDA  #32      ;CLEAR OUT OTHER DATA
984C 99 CØ 8F            STA  CRT+96Ø,Y
984F 88                  DEY
985Ø 1Ø E7               BPL  XFER1
9852 A9 1B               LDA  #27      ;STANDARD CHARACTERS
9854 85 4B               STA  HIR1     ;VIC+21 SHADOW
9856 A9 35               LDA  #53      ;UPPER CASE ROM CHARACTERS
9858 85 4C               STA  HIR2     ;VIC+24 SHADOW
985A A9 Ø8               LDA  #8       ;NON-MULTICOLOR CHARACTERS
985C 85 4D               STA  HIR3     ;VIC+22 SHADOW
985E 6Ø                  RTS
               ;
985F AØ 27     IN        LDY  #39      ;COPY 4Ø BYTES
9861 B9 4Ø BF  XFER2     LDA  SCLINE,Y  ;FROM TABLE
9864 99 CØ 8F            STA  CRT+96Ø,Y  ;TO SCREEN
9867 B9 7Ø BF            LDA  COLINE,Y  ;FROM TABLE
986A 99 CØ DB            STA  COLCRT+96Ø,Y  ;TO COLOR SCREEN
986D 88                  DEY
986E 1Ø F1               BPL  XFER2
987Ø A9 3B               LDA  #59      ;HI-RES
9872 85 4B               STA  HIR1     ;VIC+21 SHADOW
9874 A9 38               LDA  #56      ;SECOND HI-RES SCREEN
```

```
9876 85 4C              STA  HIR2    ;VIC+24 SHADOW
9878 A5 4E              LDA  HIR4    ;OLD MULTICOLOR MODE
987A 85 4D              STA  HIR3    ;VIC+22 SHADOW
987C 60                 RTS
                  ;
                  ;ENABLE/DISABLE RASTER INTERRUPTS
                  ;
987D 78           RSTON SEI
987E A9 7F              LDA  #$7F
9880 8D 0D DC           STA  CIAICR  ;DISABLE CIA INTERRUPTS
9883 A9 01              LDA  #1
9885 8D 1A D0           STA  VIC+26  ;ENABLE RASTER INTERRUPTS
9888 A9 00              LDA  #0
988A 8D 12 D0           STA  VIC+18  ;SET IT AT LINE 0
988D AD 11 D0           LDA  VIC+17
9890 29 7F              AND  #127
9892 8D 11 D0           STA  VIC+17
9895 AD 14 03           LDA  INTPNT
9898 8D 22 99           STA  IEND+1  ;STORE THE OLD INTERRUPT
989B AD 15 03           LDA  INTPNT+1
989E 8D 23 99           STA  IEND+2
98A1 A9 D3              LDA  #<INT
98A3 8D 14 03           STA  INTPNT  ;VECTOR OUR INTERRUPT IN
98A6 A9 98              LDA  #>INT
98A8 8D 15 03           STA  INTPNT+1
98AB 58                 CLI          ;RE-ENABLE INTERRUPTS
98AC A9 01              LDA  #1
98AE 8D 15 D0           STA  VIC+21  ;TURN ON SPRITE 0
98B1 60                 RTS
                  ;
98B2 A9 00        RSTOFF LDA #0
98B4 8D 1A D0           STA  VIC+26  ;DISABLE RASTER INTERRUPTS
98B7 AD 0D DC           LDA  CIAICR
98BA 09 81              ORA  #129
98BC 8D 0D DC           STA  CIAICR  ;ENABLE CIA INTERRUPTS
98BF 78                 SEI
98C0 AD 22 99           LDA  IEND+1
98C3 8D 14 03           STA  INTPNT  ;REPOINT THE INTERRUPT
98C6 AD 23 99           LDA  IEND+2
98C9 8D 15 03           STA  INTPNT+1
98CC 58                 CLI          ;RE-ENABLE THE INTERRUPTS
98CD A9 00              LDA  #0
98CF 8D 15 D0           STA  VIC+21  ;TURN OFF THE SPRITES
98D2 60                 RTS
                  ;
                  ;RASTER INTERRUPT ROUTINE
                  ;
98D3 AD 19 D0     INT   LDA  VIC+25  ;WHAT INTERRUPTS
98D6 8D 19 D0           STA  VIC+25  ;ACKNOWLEDGE THEM
98D9 29 01              AND  #1
98DB F0 47              BEQ  SKIP
98DD A5 4B              LDA  HIR1
98DF 8D 11 D0           STA  53265   ;SET DEFAULT HIRES
98E2 A5 4C              LDA  HIR2
98E4 8D 18 D0           STA  53272   ;AND CHARACTER SET
98E7 A5 4D              LDA  HIR3
98E9 8D 16 D0           STA  53270   ;AND MULTICOLOR
98EC A2 F2              LDX  #242
98EE A0 01              LDY  #1      ;SET .X TO RASTER LINE, FLAG .Y
98F0 AD 12 D0           LDA  VIC+18  ;CHECK RASTER FOR WHICH
98F3 10 04              BPL  MID
98F5 A2 00              LDX  #0
98F7 A0 00              LDY  #0      ;FLAG RASTER AT 0, FLAG .Y
98F9 8E 12 D0     MID   STX  VIC+18  ;SET NEXT INTERRUPT
98FC AD 11 D0           LDA  VIC+17
98FF 29 7F              AND  #127
9901 8D 11 D0           STA  VIC+17  ;KEEP BIT 8 ZERO
9904 C0 00              CPY  #0
9906 D0 03              BNE  NORMAL  ;RESET SCREEN
9908 4C 1A 99           JMP  INTCK   ;DEFAULTS CORRECT
990B A9 3B        NORMAL LDA #59
990D 8D 11 D0           STA  53265   ;HIRES
```

```
9910 A9 38             LDA  #56
9912 8D 18 DØ          STA  53272    ;CHARACTERS
9915 A5 4E             LDA  HIR4
9917 8D 16 DØ          STA  5327Ø    ;MULTICOLOR
991A AD ØD DC INTCK    LDA  CIAICR
991D 29 Ø1             AND  #1
991F FØ Ø3             BEQ  SKIP     ;IF NO STANDARD INTERRUPT
9921 4C 31 EA IEND     JMP  INTRPT   ;JUMP TO NORMAL IRQ ROUTINE
9924 68       SKIP     PLA           ;ABORT THE INTERRUPT
9925 A8                TAY
9926 68                PLA
9927 AA                TAX
9928 68                PLA
9929 4Ø                RTI
              ;
              ;LOAD/SAVE HIRES SUBROUTINE
              ;
992A 2Ø 37 98 LS       JSR  OUT      ;BRING UP STATUS LINE
992D A2 FD             LDX  #<LSMESS
992F AØ 99             LDY  #>LSMESS  ;SET INPUT ROUTINE PARAMETERS
9931 2Ø 31 9A          JSR  INPUT    ;EXECUTE THE INPUT
9934 FØ 48             BEQ  ENDLS    ;IF ZERO LENGTH, ABORT
9936 A9 ØØ             LDA  #Ø
9938 85 Ø2             STA  LDSV     ;DEFAULT (Ø=LOAD,1=SAVE)
993A AD ØØ Ø2          LDA  INBUFF   ;START OF INPUT RETURN BUFFER
993D C9 4C             CMP  #"L"
993F FØ Ø6             BEQ  NXMES1   ;DEFAULT CORRECT
9941 C9 53    SCK      CMP  #"S"
9943 DØ 39             BNE  ENDLS    ;NOT S OR L, SO ABORT
9945 E6 Ø2             INC  LDSV     ;LDSV = 1 = SAVE
9947 A2 ØB    NXMES1   LDX  #<DVMESS
9949 AØ 9A             LDY  #>DVMESS  ;PARAMETERS
994B 2Ø 31 9A          JSR  INPUT
994E C9 Ø1             CMP  #1
995Ø DØ 2C             BNE  ENDLS    ;INPUT MUST BE 1 LONG
9952 AD ØØ Ø2          LDA  INBUFF
9955 38                SEC
9956 E9 3Ø             SBC  #"Ø"     ;SET ASCII "Ø" TO HEX ØØ
9958 A2 Ø3             LDX  #3       ;CHECK SECONDARY ADDRESSES
995A DD 25 9A DVLOOP   CMP  DEV,X
995D FØ Ø5             BEQ  DOLFS    ;FOUND DEVICE IN TABLE
995F CA                DEX
996Ø 1Ø F8             BPL  DVLOOP   ;NEXT DEVICE
9962 3Ø 1A             BMI  ENDLS    ;NOT IN TABLE
9964 BC 29 9A DOLFS    LDY  SA,X     ;.Y HOLDS SECONDARY ADDRESS
9967 85 3F             STA  PNTR     ;SAVE PHYSICAL DEVICE FOR REF.
9969 AA                TAX           ;IN .X FOR OPEN FILE
996A EØ Ø1             CPX  #1
996C DØ Ø2             BNE  DOSET    ;NOT A TAPE
996E A4 Ø2             LDY  LDSV     ;S.A. = Ø OR 1
997Ø A9 Ø1    DOSET    LDA  #1       ;SET LOGICAL DEVICE
9972 2Ø BA FF          JSR  SETLFS   ;SET LOGICAL,PHYSICAL,SECONDARY
9975 A2 1A             LDX  #<NMMESS
9977 AØ 9A             LDY  #>NMMESS  ;INPUT "NAME OF FILE"
9979 2Ø 31 9A          JSR  INPUT
997C DØ Ø6             BNE  HERE     ;CONTINUE
997E 2Ø 5F 98 ENDLS    JSR  IN
9981 4C 67 91          JMP  BEGIN    ;ABORT
9984 A5 3F    HERE     LDA  PNTR
9986 C9 Ø8             CMP  #8
9988 9Ø 12             BCC  NAMDO    ;NOT A DISK DRIVE
998A A5 Ø2             LDA  LDSV
998C FØ ØE             BEQ  NAMDO    ;NOT SAVING
998E AØ ØØ             LDY  #Ø       ;INITIALIZE COUNTER
999Ø B9 2D 9A SWADD    LDA  SW,Y
9993 9D ØØ Ø2          STA  INBUFF,X  ;.X SET FROM INPUT
9996 E8                INX
9997 C8                INY
9998 CØ Ø4             CPY  #4
999A DØ F4             BNE  SWADD    ;ADD ON ",S,W"
999C 8A       NAMDO    TXA           ;.A HOLDS NAME LENGTH
999D A2 ØØ             LDX  #<INBUFF  ;.X,.Y HOLDS INITIAL ADDRESS
```

```
999F A0 02               LDY  #>INBUFF
99A1 20 BD FF            JSR  SETNAM  ;SET FILE NAME
                ;
99A4 20 5F 98            JSR  IN      ;RESTORE FULL HIRES
99A7 20 B2 98            JSR  RSTOFF  ;DISABLE RASTER INTERRUPTS
99AA 20 95 9A            JSR  XFERIT  ;TRANSFER ALL DATA
99AD A9 01               LDA  #1      ;CLOSE FILE #1
99AF 20 C3 FF            JSR  CLOSE
99B2 20 7D 98            JSR  RSTON   ;RE-ENABLE RASTER INTERRUPTS
99B5 A5 3F               LDA  PNTR
99B7 C9 08               CMP  #8
99B9 90 3C               BCC  NODO    ;IF NOT A DISK
99BB 20 37 98            JSR  OUT     ;RESTORE STATUS LINE
99BE A9 0F               LDA  #15
99C0 A8                  TAY
99C1 A6 3F               LDX  PNTR
99C3 20 BA FF            JSR  SETLFS  ;SET 15,8,15
99C6 A9 00               LDA  #0
99C8 20 BD FF            JSR  SETNAM  ;NO NAME
99CB 20 C0 FF            JSR  OPEN
99CE A2 0F               LDX  #15
99D0 20 C6 FF            JSR  CHKIN   ;OPEN, AND SET FOR INPUT
99D3 A0 00               LDY  #0      ;POINTER TO SCREEN
99D5 20 CF FF ERLP       JSR  CHRIN   ;GET A BYTE
99D8 C9 0D               CMP  #13
99DA F0 0B               BEQ  WAITER  ;IF RETURN, FINISHED
99DC 29 3F               AND  #63     ;SET UP FOR POKING
99DE 99 C0 8F            STA  CRT+960,Y
99E1 C8                  INY          ;NEXT BYTE
99E2 20 B7 FF            JSR  READST  ;CHECK ERROR
99E5 F0 EE               BEQ  ERLP    ;NONE, SO BRANCH BACK
99E7 A9 0F    WAITER     LDA  #15
99E9 20 C3 FF            JSR  CLOSE   ;CLOSE 15
99EC A9 96               LDA  #150
99EE 85 A2               STA  TIME+2  ;SET A COUNT-UP TIME
99F0 A5 A2    WAITNG     LDA  TIME+2
99F2 D0 FC               BNE  WAITNG  ;WAIT 106 JIFFIES
99F4 20 5F 98            JSR  IN      ;FULL HIRES
99F7 20 E7 FF NODO       JSR  CLALL   ;CLOSE EVERYTHING
99FA 4C 67 91            JMP  BEGIN   ;RETURN TO LOOP
                ;
                ;LOAD/SAVE, DEVICE, NAME QUERIES
99FD 0C 0F 01 LSMESS    .BYT 12,15,1,4,32,15,18,32,19,1,22,5,63,0
9A0B 04 05 16 DVMESS    .BYT 4,5,22,9,3,5,32,14,21,13,2,5,18,63,0
9A1A 06 09 0C NMMESS    .BYT 6,9,12,5,32,14,1,13,5,58,0
9A25 01 02 08 DEV       .BYT 1,2,8,9  ;PHYSICAL DEVICES
9A29 01 00 02 SA        .BYT 1,0,2,2  ;SECONDARY ADDRESSES
9A2D 2C 53 2C SW        .ASC ",S,W"  ;ADD-ON FOR DISK SAVE
                ;
9A31 86 FD    INPUT      STX  MISC
9A33 84 FE               STY  MISC+1  ;SAVE .X AND .Y
9A35 A0 27               LDY  #39
9A37 A9 20               LDA  #32     ;CLEAR STATUS LINE
9A39 99 C0 8F CLRLN      STA  CRT+960,Y
9A3C 88                  DEY
9A3D 10 FA               BPL  CLRLN
9A3F C8                  INY          ;SAME AS LDY #0
9A40 B1 FD    PRINT      LDA  (MISC),Y
9A42 F0 06               BEQ  INPDO   ;IF END OF MESSAGE
9A44 99 C0 8F            STA  CRT+960,Y
9A47 C8                  INY
9A48 D0 F6               BNE  PRINT   ;BACK TO LOOP
9A4A C8       INPDO      INY          ;SPACE A CHARACTER
9A4B A2 00               LDX  #0      ;POINTER TO BUFFER
9A4D A9 A0    GET        LDA  #160
9A4F 99 C0 8F            STA  CRT+960,Y  ;CURSOR
9A52 84 FB               STY  AD
9A54 86 FC               STX  AD+1    ;ENSURE .Y & .X ARE PRESERVED
9A56 20 E4 FF GETWT      JSR  GETIN
9A59 F0 FB               BEQ  GETWT   ;WAIT ON INPUT
9A5B A4 FB               LDY  AD
9A5D A6 FC               LDX  AD+1    ;BRING BACK .X,.Y
```

```
9A5F C9 ØD                CMP  #13
9A61 DØ Ø7                BNE  DELCK     ;NOT A RETURN
9A63 A9 2Ø                LDA  #32
9A65 99 CØ 8F             STA  CRT+96Ø,Y  ;KILL CURSOR
9A68 8A                   TXA            ;SET ZERO FLAG AND LEAVE IN .A
9A69 6Ø                   RTS            ;END OF INPUT
9A6A C9 14     DELCK      CMP  #2Ø
9A6C DØ ØE                BNE  RNGCK     ;NOT A DELETE
9A6E EØ ØØ                CPX  #Ø
9A7Ø FØ DB                BEQ  GET       ;NOTHING IN BUFFER
9A72 A9 2Ø                LDA  #32
9A74 99 CØ 8F             STA  CRT+96Ø,Y  ;CLEAR CURSOR
9A77 88                   DEY            ;BACKSPACE ON SCREEN
9A78 CA                   DEX            ;BACK UP IN BUFFER
9A79 4C 4D 9A             JMP  GET       ;BACK TO INPUT
9A7C C9 2Ø     RNGCK      CMP  #32
9A7E 9Ø CD                BCC  GET       ;CONTROL CODES NOT ACCEPTED
9A8Ø C9 6Ø                CMP  #96
9A82 BØ C9                BCS  GET       ;NOR GRAPHICS CODES
9A84 CØ 27                CPY  #39
9A86 FØ C5                BEQ  GET       ;AT END OF LINE
9A88 9D ØØ Ø2             STA  INBUFF,X  ;IN LINE BUFFER
9A8B 29 3F                AND  #63       ;MODIFY FOR POKING
9A8D 99 CØ 8F             STA  CRT+96Ø,Y  ;PUT ON SCREEN
9A9Ø C8                   INY            ;MOVE TO NEXT SPACE
9A91 E8                   INX
9A92 4C 4D 9A             JMP  GET
               ;
9A95 2Ø CØ FF  XFERIT     JSR  OPEN      ;OPEN THE FILE
9A98 BØ 1Ø                BCS  XFSTOP    ;CARRY SET = ERROR
9A9A 2Ø B7 FF             JSR  READST
9A9D DØ ØB                BNE  XFSTOP    ;ZERO CLEAR = ERROR
9A9F A2 Ø1                LDX  #1        ;LOGICAL FILE
9AA1 A5 Ø2                LDA  LDSV
9AA3 DØ Ø6                BNE  OUTPT     ;SAVING
9AA5 2Ø C6 FF             JSR  CHKIN     ;SET UP AS INPUT
9AA8 9Ø Ø6                BCC  START     ;IF NO ERROR
9AAA 6Ø        XFSTOP     RTS            ;ERROR - RETURN
9AAB 2Ø C9 FF  OUTPT      JSR  CHKOUT    ;SET UP AS OUTPUT
9AAE BØ FA                BCS  XFSTOP    ;IF AN ERROR
9ABØ 2Ø B7 FF  START      JSR  READST
9AB3 DØ F5                BNE  XFSTOP    ;IF AN ERROR
9AB5 A9 DØ                LDA  #>VIC
9AB7 85 FC                STA  AD+1      ;SET I/O TO BORDER COLOR
9AB9 A9 2Ø                LDA  #32
9ABB 85 FB                STA  AD
9ABD 2Ø 5D 9B             JSR  IO        ;SEND OR RECEIVE FROM/TO 5328Ø
9ACØ E6 FB                INC  AD        ;NOW AT 53281
9AC2 2Ø 5D 9B             JSR  IO        ;SEND/RECEIVE
               ;
9AC5 A9 ØØ                LDA  #<CRT
9AC7 85 FB                STA  AD        ;SET UP I/O AT START OF SCREEN
9AC9 A9 8C                LDA  #>CRT
9ACB 85 FC                STA  AD+1      ;HIGH BYTE
9ACD 2Ø 5D 9B  TLP1       JSR  IO
9ADØ E6 FB                INC  AD
9AD2 DØ F9                BNE  TLP1      ;WAIT TILL END OF PAGE
9AD4 E6 FC                INC  AD+1      ;NEXT PAGE
9AD6 A5 FC                LDA  AD+1
9AD8 C9 8F                CMP  #>3*256+CRT
9ADA DØ F1                BNE  TLP1      ;NOT FINISHED
9ADC 2Ø 5D 9B  TLP2       JSR  IO        ;TRANSMIT/RECEIVE
9ADF E6 FB                INC  AD        ;NEXT BYTE
9AE1 A5 FB                LDA  AD
9AE3 C9 E8                CMP  #232
9AE5 DØ F5                BNE  TLP2      ;I/O 232 BYTES
               ;
9AE7 A9 ØØ                LDA  #<COLCRT
9AE9 85 FB                STA  AD        ;SET FOR COLOR SCREEN
9AEB A9 D8                LDA  #>COLCRT
9AED 85 FC                STA  AD+1
9AEF 2Ø 5D 9B  CLP1       JSR  IO
```

```
9AF2 E6 FB              INC   AD
9AF4 DØ F9              BNE   CLP1
9AF6 E6 FC              INC   AD+1
9AF8 A5 FC              LDA   AD+1
9AFA C9 DB              CMP   #>3*256+COLCRT
9AFC DØ F1              BNE   CLP1     ;NOT FINISHED
9AFE 2Ø 5D 9B CLP2      JSR   IO
9BØ1 E6 FB              INC   AD
9BØ3 A5 FB              LDA   AD
9BØ5 C9 E8              CMP   #232
9BØ7 DØ F5              BNE   CLP2     ;I/O LAST 232
               ;
9BØ9 A9 ØØ              LDA   #<HIPAGE
9BØB 85 FB              STA   AD       ;HIRES PAGE
9BØD A9 AØ              LDA   #>HIPAGE
9BØF 85 FC              STA   AD+1
9B11 2Ø 5D 9B HLP1      JSR   IO
9B14 AØ ØØ              LDY   #Ø
9B16 B1 FB              LDA   (AD),Y
9B18 DØ 3D              BNE   HLPDO    ;IF NON-ZERO
               ;
               ;EXECUTE NON-STANDARD ZERO-FLAGGED DATA
9B1A A5 Ø2    ZERDO     LDA   LDSV
9B1C DØ 22              BNE   OUTZR    ;IF SAVING
9B1E 2Ø CF FF           JSR   CHRIN
9B21 85 FD              STA   MISC     ;READ ADDRESS OF FIRST
9B23 2Ø CF FF           JSR   CHRIN
9B26 85 FE              STA   MISC+1   ;NON-ZERO BYTE FOLLOWING
9B28 A9 ØØ    NXTZ      LDA   #Ø
9B2A A8                 TAY
9B2B 91 FB              STA   (AD),Y   ;ZERO THE ADDRESS
9B2D 2Ø 92 9B           JSR   NEXTY    ;NEXT ADDRESS
9B3Ø BØ 2A              BCS   CLS      ;CARRY SET = END OF I/O
9B32 A5 FB              LDA   AD
9B34 C5 FD              CMP   MISC
9B36 DØ FØ              BNE   NXTZ     ;NOT TO END OF ZEROS
9B38 A5 FC              LDA   AD+1
9B3A C5 FE              CMP   MISC+1
9B3C DØ EA              BNE   NXTZ     ;NOT TO END OF ZEROS
9B3E FØ D1              BEQ   HLP1     ;RETURN TO STANDARD I/O
9B4Ø 2Ø 92 9B OUTZR     JSR   NEXTY    ;NEXT ADDRESS
9B43 9Ø Ø6              BCC   ZECHEK   ;NOT AT END OF I/O
9B45 2Ø 87 9B           JSR   SENDZE   ;SEND FINAL ADDRESS
9B48 4C 5C 9B           JMP   CLS      ;END OF SUBROUTINE
9B4B AØ ØØ    ZECHEK    LDY   #Ø
9B4D B1 FB              LDA   (AD),Y
9B4F FØ EF              BEQ   OUTZR    ;STILL A ZERO BYTE
9B51 2Ø 87 9B           JSR   SENDZE   ;SEND ADDRESS
9B54 2Ø 5D 9B           JSR   IO       ;SEND THE NON-ZERO BYTE
               ;RESUME STANDARD I/O
9B57 2Ø 92 9B HLPDO     JSR   NEXTY
9B5A 9Ø B5              BCC   HLP1     ;NOT AT END YET
9B5C 6Ø       CLS       RTS            ;FINISHED
               ;
9B5D A5 Ø2    IO        LDA   LDSV
9B5F DØ 15              BNE   OUTPUT   ;IF SAVING
9B61 2Ø CF FF           JSR   CHRIN    ;GET THE BYTE
9B64 48                 PHA            ;PUSH ON STACK
9B65 BØ 1C              BCS   STP2     ;ABORT IF AN ERROR
9B67 2Ø B7 FF           JSR   READST
9B6A FØ Ø4              BEQ   ENCKST   ;NO ERROR
9B6C C9 4Ø              CMP   #64
9B6E DØ 13              BNE   STP2     ;A NON-EOT CONDITION
9B7Ø 68       ENCKST    PLA            ;RECALL THE CHRIN
9B71 AØ ØØ              LDY   #Ø       ;ZERO .Y FOR INDIRECT
9B73 91 FB              STA   (AD),Y   ;STORE ON SCREEN
9B75 6Ø                 RTS            ;END OF I/O SUBROUTINE
9B76 AØ ØØ    OUTPUT    LDY   #Ø
9B78 B1 FB              LDA   (AD),Y   ;GET A BYTE FROM SCREEN
9B7A 2Ø D2 FF           JSR   CHROUT   ;SEND IT OUT
9B7D 2Ø B7 FF           JSR   READST
9B8Ø DØ Ø2              BNE   STP      ;IF ERROR
```

```
9B82 60                 RTS
9B83 68        STP2     PLA             ;PULL EXTRA PUSH
9B84 68        STP      PLA
9B85 68                 PLA             ;REMOVE A LEVEL OF SUBROUTINE
9B86 60                 RTS             ;EXIT DIRECTLY TO MAIN ROUTINE
9B87 A5 FB     SENDZE   LDA  AD
9B89 20 D2 FF           JSR  CHROUT     ;SEND LOW BYTE
9B8C A5 FC              LDA  AD+1
9B8E 20 D2 FF           JSR  CHROUT     ;SEND HIGH BYTE
9B91 60                 RTS
9B92 E6 FB     NEXTY    INC  AD
9B94 D0 04              BNE  ENDY       ;NO PAGE
9B96 E6 FC              INC  AD+1       ;NEXT PAGE
9B98 18                 CLC             ;CLEAR CARRY = NO END OF PAGE
9B99 60                 RTS             ;BACK TO XFERIT SUBROUTINE
9B9A A5 FC     ENDY     LDA  AD+1
9B9C C9 BF              CMP  #>31*256+HIPAGE  ;END OF HIRES PAGE
9B9E 90 04              BCC  RETY       ;NOT AT END
9BA0 A5 FB              LDA  AD
9BA2 C9 41              CMP  #65        ;CARRY SET IF PAST 64
9BA4 60        RETY     RTS
               ;
               ;DATA FOR CHARACTER CHECKING
               ;
9BA5 12 17 0A DIRKEY    .BYT 18,23,10,9,20,12,62,14
9BAD 08 02 04 DIRECT    .BYT %1000,%0010,%0100,%0001
9BB1 0A 06 05           .BYT %1010,%0110,%0101,%1001
               ;
9BB5 90 05 1C COLORS    .BYT 144,5,28,159,156,30,31,158
9BBD 81 95 96           .BYT 129,149,150,151,152,153,154,155
               ;
9BC5 88 87 86 FNCTNS    .BYT 136,135,134,133
               ;
9BC9 48 49 52 QUESTN    .ASC "HIRES SKETCHPAD - BY CHRIS METCALF"
9BEB 0D                 .BYT 13
9BEC 4D 55 4C           .ASC "MULTICOLOR MODE? N"
9BFE 9D 00              .BYT 157,0
               ;
9C00 10 0C 0F STLINE    .BYT 16,12,15,20,58,6,32,58,32,32,32,32
9C0C 20 0C 09           .BYT 32,12,9,14,5,32,4,18,1,23,58,32,32,32,32,3
9C1C 20 06 09           .BYT 32,6,9,12,12,20,15,58,32,32,32,32
9C28 37 35 33 FNUM      .ASC "7531"
9C2C 0F 06 06 DRMD      .BYT 15,6,6,32,15,14,32,32
9C34 0F 06 06 LNMD      .BYT 15,6,6,32,6,18,15,13,20,15,32,32,12,9,14,5
9C44 13 01 0D FLMD      .BYT 19,1,13,5,1,14,25,32
               ;
               ;THESE ARE THE SPRITE MATRICES
               ;
               ;SPRITE FOR HIRES MODE
9C4C 00 38 00 SPRITE    .BYT %00000000,%00111000,%00000000
9C4F 00 44 00           .BYT %00000000,%01000100,%00000000
9C52 00 44 00           .BYT %00000000,%01000100,%00000000
9C55 06 FE C0           .BYT %00000110,%11111110,%11000000
9C58 09 01 20           .BYT %00001001,%00000001,%00100000
9C5B 06 00 C0           .BYT %00000110,%00000000,%11000000
9C5E 04 00 40           .BYT %00000100,%00000000,%01000000
9C61 04 00 40           .BYT %00000100,%00000000,%01000000
9C64 04 00 40           .BYT %00000100,%00000000,%01000000
9C67 06 00 C0           .BYT %00000110,%00000000,%11000000
9C6A 09 01 20           .BYT %00001001,%00000001,%00100000
9C6D 06 FE C0           .BYT %00000110,%11111110,%11000000
9C70 00 38 00           .BYT %00000000,%00111000,%00000000
9C73 00 0C              .BYT %00000000,%00001100
               ;
               ;THE SPRITE FOR MULTICOLOR MODE
9C75 00 18 00           .BYT %00000000,%00011000,%00000000
9C78 00 24 00           .BYT %00000000,%00100100,%00000000
9C7B 00 24 00           .BYT %00000000,%00100100,%00000000
9C7E 03 7E C0           .BYT %00000011,%01111110,%11000000
9C81 04 81 20           .BYT %00000100,%10000001,%00100000
9C84 03 00 C0           .BYT %00000011,%00000000,%11000000
9C87 02 00 40           .BYT %00000010,%00000000,%01000000
```

```
9C8A Ø2  ØØ  4Ø          .BYT %ØØØØØØ1Ø,%ØØØØØØØØ,%Ø1ØØØØØØ
9C8D Ø2  ØØ  4Ø          .BYT %ØØØØØØ1Ø,%ØØØØØØØØ,%Ø1ØØØØØØ
9C9Ø Ø3  ØØ  CØ          .BYT %ØØØØØØ11,%ØØØØØØØØ,%11ØØØØØØ
9C93 Ø4  81  2Ø          .BYT %ØØØØØ1ØØ,%1ØØØØØØ1,%ØØ1ØØØØØ
9C96 Ø3  7E  CØ          .BYT %ØØØØØØ11,%Ø111111Ø,%11ØØØØØØ
9C99 ØØ  18  ØØ          .BYT %ØØØØØØØØ,%ØØØ11ØØØ,%ØØØØØØØØ
9C9C ØØ  ØC  ØØ          .BYT %ØØØØØØØØ,%ØØØØ11ØØ,Ø
```

Chapter Three

Redefining Character Sets

Make Your Own Characters

Orson Scott Card

Using custom characters to draw pictures and animate them is clearly explained in several demonstration programs.

The easiest type of graphics to use on the Commodore 64 is character graphics. After all, you're using character graphics every time you type. There's a built-in text editor that decides what letter should appear in a certain position on a screen when you press a certain key, but the graphics part of the operation, the actual placement of a character on the screen, is really quite simple.

This is how it works.

The VIC-II, the video chip in your computer, tells the television what to display. As long as the VIC-II is working, the screen is never truly blank. Sixty times a second, the VIC-II tells the television what to put in every single position on the screen. Displaying a blank screen is no faster or easier than displaying a screen with a lot of text — to the VIC-II chip, it's all the same. A blank area on the screen is filled up with characters, just like screen areas with lots of writing. The difference is that the blank area is filled with the blank character — the character you get when you hit the space bar.

So the screen is always full of characters — all you do is change which character is in a particular place.

Talking to the TV

You don't do that, however, by sending messages to the television. The TV has no memory — you can tell it to put a dot on the screen in a certain place, and it might do it, but exactly 1/60 second later, the TV would erase that dot unless you said to display it again. You have to send the TV a new instruction for every dot on the screen, 60 times each second. If you had to program all that yourself, even in machine language, there'd be little time left for anything else. And in BASIC, you couldn't do anything at all.

Fortunately, the VIC-II chip knows how to speak the TV's language, just as fast as the TV can go. All you have to do is tell the VIC-II what you want, and it will automatically tell the TV to do it. Most important, it will *keep* telling the TV the same thing until you change it. You have to give the instruction to the VIC-II only one time; it will give the instruction to the TV from then on, until you change the instruction or turn off the computer.

Screen Memory

The VIC-II understands many different instructions. But you don't give those instructions to the VIC-II directly. You never have to create a VIC-II program and then type RUN. The VIC-II is already running, from the minute you turn on the computer. (There are ways to stop the VIC-II while the computer is running, but that's another story.)

What is the VIC-II doing? It scans through memory, again and again, looking for specific instructions and telling the TV what to do. But it doesn't scan through *all* of memory. It looks in certain locations to find certain things. After all, the only thing that any memory location can hold is a number from 0 to 255. The same number can mean different things, depending on where the VIC-II finds it.

To carry out character graphics, as opposed to high-resolution graphics and sprite graphics, you only have to know some of the areas that the VIC-II scans.

Screen memory. Screen memory is a thousand bytes long. Each byte represents a small area on the screen. Screen memory, like all computer memory, is just one long row of memory locations. But the VIC-II reads it like the page of a book.

The VIC-II scans screen memory from 0 to 999. Byte 0 is the upper-left-hand corner of the screen, just as you began reading this page starting in the upper-left-hand corner.

The byte that the VIC-II finds in that position is a code number for a character. This is not the ASCII character code, however. This is the *screen code.* Appendix G is a complete table of all the screen codes that the VIC-II recognizes. The VIC-II will tell the TV to display whatever character is called for in the upper-left-hand corner of the screen.

After recognizing the first screen code, the VIC-II reads the next memory location, byte 1. The character called for will be displayed just to the right of character 0. Byte 2 controls the next character to the right, and so on. After byte 39, the VIC-II drops down to the leftmost position on the second row on the screen, just as your eyes do after reading the last word on a line. Bytes 0 to 39 are the first row on the screen; bytes 40 to 79 are the second row; bytes 80 to 119 are the third row; and so on, until bytes 960 to 999 make up the last row.

Then the VIC-II is finished with screen memory. But less than 1/60 second later, it will come back and scan screen memory again. If you have changed the code anywhere in screen memory since the last time the VIC-II read that location, it will read the new value and put a different character on the screen in that location. The VIC-II doesn't care what code it finds in any location, nor does it have to do anything special if you have changed the character — the VIC-II just takes what it finds and passes it on to the TV.

Screen memory, then, becomes a map of the screen. By PEEKing into screen memory, your program can find out what characters are being displayed on the television, and by POKEing into screen memory, your program can change what the TV shows.

Character memory. The screen code itself is not enough to put a complete character on the screen. The screen code is merely an *index* into the character set.

Like screen memory, character memory is just a section of memory. Each character pattern is stored as a series of eight bytes. Since the Commodore 64 can access 512 different characters (two sets of 128 characters and 128 inverse characters each), character memory consists of eight times that many bytes — 4096 bytes, to be exact. That's 4K, much larger than screen memory.

The eight-byte character patterns are stored in the same order as the screen codes. The first screen code is 0, which stands for the character @. The second screen code, 1, is A; the next eight bytes in character memory hold pattern for A.

That means that to find the pattern for any character in character memory, all you have to do is take the character's screen code and multiply it by 8, then count in that many bytes from the start of character memory. The screen code for Z is 26. Eight times 26 equals 208. So the first byte of the pattern for the letter Z is byte 208 in character memory.

Whenever the VIC-II reads a screen code number, it counts the right number of bytes from the start of character memory to find that character's pattern. Then it tells the television to display that character pattern.

Chapter Three

How do you create your own special characters? By changing the pattern stored in character memory. The VIC-II can't read the alphabet. It doesn't care whether the character pattern for screen code 1 looks like an A or not. It will print whatever pattern is in that eight-byte section of character memory, no questions asked. It could be a letter in the Cyrillic alphabet or a picture of a duck — once the pattern in the character set has been changed, that is what will be displayed every time the VIC-II finds that particular screen code.

Color memory. Besides screen memory, which is a map of the characters on the screen, there is a second map to keep track of the colors on the screen. You can select the color for each character in screen memory individually, by changing the corresponding location in color memory.

The character map of screen codes and the color map of color codes have an exact one-to-one relationship. That is, whatever character is called for in byte 299 of screen memory, it will be displayed with whatever color is called for in byte 299 of color memory.

Moving Character Memory

When your computer powers up, screen memory starts at location 1024, color memory starts at 55296, and character memory starts at 53248. But that isn't necessarily permanent. You can tell the VIC-II to look for screen memory, color memory, and character memory somewhere else. For our purposes right now we don't have to move screen memory or color memory. We *will*, however, move character memory.

Why do we have to move it? Why can't we just POKE new character patterns into character memory where it is?

The reason is simple enough. The character set is in ROM, Read Only Memory, which cannot be changed. As long as the VIC-II is looking for character patterns in ROM, you can't change the character patterns. That's why the character set isn't erased every time you turn off your computer.

So before the VIC-II can start to use your new character set, you have to tell it to look for character memory somewhere else. You do this by changing the number stored at location 53272.

Where to put character memory. It is tempting to get deeply into the complexities of graphics memory organization, but really unnecessary for our purpose. It's enough to say that the VIC-II chip can look in only one 16K block of RAM at a time, so that screen memory and the character set always have to be in the same block, unless you're using the built-in character set.

Since screen memory and character memory have to be in the same block, the VIC-II only looks for instructions telling *which* 2K block of memory within that 16K block contains the character set. Therefore, there are only eight possible location instructions for character memory. The code numbers for the blocks are the even numbers from 0 to 14. These code numbers, stored in location 53272, tell the VIC-II to look for character memory in one of the eight possible locations within the block:

instruction	**starting address within block**
0	0
2	2048
4	4096
6	6144
8	8192
10	10240
12	12288
14	14336

(For a more complete discussion of graphics memory blocks, see "Graphics Memory" in Chapter 2. For now, let's just assume that we are using the default graphics block, the one that starts at location 0 and goes to location 16383.)

Character memory instructions. Why does it take only 2K to hold character memory, when the ROM character set is 4096 bytes long? That's because the ROM character set is really *two* character sets. The VIC-II can read either of them, but only one of them at a time. Each is only 2K long. You can create as many as seven character sets at one time, and switch from one to another just by changing location 53272. It's the same thing that happens when you press SHIFT and the Commodore key at the same time — you're just switching character sets.

Changing the code at 53272 isn't as simple, however, as POKEing the code number. That's because the character memory location is stored in bits 1-3. Bits 4-7 are used to hold the screen memory location. If you POKEd the character memory instruction 12 into location 53272, the binary number stored there would be:

$$\begin{array}{lllllllll} & \textbf{bit:} & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \textbf{12} & & 0 & 0 & 0 & 0 & \underline{1} & \underline{1} & \underline{0} & 0 \\ & & & & & & \multicolumn{2}{c}{\textbf{code}} & & \end{array}$$

Notice that bits 4-7, which contain the screen memory instruction, contain only zeros. Therefore, the VIC-II will look for screen memory at location 0 within the block. Since that section of memory is used for vital machine language functions, your TV screen will look quite odd.

Fortunately, Commodore 64 BASIC includes two operations that allow you to change individual bits in a byte without affecting the rest of the byte: bitwise AND and bitwise OR. (If you don't already know how to use these operations, you might want to see "Bitmapped Graphics" in Chapter 2.) Here's how to change the character set location to code 12 without changing the screen memory instruction:

```
POKE 53272, (PEEK (53272) AND 240) OR 12
```

To specify a different character memory location, change the 12 to a different even number from 0 to 14.

Mixed character sets. Changing the location where the VIC looks for the character set doesn't actually put the character patterns there, however. The only way to get character patterns into character memory, once you've changed from using the ROM character set, is to put the patterns there yourself.

Often you will want to mix character sets. That is, you'll want to use the alphabet and some of the symbols from the ROM set and some of your own custom characters at the same time. The easiest way to do this is to copy the ROM character set — or part of it — into the new character memory area. Once it's in place, just change the patterns for a few of the characters, the ones you want to customize. The rest will stay the same as the ROM set.

To copy the ROM set, you have to do a couple of POKEs first. You don't have to understand all the engineering behind it. The character ROM is in a bank of memory that is normally switched out, where you can't PEEK it. You have to switch it in. And before you switch it in, you have to disable all interrupts. Then, when you're through copying the ROM character set, you have to switch it out and reenable interrupts. Here are the instructions that do the job:

Disable interrupts: POKE 56334, PEEK (56334) AND 254
Switch in character ROM: POKE 1, PEEK (1) AND 251
(now you copy the character set)
Switch out character ROM: POKE 1, PEEK (1) OR 4
Enable interrupts: POKE 56334, PEEK (56334) OR 1

Chapter Three

Now we can do our first simple character set operation. This program will copy the ROM character set and tell the VIC-II to find character memory at location code 12.

```
1Ø CM=12288:CX=53248
15 GOSUB 8ØØ
2Ø POKE 53272,(PEEK(53272)AND 24Ø)OR 12
199 END
8ØØ POKE 56334,PEEK(56334) AND 254
8Ø5 POKE 1,PEEK(1) AND 251
81Ø FOR I=Ø TO 1Ø23:POKE CM+I,PEEK(CX+I):NEXT
815 POKE 1,PEEK(1) OR 4
82Ø POKE 56334,PEEK(56334) OR 1
825 RETURN
```

The trouble with this program is that it doesn't do anything you can see. It just copies ROM, so that as far as you can tell the computer is just as it always is. It's time to start changing the character patterns.

Character Patterns

Each character pattern consists of eight bytes. Each byte consists of eight bits. That means that each character can be mapped like this:

bits	7	6	5	4	3	2	1	0
bytes								
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

The television screen is divided into 25 rows of 40 characters each. Every character consists of a small rectangle eight dots wide by eight dots high. This rectangle has a one-to-one relationship with the bits in the eight bytes of the character pattern. Every bit must be either a 1 or a 0. If the bit is a 0, then that dot on the screen is *off* — the background color is displayed. If the bit is a 1, then that dot is *on*, and the foreground color is displayed.

Figure 1 shows a pattern that will produce the letter A. Each *on* bit, or 1, will be bright on the screen; each *off* bit, or 0, will be dark. With the zeros removed, you can easily see the pattern that would actually be displayed.

To the right of each byte in the pattern is a decimal number. This is the number that you would have to POKE into that position in the character pattern in order to get that byte, with its pattern of *on* and *off* bits.

To show how you can change these patterns, here's a program that adds a solid line right through each of the first eight characters in the character set: @, A, B, C, D, E, F, and G. It is identical to the first program, except for line 25. To see how the new letters look, just type them as soon as the program is through running. The READY message will already show you what happened to the A and D.

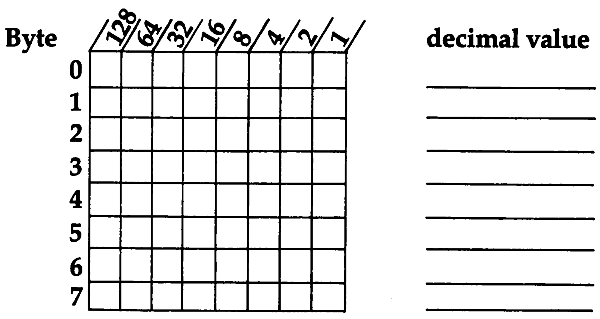
```
10 CM=12288:CX=53248
15 GOSUB 8ØØ
2Ø POKE 53272,(PEEK(53272)AND 24Ø)OR 12
25 FOR I=Ø TO 63 STEP 9:POKE CM+I,255:NEXT
199 END
8ØØ POKE 56334,PEEK(56334) AND 254
8Ø5 POKE 1,PEEK(1) AND 251
81Ø FOR I=Ø TO 1Ø23:POKE CM+I,PEEK(CX+I):NEXT
815 POKE 1,PEEK(1) OR 4
82Ø POKE 56334,PEEK(56334) OR 1
825 RETURN
```

What does line 25 actually do? First, remember that the number 255 is the highest possible eight-bit number. Every bit in the number is a 1. Therefore, if a byte in a character pattern is the number 255, it will be displayed as a thin horizontal line.

Figure 1. Character Map of the Letter A

				bits											
bytes	7	6	5	4	3	2	1	0							decimal
0	0	0	0	0	0	0	0	0							0
1	0	0	0	1	1	0	0	0			1	1			24
2	0	0	1	1	1	1	0	0		1	1	1	1		60
3	0	1	1	0	0	1	1	0	1	1			1	1	102
4	0	1	1	1	1	1	1	0	1	1	1	1	1	1	126
5	0	1	1	0	0	1	1	0	1	1			1	1	102
6	0	1	1	0	0	1	1	0	1	1			1	1	102
7	0	0	0	0	0	0	0	0							0

Figure 2. Character Design Matrix



Line 25, then, POKEs 255 once into each of the first eight character patterns. Since the FOR-NEXT loop STEPs 9 instead of 8, the line will be one position lower in each pattern than in the character before.

Combining shapes. It's possible to combine character shapes, too. In this program, lines 25 and 30 pick up the patterns of characters 73 and 74 (little circle segments) and make them into a single pattern in place of the @ character. Character patterns are combined by ORing byte 0 in one pattern with byte 0 in the other, then doing the same with bytes 1 through 7. To see different character combinations, just change the values of X and Y in line 25. Most combinations, however, won't be too clear.

```
10 CM=12288:CX=53248
15 GOSUB 800
20 POKE 53272,(PEEK(53272)AND 240)OR 12
25 X=73:Y=74
30 FOR I=0 TO 7:POKE CM+I,PEEK(CM+I+8*X)OR PEEK(CM
   +I+8*Y):NEXT
199 END
800 POKE 56334,PEEK(56334) AND 254
805 POKE 1,PEEK(1) AND 251
810 FOR I=0 TO 1023:POKE CM+I,PEEK(CX+I):NEXT
815 POKE 1,PEEK(1) OR 4
820 POKE 56334,PEEK(56334) OR 1
825 RETURN
```

Animation. By making several characters that are only slightly different from each other, and then POKEing the characters successively into the same location in screen memory, it is possible to give the illusion of motion. In this program, lines 20-30 create a series of eight characters. All of them have a diagonal line and a vertical line, but in each character the vertical line is in a slightly different position. This program only creates the characters — we'll add the motion in a moment:

```
10 CM=12288:CX=53248
15 GOSUB 800
20 POKE 53272,(PEEK(53272)AND 240)OR 12
25 FOR I=0 TO 7:FOR J=0 TO 7:W=I*8+J:POKE CM+W,(2↑
   (I+2))/4 OR (2↑(J+2))/4
30 NEXT:NEXT
199 END
800 POKE 56334,PEEK(56334) AND 254
805 POKE 1,PEEK(1) AND 251
810 FOR I=0 TO 1023:POKE CM+I,PEEK(CX+I):NEXT
815 POKE 1,PEEK(1) OR 4
820 POKE 56334,PEEK(56334) OR 1
825 RETURN
```

By typing the characters @ through G from the keyboard, you can see what the newly formed characters look like.

Now it's time to add the animation routine. In this program, line 10 is changed—SM is set to the starting address of screen memory, and CL is set to the starting address of color memory. The main loop of the program, lines 100-130, POKEs the color white into location 500 in color memory, and then POKEs screen codes 0 through 7 successively into location 500 in screen memory. When that loop is complete, line 120 does the same thing, only in reverse order.

Line 200 is a delay subroutine. Without it, the movement would be too fast to see. When the program runs, it will seem as though the vertical line is moving back and forth across the diagonal line.

```
10 CM=12288:CX=53248:SM=1024:CL=55296
15 GOSUB 800
20 POKE 53272,(PEEK(53272)AND 240)OR 12
25 FOR I=0 TO 7:FOR J=0 TO 7:W=I*8+J:POKE CM+W,(2↑
   (I+2))/4 OR (2↑(J+2))/4
30 NEXT:NEXT
100 POKE CL+500,1
110 FOR I=0 TO 7:POKE SM+500,I:GOSUB 200:NEXT
120 FOR I=7 TO 0 STEP -1:POKE SM+500,I:GOSUB 200:N
    EXT
130 GOTO 110
199 END
200 FOR J=0 TO 50:NEXT:RETURN
800 POKE 56334,PEEK(56334) AND 254
805 POKE 1,PEEK(1) AND 251
```

```
810 FOR I=0 TO 1023:POKE CM+I,PEEK(CX+I):NEXT
815 POKE 1,PEEK(1) OR 4
820 POKE 56334,PEEK(56334) OR 1
825 RETURN
```

Drawing with DATA statements. So far, all the new characters have been drawn using mathematical expressions. Actually, that method is almost never used in programming. Instead, you will plot out each character pattern separately, and enter them using DATA statements. The clearest method is to put all eight DATA statements for a particular character pattern on one line:

DATA 0,9,22,128,255,128,66,0

Notice that where you want a blank row, you must insert a zero. Every character pattern must have eight bytes — blank ones must be represented by 0.

How can you plot out your characters and figure the decimal values? The easiest way is to use a character editor, which allows you to see the exact character pattern you're creating. "Ultrafont", the program in the next article, is an excellent character editor, which allows you to create your own character set without ever having to calculate the bit patterns and decimal values of any character.

However, to create just a few characters you can use this simple method. Draw an 8-by-8 grid (or mark an 8-square-by-8-square section on regular graph paper). Fill in any squares that you want to have lit up; leave blank any squares that should be the background color. When you have the pattern you want, each horizontal row represents a single byte in the pattern, arranged in order from top to bottom. Each filled-in square represents an *on* bit, or a 1. Each *on* bit will have a different value, depending on its position in the row. A 1 in the leftmost position has a value of 128; a 1 in the rightmost position has a value of 1. A zero or blank in any position has a value of 0.

This chart shows the values. To calculate the decimal value of the binary number 01110011 (shown by the pattern of Xs), add up the values of the *on* bits. In this case, the values, from left to right, are 64, 32, 16, 2, and 1. Therefore, the decimal number that will produce this bit pattern is 64+32+16+2+1, or 115. This is the number you would POKE into character memory to produce this bit pattern.

```
bits        7   6   5   4   3  2  1   0
                X   X   X         X   X
decimal    128     32      8     2
value           64     16     4     1
of on
bit
```

If this is the top row of the character pattern, put this number first in the DATA statement; if it is the bottom row, you'll want to put it last. Then, READ the DATA statements in a loop and POKE the values into character memory. If you have arranged your DATA statements in the right order, the character patterns will all be correct at the end of the loop.

This program makes a very simple character to replace the @ character. It will be four horizontal lines. The horizontal lines are the value 255; the blanks between them are, of course, 0.

```
10 CM=12288:CX=53248
15 GOSUB 800
20 POKE 53272,(PEEK(53272)AND 240)OR 12
25 FOR I=0 TO 7:READ N:POKE CM+I,N:NEXT
199 END
800 POKE 56334,PEEK(56334) AND 254
805 POKE 1,PEEK(1) AND 251
810 FOR I=0 TO 1023:POKE CM+I,PEEK(CX+I):NEXT
815 POKE 1,PEEK(1) OR 4
820 POKE 56334,PEEK(56334) OR 1
825 RETURN
900 DATA 255,0,255,0,255,0,255,0
```

After this program is run, type the character @ to see the new character.

Combined character animation. This program shows the smooth animation that is possible using carefully planned custom characters. The picture will always consist of two characters, one representing the top half of a human figure, the other representing the bottom half. By POKEing the top half into screen memory location 500 and the bottom half into location 540 (exactly one row below it), it will seem to be a complete human figure.

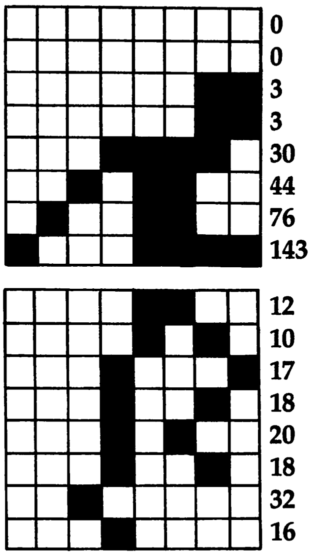
There are eight characters involved in the animation sequence, four for the top and four for the bottom. Each character is only slightly changed from the one before. By POKEing them into memory in the right order, the figure will seem to be running in place. Since this program does not use any of the regular ROM character set, the routine to copy the ROM set is not included.

```
10 CM=12288:SM=1024:CL=55296
20 POKE 53272,(PEEK(53272)AND 240)OR 12
```

```
25 FOR I=Ø TO 63:READ N:POKE CM+I,N:NEXT
100 POKE CL+5ØØ,1:POKE CL+54Ø,1
11Ø FOR I=Ø TO 6 STEP 2:POKE SM+5ØØ,I:POKE SM+54Ø,
    I+1:GOSUB 2ØØ:NEXT:GOTO 11Ø
13Ø GOTO 11Ø
199 END
2ØØ FOR J=Ø TO 99:NEXT:RETURN
9ØØ DATA Ø,Ø,3,3,3Ø,44,76,14Ø
91Ø DATA 12,1Ø,17,18,2Ø,18,32,16
92Ø DATA Ø,Ø,6,6,12,28,28,3Ø
93Ø DATA 12,1Ø,18,17,34,51,13Ø,64
94Ø DATA Ø,12,12,24,24,28,3Ø,14
95Ø DATA 12,1Ø,1Ø,1Ø,18,34,66,1
96Ø DATA Ø,Ø,6,6,8,29,46,76
97Ø DATA 12,12,12,66,72,8,8,12
```

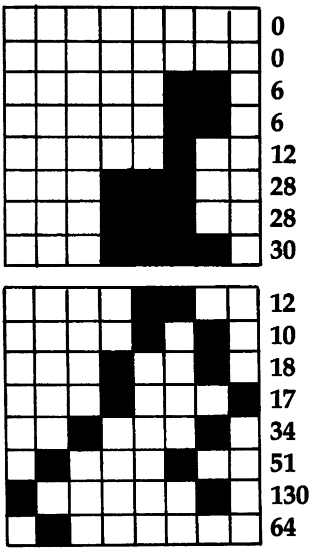
Figures 3 through 6 show the eight character patterns used to create the animation effect, along with the decimal numbers actually POKEd into memory.

Figure 3.



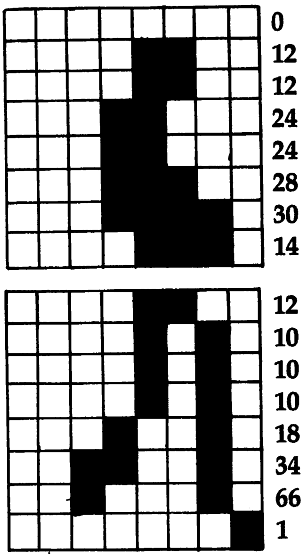
First Running Figure

Figure 4.



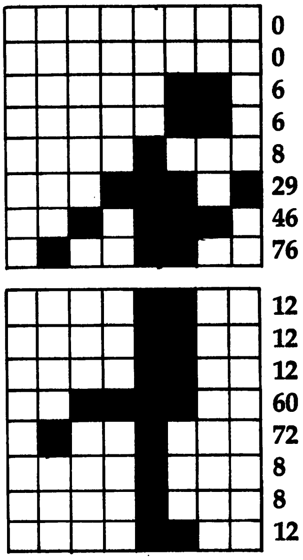
Second Running Figure

Figure 5.



Third Running Figure

Figure 6.



Fourth Running Figure

Ultrafont Character Editor

Charles Brannon

This fast, feature-packed, all machine language utility makes custom characters a breeze. Its unique features let you concentrate on your artwork instead of programming.

Anyone who has used graph paper to plot out characters, then tediously converted the rows into decimal numbers can appreciate a character editor. Instead of drawing and erasing on paper, you can draw your characters freehand with a joystick. "Ultrafont" has been written to offer almost every conceivable aid to help you design whole character sets.

Typing It In

Ultrafont is written entirely in machine language, giving you speed and efficiency that BASIC can't match. While this gives you a product of commercial quality, it does carry the liability of lots of typing. Ultrafont is actually rather short, using less than 3K of memory at hexadecimal location $C000, which is reserved for programs like Ultrafont. Therefore, you don't lose one byte of BASIC programming space.

However, 3,000 characters require three times as much typing, since each byte must be represented by a three-digit number (000-255). With that much typing, mistakes are inevitable. To make things manageable, we've prepared Ultrafont to be typed in using MLX, the Machine Language Editor. Full instructions on using the machine language editor are provided in the MLX article (Appendix I). So, despite the typing, rest assured that a few afternoons at the keyboard will yield a substantial reward.

Once you've entered, SAVEd, and RUN MLX, answer the two questions, starting address and ending address, with 49152 and 52139, respectively. After you've Saved the program with

MLX, you can load it from disk with LOAD "filename", 8,1 or LOAD "filename", 1,1 from tape. After it's loaded, enter NEW, then SYS 49152. This command "runs" the machine language program at $C000 (12*4096 = 49152).

The Display

After you SYS to Ultrafont, you should see the work area. At the bottom of the screen are eight lines of characters. These are the 256 characters you can customize, arranged in eight rows of 32 characters. A flashing square is resting on the @ symbol, the home position of the character set. Above the eight rows is the main grid, a blown-up view of ten characters. The last row of the screen is reserved for messages.

About the Grid

The grid is like a large-size window on the character set. You see the first five characters and the five beneath them. A large blue cursor shows you which character you are currently editing, and a smaller flashing square is the cursor you use to set and clear points.

Moving Around

You can use the cursor keys (up, down, left, right) to move the large blue cursor to any character you want to edit. If you move to a character not on the large grid (out of the window), the window will automatically scroll to make the character appear. You can also look at the bottom of the screen to move the larger cursor, as the flashing square on the character set moves with the main grid.

The HOME key moves the small cursor to the upper-left corner of the screen. If you press it twice it will take you back to the top of the character set — to @.

You move the small cursor within the grid using a joystick plugged into Port 2. If you move the cursor out of the current character, the blue cursor will jump to the next character in whatever direction you want to move. The display at the bottom will adjust, and the grid will scroll as necessary. This means that you can ignore the traditional boundaries between characters, and draw shapes as big as the entire character set (256 x 64 pixels — a pixel is a picture element, or dot). You can still edit one character at a time, or make a shape within a 2 x 2 box of characters.

The fire button is used to set and clear points. If the cursor is resting on a solid square, it will be turned off. If the square is off, it will be turned on. If you hold down fire while you move the joystick, you can stay in the same drawing mode. If you set a point, you will continue to draw as you move. If you clear a point, you can move around and erase points all over the screen.

If the drawing cursor is too fast or too slow to use, just press V to set the cursor velocity (speed). Answer the prompt with a speed from 0 (slow) to 9 (too fast for practical use).

Manipulations

There are several functions that affect the current character (where the blue box is). You can rotate, shift, mirror, reverse, erase, replace, and copy characters. The best way to learn is to play with the functions. It's really a lot of fun! The following keys control each function:

Function keys:

f1: Scroll character right. All pixels move right. The rightmost column of pixels wraps around to the left.
f2: Scroll character left. Wraparound is like f1.
f3: Scroll character down. All pixels move down. The last row of pixels wraps around to the top.
f4: Scroll character up. Wraparound is like f3.

R: Rotate. Rotates the character 90 degrees. Press twice to flip the character upside-down.
M: Mirror. Creates a mirror image of the character left to right.

CLR (SHIFT CLR/HOME): Erases current character.

CTRL-R or CTRL-9: Reverses the character. All set dots are clear, and all empty dots are set. The bottom half of the character set is the reversed image of the top half.

F: Fix. Use this if you want to restore the normal pattern for the character. If you've redefined A and press F, the Commodore pattern for A will be copied back from ROM.

Saving and Loading Character Sets

To save your creation to tape or disk, press S. Then press either T for tape or D for disk. When requested, enter the filename, up to 16 characters. Don't use the "0:" prefix if you're using a disk drive. The screen will clear, display the appropriate messages, and then return to the editing screen if there are no errors. If there are errors, such as the disk being full, Ultrafont will read the disk error message and display it at the bottom of the screen.

Press a key after you've read the message and try to correct the cause of the error before you save again. I don't think the computer can detect an error during a tape SAVE.

To load a character set previously saved, press L and answer the "Tape or Disk" message. Enter the filename. If you're using tape, be sure the tape is rewound and ready. After the load, you will be returned to the editing screen, and a glance is all it takes to see that the set is loaded. If an error is detected on tape load, you will see the message "Error on Save/Load". Once again, if you are using disk, the error message will be displayed. Press a key to return to editing so you can try again.

Copying and Moving Characters

You can copy one character to another with function keys 7 and 8. When you press f7, the current character will flash briefly, and it will be copied into a little buffer. Ultrafont will remember that character pattern. You can then position the cursor where you want to copy the character and press SHIFT-f7 (f8). The memorized character will then replace the character the cursor is resting on. You can also use the buffer as a fail-safe device. Before you begin to edit a character you've already worked on, press f7 to store it safely away. That way, if you accidentally wipe it out or otherwise garble the character, you can press f8 to bring back your earlier character.

Creating DATA Statements

A very useful command, CTRL-D (hold down CTRL and press D), allows you to create DATA statements for whatever characters you've defined. Ultrafont doesn't make DATA statements for all the characters, just the ones you've changed. After you press CTRL-D, Ultrafont adds the characters to the end of whatever program you have in BASIC memory. If there is no program, the DATA statements exist alone. You can LOAD Ultrafont, enter NEW to reset some BASIC pointers, LOAD a program you are working on, then SYS 49152 to Ultrafont to add DATA to the end of the program. The DATA statements always start at line 60000, so you may want to renumber them. If you press CTRL-D twice, another set of DATA statements will be appended, also numbered from line numbers 60000 and up. See the notes at the end of the article for more details on using the DATA statements in your own programs.

Chapter Three

Exiting Ultrafont

After you create the DATA, you'll still be in Ultrafont. If you want to exit to see the DATA statements or go on to other things, press CTRL-X. The screen will reset to the normal colors and you'll see READY. If you've made DATA, a LIST will dramatically reveal it. I recommend you enter the command CLR to make sure BASIC is initialized properly after exiting Ultrafont. One thing to watch out for: don't use RUN/STOP — RESTORE to exit Ultrafont. Ultrafont moves screen memory from the default area at 1024, and the RUN/STOP — RESTORE combination does not reset the operating system pointers to screen memory. If you do press it, you will not be able to see what you are typing. To fix it, type blind POKE 648,4 or SYS 49152 to reenter Ultrafont so you can exit properly.

Reentering Ultrafont

After you've exited, you can reRUN Ultrafont with SYS 49152. You'll see the character set you were working on previously, along with the message USE ROM SET (Y/N). Usually, Ultrafont will copy the ROM character patterns into RAM where you can change them. If you press N, however, the set you were working on previously is left untouched. Press any other key, like RETURN, to reset the characters to the ROM standard.

A Whole New World

We're not finished yet. There is a whole other mode of operation within Ultrafont, the multicolor mode. In multicolor mode, any character can contain up to four colors (including the background) simultaneously. Multicolor changes the way the computer interprets character patterns. Instead of a 1 bit representing a solid pixel, and 0 representing a blank, the eight bits are organized as four pairs of bits. Each pair can represent four possibilities: 00, 01, 10, and 11. Each of these also is a number in decimal from 0-3. Each two-bit bit-pattern represents one of the four colors. Programming and using multicolor characters is described in the following article, "Advanced Use of Character Graphics."

Ultrafont makes multicolor easy. You don't have to keep track of bit pairs, any more than you have to convert binary to decimal. Just press the f5 function key. Presto! The whole screen changes. The characters are rather unrecognizable, and the drawing cursor is twice as wide (since eight bits have been reduced to four pixel-pairs, making each dot twice as wide). You only have four dots horizontally per character, but you can easily combine many characters to form larger shapes.

Multicolor redefines the way the joystick and fire button works. The fire button always lays down a colored rectangle in the color you are currently working with. The color it lays down is shown in the center of the drawing cursor. Press the number keys 1, 2, 3, or 4 to choose different colors to draw with. The number of the key is one more than the bit pattern, so color 1 is bit pattern 00, and color 4 is bit pattern 11. When you first SYS to Ultrafont, the four colors are black (background), white, cyan, and purple. These four colors show up distinctly on a black and white TV or monitor.

You can easily change the colors. Just hold down SHIFT and press the appropriate number key to change that number's color. You will see the message PRESS COLOR KEY. Now press one of the color keys from CTRL-1 to CTRL-8 or Commodore-1 to Commodore-8. Hold down CTRL or the Commodore logo key as you do this. Instantly, that color, and everything previously drawn in that color, is changed.

Three of the colors (including 1, the background color) can be any of the 16 colors. But because of the way multicolor works, color 4, which is represented by bit pattern 11, or 3 in decimal, can only be one of the 8 CTRL-colors. Assigning it one of the Commodore logo colors just picks the color shown on the face of the color key. Incidentally, it is the color of bit pattern 3 (color 4) that changes according to the character color as set in color memory. The other colors are programmed in multicolor registers 1 and 2 (POKE 53282 and 53283), so all characters share these two colors. When you want to vary a certain color without affecting the rest of the characters, you'll want to draw it in color 4.

Some of the commands in the multicolor mode aren't as useful as others. You have to press f1 and f2 twice to shift a character, since they only shift one bit, which causes all the colors to change. You can use CTRL-R, Reverse, to reverse all the colors (color 1 becomes color 4, color 2 becomes color 3, and color 3 becomes color 2). R: Rotate changes all the colors and is rather useless unless you press it twice to just turn the characters upside down. M: Mirror will switch colors 2 and 3, since bit pattern 01 (color 2) becomes 10 (color 3). You can still copy characters using f7 and f8 (see above).

Returning to Normal

You can switch back instantly to the normal character mode by pressing f6 (SHIFT-f5). If you were drawing in multicolor, you can see the bit patterns that make up each color. In other words, multicolor characters look just as strange in normal mode as normal characters look in multicolor.

If you changed colors in the multicolor mode, some of the colors in the normal mode may have changed. You can change these colors as in multicolor mode. Press SHIFT-1 to change the color of the empty pixels, and SHIFT-3 to change the color of the eight rows of characters. Use SHIFT-2 to change the color of the *on* pixels.

Programming

The following article shows you how you can make the most of characters. It includes several short machine language utilities that you can use when writing games or other programs using these custom characters. It shows how your program can read the SAVEd files directly, without having to POKE from DATA statements. You should still have a good grasp of the essentials of programming characters (see Scott Card's "Make Your Own Characters"). Ultrafont is intended as an artistic aid in your creations, letting the computer take over the tedious tasks it is best suited for.

Notes: How to Use the DATA Statements

The DATA statements are created from lines 60000 and up, as many as necessary. Each line of data has nine numbers. The first number is the internal code of the character (the code you use when POKEing to the screen). It represents an offset into the table of character patterns. The eight bytes that follow are the decimal numbers for the eight bytes it takes to define any character. A sample program to read them and display them could be:

```
10 POKE 52,48:POKE 56,48:CLR
50 READ A:IF A=-1 THEN 70
60 FOR I=0 TO7:READ B:POKE 12288+A*8+I,B:NEXT:GOTO
   50
70 PRINT CHR$(147);"{10 DOWN}":REM TEN CURSOR DOWN
   S
80 FOR I=0TO7:FORJ=0TO31:POKE1028+J+I*40,I*32+J:PO
   KE 55300+J+I*40,1:NEXT:NEXT
90 POKE 53272,(PEEK(53272)AND240)OR12:END
```

Add:

```
63999 DATA -1
```

If you want to have your cake and eat it too (that is, also have the normal ROM patterns), copy them from ROM down to RAM by adding:

```
20 POKE 56334,PEEK(56334)AND254:POKE 1,PEEK(1)AND2
   51
30 FOR I=0 TO 2047:POKE 12288+I,PEEK(53248+I):NEXT
40 POKE 1,PEEK(1)OR4:POKE 56334,PEEK(56334)OR1
```

Quick Reference: Ultrafont Commands (joystick in port #2)

Cursor keys:	Move to next character
HOME (CLR/HOME):	Moves the cursor to upper left corner. Press twice to go back to start.
V:	Cursor velocity. Answer from 0 (slow) to 9 (fast).
f1:	Scroll right with wraparound
f2 (SHIFT-f1):	Scroll left
f3:	Scroll down
f4 (SHIFT-f3):	Scroll up
R:	Rotate 90 degrees. Press twice to invert.
M:	Mirror image
SHIFT CLR/HOME:	Erase current character
CTRL-R, CTRL-9:	Reverse pixels
F:	Fix character from ROM pattern
L:	Load. Tape or Disk, Filename
S:	Save. Tape or Disk, Filename
f7:	Memorize character (keep)
f8 (SHIFT-f7):	Recall character (put)

Ultrafont

```
49152 :076,029,196,000,001,003,049
49158 :004,000,173,048,002,072,049
49164 :173,045,002,141,048,002,167
49170 :141,079,002,032,043,193,252
49176 :104,141,048,002,169,100,076
49182 :133,252,169,000,133,251,200
49188 :133,167,169,216,133,168,254
49194 :169,008,141,040,002,169,059
```

```
49200 :ØØ2,141,Ø42,ØØ2,169,ØØ5,153
49206 :141,Ø41,ØØ2,174,ØØ3,192,Ø95
49212 :173,Ø79,ØØ2,2Ø5,Ø48,ØØ2,Ø57
49218 :2Ø8,ØØ2,162,ØØ6,142,Ø8Ø,154
49224 :ØØ2,16Ø,ØØØ,177,253,17Ø,Ø66
4923Ø :173,Ø63,ØØ2,24Ø,ØØ3,Ø76,123
49236 :229,192,169,2Ø7,145,251,253
49242 :138,Ø1Ø,17Ø,176,ØØ8,173,253
49248 :Ø8Ø,ØØ2,145,167,Ø76,1Ø8,162
49254 :192,173,ØØ4,192,145,167,2Ø7
4926Ø :2ØØ,192,ØØ8,2Ø8,221,Ø24,193
49266 :165,251,1Ø5,ØØ8,133,251,ØØ3
49272 :133,167,165,252,1Ø5,ØØØ,174
49278 :133,252,1Ø5,116,133,168,ØØ9
49284 :Ø24,165,253,1Ø5,ØØ8,133,Ø52
4929Ø :253,165,254,1Ø5,ØØØ,133,Ø24
49296 :254,Ø56,238,Ø79,ØØ2,2Ø6,211
493Ø2 :Ø41,ØØ2,173,Ø41,ØØ2,2Ø8,1Ø5
493Ø8 :156,Ø56,173,Ø79,ØØ2,233,Ø87
49314 :ØØ5,141,Ø79,ØØ2,Ø56,165,Ø98
4932Ø :253,233,Ø39,133,253,165,22Ø
49326 :254,233,ØØØ,133,254,2Ø6,23Ø
49332 :Ø4Ø,ØØ2,173,Ø4Ø,ØØ2,24Ø,165
49338 :ØØ3,Ø76,Ø52,192,2Ø6,Ø42,245
49344 :ØØ2,173,Ø42,ØØ2,24Ø,Ø3Ø,169
4935Ø :169,ØØ8,141,Ø4Ø,ØØ2,Ø24,Ø7Ø
49356 :173,Ø79,ØØ2,1Ø5,Ø32,141,224
49362 :Ø79,ØØ2,Ø24,165,253,1Ø5,Ø7Ø
49368 :248,133,253,165,254,1Ø5,Ø94
49374 :ØØØ,133,254,Ø76,Ø52,192,161
4938Ø :Ø96,134,Ø97,169,ØØØ,141,Ø97
49386 :Ø43,ØØ2,ØØ6,Ø97,Ø46,Ø43,215
49392 :ØØ2,ØØ6,Ø97,Ø46,Ø43,ØØ2,18Ø
49398 :174,Ø43,ØØ2,169,2Ø7,145,218
494Ø4 :251,2ØØ,169,247,145,251,235
4941Ø :136,189,ØØ3,192,145,167,Ø66
49416 :2ØØ,145,167,2ØØ,192,ØØ8,152
49422 :2Ø8,215,Ø76,113,192,169,219
49428 :ØØØ,141,Ø26,2Ø8,165,ØØ1,Ø49
49434 :Ø41,251,133,ØØ1,Ø96,165,2Ø1
4944Ø :ØØ1,ØØ9,ØØ4,133,ØØ1,169,Ø93
49446 :ØØ1,141,Ø26,2Ø8,Ø96,169,167
49452 :ØØØ,133,254,173,Ø48,ØØ2,142
49458 :133,253,ØØ6,253,Ø38,254,219
49464 :ØØ6,253,Ø38,254,ØØ6,253,Ø98
4947Ø :Ø38,254,Ø24,169,112,1Ø1,248
49476 :254,133,254,Ø96,Ø32,Ø43,112
49482 :193,16Ø,ØØØ,177,253,Ø73,162
49488 :255,145,253,2ØØ,192,ØØ8,1Ø9
```

```
49494 :208,245,032,008,192,096,099
49500 :169,102,133,252,169,218,111
49506 :133,168,169,132,133,251,060
49512 :133,167,162,008,169,000,231
49518 :133,097,160,000,165,097,250
49524 :145,251,230,097,173,058,046
49530 :002,145,167,200,192,032,092
49536 :208,240,024,165,251,105,097
49542 :040,133,251,133,167,165,255
49548 :252,105,000,133,252,105,219
49554 :116,133,168,202,208,216,165
49560 :096,032,019,193,169,112,005
49566 :133,252,162,008,169,208,066
49572 :133,254,169,000,133,253,082
49578 :133,251,168,177,253,145,017
49584 :251,200,208,249,230,254,032
49590 :230,252,202,208,242,165,201
49596 :252,201,128,208,223,032,208
49602 :031,193,096,032,043,193,014
49608 :160,000,177,253,010,008,040
49614 :074,040,042,145,253,200,192
49620 :192,008,208,242,076,008,178
49626 :192,032,043,193,160,000,070
49632 :177,253,074,008,010,040,018
49638 :106,145,253,200,192,008,110
49644 :208,242,076,008,192,032,226
49650 :043,193,160,000,177,253,044
49656 :133,097,200,177,253,136,220
49662 :145,253,200,200,192,008,228
49668 :208,245,165,097,136,145,232
49674 :253,076,008,192,032,043,102
49680 :193,160,007,177,253,133,171
49686 :097,136,177,253,200,145,006
49692 :253,136,016,247,200,165,021
49698 :097,145,253,076,008,192,037
49704 :032,043,193,160,000,169,125
49710 :000,133,097,162,008,177,111
49716 :253,010,102,097,202,208,156
49722 :250,165,097,145,253,200,144
49728 :192,008,208,233,076,008,021
49734 :192,032,043,193,160,008,186
49740 :169,000,153,048,002,136,072
49746 :208,250,169,007,133,097,178
49752 :152,170,169,000,133,007,207
49758 :177,253,074,145,253,038,010
49764 :007,202,016,251,166,097,071
49770 :165,007,029,049,002,157,003
49776 :049,002,198,097,165,097,208
49782 :016,224,200,192,008,208,198
```

```
49788 :215,136,185,Ø49,ØØ2,145,Ø88
49794 :253,136,Ø16,248,Ø76,ØØ8,Ø99
498ØØ :192,Ø32,Ø43,193,16Ø,ØØØ,244
498Ø6 :152,145,253,2ØØ,192,ØØ8,Ø68
49812 :2Ø8,249,Ø76,ØØ8,192,12Ø,233
49818 :169,127,141,Ø13,22Ø,169,225
49824 :ØØ1,141,Ø26,2Ø8,169,177,114
4983Ø :141,Ø18,2Ø8,169,Ø27,141,1Ø2
49836 :Ø17,2Ø8,169,186,141,Ø2Ø,145
49842 :ØØ3,169,194,141,Ø21,ØØ3,197
49848 :Ø88,Ø96,173,Ø18,2Ø8,2Ø1,2ØØ
49854 :177,2Ø8,Ø39,169,242,141,142
4986Ø :Ø18,2Ø8,173,Ø44,ØØ2,141,Ø14
49866 :Ø24,2Ø8,173,Ø22,2Ø8,Ø41,11Ø
49872 :239,Ø13,Ø63,ØØ2,141,Ø22,176
49878 :2Ø8,173,Ø57,ØØ2,141,Ø33,Ø6Ø
49884 :2Ø8,169,ØØ1,141,Ø25,2Ø8,2Ø4
4989Ø :1Ø4,168,1Ø4,17Ø,1Ø4,Ø64,172
49896 :169,177,141,Ø18,2Ø8,169,Ø9Ø
499Ø2 :158,141,Ø24,2Ø8,173,Ø32,2Ø6
499Ø8 :2Ø8,141,Ø33,2Ø8,169,2ØØ,179
49914 :141,Ø22,2Ø8,238,Ø37,2Ø8,Ø8Ø
4992Ø :169,ØØ1,141,Ø25,2Ø8,Ø76,1Ø8
49926 :Ø49,234,Ø85,Ø64,ØØØ,Ø64,246
49932 :Ø64,ØØØ,Ø76,Ø64,ØØØ,Ø76,Ø36
49938 :Ø64,ØØØ,Ø76,Ø64,ØØØ,Ø76,Ø42
49944 :Ø64,ØØØ,Ø64,Ø64,ØØØ,Ø85,Ø45
4995Ø :Ø64,ØØØ,ØØØ,ØØØ,Ø85,Ø8Ø,ØØ3
49956 :ØØØ,Ø64,Ø16,ØØØ,Ø64,Ø16,196
49962 :ØØØ,Ø64,Ø16,ØØØ,Ø64,Ø16,2Ø2
49968 :ØØØ,Ø64,Ø16,ØØØ,Ø64,Ø16,2Ø8
49974 :ØØØ,Ø64,Ø16,ØØØ,Ø64,Ø16,214
4998Ø :ØØØ,Ø85,Ø8Ø,ØØØ,ØØØ,ØØØ,225
49986 :ØØØ,255,255,255,ØØØ,ØØ1,Ø64
49992 :ØØ1,ØØ1,ØØØ,255,ØØ1,ØØØ,Ø74
49998 :ØØØ,255,ØØ1,ØØØ,ØØØ,255,Ø77
5ØØØ4 :ØØ1,Ø18,Ø85,Ø76,Ø84,Ø82,174
5ØØ1Ø :Ø65,Ø7Ø,Ø79,Ø78,Ø84,146,1ØØ
5ØØ16 :Ø95,Ø69,Ø82,Ø82,Ø79,Ø82,Ø73
5ØØ22 :Ø32,Ø79,Ø78,Ø32,Ø83,Ø65,215
5ØØ28 :Ø86,Ø69,Ø47,Ø76,Ø79,Ø65,Ø18
5ØØ34 :Ø68,Ø95,Ø18,Ø84,146,Ø65,Ø78
5ØØ4Ø :Ø8Ø,Ø69,Ø32,Ø79,Ø82,Ø32,238
5ØØ46 :Ø18,Ø68,146,Ø73,Ø83,Ø75,Ø77
5ØØ52 :Ø63,Ø95,Ø7Ø,Ø73,Ø76,Ø69,Ø66
5ØØ58 :Ø78,Ø65,Ø77,Ø69,Ø58,Ø95,Ø68
5ØØ64 :Ø69,Ø78,Ø84,Ø69,Ø82,Ø32,Ø46
5ØØ7Ø :Ø67,Ø79,Ø76,Ø79,Ø82,Ø32,Ø53
5ØØ76 :Ø75,Ø69,Ø89,Ø95,Ø85,Ø83,14Ø
```

```
5ØØ82  :Ø69,Ø32,Ø82,Ø79,Ø77,Ø32,Ø21
5ØØ88  :Ø83,Ø69,Ø84,Ø63,Ø32,Ø4Ø,Ø27
5ØØ94  :Ø89,Ø47,Ø78,Ø41,Ø95,169,181
5Ø1ØØ  :Ø85,16Ø,195,133,251,132,112
5Ø1Ø6  :252,16Ø,Ø4Ø,169,Ø32,153,224
5Ø112  :191,1Ø3,136,2Ø8,25Ø,177,233
5Ø118  :251,2ØØ,2Ø1,Ø95,2Ø8,249,122
5Ø124  :136,132,Ø97,152,Ø74,Ø73,1ØØ
5Ø13Ø  :255,Ø56,1Ø5,Ø2Ø,168,162,2Ø8
5Ø136  :Ø24,Ø24,Ø32,24Ø,255,16Ø,183
5Ø142  :ØØØ,177,251,Ø32,21Ø,255,123
5Ø148  :2ØØ,196,Ø97,144,246,Ø96,183
5Ø154  :133,251,132,252,16Ø,Ø4Ø,178
5Ø16Ø  :169,Ø32,153,191,1Ø3,136,ØØØ
5Ø166  :2Ø8,25Ø,162,Ø24,16Ø,ØØØ,Ø26
5Ø172  :Ø24,Ø32,24Ø,255,16Ø,ØØØ,195
5Ø178  :177,251,2Ø1,Ø95,24Ø,ØØ6,2Ø4
5Ø184  :Ø32,21Ø,255,2ØØ,2Ø8,244,133
5Ø19Ø  :Ø96,174,Ø76,ØØ2,24Ø,ØØ8,Ø98
5Ø196  :16Ø,ØØØ,2ØØ,2Ø8,253,2Ø2,Ø19
5Ø2Ø2  :2Ø8,25Ø,Ø96,173,ØØ2,221,2Ø8
5Ø2Ø8  :ØØ9,ØØ3,141,ØØ2,221,173,Ø69
5Ø214  :ØØØ,221,Ø41,252,ØØ9,ØØ2,Ø51
5Ø22Ø  :141,ØØØ,221,169,1ØØ,141,Ø48
5Ø226  :136,ØØ2,169,147,Ø32,21Ø,234
5Ø232  :255,169,144,Ø32,21Ø,255,Ø97
5Ø238  :169,ØØ8,Ø32,21Ø,255,16Ø,128
5Ø244  :ØØØ,152,153,128,Ø99,2ØØ,Ø32
5Ø25Ø  :Ø16,25Ø,168,185,ØØ8,195,128
5Ø256  :153,128,Ø99,2ØØ,192,Ø23,1Ø7
5Ø262  :2Ø8,245,16Ø,ØØØ,185,Ø31,147
5Ø268  :195,153,192,Ø99,2ØØ,192,Ø99
5Ø274  :Ø32,2Ø8,245,169,156,141,Ø25
5Ø28Ø  :Ø44,ØØ2,169,Ø12,141,Ø32,248
5Ø286  :2Ø8,169,128,141,138,ØØ2,128
5Ø292  :Ø32,153,194,169,Ø48,141,Ø85
5Ø298  :Ø76,ØØ2,169,Ø11,141,Ø57,Ø66
5Ø3Ø4  :ØØ2,169,ØØ7,169,ØØØ,141,1Ø4
5Ø31Ø  :Ø48,ØØ2,141,Ø45,ØØ2,141,ØØ1
5Ø316  :Ø63,ØØ2,173,ØØ6,192,ØØ9,Ø73
5Ø322  :ØØ8,141,Ø58,ØØ2,173,ØØ4,Ø2Ø
5Ø328  :192,141,Ø34,2Ø8,173,ØØ5,137
5Ø334  :192,141,Ø35,2Ø8,Ø32,ØØ8,ØØ6
5Ø34Ø  :192,Ø32,Ø92,193,169,2Ø3,Ø21
5Ø346  :2Ø5,ØØ7,192,24Ø,ØØ6,141,193
5Ø352  :ØØ7,192,Ø76,197,196,169,245
5Ø358  :16Ø,16Ø,195,Ø32,183,195,Ø83
5Ø364  :Ø32,228,255,24Ø,251,2Ø1,115
5Ø37Ø  :Ø78,24Ø,ØØ6,Ø32,153,193,128
```

```
50376 :032,008,192,032,179,195,070
50382 :169,142,141,248,103,169,154
50388 :143,141,249,103,169,003,252
50394 :141,021,208,169,024,141,154
50400 :000,208,169,000,141,016,246
50406 :208,169,051,141,001,208,240
50412 :169,176,141,003,208,169,078
50418 :053,141,002,208,169,000,047
50424 :141,029,208,141,023,208,230
50430 :141,038,208,169,003,141,186
50436 :028,208,169,000,141,059,097
50442 :002,141,060,002,173,000,132
50448 :220,072,041,015,073,015,196
50454 :141,061,002,104,041,016,131
50460 :141,062,002,032,228,255,236
50466 :240,006,032,163,198,076,237
50472 :014,197,032,015,196,173,155
50478 :062,002,208,003,032,054,151
50484 :198,173,062,002,073,016,064
50490 :141,075,002,173,061,002,000
50496 :240,204,174,061,002,189,166
50502 :063,195,172,063,002,240,037
50508 :001,010,024,109,059,002,025
50514 :141,059,002,024,173,060,029
50520 :002,125,074,195,141,060,173
50526 :002,174,059,002,016,025,116
50532 :162,000,142,059,002,173,126
50538 :045,002,240,015,206,045,147
50544 :002,162,007,173,063,002,009
50550 :240,002,162,006,142,059,217
50556 :002,174,059,002,224,040,113
50562 :144,022,162,039,142,059,186
50568 :002,173,045,002,201,219,010
50574 :176,010,105,001,141,045,108
50580 :002,162,032,142,059,002,035
50586 :172,060,002,016,022,160,074
50592 :000,140,060,002,173,045,068
50598 :002,201,032,144,010,233,020
50604 :032,141,045,002,160,007,047
50610 :140,060,002,172,060,002,102
50616 :192,016,144,022,160,015,221
50622 :140,060,002,173,045,002,100
50628 :201,192,176,010,105,032,144
50634 :141,045,002,160,008,140,186
50640 :060,002,173,059,002,172,164
50646 :060,002,074,074,074,192,178
50652 :008,144,002,105,031,109,107
50658 :045,002,141,048,002,041,249
50664 :224,074,074,105,176,141,002
```

```
50670 :003,208,173,048,002,041,201
50676 :031,010,010,010,105,053,207
50682 :141,002,208,169,000,105,107
50688 :000,133,097,173,060,002,209
50694 :010,010,010,105,051,141,077
50700 :001,208,173,059,002,010,209
50706 :010,010,038,097,105,024,046
50712 :141,000,208,165,097,105,228
50718 :000,141,016,208,173,048,104
50724 :002,205,081,002,240,009,063
50730 :032,008,192,173,048,002,241
50736 :141,081,002,076,014,197,047
50742 :032,043,193,173,060,002,045
50748 :041,007,168,173,059,002,254
50754 :041,007,073,007,170,232,084
50760 :134,097,056,169,000,042,058
50766 :202,208,252,174,063,002,211
50772 :208,048,133,097,173,075,050
50778 :002,208,022,169,000,141,120
50784 :064,002,141,038,208,177,214
50790 :253,037,097,208,008,169,106
50796 :001,141,064,002,141,038,239
50802 :208,165,097,073,255,049,193
50808 :253,174,064,002,240,002,087
50814 :005,097,145,253,032,008,154
50820 :192,096,133,098,074,005,218
50826 :098,073,255,049,253,166,008
50832 :097,202,133,097,173,066,144
50838 :002,074,042,202,208,252,162
50844 :005,097,145,253,076,008,228
50850 :192,141,065,002,174,191,159
50856 :198,221,191,198,240,004,196
50862 :202,208,248,096,202,138,244
50868 :010,170,189,224,198,072,019
50874 :189,223,198,072,096,031,227
50880 :133,137,134,138,077,082,125
50886 :147,018,145,017,157,029,199
50892 :070,135,139,049,050,051,186
50898 :052,019,136,140,033,034,112
50904 :035,036,086,083,076,024,044
50910 :004,218,193,196,193,013,015
50916 :194,240,193,039,194,070,134
50922 :194,136,194,071,193,028,026
50928 :199,050,199,072,199,094,029
50934 :199,118,199,153,199,196,030
50940 :199,208,199,208,199,208,193
50946 :199,208,199,225,199,250,002
50952 :199,016,200,050,200,050,211
50958 :200,050,200,050,200,121,067
```

```
50964 :200,172,201,056,202,072,155
50970 :202,187,202,173,060,002,084
50976 :041,007,133,097,056,173,027
50982 :060,002,233,008,056,229,114
50988 :097,141,060,002,076,114,022
50994 :199,173,060,002,041,007,020
51000 :133,097,024,173,060,002,033
51006 :105,008,056,229,097,141,186
51012 :060,002,076,114,199,173,180
51018 :059,002,041,007,133,097,157
51024 :056,173,059,002,233,008,099
51030 :056,229,097,141,059,002,158
51036 :076,114,199,173,059,002,203
51042 :041,007,133,097,024,173,061
51048 :059,002,105,008,056,229,051
51054 :097,141,059,002,104,104,105
51060 :076,095,197,032,043,193,240
51066 :032,019,193,160,007,024,045
51072 :165,254,105,096,141,142,007
51078 :199,165,253,141,141,199,208
51084 :185,000,208,145,253,136,043
51090 :016,248,032,031,193,076,230
51096 :008,192,169,016,141,063,229
51102 :002,169,001,141,029,208,196
51108 :032,008,192,173,058,002,117
51114 :009,008,141,058,002,032,164
51120 :092,193,169,050,141,065,118
51126 :002,032,209,199,173,059,088
51132 :002,041,254,141,059,002,175
51138 :076,114,199,169,000,141,125
51144 :063,002,141,029,208,032,163
51150 :008,192,096,056,173,065,028
51156 :002,233,049,141,066,002,193
51162 :170,189,003,192,141,038,183
51168 :208,096,173,059,002,013,007
51174 :060,002,208,003,141,045,177
51180 :002,169,000,141,059,002,097
51186 :141,060,002,032,008,192,165
51192 :076,114,199,032,072,193,166
51198 :032,072,193,032,043,193,051
51204 :160,000,177,253,153,067,046
51210 :002,200,192,008,208,246,098
51216 :096,032,043,193,160,000,028
51222 :185,067,002,145,253,200,106
51228 :192,008,208,246,076,008,254
51234 :192,144,005,028,159,156,206
51240 :030,031,158,129,149,150,175
51246 :151,152,153,154,155,169,212
51252 :144,160,195,032,183,195,193
```

```
51258 :032,228,255,240,251,162,202
51264 :000,221,035,200,240,008,000
51270 :232,224,016,208,246,076,048
51276 :179,195,056,173,065,002,234
51282 :233,033,168,138,153,003,042
51288 :192,192,003,240,010,192,149
51294 :000,240,019,153,033,208,235
51300 :076,116,200,174,063,002,219
51306 :240,002,009,008,141,058,052
51312 :002,032,092,193,032,008,215
51318 :192,076,179,195,169,158,063
51324 :160,200,032,183,195,032,158
51330 :228,255,056,233,048,048,230
51336 :248,201,010,176,244,133,124
51342 :097,056,169,009,229,097,031
51348 :010,010,010,010,141,076,149
51354 :002,076,179,195,067,085,246
51360 :082,083,079,082,032,086,092
51366 :069,076,079,067,073,084,102
51372 :089,032,040,048,045,057,227
51378 :041,063,095,160,000,140,165
51384 :078,002,169,164,032,210,071
51390 :255,169,157,032,210,255,244
51396 :032,228,255,240,251,172,094
51402 :078,002,133,097,169,032,201
51408 :032,210,255,169,157,032,039
51414 :210,255,165,097,201,013,131
51420 :240,039,201,020,208,013,173
51426 :192,000,240,209,136,169,148
51432 :157,032,210,255,076,183,121
51438 :200,041,127,201,032,144,215
51444 :194,192,020,240,190,165,221
51450 :097,153,000,002,032,210,232
51456 :255,200,076,183,200,169,059
51462 :095,153,000,002,152,096,248
51468 :032,231,255,169,116,160,207
51474 :195,032,183,195,032,228,115
51480 :255,240,251,162,001,201,110
51486 :084,240,011,162,008,201,224
51492 :068,240,005,104,104,076,121
51498 :179,195,141,077,002,160,028
51504 :001,169,001,032,186,255,180
51510 :169,134,160,195,032,234,210
51516 :195,032,181,200,208,007,115
51522 :173,077,002,201,084,208,043
51528 :237,173,077,002,201,068,062
51534 :208,066,169,064,141,020,234
51540 :002,169,048,141,021,002,211
51546 :169,058,141,022,002,160,130
```

```
51552 :ØØØ,185,ØØØ,ØØ2,153,Ø23,2Ø3
51558 :ØØ2,2ØØ,2Ø4,Ø78,ØØ2,2Ø8,Ø28
51564 :244,169,Ø44,153,Ø23,ØØ2,231
5157Ø :169,Ø8Ø,153,Ø24,ØØ2,173,2Ø3
51576 :Ø65,ØØ2,2Ø1,Ø83,2Ø8,Ø12,179
51582 :169,Ø44,153,Ø25,ØØ2,169,176
51588 :Ø87,153,Ø26,ØØ2,2ØØ,2ØØ,Ø32
51594 :2ØØ,2ØØ,2ØØ,2ØØ,2ØØ,Ø76,19Ø
516ØØ :16Ø,2Ø1,16Ø,ØØØ,185,ØØØ,Ø82
516Ø6 :ØØ2,153,Ø2Ø,ØØ2,2ØØ,2Ø4,219
51612 :Ø78,ØØ2,2Ø8,244,152,162,234
51618 :Ø2Ø,16Ø,ØØ2,Ø32,189,255,Ø52
51624 :169,16Ø,133,178,Ø96,Ø32,168
5163Ø :Ø12,2Ø1,Ø32,Ø79,2Ø2,169,1Ø1
51636 :ØØØ,133,253,133,251,169,Ø95
51642 :112,133,252,162,255,16Ø,236
51648 :119,169,251,Ø32,216,255,21Ø
51654 :176,Ø11,Ø32,183,255,2Ø8,Ø39
5166Ø :ØØ6,Ø32,1Ø2,2Ø2,Ø76,179,Ø33
51666 :195,Ø32,1Ø2,2Ø2,Ø32,231,236
51672 :255,173,Ø77,ØØ2,2Ø1,Ø68,224
51678 :24Ø,Ø15,169,Ø97,16Ø,195,Ø74
51684 :Ø32,183,195,Ø32,228,255,129
5169Ø :24Ø,251,Ø76,179,195,169,Ø64
51696 :ØØØ,Ø32,189,255,169,Ø15,132
517Ø2 :162,ØØ8,16Ø,Ø15,Ø32,186,Ø41
517Ø8 :255,Ø32,192,255,162,Ø15,139
51714 :Ø32,198,255,16Ø,ØØØ,Ø32,167
5172Ø :2Ø7,255,2Ø1,Ø13,24Ø,ØØ7,163
51726 :153,ØØØ,ØØ2,2ØØ,Ø76,ØØ7,196
51732 :2Ø2,169,Ø95,153,ØØØ,ØØ2,129
51738 :Ø32,2Ø4,255,169,ØØØ,16Ø,Ø78
51744 :ØØ2,Ø32,183,195,162,Ø15,1Ø9
5175Ø :Ø32,2Ø1,255,169,Ø73,Ø32,Ø32
51756 :21Ø,255,169,Ø13,Ø32,21Ø,165
51762 :255,Ø32,231,255,Ø76,231,1Ø6
51768 :2Ø1,Ø32,Ø12,2Ø1,Ø32,Ø79,1Ø1
51774 :2Ø2,169,ØØØ,Ø32,213,255,165
5178Ø :176,141,Ø76,1Ø2,2Ø2,169,166
51786 :ØØ4,141,136,ØØ2,ØØØ,169,Ø14
51792 :ØØØ,141,Ø26,2Ø8,173,Ø13,129
51798 :22Ø,ØØ9,128,141,Ø13,22Ø,Ø49
518Ø4 :169,ØØØ,141,Ø21,2Ø8,169,Ø32
5181Ø :147,Ø76,21Ø,255,Ø32,153,2Ø3
51816 :194,169,ØØ3,141,Ø21,2Ø8,Ø72
51822 :Ø32,ØØ8,192,Ø32,Ø92,193,147
51828 :Ø76,179,195,248,169,ØØØ,215
51834 :141,ØØØ,ØØ1,141,ØØ1,ØØ1,151
5184Ø :224,ØØØ,24Ø,Ø21,2Ø2,Ø24,Ø71
```

```
51846 :173,000,001,105,001,141,043
51852 :000,001,173,001,001,105,165
51858 :000,141,001,001,076,128,237
51864 :202,216,173,001,001,009,242
51870 :048,141,002,001,173,000,011
51876 :001,041,240,074,074,074,156
51882 :074,009,048,141,001,001,188
51888 :173,000,001,041,015,009,159
51894 :048,141,000,001,096,096,052
51900 :056,165,045,233,002,133,054
51906 :045,165,046,233,000,133,048
51912 :046,169,024,133,057,169,030
51918 :246,133,058,169,000,141,185
51924 :079,002,133,251,133,253,039
51930 :169,112,133,254,169,208,239
51936 :133,252,032,019,193,160,245
51942 :000,177,251,209,253,208,048
51948 :058,200,192,008,208,245,123
51954 :238,079,002,024,165,253,235
51960 :105,008,133,253,133,251,107
51966 :165,254,105,000,133,254,141
51972 :105,096,133,252,201,216,239
51978 :208,217,169,000,168,145,149
51984 :045,200,145,045,024,165,128
51990 :045,105,002,133,045,165,005
51996 :046,105,000,133,046,032,134
52002 :031,193,076,051,165,160,198
52008 :000,024,165,045,105,041,164
52014 :145,045,200,165,046,105,240
52020 :000,145,045,200,165,057,152
52026 :145,045,200,165,058,145,048
52032 :045,200,169,131,145,045,031
52038 :174,079,002,032,119,202,166
52044 :200,173,002,001,145,045,130
52050 :200,173,001,001,145,045,135
52056 :200,173,000,001,145,045,140
52062 :200,132,097,160,000,132,047
52068 :098,177,253,170,032,119,181
52074 :202,164,097,169,044,145,159
52080 :045,200,173,002,001,145,166
52086 :045,173,001,001,200,145,171
52092 :045,173,000,001,200,145,176
52098 :045,200,132,097,164,098,098
52104 :200,192,008,208,214,164,098
52110 :097,169,000,145,045,160,246
52116 :000,177,045,072,200,177,051
52122 :045,133,046,104,133,045,148
52128 :230,057,208,002,230,058,177
52134 :076,242,202,013,013,013,213
```

Advanced Use of Character Graphics

Charles Brannon

The many graphics capabilities of the Commodore 64 are enough to boggle any programmer. Sprites, 16 colors, high-resolution — it's all there. But many people have overlooked one of the most powerful graphics techniques: custom characters.

Custom characters are handy in games. You can redefine any letter of the alphabet and move that character about the screen with a simple PRINT or POKE statement. You can even program a foreign language alphabet, or a special set of technical or mathematical symbols.

Usually, we use custom characters in the normal text mode, but there are several character variations. Most promising are multicolor characters. Normally, a character is composed only of one color and the background color. The character can be any of 16 text colors. In multicolor mode, a single character can be defined with three colors, plus the background. It's all in how the character is defined.

In the normal text mode, a binary pattern determines the shape of a character. Each of the eight rows in a character is defined by a binary byte of eight bits: one bit per column. The binary representation defines 1 as a lit pixel, or dot, and 0 denotes an empty square. In this manner, an entire character is built up.

Multicolor mode "teams" bits together to allow more than one color per pixel. Instead of 1 representing a lit pixel, and 0 a blank, the bits are paired to form four binary patterns: 00, 01,10, and 11. Each binary pattern is a decimal number from 0-3. You can therefore have four colors (four possible bit-pairs), but you cut horizontal resolution in half, since the VIC-II chip still uses only one byte per row.

This limitation is easily overcome by combining several characters to form a larger shape. Each character has a resolution of four colored dots horizontally and eight vertically. You can gang up 2x2 characters to get a 8x16 matrix, or 3x2 for a 12x16 grid. The character editor "Ultrafont" allows you to easily design shapes larger than one character. You can also define some building-block shapes that you use to build large areas such as brick walls, ladders, or mountain ranges. You can use multicolor characters to draw nice-looking landscapes for sprite games without using the high-resolution page.

Tech Tips

Here's how the four colors are set. Each bit pattern is a code for where the color comes from. Bit pattern 00 is blank, and appears as the background color showing through. Pattern 01's color comes from memory location 53282. It can hold any color from 0-15, and all dots drawn in bit pattern 01 will appear in this color. Pattern 10 (decimal 2) gets its color from multicolor register 53283 in the same manner. Bit pattern 11 (decimal 3) is a different story. Its color comes from the character color in color memory. But you can use only colors 0-7.

Setting Multicolor Mode

Here's why: Commodore lets you have both normal and multicolor characters on the screen simultaneously. In the normal mode, colors 8-15, accessed with the Commodore logo key, are bonus colors, eight more text colors than the VIC-20. In multicolor mode, however, any color greater than 7 signals multicolor. The color of bit pattern 11 is the color 8 through 15 minus 8.

A character in multicolor mode printed in color Commodore-1 will not use orange, but black (CTRL-1). In other words, the lower three bits of the number are used in setting the color of multicolor bit pattern 11. Bit 4 (decimal 8) signals multicolor, so this precludes the use of the Commodore colors.

Selecting an Option

Multicolor mode is allowed if you set bit 4 of the VIC-II register 53270. In BASIC, you would enter:

```
10 POKE 53270,PEEK(53270)OR16
```

You can cancel multicolor mode with:

```
20 POKE 53270,PEEK(53270)AND239
```

Since all multicolor characters have the same colors for bit pattern 01 and 10 (from registers 53282 and 53283), you usually want to use bit pattern 11 in defining a character for the colors you want to change. For example, any spaceships you define will share the two multicolor registers, but you can have a red and a blue spaceship if you program one ship with the body color in bit pattern 11. You then use different character colors in color memory when you display the ships.

You can, of course, use sprites simultaneously with multicolor characters. There is a collision register that determines if a sprite and a background character have touched. The collision register at 53279 ($D01F) holds a value from bit position 0 to 7 representing which sprite hit the character. If sprite 3 (numbering from 0) hits a character, the register will hold 2 raised to the third power, or 8.

In multicolor, collisions are generated in the same way, but the hardware does not detect a collision between a sprite and bit pattern 01. You can make portions of the landscape you want to make transparent (collision-wise) by programming the noncolliding shapes in bit pattern 01. You can then discriminate between two different character colors by whether or not you get a collision at 53279 ($D01F).

Extended Background Color Mode

A holdover from the VIC-20, this mode gives you four background colors per character, in addition to any of 16 foreground colors. You can use this to highlight areas of the screen in a different background color without resorting to raster interrupts. You can only use the first 64 characters in the internal character set, however.

Enter the extended background color mode with:

```
POKE 53265,PEEK(53265)OR64
```

Use this line to turn off extended background color mode:

```
POKE 53265,PEEK(53265)AND191
```

Try typing some letters, including shifted letter, punctuation, and graphics symbols. You see that you can get different background colors within each character, but you also probably noticed that you couldn't access graphics. You just get the 64 standard alphanumeric character set, the lower 64 characters of the *internal* character set.

Out of 256 possible character codes (eight bits), only five bits (0-63) are used to tell *which* character to display. The upper two bits (64 and 128) tell what *color* the background should be. The color of the character itself (foreground) is defined in color memory, as usual. Use the following chart (adapted from the *Commodore 64 Programmer's Reference Guide*) to work with extended background color mode. Remember that the numbers are based on the screen POKE codes, not ASCII.

Screen/Internal Code	Bit 7	Bit 6	Color Register
From: 0-63	0	0	53281
64-127	0	1	53282
128-191	1	0	53283
192-255	1	1	53284

Just POKE 0-63 into screen memory, and the color will come from 53281 (as normally). Add 64 to the base number, and the character will get its background color from 53282. You can add either 128 or 192 to get the other two colors. This program places the letter A in four background colors:

```
1Ø POKE 53265,PEEK(53265)OR64:BM(Ø)=Ø:BM(1)=64:BM(
   2)=128:BM(3)=192
2Ø POKE5328Ø,Ø:POKE53281,Ø:POKE53282,2:POKE53283,8
   :POKE53284,7
3Ø PRINT"{CLR}{WHT}EX{RVS}TE{OFF}ND{RVS}ED"
4Ø FORI=ØTO15:OF=I*4Ø:FORJ=ØTO3:POKE1Ø4Ø+OF+J,1+BM
   (J):POKE55312+OF+J,I:NEXT:NEXT
5Ø FORW=1TO2ØØØ:NEXT:POKE53265,PEEK(53265)AND191:P
   RINT"{HOME}NORMAL MODE"
```

Extended background mode uses some of the same registers as multicolor mode. Commodore says not to use both modes simultaneously. Don't take their word for it — try it and see!

The rest of this article is dedicated to several short machine language routines that come in handy when programming characters. You can use all of them simultaneously, or separately. Each one is a set of DATA statements you can add to your program. GOSUB the appropriate line number to READ and POKE the machine language DATA into memory. You then use SYS to call each one.

Copydown

Decide where you'll put the character set and use the appropriate POKE to 53272 to select which 2K bank to use. SYS 49152 to copy down the uppercase ROM set to the RAM bank you've selected. Change the 208 in line 49176 to 216 if you'd rather copy down lowercase. (If you do this, the checksum on line 49000 won't match up, of course.)

Program 1. Copydown

```
49000 FORI=49152TO49235:READA:CK=CK+A:POKEI,A:NEXT
      :IFCK=10131THENRETURN
49010 PRINT"{RVS}ERROR IN DATA STATEMENTS:CHECK TY
      PING":STOP
49152 DATA 120,173,014,220,041,254
49158 DATA 141,014,220,165,001,041
49164 DATA 251,133,001,173,024,208
49170 DATA 041,014,010,010,133,167
49176 DATA 169,208,133,252,173,000
49182 DATA 221,041,003,073,003,010
49188 DATA 010,010,010,010,010,005
49194 DATA 167,133,254,169,000,133
49200 DATA 251,133,253,168,162,008
49206 DATA 177,251,145,253,200,208
49212 DATA 249,230,252,230,254,202
49218 DATA 208,242,165,001,009,004
49224 DATA 133,001,173,014,220,009
49230 DATA 001,141,014,220,088,096
```

Load Character Set from Tape or Disk

This can be done without any extra machine language. You can directly call the Kernal LOAD routine. Just change the OPEN statement below to the filename of your character set. Change the eight to a one for tape. And change CHSET to where you want the character set to load.

Program 2. Load Character Set

```
5000 REM CHARACTER SET LOADER
5010 OPEN 1,8,8,"FILENAME"+",P,R"
5020 CHSET=14336:REM WHERE TO LOAD
5030 POKE780,1:POKE781,8:POKE782,0:SYS 65466
5040 POKE 780,0:POKE781,0:POKE782,CHSET/256:SYS 65
     493
5050 IFPEEK(783)AND1THENPRINT"LOAD ERROR":STOP
5060 CLOSE1:RETURN
```

Raster Interrupt

You can use this program to divide the screen into two areas, each with a different character set and screen background color. With it, you can fill up a character set with graphics characters, and use the raster interrupt to let the score line use the normal

ROM characters. You can divide the screen into uppercase and lowercase areas. You can even have two different character sets in the two areas.

Use SYS 49236 to initialize, then POKE in the number you would POKE 53272 with for each character set (use 21 for the default character set, 28 for 12288) into locations 831 and 832, respectively. POKE in the background color for each area into 829 and 830, respectively. You can also choose at which screen line you want the division to occur: just POKE in 50+ (character row) *8 into location 828.

Here are the DATA statements for the raster interrupt:

Program 3. Raster Interrupt

```
49020 CK=0:FORI=49236TO49331:READA:CK=CK+A:POKEI,A
      :NEXT:IFCK=10328THENRETURN
49030 PRINT"{RVS}ERROR IN DATA STATEMENTS:CHECK TY
      PING":STOP
49236 DATA 120,169,127,141,013,220
49242 DATA 169,001,141,026,208,173
49248 DATA 060,003,141,018,208,169
49254 DATA 027,141,017,208,169,118
49260 DATA 141,020,003,169,192,141
49266 DATA 021,003,088,096,173,018
49272 DATA 208,205,060,003,208,028
49278 DATA 169,000,141,018,208,173
49284 DATA 064,003,141,024,208,173
49290 DATA 062,003,141,033,208,169
49296 DATA 001,141,025,208,104,168
49302 DATA 104,170,104,064,173,060
49308 DATA 003,141,018,208,173,061
49314 DATA 003,141,033,208,173,063
49320 DATA 003,141,024,208,169,001
49326 DATA 141,025,208,076,049,234
```

Below is a line that initializes it and POKEs in some example values. The top of the screen is black, the bottom white. The upper half is in uppercase, the lower half in lowercase. The division is set to occur at the twelfth line (12*8+50=146).

```
10 GOSUB 49020:POKE 828,146:POKE829,0:POKE830,1:PO
   KE831,21:POKE832,23:END
```

Do not execute "Copydown" (SYS 49152) while you have the raster routine enabled; disable it first. You will also want to disable the raster interrupt (POKE 53274,0) before you do tape I/O, then reenable with SYS 49236.

Also listed (Program 4) is the source code for both Program 1 and Program 3.

Program 4. Source Code

```
110:    C000                           .OPT P4
120:    C000                           *=   $C000
130:    C000                 SRC       =    $FB
140:    C000                 DEST      =    $FD
150:    C000                 TEMP      =    $A7
                             ;
170:    C000 78              COPYDOWN  SEI
170:    C001 AD 0E DC                  LDA  56334
170:    C004 29 FE                     AND  #254
170:    C006 8D 0E DC                  STA  56334
170:    C009 A5 01                     LDA  1
170:    C00B 29 FB                     AND  #251
170:    C00D 85 01                     STA  1
180:    C00F AD 18 D0                  LDA  53272
180:    C012 29 0E                     AND  #%1110
180:    C014 0A                        ASL
180:    C015 0A                        ASL
180:    C016 85 A7                     STA  TEMP
180:    C018 A9 D0                     LDA  #$D0
180:    C01A 85 FC                     STA  SRC+1
190:    C01C AD 00 DD                  LDA  56576
190:    C01F 29 03                     AND  #%11
190:    C021 49 03                     EOR  #%11
190:    C023 0A                        ASL
190:    C024 0A                        ASL
190:    C025 0A                        ASL
190:    C026 0A                        ASL
190:    C027 0A                        ASL
190:    C028 0A                        ASL
190:    C029 05 A7                     ORA  TEMP
190:    C02B 85 FE                     STA  DEST+1
200:    C02D A9 00                     LDA  #0
200:    C02F 85 FB                     STA  SRC
200:    C031 85 FD                     STA  DEST
200:    C033 A8                        TAY
200:    C034 A2 08                     LDX  #8
210:    C036 B1 FB           INLOOP    LDA  (SRC),Y
210:    C038 91 FD                     STA  (DEST),Y
210:    C03A C8                        INY
210:    C03B D0 F9                     BNE  INLOOP
220:    C03D E6 FC                     INC  SRC+1
220:    C03F E6 FE                     INC  DEST+1
220:    C041 CA                        DEX
220:    C042 D0 F2                     BNE  INLOOP
```

```
230:    C044 A5 01               LDA   1
230:    C046 09 04               ORA   #4
230:    C048 85 01               STA   1
230:    C04A AD 0E DC            LDA   56334
230:    C04D 09 01               ORA   #1
230:    C04F 8D 0E DC            STA   56334
230:    C052 58                  CLI
230:    C053 60                  RTS
                        ;
250:    C054 78         RASTER   SEI
250:    C055 A9 7F               LDA   #$7F
250:    C057 8D 0D DC            STA   $DC0D
250:    C05A A9 01               LDA   #1
250:    C05C 8D 1A D0            STA   $D01A
260:    C05F AD 3C 03            LDA   FIRST
260:    C062 8D 12 D0            STA   $D012
260:    C065 A9 1B               LDA   #27
260:    C067 8D 11 D0            STA   $D011
270:    C06A A9 76               LDA   #<IRQ
270:    C06C 8D 14 03            STA   $314
270:    C06F A9 C0               LDA   #>IRQ
270:    C071 8D 15 03            STA   $315
270:    C074 58                  CLI
270:    C075 60                  RTS
                        ;
290:    C076 AD 12 D0   IRQ      LDA   $D012
290:    C079 CD 3C 03            CMP   FIRST
290:    C07C D0 1C               BNE   TWO
300:    C07E A9 00               LDA   #0
300:    C080 8D 12 D0            STA   $D012
300:    C083 AD 40 03            LDA   SHAD2
300:    C086 8D 18 D0            STA   53272
310:    C089 AD 3E 03            LDA   COL2
310:    C08C 8D 21 D0            STA   53281
310:    C08F A9 01               LDA   #1
310:    C091 8D 19 D0            STA   $D019
320:    C094 68         IREXIT   PLA
320:    C095 A8                  TAY
320:    C096 68                  PLA
320:    C097 AA                  TAX
320:    C098 68                  PLA
320:    C099 40                  RTI
                        ;
340:    C09A AD 3C 03   TWO      LDA   FIRST
340:    C09D 8D 12 D0            STA   $D012
340:    C0A0 AD 3D 03            LDA   COL1
340:    C0A3 8D 21 D0            STA   53281
340:    C0A6 AD 3F 03            LDA   SHAD1
340:    C0A9 8D 18 D0            STA   53272
```

```
350:    CØAC A9 Ø1                  LDA   #1
350:    CØAE 8D 19 DØ               STA   $DØ19
350:    CØB1 4C 31 EA               JMP   $EA31
                          ;
370:    CØB4              FIRST     =     828
380:    CØB4              COL1      =     829
390:    CØB4              COL2      =     83Ø
400:    CØB4              SHAD1     =     831
410:    CØB4              SHAD2     =     832
```

Chapter Four

Action

Chapter Four

Sprite Editor

Stephen Meirowsky

Create and modify multicolored sprites on the Commodore 64, the easy way.

Graphics Potential

The 64 has text graphics with a 40 x 25 character format, just like the PET. Plus, it has *sprites* to use with the text graphics. These tools allow you to design your own pictures in four different colors (the manual shows how to use only one color), just like arcade videogames. Sprites can be one of 16 colors in the single-color mode, and four of eight colors in the multicolor mode.

Eight sprites are available for screen display in a 24 horizontal by 21 vertical pixel format. Each sprite has a different display hierarchy when crossing over another sprite. Sprite 0 would move in front of sprite 1; sprite 1 and sprite 0 would move in front of sprite 2, and so on up to sprite 7. All other sprites would move in front of sprite 7. Also, you can tell each sprite whether it moves in front of or behind the normal background text graphics.

Each sprite can be expanded to twice its size, horizontally, vertically, or both. Automatic collision detection tells you when sprites have hit each other or when a sprite has hit the background text graphics.

Commodore's manual gives the register number in the graphics chip which gives access to the collision information. First of all, the sprite-to-sprite collision is register 30 decimal. When sprites collide, the graphics chip sets their bits in this register. Second, the sprite to background graphics collision is register 31 decimal. When a sprite collides with the background, its bit is set.

Creating a Sprite

To make a sprite, you must first draw it on a 24 x 21 grid. Then you convert the set dots in each row into three separate bytes of data, using binary code. For each byte, add up the number according to its bit. The numbers for each bit in a byte are 128, 64, 32, 16, 8, 4, 2, 1.

Example of converting the grid:

```
Row 1 + . . . . . . +. . . . . . . + + + + + + + + +
Row 2 + . .+. . . +. . . . . . . + + + + + + + + +
Row 3  . . .+. . . +. . . . . . . + + +  . . .  + + +

101  DATA 129,1,255:REM DATA FOR ROW 1
102  DATA 145,1,255:REM DATA FOR ROW 2
103  DATA 17,1,199  :REM DATA FOR ROW 3
104  DATA
```

Next, POKE into memory the 63 bytes of data to describe the sprite to the computer. The conversion of the grid into 63 bytes is not hard, but it is very time-consuming. This is the reason for "Sprite Editor."

The Easy Way

Sprite Editor gives many easy, single-key commands to edit the sprite, display it, and save it. When the program is executed, commands are printed along the left side of the screen. On the right side of the screen is a 24 x 21 grid which is used to edit a sprite. To move the cursor, use the cursor keys. If you want a pixel set on the sprite, press the 1, 2, or 3 key. If you want the pixel erased, press the ← key. Anytime you want to see the actual sprite, press the = key, and it will compute the grid into the byte form and display the sprite in the lower-left corner of the screen.

If you make any updates on the grid, they will not be displayed in the corner until the = key is pressed again. Once the sprite has been displayed, it can be enlarged horizontally or vertically by pressing X or Y. Also, you can display the data for using this sprite in a program by pressing B.

On all four of the following commands, the computer will ask if it is the correct command to be executed. The four commands are N for erasing the grid and the sprite to edit a new sprite; S for saving sprite data to cassette; L for loading a sprite from cassette; and Q for quitting the program.

It is a good idea to compute the sprite (press =) before displaying data (press B) and saving a sprite (press S) to be sure the sprite has been updated.

To change colors while creating a sprite, use the f1, f3, f5, and f7 keys.

Disk or Tape

The program as written is set up for use with a tape. To SAVE a sprite on disk, it is necessary to change lines 196 and 200 as indicated in the REM statements on lines 196 and 201.

Sprite Editor

```
10 POKE53281,6:DIM A(21,24),B(63),A$(15) :X=Ø:Y=Ø:
   R=Ø:C=Ø:S=1Ø39:S1=55311
11 V=53248:POKEV+21,Ø:POKEV+23,Ø:POKEV+29,Ø:RESTOR
   E:FORX=ØTO15:READA$(X):NEXT
12 PRINT"{CLR}":FORR=1 TO 21:FOR C=1 TO 24:A(R,C)=
   46:NEXT:NEXT
13 FOR X=1 TO 63:B(X)=Ø:NEXT
14 POKEV+4,6Ø:POKEV+5,2ØØ:POKE2Ø42,13:POKEV+37,Ø:P
   OKEV+41,14:POKEV+38,1
16 FORX=1TO63:POKE831+X,B(X):NEXT:POKEV+21,4:POKE
   {SPACE}V+28,4
2Ø PRINT"{CLR}{DOWN}⁅7⁆MC SPRITE EDITOR{DOWN}"
22 PRINT"← ERASE"
23 PRINT"1 MC Ø-"A$(PEEK(V+37) AND 15)
24 PRINT"2 SC{2 SPACES}-"A$(PEEK(V+41) AND 15)
25 PRINT"3 MC 1-"A$(PEEK(V+38) AND 15)
32 PRINT"= COMPUTE SPRITE"
33 PRINT"X SCALE 'X'"
34 PRINT"Y SCALE 'Y'"
35 PRINT"B BASIC DATA"
36 PRINT"N NEW SCREEN"
37 PRINT"S SAVE SPRITE"
38 PRINT"L LOAD SPRITE"
39 PRINT"Q QUIT{DOWN}"
5Ø Y=Ø:FORR=1TO21:FORC=1TO24:Y=Y+1:POKES+Y,A(R,C):
   POKES1+Y,14:NEXT:Y=Y+16:NEXT
55 X=1:Y=1:GOTO79
6Ø GETA$:IFA$=""THEN6Ø
61 R=S+X+(Y-1)*4Ø:C=A(Y,X):POKER,C:POKER+1,C
62 IFA$="{DOWN}"THENY=Y+1:IFY>21THENY=1
63 IFA$="{UP}"THENY=Y-1:IFY<1THENY=21
64 IFA$="{RIGHT}"THENX=X+2:IFX>24THENX=1
65 IFA$="{LEFT}"THENX=X-2:IFX<1THENX=23
66 IFA$="←"THENA(Y,X)=46:A(X,Y+1)=46
67 IFA$>"Ø"AND A$<"4"THENR=48+VAL(A$):A(Y,X)=R:A(Y
   ,X+1)=R
68 IFA$="="THEN1ØØ
69 IFA$="X"THENPOKEV+29,ABS(PEEK(V+29)-4)
7Ø IFA$="Y"THENPOKEV+23,ABS(PEEK(V+23)-4)
71 IFA$="B"THEN12Ø
72 IFA$="L"ORA$="S"ORA$="N"ORA$="Q"THEN19Ø
73 IF A$="{F1}"THENR=33:GOSUB 13Ø
74 IF A$="{F3}"THENR=37:GOSUB 13Ø
75 IF A$="{F5}"THENR=41:GOSUB 13Ø
```

```
76 IF A$="{F7}"THENR=38:GOSUB 130
79 R=S+X+(Y-1)*40:C=A(Y,X)+128:POKER,C:POKER+1,C:G
   OTO60
100 Y=0:FORR=1TO21:FORX=0TO2:Y=Y+1:B(Y)=0:FORC=1TO
    7STEP2:Q=A(R,X*8+C)-48
102 IFQ<0 OR Q>3 THEN Q=0
104 B(Y)=B(Y)+2↑(7-C)*Q:NEXT:NEXT:NEXT:FORX=1TO63:
    POKE831+X,B(X):NEXT:GOTO55
110 PRINT"{RVS}"A$": YES OR NO"
111 FORX=1TO10:GETN$:NEXT
112 GETN$:IFN$=""THEN112
114 PRINT"{UP}{16 SPACES}{UP}":RETURN
115 PRINT"{RVS}CONTINUE":GOTO111
119 REM
120 PRINT"{CLR}":FORX=1TO7:PRINT"DATA";:FORY=1TO9:
    PRINTB((X-1)*9+Y)"{LEFT},";:NEXT
122 PRINT"{LEFT} ":NEXT:PRINT:GOSUB115:GOTO20
130 C=PEEK(V+R)AND15:C=C+1:IF C>15 THEN C=0
132 POKE V+R,C:PRINT"{HOME}{3 DOWN}";:IFR=33 THEN
    {SPACE}136
133 PRINT"{DOWN}";:IF R=37 THEN 136
134 PRINT"{DOWN}";:IF R=41 THEN 136
135 PRINT"{DOWN}";
136 PRINT"{7 RIGHT}"A$(C)"{2 SPACES}":RETURN
190 GOSUB110:IFN$<>"Y"THEN79
191 GETN$:GETN$:IFA$="N"THEN11
192 IFA$="Q"THENPOKEV+21,0:PRINT"{4 DOWN}":END
194 PRINT"{CLR}":POKEV+21,0:INPUT"NAME OF SPRITE";
    N$:PRINT
196 IFA$="L"THENOPEN1,1,0,N$:GOTO300:REM DISK USER
    S OPEN 1,8,2,N$
200 OPEN1,1,1,N$:FORX=1TO63:PRINT#1,B(X):NEXT:CLOS
    E1:GOTO16
201 REM DISK USERS OPEN 1,8,2,N$+",S,W" ON LINE 20
    0
300 FORX=1TO63:INPUT#1,B(X):NEXT:CLOSE1:PRINT"
    {DOWN}COMPUTING SPRITE MATRIX"
310 Y=0:FORR=1TO21:FORX=0TO2:Y=Y+1:FORC=2TO8STEP2:
    Q=X*8+C:P=2↑(8-C)
312 S=B(Y)AND(P*3):A(R,Q)=46:A(R,Q-1)=46
314 IF S>0 THEN A(R,Q)=S/P+48:A(R,Q-1)=S/P+48
330 NEXT:NEXT:NEXT:S=1039:GOTO16
500 DATA BLACK,WHITE,RED,CYAN,PURPLE,GREEN,BLUE,YE
    LLOW
510 DATA ORANGE,BROWN,LT RED,GRAY1,GRAY2,LT GRN,LT
     BLUE,GRAY3
```

Creating Sprite Animation

Eric Brandon

Presented here is a detailed discussion of how the author, using sprites, was able to create a realistic simulation of a space shuttle takeoff. Many useful animation techniques are explained.

After going to Florida to see the space shuttle Challenger take off, I wrote a short program to show people what it looked like. By coincidence, that same day I was writing a Commodore 64 version of Matt Giwer's game "Moving Maze." Thus "Shuttle Escape" was born.

The program you see here is the title page of Shuttle Escape. It is essentially my original program, polished up a bit for the game. Many interesting techniques are used in this program, and can be easily adapted to any animation or cartoon programming project.

First Things First

The first thing the program does is GOSUB to line 3000, where it executes a subroutine that prints the words SHUTTLE ESCAPE on the screen in large letters. Then, in line 110, it prints a CHR$(142) to insure that the display is in Graphics mode and not upper/lowercase.

Line 120 checks to see if the sprite data is already in memory. If it is not, it goes to a subroutine at line 10000 that POKEs in the sprite data.

The Reasons for All Those Numbers

Lines 10000 through 10026 are a simple loop to READ the values in the DATA statements and POKE them in. In line 10000 the internal clock, TI$, is reset to zero. In line 10005, a 39-second countdown is displayed by subtracting TI/60 from the number 39 and displaying it in the middle of a message. TI$ contains the time in HHMMSS (hours, minutes, seconds) format, but TI tells the same time in sixtieths of a second, called jiffies. You may wonder how I knew beforehand that the countdown would last 39 seconds. The answer is that I didn't. I tried different values until I found one that made the countdown end with 0 seconds on the clock. Much of this sort of programming is tinkering with the data until it looks right. The checksum on line 10025 was arrived at by writing a short program to READ and add the contents of the DATA statements. The reason for the checksum is to let you know if you've made a typing error.

In line 10030 begin the DATA statements that contain the shape of all the sprites. The shape of each sprite consists of 64 numbers. Each sprite has a width of 24 pixels and a height of 21. This is 504 dots. Each dot is represented by one bit; 504 bits divided by 8 bits per byte = 63 bytes. A 64th byte is required to pad out the sprite definition. The shapes, in the order that they appear in the DATA statements, are:

Page	**Shape**
244	Shuttle in horizontal position.
245	Shuttle in vertical position with bottom part of fuel tank.
246	Cap of fuel tank.
247	Small flame.
248	Larger flame.
249	Shuttle in vertical position without fuel tank.
250	Bottom of fuel tank (without shuttle).
251	Partially disintegrated fuel tank.
252	More disintegrated fuel tank.
253	Partially disintegrated fuel tank cap.
254	Different shape, same size as page 248.

What do I mean by *page*? Each sprite has a pointer at memory locations 2040-2047 that tells it where to find the data that holds its shape. The number you put in this pointer is its *page* number. Each page is 64 bytes long, so page 244 starts at memory location 244*64, or 15616. The reason for having so many different sprites will become apparent as we walk through the program.

Now, What Do I Do with These Sprites?

On line 130 the variable V is set equal to 13*4096 ($D000 and 53248). V points to the VIC-II video chip where all the action on the screen takes place. CO (column) is set equal to 50. The CO function is to offset all the X positions of the various sprites by 50 pixels. By putting this value in a variable, I can move the whole takeoff sequence right or left on the screen as needed when more things are added to the title page.

At this point, I had to decide which sprite would display which image. I arbitrarily chose sprite 0 for the shuttle, 1 for the fuel tank, 2 for the flame, and 3 for the cap of the fuel tank. The reason the cap came after the flame was that I already had a flame in the picture before I decided to elongate the stubby fuel tank.

This takes us to line 140, where I begin a wanton POKEfest. POKE V + 16,0 clears bit 9 of the X positions of all the sprites. This way we can be sure that part of our takeoff won't be happening independently on the right edge of the screen. To get a better idea of what this means, try changing this line by POKEing values other than 0 into V + 16.

Line 160 sets the X positions of the shuttle and its attached fuel tank to CO. The X position of the flame at the base of the shuttle is set at CO – 2. POKE V + 5,221 sets the vertical position of the flame. The next line sets the Y position of the shuttle/tank assembly at 200. The flame is at 221, and 221 – 200 = 21, which is the height of a sprite. Many of the sprites in this program are 21 pixels apart from each other in the Y direction for the same reasons.

Lines 180 and 190 set the cap of the tank just above the shuttle. CO is its X position, and its Y position is 200 – 21 = 179.

Line 210 sets the color of sprites 0, 1, and 3 to white, color 1. It also sets the color of sprite 2 to orange, color 8.

In lines 220 to 240, the page values are POKEd into the sprite data pointers. Note how the page values correspond with the table above.

In line 250, we have a short delay, and then with POKE V + 21,7, we put three sprites on the screen. The contents of V + 21 tells the VIC-II chip which sprites to display. Whenever a bit of that byte is set, a sprite will appear. Since 7 is 00000111 in binary, this POKE turns on sprites 0, 1, and 2.

A Bit of Noise

Line 260 sends us to the sound subroutine at line 2000. Here S is initialized to 54272. S contains the starting address of the sound chip. In line 2010, the volume is set to maximum, and both the Low Pass and High Pass filters are set. This means that the higher and lower frequencies pass unaltered through the SID (sound chip), but that the midrange is attenuated. How do I set these filters? Easy. POKE S + 24,15 sets the volume to maximum and does nothing else. Adding 16 sets bit 4, and adding 32 sets bit 5. These two bits control the filters.

The next POKE, to S + 23, has two effects. Setting bit 0 sends the output of voice 1 through the filters. Putting a five in the upper 4 bits sets the resonance to 5. Resonance determines the sharpness of the sound at the cutoff point of the filters.

The attack and decay are set to 0 in line 2020, for a sound that begins instantaneously. Line 2030 sets the sustain and decay to their maximum values. In line 2040 the noise waveform is turned on, and finally in line 2050 the high byte of the frequency is set to 11.

Sound is undoubtedly the most difficult part of a program such as this one, but by intelligent trial and error, you should be able to get some good sounds. For example, if you don't have a clue what effect a filter would produce on your sound, turn it on and listen. If you don't like the results, you can always change it back. (The programs in Chapter 4 will help you experiment with sound.)

We Have Ignition!

I is the vertical position of the shuttle and is set in line 270. Space shuttles accelerate as they leave the ground; therefore, the vertical position of the shuttle must steadily subtract greater numbers. The first thing that comes to mind is to use a parabolic equation (the velocity is proportional to the square of how many times the loop has run). When I tried this, the shuttle did not seem to accelerate fast enough, so I added P, a third order coefficient. What this all means in English is that Q is the speed with which the shuttle moves. This speed is increased by adding .01*P to it over and over, and P is increased to account for the steadily reducing weight of the shuttle as it burns off fuel.

C in line 300 is a counter. It tells us how far the program has gone so far, and when to start the various stages of the takeoff.

Lines 320 and 330 alternate the large flame between the two slightly different images at page 248 and page 254. Since at the start of the program the flame is the smaller image at page 247, these lines have no effect until later on in the program. They do illustrate one important feature of sprite animation, however. By changing what the sprite pointer is looking at, you can instantly, and with almost no program overhead, change the shape of an object on the screen. This is used over and over in this program and in any program of this type.

Line 340 sets the vertical position of the shuttle, fuel cap, and flame. They are all in relation to the same variable, I, so that by changing the value of one variable, you can change the position of all three sprites.

Lines 350 to 360 dynamically change the filter cutoff frequency. This gives the sound a sort of roar which is much more realistic than just a steady buzz. In line 360, P2, the cutoff frequency, is increased.

Lines 370-390 control the flame. Register V + 23 controls whether sprites are double sized in the Y direction. By putting a 4 in this register, we double the vertical length of sprite 2, the flame. The first line executed is 380, when the counter reaches a value of 20. This line doubles the size of the small sprite. The next line, 390, will execute when the counter reaches 40. This line turns off the Y expansion, but changes the sprite pointer to one of the larger flames. From now on, lines 320 and 330 will alternate between the two larger flames. Line 370 will finally execute when the counter has reached 60, and will double the size of the larger flames. With very little trouble, we have managed to display six different flame shapes.

Line 400 keeps the loop executing as long as the counter is less than 70.

We Have Separation!

After the counter reaches 70, it is time for the fuel tank to fall off. By now some of you must be wondering where the booster rockets are. They were not put in because the takeoff looked good enough as it was.

If the tank and the shuttle are to go their separate ways, they must become two different sprites. This is why, in lines 410 and 420, the shuttle takes on the form of page 249, and the tank looks like page 250. The X and Y coordinates of the tank (sprite 3) are set, and then with POKE V + 21,15 we add sprite 3 to the others that are already visible.

Lines 430 through 470 are clones of lines 290 through 330 and serve exactly the same function in this new loop. The only difference is that a new variable C2 has been introduced. This variable, which has 0.6 added to it each iteration of the loop, is used to calculate the position of the falling fuel tank.

Line 490 calculates the position of the falling fuel tank. The Y position is in NR (new row) and is calculated by I + C2*C2. The reason for this formula is that when C2 is small, I is steadily decreasing, and the fuel tank rises. As C2 increases, it begins to dominate and the tank falls. This gives the illusion that the tank is being carried up a bit by its momentum, and then pulled down increasingly fast by gravity. The X position of the tank, NC (new column), is calculated simply as a multiple of the counter C2 added to CO, the base column.

Line 500 POKEs in the X and Y position of the fuel tank and the fuel tank cap.

Lines 510 through 530 control the disintegration of the fuel tank. When the Challenger releases its fuel tank, it falls a few miles and then burns up in the atmosphere. This program simulates this by placing the page numbers of more and more disintegrated tanks in the pointer for sprite 3. Line 510 changes both the tank and tank cap pointers when the counter, C, reaches 83. When C is 86, line 520 changes the tank pointer again. The cap does not change further, but since it is small this does not degrade the effect. Finally, when the counter reaches 89, line 530 does a POKE V + 21,5, leaving only sprites 0 and 2 on the screen, eliminating the fuel tank altogether.

The sound begins to decay with line 570; it will take 24 seconds to reach minimum volume. This gives an audio effect of the shuttle receding into the distance even after it is no longer on the screen.

Orbit Achieved

Now the shuttle crosses the words SHUTTLE ESCAPE as a symbol that orbit has been achieved.

Line 600 POKEs V + 21 with 1, leaving only sprite 0 potentially visible.

Line 610 is a loop giving the sound some time to die away.

Line 620 PRINTs "Orbit achieved . . ." across the top of the screen to avoid confusion about what is happening.

Line 640 POKEs the sprite pointer to page 244, the horizontal shuttle.

Line 650 sets the shuttle to the left edge of the screen and 117 pixels down.

Lines 660 through 680 move the shuttle across the screen two pixels at a time. In line 670 we have to deal for the first time with the "sprite seam" at X location 255. Since each byte can only hold a value between 0 and 255, yet there are 320 pixels across the screen, the X position of each sprite must be held in more than one byte. At V + 16, each sprite has a bit which when set makes its X position based not at the left edge of the screen, but offset by 256 pixels to the right. Line 670 takes care of that by only POKEing the lower eight bits of I into V, and POKEing the ninth bit into V + 16.

Your Turn

While the program is complex, the techniques it uses are simple and applicable in your own programs.

The most important thing to be learned from this program is that cartoons don't have to follow the real laws of physics to look realistic; the equations can be simplified. Another important point is that almost all effects will look wrong when first programmed. They must be pushed and prodded until they look like you think they should.

Shuttle Escape

```
100 GOSUB3000
110 PRINTCHR$(142)
120 IF PEEK(15625)<>24 THEN GOSUB 10000
130 V=13*4096:CO=50
140 POKE V+16,0
160 POKE V+0,CO:POKEV+4,CO-2:POKEV+5,221
170 POKE V+1,200
180 POKE V+2,CO
190 POKE V+3,179
210 POKE V+39,1:POKEV+40,1:POKEV+41,8:POKEV+42,1
220 POKE 2040,245
230 POKE 2041,246
240 POKE 2042,247
250 FOR K=1 TO 500 : NEXT K:POKEV+21,7
260 GOSUB 2000
270 I=200
280 P=1
290 Q=Q+.01*P
300 P=P+.1:C=C+1
310 I=I-Q
320 IF PEEK(2042)=248 THEN POKE 2042,254:GOTO340
330 IF PEEK(2042)=254 THEN POKE 2042,248
340 POKE V+1,I:POKEV+3,I-21:POKEV+5,I+21
350 POKES+22,P2:POKES+23,1OR(16-P2/16)*16
360 P2=P2+P2/244
370 IF C=60THEN POKEV+23,4
380 IF C=20THEN POKEV+23,4
390 IF C=40 THEN POKEV+23,0:POKE2042,248
```

```
400 IF C<70 THEN 290
410 POKE 2040,249
420 POKE 2043,250:POKEV+6,CO:POKEV+7,I:POKEV+21,15
430 Q=Q+.01*P
440 P=P+.1:C=C+1:C2=C2+.6
450 I=I-Q
460 IF PEEK(2042)=248 THEN POKE 2042,254:GOTO480
470 IF PEEK(2042)=254 THEN POKE 2042,248
480 POKE V+1,I:POKEV+5,I+21
490 NR=I+C2*C2:NC=CO+C2*3
500 POKE V+7,NR:POKEV+3,NR-21:POKEV+6,NC:POKEV+2,N
    C
510 IF C=83 THEN POKE 2043,251:POKE2041,253
520 IF C=86 THEN POKE 2043,252
530 IF C=89 THEN POKE V+21,5
540 POKES+22,P2:POKES+23,1OR(16-P2/16)*16
550 P2=P2+P2/244
560 IF I>25 THEN 430
570 POKE S+4,128
580 POKE V+5,I+21
590 I=I-2:IFI>0 THEN580
600 POKE V+21,1
610 FOR J=1 TO 2000:NEXT
620 PRINT"{HOME}{10 RIGHT}{WHT}{2 SPACES}ORBIT ACH
    IEVED...."
630 FOR I=1 TO 1000:NEXT
640 POKE 2040,244
650 POKE V,0:POKEV+1,117
660 FOR I=0 TO 348 STEP2
670 POKE V,I AND 255:POKEV+16,I/255
680 NEXT
690 FOR I=0 TO 1000:NEXT
720 END
2000 S=54272
2010 POKES+24,15+16+32:POKES+23,1+16*5
2020 POKES+5,0
2030 POKES+6,16*15+15
2040 POKES+4,129
2050 POKES+1,11
2060 P2=100:RETURN
3000 POKE 53281,0:POKE53280,0
3010 PRINT"{CLR}"
3020 PRINT"{5 DOWN}"
3040 T=12
3050 PRINTTAB(T)"{7}{RVS}£{2 SPACES}{RIGHT}
     {RIGHT} {RIGHT} {RIGHT} {RIGHT}{3 SPACES}
     {RIGHT}{3 SPACES}{RIGHT} {3 RIGHT}£
     {2 SPACES}"
```

```
3060 PRINTTAB(T)"{RVS} {3 RIGHT} {RIGHT} {RIGHT}
     {RIGHT} {2 RIGHT} {3 RIGHT} {2 RIGHT}
     {3 RIGHT} "
3070 PRINTTAB(T)"E*3{RVS} E*3{RIGHT}{3 SPACES}
     {RIGHT} {RIGHT} {2 RIGHT} {3 RIGHT} {2 RIGHT}
     {3 RIGHT}{2 SPACES}"
3080 PRINTTAB(T)"{RVS}{2 RIGHT} {RIGHT} {RIGHT}
     {RIGHT} {RIGHT} {2 RIGHT} {3 RIGHT} {2 RIGHT}
     {3 RIGHT} "
3090 PRINTTAB(T)"{RVS}{2 SPACES}{OFF}£{RVS}
     {RIGHT} {RIGHT} {RIGHT}{OFF}E*3{RVS} {OFF}
     £{RVS}{2 RIGHT} {3 RIGHT} {2 RIGHT}{OFF}
     E*3{RVS}{2 SPACES}{RIGHT}{OFF}E*3{RVS}
     {2 SPACES}"
3100 PRINT
3110 PRINTTAB(T)"{CYN}{RVS}£{2 SPACES}{RIGHT}£
     {2 SPACES}{RIGHT}£{2 SPACES}{RIGHT}£ E*3
     {RIGHT}{2 SPACES}E*3{RIGHT}£{2 SPACES}"
3120 PRINTTAB(T)"{RVS} {3 RIGHT} {3 RIGHT}
     {3 RIGHT} {RIGHT} {RIGHT} {RIGHT} {RIGHT} "
3130 PRINTTAB(T)"{RVS}{2 SPACES}{2 RIGHT}{OFF}
     E*3{RVS} E*3{RIGHT} {3 RIGHT}{3 SPACES}
     {RIGHT}{2 SPACES}{OFF}£{RIGHT}{RVS}
     {2 SPACES}"
3140 PRINTTAB(T)"{RVS} {5 RIGHT} {RIGHT} {3 RIGHT}
     {RIGHT} {RIGHT} {3 RIGHT} "
3150 PRINTTAB(T)"E*3{RVS}{2 SPACES}{RIGHT}
     {2 SPACES}{OFF}£ E*3{RVS}{2 SPACES}{RIGHT}
     {RIGHT} {RIGHT} {3 RIGHT}{OFF}E*3{RVS}
     {2 SPACES}"
3999 RETURN
10000 I=15616:TI$="000000"
10005 PRINT"{HOME}{WHT}{12 RIGHT}READY IN"LEFT$(ST
      R$(39-INT(TI/60)),4)" SECONDS "
10010 READ A:IF A=256 THEN PRINT"{HOME}{12 RIGHT}
      {20 SPACES}" :GOTO10025
10020 C1=C1+A:POKE I,A:I=I+1:GOTO 10005
10025 IF C1<>30584 THEN PRINT"CHECKSUM ERROR IN LI
      NE 10025":END
10026 RETURN
10030 DATA 0,0,0,0,0,0,0
10040 DATA 0,0,24,0,0,28,0
10050 DATA 0,31,0,0,31,255,240
10060 DATA 31,255,8,20,255,254,31
10070 DATA 127,255,30,63,254,24,0
10080 DATA 0,0,0,0,0,0,0
10090 DATA 0,0,0,0,0,0,0
10100 DATA 0,0,0,0,0,0,0
10110 DATA 0,0,0,0,0,0,0
```

```
10120 DATA 0,0,71,192,0,247,192
10130 DATA 0,247,192,1,255,192,2
10140 DATA 255,192,2,255,192,2,247
10150 DATA 192,2,247,192,3,247,192
10160 DATA 3,247,192,3,247,192,3
10170 DATA 247,192,3,247,192,3,247
10180 DATA 192,3,255,192,3,255,192
10190 DATA 7,103,192,7,103,192,15
10200 DATA 229,128,31,119,128,31,240
10210 DATA 0,0,0,0,0,0,0
10220 DATA 0,0,0,0,0,0,0
10230 DATA 0,0,0,0,0,0,0
10240 DATA 0,0,0,0,0,0,0
10250 DATA 0,0,0,0,0,0,0
10260 DATA 0,0,0,0,0,0,0
10270 DATA 0,0,0,0,0,0,0
10280 DATA 0,0,3,128,0,15,192
10290 DATA 0,15,192,0,15,192,0
10300 DATA 15,192,0,1,252,0,1
10310 DATA 116,0,1,212,0,0,88
10320 DATA 0,0,80,0,0,0,0
10330 DATA 0,0,0,0,0,0,0
10340 DATA 0,0,0,0,0,0,0
10350 DATA 0,0,0,0,0,0,0
10360 DATA 0,0,0,0,0,0,0
10370 DATA 0,0,0,0,0,0,0
10380 DATA 0,0,0,0,0,0,0
10390 DATA 0,0,0,0,1,252,0
10400 DATA 1,252,0,1,252,0,1
10410 DATA 254,0,7,248,0,6,249
10420 DATA 0,2,251,0,6,122,0
10430 DATA 3,242,0,0,248,0,0
10440 DATA 248,0,0,60,0,0,120
10450 DATA 0,0,56,0,0,56,0
10460 DATA 0,96,0,0,96,0,0
10470 DATA 8,0,0,32,0,0,0
10480 DATA 0,0,0,0,0,0,64

10490 DATA 0,0,240,0,0,240,0
10500 DATA 1,240,0,2,240,0,2
10510 DATA 240,0,2,240,0,2,240
10520 DATA 0,3,240,0,3,240,0
10530 DATA 3,240,0,3,240,0,3
10540 DATA 240,0,3,240,0,3,240
10550 DATA 0,3,240,0,7,96,0
10560 DATA 7,96,0,15,224,0,31
10570 DATA 112,0,31,240,0,0,0
10580 DATA 7,192,0,7,192,0,7
10590 DATA 192,0,7,192,0,7,192
```

```
10600 DATA 0,7,192,0,7,192,0
10610 DATA 7,192,0,7,192,0,7
10620 DATA 192,0,7,192,0,7,192
10630 DATA 0,7,192,0,7,192,0
10640 DATA 7,192,0,7,192,0,7
10650 DATA 192,0,7,192,0,7,192
10660 DATA 0,3,128,0,0,0,0
10670 DATA 0,2,0,0,7,192,0
10680 DATA 7,192,0,6,192,0,4
10690 DATA 192,0,3,64,0,6,192
10700 DATA 0,1,192,0,4,0,0
10710 DATA 7,192,0,7,128,0,7
10720 DATA 64,0,7,192,0,1,192
10730 DATA 0,5,192,0,6,64,0
10740 DATA 7,192,0,7,192,0,0
10750 DATA 128,0,3,128,0,0,0
10760 DATA 0,0,2,0,0,1,0
10770 DATA 0,6,64,0,0,64,0
10780 DATA 4,128,0,3,64,0,6
10790 DATA 0,0,1,0,0,0,0
10800 DATA 0,0,0,0,0,128,0
10810 DATA 1,64,0,6,0,0,1
10820 DATA 0,0,5,0,0,6,64
10830 DATA 0,0,0,0,4,0,0
10840 DATA 0,128,0,3,128,0,0
10850 DATA 0,0,0,0,0,0,0
10860 DATA 0,0,0,0,0,0,0
10870 DATA 0,0,0,0,0,0,0
10880 DATA 0,0,0,0,0,0,0
10890 DATA 0,0,0,0,0,0,0
10900 DATA 0,0,0,0,0,0,0
10910 DATA 0,0,0,0,0,0,0
10920 DATA 0,0,1,128,0,6,128
10930 DATA 0,2,64,0,5,192,0
10940 DATA 3,128,0,1,252,0,1
10950 DATA 252,0,1,236,0,1,126
10960 DATA 0,3,248,0,2,120,0
10970 DATA 0,248,0,0,120,0,0
10980 DATA 112,0,0,120,0,0,120
10990 DATA 0,0,48,0,0,48,0
11000 DATA 0,0,0,0,0,0,0
11010 DATA 0,0,0,0,0,0,0
11020 DATA 0,0,0,0,0,0,0
11030 DATA 0,0,0,0,256
```

Chapter Four

Moving Maze

Matt Giwer   Commodore 64 version by Eric Brandon

"Moving Maze" is a different kind of maze game: the walls keep moving. This challenging game illustrates the use of sprites in a game. Also included is an explanation of how to combine the "Shuttle Escape" program with the game Moving Maze.

The objective of "Moving Maze" is to move the spaceship from the left side of the screen to the right side. You begin with 2000 fuel units which you lose at the rate of 60 units each second whether the shuttle is moving or not. If you touch a wall or one of the roving droids, you lose 100 units each 1/60 second. When you have run out of fuel, the game is over. Fortunately, you can refill your tanks by reaching the right-hand side of the screen.

If you want to stop the game for a moment, just hold down the SHIFT key. If you want to stop the game for a longer period of time, use SHIFT LOCK, but be careful — it's just beside the RUN/STOP key.

You can speed up the movement of the walls by holding down the fire button on the joystick. This won't make gaps appear any sooner, but it will speed up any gaps that are already there. The penalty is that while the fire button is down, your fuel disappears twice as fast.

Programming Moving Maze revealed some interesting problems. The first is that "sparkle" (little specks of snow) appears on the screen. Usually this causes no difficulty, but when you try to use the VIC-II's sprite-background collision detection register, it turns out that sprites can collide with sparkle.

What this meant to Moving Maze was that occasionally, for no apparent reason, the shuttle would collide and you would lose 100 fuel units. Since moving the character set eliminates sparkle, it was relocated to $3000.

Another quirk of the 64 is that the VIC-II chip can look at only 16K of memory at a time. When you turn on your machine, it is looking at the first 16K block from $0000-$3FFF. It was decided to leave it there for simplicity. This meant that the sprite data, the relocated character set, and the entire BASIC program all had to be squeezed into 16K. Because of this memory limitation, when the machine language creates a character set at $3000, it destroys the DATA statements in the program. Fortunately, the DATA statements are no longer needed since they have already been POKEd into memory.

Because RUNning the program destroys it, be extra sure that when you type it in, you SAVE it before you try to RUN.

Typing in the Program

Moving Maze will run as a game if Program 1 is entered correctly. Because RUNning the program destroys it, be extra sure when you type the program in that you SAVE it before RUNning it.

If you wish to combine the game of Moving Maze with the very impressive title screen from "Shuttle Escape" to form one program, follow the procedure below.

1. Type in the *entire* listing of Shuttle Escape.
2. From Program 1, Moving Maze, type in lines 4000 to 4210, and lines 11040 to 52010.
3. Type in Program 2.
4. SAVE your program.

Program 1. Moving Maze

```
50 POKE53281,0:POKE53280,0:PRINT"{CLR}"
110 PRINTCHR$(142)
120 IF PEEK(16378)<>16 THEN GOSUB 10000:GOSUB 5000
    0
2000 S=54272
2010 POKES+24,15+16+32:POKES+23,1+16*5
2020 POKES+5,7*16
2030 POKES+6,249
2050 POKES+1,11
4000 V=13*4096
4005 POKE 2034,1:POKE2044,1:POKE2054,1
4010 POKE V+21,0
4020 POKES+4,128
4030 FOR I=1 TO 6
4040 POKE V+39+I,7+4*(INT(I/2)<>I/2): POKE V+2*I,(
     36+40*I)AND255:NEXT
4050 POKE V+16,64:POKE 2040,254:POKEV,30:POKEV+1,1
     48
4060 FOR I=2041 TO 2047:POKEI,255:NEXT
4065 POKE V+21,127
4070 PRINT"{CYN}{CLR}FUEL
4080 PRINT"02000"
```

```
4090 PRINT"SCORE:"
4100 PRINT"00000"
4110 P(0)=1029:P(4)=1994:P(1)=1039:P(5)=2004:P(2)=
     1049:P(6)=2014:P(3)=1059
4120 SYS 49152
4130 POKE P(0),227
4140 IF PEEK(2)=255 THEN 20000
4150 IF PEEK(653)=1 THEN 4150
4160 IF RND(1)>.05 THEN 4140
4170 IF RND(1)>.5 THEN 4200
4180 P=RND(1)*5:IF PEEK(P(P))<>160 THEN 4180
4190 POKE P(P),227:GOTO4140
4200 P=RND(1)*3+4:IF PEEK(P(P))<>160 THEN 4200
4210 POKE P(P),228:GOTO4140
10000 I=16256:TI$="000000"
10005 PRINT"{HOME}{WHT}{12 RIGHT}READY IN"LEFT$(ST
      R$(93-INT(TI/60)),4)" SECONDS "
10010 READ A:IF A=256 THEN GOTO10025
10020 C1=C1+A:POKE I,A:I=I+1:GOTO 10005
10025 IF C1<>6062 THEN PRINT"CHECKSUM ERROR IN LIN
      E 10025":END
10026 RETURN
10030 DATA 0,0,0,0,0,0,0
10040 DATA 0,0,24,0,0,28,0
10050 DATA 0,31,0,0,31,255,240
10060 DATA 31,255,8,20,255,254,31
10070 DATA 127,255,30,63,254,24,0
10080 DATA 0,0,0,0,0,0,0
10090 DATA 0,0,0,0,0,0,0
10100 DATA 0,0,0,0,0,0,0
10110 DATA 0,0,0,0,0,0,0,0
11030 DATA 0,0,0
11040 DATA 0,16,0,0,16,0,16
11050 DATA 56,16,10,16,160,4,16
11060 DATA 64,10,124,160,1,255,0
11070 DATA 1,255,0,11,255,144,127
11080 DATA 255,252,11,255,144,1,255
11090 DATA 0,1,255,0,10,124,160
11100 DATA 4,16,64,10,16,160,16
11110 DATA 56,16,0,16,0,0,16
11120 DATA 0,0,0,0,0,0,256
20000 SC=0:FOR I=0 TO 4:SC=SC+(PEEK(1148-I)-48)*10
      ↑I:NEXT I
20010 IF H<SC THEN H=SC
20020 POKE S+4,128
20030 POKE 13*4096+21,0
20040 FOR I=1 TO 1000:NEXT I
20050 PRINT"{CLR}OUT OF FUEL...{DOWN}
20060 PRINT"YOU SCORED{WHT}"SC"{CYN}POINTS
```

```
20070 PRINT"HIGH SCORE{WHT}"H"{CYN}
20080 PRINT"{3 DOWN}{11 SPACES}AGAIN? (Y OR N)"
20090 PRINT"{DOWN} OR PRESS FIRE BUTTON TO START A
      GAIN"
20100 GETA$
20110 IF A$="N"THEN END
20120 IF (PEEK(56320) AND 16)=0 THEN GOTO4000
20130 IF A$<>"Y" THEN 20100
20140 GOTO4000
50000 I=49152:TI$="000000"
50010 PRINT"{HOME}{WHT}{12 RIGHT}READY IN"LEFT$(ST
      R$(86-INT(TI/60)),4)" SECONDS "
50015 READ A:IF A=256 THEN PRINT"{HOME}{10 RIGHT}
      {21 SPACES}{SHIFT-SPACE}":GOTO50045
50020 IF A=-1 THEN I=49920 : GOTO 50010
50030 IF A=-2 THEN I=50688 : GOTO 50010
50040 C2=C2+A:POKE I,A:I=I+1:GOTO 50010
50045 IF C2<>188431 THEN PRINT"CHECKSUM ERROR IN L
      INE 50045":END
50046 RETURN
50050 DATA 120,169,0,141,20,3,169
50060 DATA 195,141,21,3,88,173,14
50070 DATA 220,41,254,141,14,220,165
50080 DATA 1,41,251,133,1,160,0
50090 DATA 185,0,208,153,0,48,185
50100 DATA 0,50,153,0,50,185,0
50110 DATA 209,153,0,49,185,0,211
50120 DATA 153,0,51,185,0,212,153
50130 DATA 0,52,185,0,213,153,0
50140 DATA 53,185,0,214,153,0,54
50150 DATA 185,0,215,153,0,55,169
50160 DATA 15,141,156,200,200,208,200
50170 DATA 165,1,9,4,133,1,173
50180 DATA 14,220,9,1,141,14,220
50190 DATA 169,28,141,24,208,169,15
50200 DATA 141,156,200,169,255,141,15
50210 DATA 212,169,128,141,18,212,169
50220 DATA 0,133,2,141,224,207,141
50230 DATA 255,207,141,254,207,141,253
50240 DATA 207,141,252,207,141,249,207
50250 DATA 160,6,169,20,153,0,207
50260 DATA 169,0,153,16,207,136,208
50270 DATA 243,169,251,141,251,207,160
50280 DATA 0,169,4,133,252,132,251
50290 DATA 169,216,133,254,132,253,169
50300 DATA 160,160,5,145,251,160,10
50310 DATA 145,251,160,15,145,251,160
50320 DATA 20,145,251,160,25,145,251
50330 DATA 160,30,145,251,160,35,145
```

```
50340 DATA 251,165,251,24,105,40,133
50350 DATA 251,144,2,230,252,201,232
50360 DATA 208,211,169,1,160,10,145
50370 DATA 253,169,4,160,5,145,253
50380 DATA 169,7,160,15,145,253,169
50390 DATA 14,160,20,145,253,169,8
50400 DATA 160,25,145,253,169,13,160
50410 DATA 30,145,253,169,3,160,35
50420 DATA 145,253,165,253,24,105,40
50430 DATA 133,253,144,2,230,254,201
50440 DATA 232,208,199,96,-1
50450 DATA 173,141
50460 DATA 2,201,1,208,3,76,49
50470 DATA 234,230,2,165,2,201,2
50480 DATA 240,3,76,49,234,169,0
50490 DATA 133,2,169,3,133,252,169
50500 DATA 216,133,251,160,45,177,251
50510 DATA 32,79,195,160,55,177,251
50520 DATA 32,79,195,160,65,177,251
50530 DATA 32,79,195,160,75,177,251
50540 DATA 32,79,195,165,251,24,105
50550 DATA 40,133,251,144,2,230,252
50560 DATA 201,192,208,213,76,0,198
50570 DATA 201,160,240,19,201,32,240
50580 DATA 37,162,1,232,221,174,195
50590 DATA 208,250,202,189,174,195,145
50600 DATA 251,96,152,56,233,40,168
50610 DATA 177,251,201,32,240,1,96
50620 DATA 152,24,105,40,168,169,227
50630 DATA 145,251,96,165,252,201,3
50640 DATA 240,22,152,56,233,40,168
50650 DATA 177,251,201,160,240,1,96
50660 DATA 152,24,105,40,168,169,99
50670 DATA 145,251,96,152,24,105,120
50680 DATA 168,177,251,201,100,240,1
50690 DATA 96,152,56,233,120,168,169
50700 DATA 99,145,251,96,160,228,239
50710 DATA 249,226,120,119,99,32,32
50720 DATA 100,111,121,98,248,247,227
50730 DATA -2,169,7,133,252
50740 DATA 169,32,133,251,160,170,177
50750 DATA 251,32,47,198,160,180,177
50760 DATA 251,32,47,198,160,190,177
50770 DATA 251,32,47,198,165,251,56
50780 DATA 233,40,133,251,176,2,198
50790 DATA 252,201,56,208,220,76,160
50800 DATA 198,201,160,240,19,201,32
50810 DATA 240,37,162,1,232,221,142
50820 DATA 198,208,250,202,189,142,198
```

```
50830 DATA 145,251,96,152,24,105,40
50840 DATA 168,177,251,201,32,240,1
50850 DATA 96,152,56,233,40,168,169
50860 DATA 228,145,251,96,165,251,201
50870 DATA 32,240,22,152,24,105,40
50880 DATA 168,177,251,201,160,240,1
50890 DATA 96,152,56,233,40,168,169
50900 DATA 100,145,251,96,152,56,233
50910 DATA 120,168,177,251,201,99,240
50920 DATA 1,96,152,24,105,120,168
50930 DATA 169,100,145,251,96,32,99
50940 DATA 119,120,226,249,239,228,160
50950 DATA 160,227,247,248,98,121,111
50960 DATA 100,32,173,0,220,72,41
50970 DATA 15,201,15,240,8,169,129
50980 DATA 141,4,212,76,183,198,169
50990 DATA 128,141,4,212,104,41,16
51000 DATA 205,255,207,240,48,141,255
51010 DATA 207,201,16,208,24,169,2
51020 DATA 141,15,195,169,1,141,252
51030 DATA 198,141,229,200,169,0,141
51040 DATA 250,207,141,224,207,76,239
51050 DATA 198,169,1,141,15,195,169
51060 DATA 2,141,252,198,141,229,200
51070 DATA 169,0,133,2,32,245,198
51080 DATA 76,32,200,238,250,207,173
51090 DATA 250,207,201,1,240,1,96
51100 DATA 169,0,141,250,207,173,0
51110 DATA 220,141,254,207,41,1,208
51120 DATA 13,173,253,207,201,253,240
51130 DATA 23,206,253,207,76,45,199
51140 DATA 173,254,207,41,2,208,10
51150 DATA 173,253,207,201,3,240,3
51160 DATA 238,253,207,173,254,207,41
51170 DATA 8,208,13,173,252,207,201
51180 DATA 3,240,23,238,252,207,76
51190 DATA 82,199,173,254,207,41,4
51200 DATA 208,10,173,252,207,201,253
51210 DATA 240,3,206,252,207,173,254
51220 DATA 207,41,3,201,3,208,16
51230 DATA 173,253,207,240,11,16,6
51240 DATA 238,253,207,76,107,199,206
51250 DATA 253,207,173,254,207,41,12
51260 DATA 201,12,208,16,173,252,207
51270 DATA 240,11,16,6,238,252,207
51280 DATA 76,132,199,206,252,207,174
51290 DATA 249,207,208,32,174,240,207
51300 DATA 224,60,176,25,173,253,207
51310 DATA 24,109,1,208,201,80,176
```

```
51320 DATA 5,169,244,76,191,199,201
51330 DATA 244,144,27,169,80,76,191
51340 DATA 199,173,253,207,24,109,1
51350 DATA 208,201,41,176,5,169,244
51360 DATA 76,191,199,201,244,144,2
51370 DATA 169,41,141,1,208,173,252
51380 DATA 207,48,32,24,109,0,208
51390 DATA 141,240,207,173,249,207,105
51400 DATA 0,141,249,207,201,1,208
51410 DATA 42,173,240,207,201,55,144
51420 DATA 35,32,155,200,76,4,200
51430 DATA 24,109,0,208,141,240,207
51440 DATA 173,249,207,105,255,141,249
51450 DATA 207,208,12,173,240,207,201
51460 DATA 25,176,5,169,25,141,240
51470 DATA 207,173,240,207,141,0,208
51480 DATA 173,16,208,41,254,13,249
51490 DATA 207,141,16,208,173,31,208
51500 DATA 41,1,240,3,76,101,200
51510 DATA 96,162,5,189,119,4,201
51520 DATA 57,240,6,254,119,4,76
51530 DATA 58,200,169,48,157,119,4
51540 DATA 202,208,235,76,58,200,162
51550 DATA 5,189,39,4,201,48,240
51560 DATA 6,222,39,4,76,222,200
51570 DATA 169,57,157,39,4,202,208
51580 DATA 235,120,169,234,141,21,3
51590 DATA 169,49,141,20,3,88,169
51600 DATA 255,133,2,76,222,200,0
51610 DATA 162,0,160,240,238,32,208
51620 DATA 232,208,250,200,208,247,169
51630 DATA 0,141,32,208,162,3,189
51640 DATA 39,4,201,48,240,4,222
51650 DATA 39,4,96,169,57,157,39
51660 DATA 4,202,208,237,162,5,169
51670 DATA 48,157,39,4,202,208,250
51680 DATA 104,104,76,81,200,160,15
51690 DATA 162,3,189,39,4,201,57
51700 DATA 240,6,254,39,4,76,180
51710 DATA 200,169,48,157,39,4,202
51720 DATA 208,235,136,208,230,169,0
51730 DATA 141,249,207,169,25,141,240
51740 DATA 207,169,148,141,1,208,172
51750 DATA 156,200,192,9,240,4,136
51760 DATA 140,156,200,173,5,4,201
51770 DATA 160,208,5,169,227,141,5
51780 DATA 4,96,238,224,207,173,224
51790 DATA 207,201,1,240,3,76,124
51800 DATA 201,169,0,141,224,207,173
```

```
51810 DATA 27,212,201,7,176,25,168
51820 DATA 185,0,207,201,20,208,8
51830 DATA 169,1,153,16,207,76,16
51840 DATA 201,201,255,208,5,169,255
51850 DATA 153,16,207,160,6,185,0
51860 DATA 207,24,121,16,207,153,0
51870 DATA 207,72,152,10,170,104,157
51880 DATA 1,208,136,208,235,160,6
51890 DATA 185,0,207,201,20,240,10
51900 DATA 201,255,240,6,136,208,242
51910 DATA 76,66,201,169,0,153,16
51920 DATA 207,76,52,201,173,30,208
51930 DATA 41,1,240,51,162,0,160
51940 DATA 240,238,32,208,232,208,250
51950 DATA 200,208,247,169,0,141,32
51960 DATA 208,162,3,189,39,4,201
51970 DATA 48,240,6,222,39,4,76
51980 DATA 49,234,169,57,157,39,4
51990 DATA 202,208,235,162,5,169,48
52000 DATA 157,39,4,202,208,250,76
52010 DATA 49,234,256
```

Program 2. Link Shuttle Escape with Moving Maze

```
90 POKE45,15000AND255:POKE46,15000/256:CLR
91 REM NO SPACES IN LINE 90!! VERY IMPORTANT!
120 IF PEEK(49153)<>169 THEN GOSUB 10000:GOSUB5000
    0:C2=0
720 GOTO 4000
4005 POKE V+23,0
4020 POKES+5,7*16:POKES+6,249:POKES+4,128
4050 POKE V+16,64:POKE 2040,244:POKEV,30:POKEV+1,1
     48
10005 PRINT"{HOME}{WHT}{12 RIGHT}READY IN"LEFT$(ST
      R$(146-INT(TI/60)),4)" SECONDS "
10010 READ A:IF A=256 THEN 10025
10025 IF C1<>34430 THEN PRINT"CHECKSUM ERROR IN LI
      NE 10025":END
11030 DATA 0,0,0,0,0,0,0
50010 PRINT"{HOME}{WHT}{12 RIGHT}READY IN"LEFT$(ST
      R$(101-INT(TI/60)),4)" SECONDS "
```

Chapter Five

Sound

Enlivening Programs with Sound

Gregg Peele

Have you been to a video arcade lately? If you have, then you know the impact that sound has on the excitement of a videogame. Whizzes, bangs, and explosions of all sorts are mixed with melodies and other special effects. Although the visuals provide most of the stimuli within a game, good sound effects add that final professional touch.

How can sound be used effectively within a program? Naturally, collisions, explosions, and other climactic events occurring on the screen need the added realism of sound. But don't limit its use to these special effects.

Sound can add a spark of interest to a particularly dull section of a game. Maybe it takes 10 or 20 seconds to set up the screen for your game. By adding sound to this part of your program, you can maintain the interest even though, visually, not much is happening.

Sound can also serve more practical purposes within other types of programs. A small beep can signal an error condition or remind the user that the computer needs attention.

Fortunately, Commodore has built excellent sound capabilities into the Commodore 64. In fact, the 64 contains one of the most sophisticated sound-producing systems of all personal computers, a true synthesizer-on-a-chip.

Fanfare

Below is a program that creates a sound effect which may be used to add a bit of excitement to almost any program. The routine produces an arcade-style fanfare for some triumphant moment within a game.

The addition of sound can enhance almost any computer program. Don't neglect the added dimension that sound can add to your computing.

Chapter Five

Fanfare

```
10 REM FANFARE
20 B=54272:FORCLEAR=B TO B+24:POKE CLEAR,0:NEXT
30 FOR R=1 TO 4
40 POKE B+5,85:POKE B+6,85:POKE B+12,85:POKE B+13,
   85
50 POKE B+24,15:POKE B+4,33:POKE B+11,17
60 FOR X=1 TO 6:READ H1,L1,H2,L2:POKE B+1,H1:POKE
   {SPACE}B,L1
70 POKE B+8,H2:POKE B+7,L2
80 IF H1=50 THEN FOR T=1 TO 200:NEXT
90 FOR T=1 TO 100:NEXT
100 DATA 25,30,18,209,33,135,25,30,42,62,31,165,50
    ,60,37,162,42,62,31,165,50,60
110 DATA 37,162
120 NEXT X
130 POKE B+4,32:POKE B+11,16:FOR W=1 TO 500:NEXT
140 RESTORE:NEXT R
150 FORCLEAR=B TO B+24:POKE CLEAR,0:NEXT
```

Chapter Five

Understanding Sound
Part 1

Gregg Peele

This article will help you understand how to create sounds on the Commodore 64. Also, there is a utility program which makes it easier to design sounds on the 64 and add them to your own programs.

The Amazing SID Chip

The Commodore 64 has three independent *voices* (sound channels), each having one of four possible *waveforms* (tone colors). These voices, produced by the MOS 6581 SID (Sound Interface Device) chip, can be set up to simulate almost any sound. In fact, the capability of the SID chip has been compared to music synthesizers costing more than the entire Commodore 64. To understand how to use the SID chip effectively, a brief discussion of the nature of sound is necessary.

Some Sound Theory

Most sounds in music and many sounds in nature have a defined *pitch*. Pitch is a way of describing how high or low a particular sound is.

The SID chip has a pitch range of nine octaves. This is about two octaves greater than a piano. When programming, these pitch values are formed from two *bytes* (a byte is a memory location which can hold a value of 0 to 255). This yields a range of more than 65,000 (256 x 256) possibilities of different pitch values for notes. The *pulse* waveform, one of the four waveforms available, allows an even broader range of pitch values.

Waveforms

Since we've already mentioned waveforms a couple of times, maybe we should clarify exactly what a waveform is. Almost every sound consists of a pulsating motion generally referred to as vibration. Different materials vibrate in different patterns. This is one reason why the different instruments of the orchestra have unique tonal qualities. The SID chip is able to produce four different waveforms: triangle, sawtooth, pulse, and noise. Each of these waveforms produces a unique sound and, along with pitch and envelope control, form the basis for sound synthesis on the Commodore 64.

A Stone's Throw

Sound waves, like waves from a stone thrown into a pond, constantly change. In fact, much of our ability to discern one sound from another is because of the unique pattern of change which fingerprints each sound. Some familiar examples are the different sounds produced when you strike something with a metal or rubber hammer. Much of the sound produced by the rubber hammer is absorbed within the hammer itself.

Envelopes

Most sounds follow a similar pattern through time. This pattern is the *envelope*. First, the initial event which creates the sound sends the volume level rapidly upward. This section of the envelope, called the *attack*, may be the major defining factor of a sound. A hand clap consists almost entirely of the attack section.

After this initial attack, the volume level decreases during the *decay* section. After this decrease, the volume level stabilizes for a time in what is called the *sustain* section. The sound then begins its final descent which terminates in silence. This descent is the *release* portion of the sound.

The combination of attack, decay, sustain, and release is the envelope, sometimes called the ADSR envelope. The SID chip provides a means to define the way a sound changes through time. This change is controlled with an *envelope generator*. The attack and decay sections are controlled within one byte — each using four bits (there are eight binary digits, or bits, in each byte). The values within this byte determine the rate that the volume changes through time. A low value for attack or decay indicates a short duration for that particular section. A larger value increases the duration of a particular section.

The sustain and release portions of the envelope also share one byte. However, sustain does not relate to a time value but to a volume level. The release section, like attack and decay, refers to a rate of change, and values for this section change the amount of time allocated for this change to occur.

Admittedly, all of this is not easy to understand at first. If you type in and run Program 1, you'll see and hear an animated demonstration of the ADSR envelope.

All Together Now

Producing sounds with the SID chip requires that certain *registers* (memory locations) within the chip contain values which represent the waveform, volume, and ADSR envelope. Also, there must be some provision for setting the length of the note. POKE commands in BASIC are used to place values for waveform, volume, and ADSR into their appropriate places.

The length of the sound is determined by using two BASIC FOR/NEXT loops as timers. The larger the value for the loops, the longer the length of the particular portion of the sound. The first loop determines the length of time allotted for the sustain portion of the sound, and the second loop determines the length of time allotted for the decay portion. The waveform byte turns the sound on. When turned off, it begins the decay, which ends the sound. One bit of that byte, referred to as the *gate bit*, is reserved for that purpose.

Here is the sequence of events: first the values for volume and ADSR are put in their proper places using the POKE command. Next, you turn on the sound by turning on the waveform byte with the gate bit set to 1. (This byte will always contain an odd value since the gate bit is the lowest bit in the byte.) Our FOR/NEXT loop is now used to provide a delay, which runs while the attack, decay, and sustain sections execute. When this loop finishes, we then replace the value that was in the waveform byte with an equivalent value minus one. This resets the gate bit and signals the release section to begin. The volume decreases until the sound is finally silent. Another FOR/NEXT loop allows the release section adequate time to execute.

An Example Program

Does all of this sound hopelessly complicated? To best illustrate the waveforms, pitches, and the envelope generator, I have included a program that allows you to manipulate all the parameters mentioned and actually create your own sound routine for use in other programs. To use Program 2, merely enter the values for volume, waveform, ADSR (attack, decay, sustain, release), and values for the length of the sustain and release. (Remember, within the range of values given, the lower values represent either low volumes or shorter lengths of time for each section.)

You also must enter two values to define the pitch of the tone. These pitch values can be derived from the table of values displayed on the screen or from the tables in the *Commodore 64 Programmer's Reference Guide* (pages 384-86).

When you are prompted with the word AGAIN?, press N if you are pleased with the sound that you have produced, or Y if you wish to continue altering the sound. If you press N, a subroutine will be created that you can add to your own programs. You will be prompted for the starting line number and the increment that you wish to leave between lines for the subroutine. Then your finished sound routine will appear on the screen. (Before you type N, make sure you have saved the original program, because it will be erased.) You may now use this new sound routine in any program or save it on disk or tape for future use.

One Small Step

We have taken only the first step toward understanding the complexities and possibilities of the SID chip. The program uses only one of the Commodore 64's three voices, and we have yet to discuss some advanced applications of the SID chip's features. However, we have taken a large step in our quest to uncover the mechanics of sound synthesis on the Commodore 64.

Program 1. ADSR Envelope

```
5 PRINT"{CLR}":POKE53281,12:POKE646,0
10 PRINTTAB(8)CHR$(18)CHR$(169)CHR$(223)"{OFF} "
20 PRINTTAB(7)CHR$(18)CHR$(169)"{2 SPACES}"CHR$(22
   3)
30 PRINTTAB(6)CHR$(18)CHR$(169)"{4 SPACES}"CHR$(22
   3)
40 PRINTTAB(5)CHR$(18)CHR$(169)"{6 SPACES}"CHR$(22
   3)
50 PRINTTAB(4)CHR$(18)CHR$(169)"{19 SPACES}"CHR$(2
   23)
60 PRINTTAB(3)CHR$(18)CHR$(169)"{21 SPACES}"CHR$(2
   23)
70 PRINTTAB(2)CHR$(18)CHR$(169)"{23 SPACES}"CHR$(2
   23)
80 PRINTTAB(1)CHR$(18)CHR$(169)"{25 SPACES}"CHR$(2
   23)
```

```
90 PRINT
100 PRINT"{4 SPACES}A{5 SPACES}D{3 SPACES}SUSTAIN
    {4 SPACES}R
110 PRINT"{4 SPACES}T{5 SPACES}E{14 SPACES}E
120 PRINT"{4 SPACES}T{5 SPACES}C{14 SPACES}L
130 PRINT"{4 SPACES}A{5 SPACES}A{14 SPACES}E
140 PRINT"{4 SPACES}C{5 SPACES}Y{14 SPACES}A
150 PRINT"{4 SPACES}K{20 SPACES}S
160 PRINT"{25 SPACES}E
170 CL=55296:S=54272:W=S+4:AD=S+5:SR=S+6:V=S+24
175 POKEV,15:POKEAD,202:POKESR,58:POKES,135:POKES+
    1,33:POKEW,33
180 FORR=CLTOCL+5:FORU=RTOCL+1024STEP40:
185 POKEU,1:NEXT:NEXT
190 FORR=CL+6TOCL+12:FORU=RTOCL+1024STEP40
195 POKEU,1:NEXT:NEXT
197 FORR=CL+13TOCL+23:FORU=RTOCL+1024STEP40
198 POKEU,1:NEXT:NEXT
200 POKEW,16:FORR=CL+24TOCL+28:FORU=RTOCL+1024STEP
    40
290 POKEU,1:NEXT:NEXT
300 FORT=STOS+28:POKET,0:NEXT
```

Program 2. Soundmaker 1

```
5 POKE53281,1:POKE646,0
10 S=54272:FORE=STOS+28:POKEE,0:NEXT
15 PRINT"{CLR}{UP}":GOSUB200
20 INPUT"ATTACK RATE 0-15";AT:INPUT"DECAY RATE 0-1
   5";DE:AD=16*AT+DE:POKE54277,AD
25 INPUT"SUSTAIN{SHIFT-SPACE}VOLUME 1-15";SU:INPUT
   "RELEASE RATE 0-15";RL:J=16*SU+RL
30 POKE54278,J:INPUT"OVERALL VOLUME 1-15";V:POKE54
   296,V
32 INPUT"HIGH BYTE";H:INPUT"LOW BYTE";L:POKE54273,
   H :POKE54272,L
34 INPUT"SUSTAIN LENGTH (* .1 SECOND)";LE:LE=LE*100
35 INPUT"WAVEFORM 17,33,65,OR 129 ";W
36 IFW=65THENINPUT"PULSE WIDTH HIGH(1-15)";PW:INPU
   T"PULSE WIDTH LOW(0-255)";P2
37 IFW=65THENPOKE54275,PW:POKE54274,P2:GOTO39
38 PRINT"{DOWN}"
39 POKE54276,W
40 FORT=1TOLE:NEXTT
42 POKE54276,(W-1)
43 FORT=1TODL:NEXT
50 S=54272
60 PRINT"{HOME}{12 DOWN}{RVS}AGAIN OR CLEAR ?{OFF}
    Y OR N"
```

```
65 GETA$:IFA$="C"THENFORAS=54272TO54272+24:POKEAS,
   Ø:NEXT
7Ø IFA$="Y"THENPRINT"{HOME}{12 DOWN}{24 SPACES}":G
   OTO2Ø
75 IFA$<>"N"THEN65
8Ø REM PRINT PROGRAM
85 INPUT"{CLR}STARTING LINE";SL:INPUT"INCREMENT";I
   N
86 PRINT"{CLR}"
88 PRINT"{3 DOWN}NEW{3 DOWN}"
89 PRINTSL;"S=54272:FORE=STOS+28:POKEE,Ø:NEXT":SL=
   SL+IN
9Ø PRINTSL;"POKE54296,";V;":POKE54277,";AD;":POKE5
   4278,";J:SL=SL+IN
1ØØ IFW=65THENPRINTSL;"POKE54275,";PW;":POKE54274,
    ";P2:SL=SL+IN
12Ø PRINTSL;"POKE 54273,";H;":POKE54272,";L;":POKE
    54276,";W:SL=SL+IN
14Ø PRINTSL;"FORT=1TO";LE;":NEXT";":POKE54276,";(W
    -1)
155 PRINT"{HOME}";:FORR=631TO644:POKER,13:NEXT
16Ø POKE198,13
165 END
2ØØ PRINT" SAMPLE DATA FOR PITCH VALUES"
2Ø5 PRINT" PITCH HIGH BYTE LOW BYTE{2 SPACES}{RVS}
    WAVEFORMS
21Ø PRINT"{3 SPACES}C{7 SPACES}33{6 SPACES}135
    {5 SPACES}TRIANGLE=17
22Ø PRINT"{3 SPACES}C#{6 SPACES}35{6 SPACES}134
    {5 SPACES}SAWTOOTH=33
23Ø PRINT"{3 SPACES}D{7 SPACES}37{6 SPACES}162
    {5 SPACES}NOISE=129
24Ø PRINT"{3 SPACES}D#{6 SPACES}39{6 SPACES}223
    {5 SPACES}PULSE=65
25Ø PRINT"{3 SPACES}E{7 SPACES}42{6 SPACES}62
26Ø PRINT"{3 SPACES}F{7 SPACES}44{6 SPACES}193
27Ø PRINT"{3 SPACES}F#{6 SPACES}47{6 SPACES}1Ø7
28Ø PRINT"{3 SPACES}G{7 SPACES}5Ø{6 SPACES}6Ø
29Ø PRINT"{3 SPACES}G#{6 SPACES}53{6 SPACES}57
3ØØ PRINT"{3 SPACES}A{7 SPACES}56{6 SPACES}99
335 PRINT
34Ø RETURN
```

Understanding Sound
Part 2

Gregg Peele

Ever wished you could create just that right sound for a game effect? Or that right tone for a song? The conclusion of this two-part article and the accompanying utility program may be just what you need to create interesting new sounds on your 64.

In Part 1 we explored some of the basics of producing sound on the Commodore 64. We discussed ADSR (attack, decay, sustain, and release) and used these parameters along with volume, pitch, and waveform to produce various sounds. In this part we will look even further into the capabilities of the 64's built-in synthesizer on a chip, the Sound Interface Device (SID). We'll discuss filters, ring modulation, and synchronization, and present a utility, "Soundmaker 2," which will make it easier to use these techniques within your own programs.

Changed Your Filters Lately?

The Commodore 64 SID chip has three filters — but unlike the filters in your car, they should never need replacing. However, they do share some similarities with car filters. Just as an oil filter allows oil to pass while blocking out other unwanted particles, the SID chip filters let parts of sounds pass — selectively *filtering* out the remainder of the sound. Synthesizer filters provide an important means of manipulating sounds to produce various effects.

The three filters are called *high pass, low pass,* and *band pass*. (See Figures 1-3.) The high-pass filter is designed to remove the lower frequencies, letting the higher frequencies pass. The low-pass filter has the opposite effect — it removes the high frequencies while allowing low frequencies to pass. The band-pass filter allows a band or group of frequencies to pass through while frequencies above and below the band are suppressed.

The filter you choose is activated by turning on bits 4 (low pass), 5 (band pass), or 6 (high pass) in SID register 24 (see "Bit-mapped Graphics" for details on turning bits on or off). These filters can be used in combinations for additional effects. For instance, adding the low- and high-pass filters together creates the inverse effect of the band-pass filter; only the higher and lower frequencies pass, suppressing the middle frequencies.

The amount of sound that is removed by a filter is determined by the *cutoff frequency*. The filter cuts off the sound beginning at this frequency. The cutoff frequency for filtering is controlled by the lower three bits in SID register 21 and all eight bits in register 22. Some of the most interesting effects possible on the 64 are created by incrementing or decrementing these series of bits while a sound is being played. Want the sound of an alien ship as it lands? Use your normal alien ship sound, add a filter, and gradually increment or decrement these eight bits as your ship descends. A certain combination of waveforms and a changing filter can create just the right sound effect for a descending alien ship.

Additive and Subtractive Synthesis

Filtering is an example of *subtractive synthesis.* Subtractive synthesis is a method of manipulating sounds by subtracting parts of a single sound — pushing other parts which normally may not be heard into the forefront. *Additive synthesis,* however, brings two sounds together to form a totally new sound. Both *ring modulation* and *synchronization* are examples of additive sound synthesis.

Ring Modulation

Ring modulation is a form of additive sound synthesis that dramatically changes the timbre or tone quality of two tones. Tones that have been fed through a ring modulator do not retain their original pitches or timbres. Instead, the sums and remainders of the two frequencies are retained. For instance, if the first sound is a tone that vibrates at 100 vibrations per second (vps), and the second tone vibrates at 200 vps, then the ring-modulated tone will be a combination of the sum (300 vps) and the difference (100 vps).

Usually the ring-modulated tone sounds very different than the two original tones. Since most tones are complex phenomena consisting of many less obvious inner frequencies (harmonics), the ring-modulated tone may be very complex in tonal character.

To achieve ring modulation on the 64, you have to set bit 2 of the waveform byte when using the triangle waveform (POKE register 4 with 21). Voice 3 must be set to some frequency. No other parameters of voice 3 have any effect on ring modulation.

Synchronization on the 64 also adds two tones together to produce a new and different sound. If bit 1 of the waveform byte is set (POKE register 4 with 19), then setting voice 3 to a definite pitch (POKE registers 14 and 15 for the pitch of voice 3) and manipulating the pitch of voice 1 (registers 0 and 1) cause the tone quality of the resulting pitch to change.

Synchronization happens when the two waveforms are linked to make the waveform of voice 1 dependent on whether it is *in sync* with the frequency produced by voice 3. Since the two waveforms are not usually in sync, the waveform is distorted, producing different and sometimes interesting waveforms. In sync mode, the pitch of the tone you hear depends on the pitch of voice 3, not voice 1, as would normally be the case.

Paddling with the SID

The SID chip also contains two registers (25-26) connected to the two joystick ports. These registers will contain a number from 0 to 255 depending on the resistance of a potentiometer attached to the ports (255 at maximum resistance). Since game paddles are really potentiometers (variable resistors), these ports can be used to register paddle movement and can easily be used to change values in other registers within the chip while sounds are being produced.

This simple routine can be added to a sound program to control the pitch of voice 1 with a paddle plugged into port 1 while a tone is being played:

```
10 POKE 54272+1,PEEK(54272+25):GOTO 10
```

This line connects the paddle value to the high byte frequency value of voice 1. It's much easier to study the effects of changing sound values if you can hear the sound playing as you experiment. That is the basis of Soundmaker 2.

Soundmaker

Soundmaker 2 allows you to create your own sounds and manipulate them by changing various parameters. Attack, decay, sustain, and release are included as well as pitch, filters, ring modulation, and synchronization. The pulse waveform may be manipulated to change the pulse width of the sound — altering the timbre of the resulting sound considerably.

To use Soundmaker 2, type it in and SAVE it on disk or tape. When you are sure you have a saved copy, run the program. After a brief delay while the program loads a small machine language routine into memory, the word Attack appears at the upper-right corner of your screen. Using the + and – keys, you can increase or decrease the attack value for your sound. The current value POKEd is represented by both a bar graph and a number. The number varies in units of 16 or 1 depending on which parameter you are working with. These values are meant to serve as a reference point only, since they may differ from the actual value by one unit. The increments were selected to make the changes in parameters very easy to hear and the program easy to use.

Once you have decided on the attack value, simply hit RETURN and the next parameter appears. Keep in mind that Sustain and Volume must be a reasonably high number for the sound to be audible. When you have picked all the parameters (Pulse wave low is the last one on the screen), then you can play the sound with the function keys. The f1 key plays the sound with the sawtooth waveform, f3 with the triangle waveform, f5 with the noise waveform, and f7 with the pulse waveform.

Ring Modulation and Sync

The up-arrow key (beside the asterisk) plays your sound as it is ring modulated with voice 3, and the left-arrow key (beside the 1) plays the synchronized sound resulting from the pitches of voice 1 and voice 3. (Ring modulation and synchronization are limited to voice 1.)

Once you have heard voice 1, simply hit the 2 key and you will again be prompted for the parameters. As with voice 1, you play voice 2 with the function keys. To hear voices 1 and 2 simultaneously, hit the space bar. To select the parameters for voice 3, press the 3 key. The space bar then plays all voices previously defined. If you have selected ring modulation or synchronization for voice 1, you may not be able to use voice 3 as a separate sound.

Changing Sounds

To alter any parameter at any time after entering it originally, merely press the key which is in reverse field on the parameter name and press the + or – key to raise or lower the value. When done, hit RETURN.

You can even change parameters as the sound is playing. To do this, hit one of the function keys or one of the arrow keys to start the note and, without releasing it, hit the reverse field character of the parameter you wish to change. Then change the sound with the + and – keys.

To use the filters as the sound is being played, you must first start the sound that you want, then, without releasing the key, hit either H (for high pass), B (for band pass), or L (for low pass). Next, hit F for filter and use the + and – keys to increment or decrement the cutoff frequency. As before, hit RETURN to end the note.

To save the sound or sounds that you have created, press Q while the note is playing. The screen clears and a program appears on the screen. Type NEW and press RETURN over the lines as they are listed on the screen. Then you can play this sound, or save it on tape or disk and use it later as a routine in your own programs. To use it as a routine, you'll need a delay loop such as this to set the duration:

```
70 FOR T=1 TO 2000:NEXT T
```

Then, to turn off the sound, use this line:

```
80 FOR T=49152+4 TO 49152+18 STEP7:POKE T,(PEEK(T)
   AND 254):NEXT:SYS 53017
```

To turn on the sound in your own program, you can either GOSUB the whole routine or use this line (with your own line number):

```
FOR T=49152+4 TO 49152+18 STEP 7:POKE T,(PEEK(T)OR
   1):NEXT:SYS 53017
```

A Bit about the Program

Soundmaker 2 uses a tiny machine language (ML) routine which copies the contents of 24 bytes starting at 49152 to the sound registers beginning at 54272. The ML routine copies the registers in the order they should be POKEd to properly create a sound.

This is done because sound registers are *write-only* registers. That is, when values are POKEd into the SID registers, they cannot be PEEKed later. Instead, you must store the values in variables or other memory locations. The ML routine stores these values in a safe area of memory and allows us to copy them at any time to the SID registers. The ability to remember the values which have been POKEd into the SID chip makes Soundmaker 2 possible.

Soundmaker 2

```
100 I=52992
110 READ A:IF A=256 THEN 190
120 POKE I,A:I=I+1:GOTO 110
130 DATA 24,5,6,0,1,2,3
140 DATA 21,12,13,7,8,9,10
150 DATA 11,19,20,14,15,16,17
160 DATA 23,4,11,18,162,0,188
170 DATA 0,207,185,0,192,153,0
180 DATA 212,232,224,25,208,242,96,256
190 POKE53281,1:POKE53280,1
200 POKE650,128
210 F$="{19 SPACES}"
220 S=49152:D=0:Q=54272:P=53017:M$="VOICE":Z$="
    {4 SPACES}{4 LEFT}"
230 FORT=STOS+30:POKET,0:NEXT:SYSP
240 PRINT"{CLR}";:FI$=" NONE "
250 FORA=1TO11:ON A GOSUB500,510,520,530,540,550,5
    60,570,590,600,610:NEXT
270 GETE$:U=PEEK(197):IFU=64ANDPEEK(S+4)THENPOKES+
    4,PEEK(S+4)AND254:SYSP
280 IFU=64ANDPEEK(S+7+4)THENPOKES+7+4,PEEK(S+7+4)A
    ND254:SYSP
290 IFU=64ANDPEEK(S+14+4)THENPOKES+14+4,PEEK(S+14+
    4)AND254:SYSP
300 IFU=62THENSYSP:GOTO1330
310 IFE$="1"ORE$="2"ORE$="3"THEND=(ASC(E$)-49)*7:P
    RINT"{CLR}";TAB(25);M$;E$:GOTO250
320 IFD>7THENPOKES+24,(PEEK(S+24)AND127):SYSP
330 IFU=4THENPOKES+4+D,33:SYSP
340 IFU=5THENPOKES+4+D,17:SYSP
350 IFU=6THENPOKES+4+D,129:SYSP
360 IFU=3THENPOKES+4+D,65:SYSP
370 IF U=39THENPOKES+24,(PEEK(S+24)AND255):FI$=" N
    ONE{6 SPACES}":POKES+23,0:SYSP
380 IF U=60 THENFORT=0TO14STEP7:POKES+4+T,PEEK(S+4
    +T)OR1:NEXT:SYSP
390 IFU=57THENPOKES+4+D,PEEK(S+4+D)OR3:SYSP
400 IFU=54THENPOKES+4+D,21:SYSP
410 V=2↑(D/7)
420 IFU=42THENFI$=" LOWPASS ":POKES+23,V:POKES+24,
    (PEEK(S+24)OR16):SYSP
430 IFU=29THENFI$=" HIGHPASS ":POKES+23,V:POKES+24
    ,(PEEK(S+24)OR64):SYSP
440 IFU=28THENFI$=" BANDPASS ":POKES+23,V:POKES+24
    ,(PEEK(S+24)OR32):SYSP
```

```
450 N$="ADSROYTVFPW":FORJ=1TO LEN(N$):G$=MID$(N$,J
    ):IF LEFT$(G$,1)=E$THEN480
460 NEXT
470 GOTO270
480 ONLEN(G$)GOSUB610,600,590,570,560,550,540,530,
    520,510,500
490 GOTO270
500 PRINT"{BLK}{HOME}{RVS}A{OFF}TTACK{2 SPACES}RAT
    E +-":GOSUB620:RETURN
510 PRINT"{BLU}{HOME}{2 DOWN}{RVS}D{OFF}ECAY
    {2 SPACES}RATE +-":GOSUB700:RETURN
520 PRINT"{RED}{HOME}{4 DOWN}{RVS}S{OFF}USTAIN LEV
    EL +-":GOSUB770:RETURN
530 PRINT"{GRN}{HOME}{6 DOWN}{RVS}R{OFF}ELEASE RAT
    E{2 SPACES}+-":GOSUB840:RETURN
540 PRINT"E1З{HOME}{8 DOWN}{RVS}O{OFF}VERALL VOL
    UME +-":GOSUB910:RETURN
550 PRINT"E2З{HOME}{10 DOWN}PITCH (HIGH B{RVS}Y
    {OFF}TE)+-":GOSUB970:RETURN
560 PRINT"{PUR}{HOME}{12 DOWN}PI{RVS}T{OFF}CH (LOW
     BYTE)+-":GOSUB1030:RETURN
570 IFD>0THENPRINT"{HOME}{14 DOWN}NO RING/SYNC FOR
     VOICES TWO AND THREE":RETURN
580 PRINT"E7З{HOME}{14 DOWN}PITCH {RVS}V{OFF}OIC
    E 3 (FOR RING)+-":GOSUB1090:RETURN
590 PRINT"E4З{HOME}{16 DOWN}{RVS}F{OFF}ILTERS
    {2 SPACES}CUTOFF{2 SPACES}+-":GOSUB1150:RETURN
600 PRINT"E3З{HOME}{18 DOWN}{RVS}P{OFF}ULSE WAVE
     HIGH{2 SPACES}+-":GOSUB1210:RETURN
610 PRINT"E2З{HOME}{20 DOWN}PULSE {RVS}W{OFF}AVE
     LOW{3 SPACES}+-":GOSUB1270:RETURN
620 POKE198,0:GETA$:IF A$<>""THEN620
630 IF PEEK(197)<>40ANDPEEK(197)<>43ANDPEEK(197)<>
    1THEN680
640 IFPEEK(197)=40ANDX1<15THENX1=X1+1
650 IFPEEK(197)=43ANDX1>0THENX1=X1-1
660 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO5
    00:NEXT:POKE198,0:PRINT:RETURN
670 PRINT"{RVS}";LEFT$(F$,X1);"{OFF}";RIGHT$(F$,15
    -X1);Z$;(PEEK(S+D+5)AND240);"{2 UP}"
680 POKES+D+5,(X1*16)+(PEEK(S+D+5)AND15):POKEQ+D+5
    ,(PEEK(S+D+5))
690 GOTO630
700 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43ANDP
    EEK(197)<>1THEN750
710 IFPEEK(197)=40ANDX2<15THENX2=X2+1
720 IFPEEK(197)=43ANDX2>0THENX2=X2-1
730 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO5
    00:NEXT:POKE198,0:PRINT:RETURN
```

```
740 PRINT"{RVS}";LEFT$(F$,X2);"{OFF}";RIGHT$(F$,15
    -X2);Z$;(PEEK(S+D+5)AND15);"{UP}"
750 POKES+D+5,X2+(PEEK(S+D+5)AND240):POKEQ+D+5,PEE
    K(S+D+5)
760 GOTO700
770 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43ANDP
    EEK(197)<>1THEN820
780 IFPEEK(197)=40ANDX3<15THENX3=X3+1
790 IFPEEK(197)=43ANDX3>0THENX3=X3-1
800 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO5
    00:NEXT:POKE198,0:PRINT:RETURN
810 PRINT"{RVS}";LEFT$(F$,X3);"{OFF}";RIGHT$(F$,15
    -X3);Z$;(PEEK(S+D+6)AND240);"{UP}"
820 POKES+D+6,(X3*16)+(PEEK(S+D+6)AND15):POKEQ+D+6
    ,PEEK(S+D+6)
830 GOTO770
840 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43ANDP
    EEK(197)<>1THEN890
850 IFPEEK(197)=40ANDX4<15THENX4=X4+1
860 IFPEEK(197)=43ANDX4>0THENX4=X4-1
870 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO5
    00:NEXT:POKE198,0:PRINT:RETURN
880 PRINT"{RVS}";LEFT$(F$,X4);"{OFF}";RIGHT$(F$,15
    -X4);Z$;(PEEK(S+D+6)AND15);"{UP}"
890 POKES+D+6,X4+(PEEK(S+D+6)AND240):POKEQ+D+6,PEE
    K(S+D+6)
900 GOTO840
910 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43ANDP
    EEK(197)<>1THEN960
920 IFPEEK(197)=40ANDX5<15THENX5=X5+1
930 IFPEEK(197)=43ANDX5>0THENX5=X5-1
940 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO5
    00:NEXT:POKE198,0:PRINT:RETURN
950 PRINT"{RVS}";LEFT$(F$,X5);"{OFF}";RIGHT$(F$,15
    -X5);Z$;(PEEK(S+24)AND15);"{UP}"
960 POKES+24,(X5+(PEEK(S+24)AND240)):SYSP:GOTO910
970 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43ANDP
    EEK(197)<>1THEN1020
980 IFPEEK(197)=40ANDX6<15THENX6=X6+1
990 IFPEEK(197)=43ANDX6>0THENX6=X6-1
1000 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO
     500:NEXT:POKE198,0:PRINT:RETURN
1010 PRINT"{RVS}";LEFT$(F$,X6);"{OFF}";RIGHT$(F$,1
     5-X6);Z$;PEEK(S+D+1);"{UP}"
1020 POKES+1+D,16*X6:POKEQ+1+D,PEEK(S+1+D):GOTO970
1030 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43AND
     PEEK(197)<>1THEN1080
1040 IFPEEK(197)=40ANDX7<15THENX7=X7+1
1050 IFPEEK(197)=43ANDX7>0THENX7=X7-1
```

```
1060 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO
     500:NEXT:POKE198,0:PRINT:RETURN
1070 PRINT"{RVS}";LEFT$(F$,X7);"{OFF}";RIGHT$(F$,1
     5-X7);Z$;PEEK(S+D);"{UP}"
1080 POKES+D,16*X7:POKEQ+D,PEEK(S+D):GOTO1030
1090 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43AND
     PEEK(197)<>1THEN1140
1100 IFPEEK(197)=40ANDX8<15THENX8=X8+1
1110 IFPEEK(197)=43ANDX8>0THENX8=X8-1
1120 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO
     500:NEXT:POKE198,0:PRINT:RETURN
1130 PRINT"{RVS}";LEFT$(F$,X8);"{OFF}";RIGHT$(F$,1
     5-X8);Z$;PEEK(S+15+D);"{UP}"
1140 POKEQ+24,PEEK(S+24)OR128:POKES+15+D,X8*16:POK
     EQ+15+D,X8*16:GOTO1090
1150 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43AND
     PEEK(197)<>1THEN1200
1160 IFPEEK(197)=40ANDX9<15THENX9=X9+1
1170 IFPEEK(197)=43ANDX9>0THENX9=X9-1
1180 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO
     500:NEXT:POKE198,0:PRINT:RETURN
1190 PRINT"{RVS}";LEFT$(F$,X9);"{OFF}";RIGHT$(F$,1
     5-X9);Z$;PEEK(S+22);"{6 RIGHT}";FI$;"{UP}"
1200 POKES+21,X9/2:POKES+22,(X9*16):POKEQ+21,7:POK
     EQ+22,(X9*16):GOTO1150
1210 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43AND
     PEEK(197)<>1THEN1260
1220 IFPEEK(197)=40ANDXA<15THENXA=XA+1
1230 IFPEEK(197)=43ANDXA>0THENXA=XA-1
1240 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO
     500:NEXT:POKE198,0:PRINT:RETURN
1250 PRINT"{RVS}";LEFT$(F$,XA);"{OFF}";RIGHT$(F$,1
     5-XA);Z$;PEEK(S+D+2);"{UP}"
1260 POKES+D+2,XA*16:POKEQ+D+2,PEEK(S+D+2):GOTO121
     0
1270 POKE198,0:IF PEEK(197)<>40ANDPEEK(197)<>43AND
     PEEK(197)<>1THEN1320
1280 IFPEEK(197)=40ANDXB<15THENXB=XB+1
1290 IFPEEK(197)=43ANDXB>0THENXB=XB-1
1300 IFPEEK(197)=1THENPOKE197,0:POKE198,0:FORT=1TO
     500:NEXT:POKE198,0:PRINT:RETURN
1310 PRINT"{RVS}";LEFT$(F$,XB);"{OFF}";RIGHT$(F$,1
     5-XB);Z$;PEEK(S+D+3);"{UP}"
1320 POKES+D+3,XB*16:GOTO1270
1330 REM SAVE ROUTINE
1340 S=49152:CO=52992
1350 PRINT"{CLR}":DIMQ(45),ML(45)
1360 FORT=0TO44:Q(T)=PEEK(S+T):ML(T)=PEEK(CO+T):NE
     XT
```

```
1370 PRINT"1 RP=52992:FORR=RPTORP+44:READGP:POKER,
     GP:NEXT"
1380 PG=Ø:FORA=ØTO4:PG=PG+3
1390 PRINT PG"DATA";:FORT=ØTO8 :PRINTML(T+9*A);:IF
      T<8 THENPRINT"{LEFT},";
1400 NEXT:PRINT:NEXT
1410 PRINT"20S=49152:FORT=STOS+24:POKET,Ø:NEXT:P=5
     3Ø17{2 SPACES}"
1420 PRINT"3ØFORT=STOS+25:READDS:POKET,DS:NEXT:SYS
     P{3 SPACES}"
1430 PO=3Ø:FORW=ØTO2:PO=PO+1Ø
1440 PRINTPO"DATA";:FORT=ØTO8:PRINTQ(T+9*W);:IFT<8
     THENPRINT"{LEFT},";
1450 NEXT:PRINT:NEXT
```

Figure 1. Low Pass Filter (Cutoff frequency is incremented through time.)

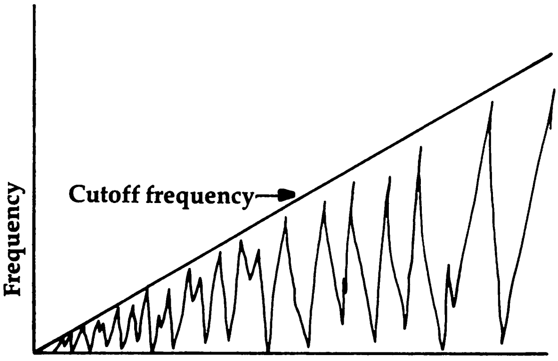


Figure 2. Band Pass Filter (Cutoff frequency is incremented through time.)

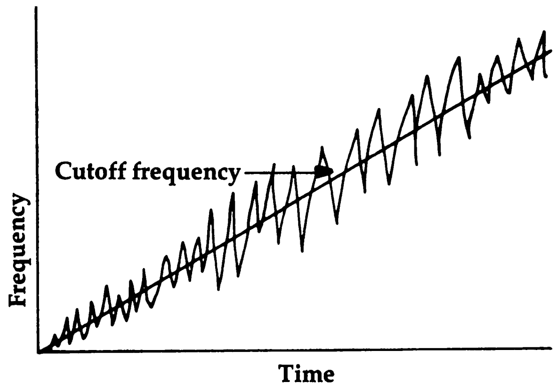
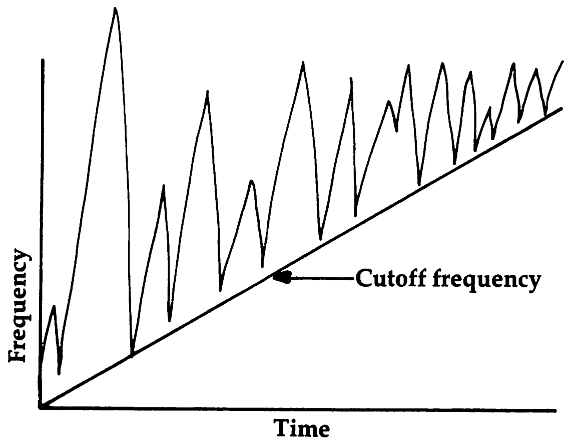


Figure 3. High Pass Filter
(Cutoff frequency is incremented through time.)



Chapter Six

Music

Songster

W. J. Crowley

Creating songs on the Commodore 64 using DATA statements can be a long process. "Songster" makes this process much easier by creating a file on disk or tape, thus avoiding the need for DATA statements. The instructions for using the program are contained within the program.

As a proud new owner of a Commodore 64, I immediately became fascinated by the capabilities of the Sound Interface Device (SID). After exploring all the examples in the *Programmer's Reference Guide,* I tried converting sheet music into DATA statements so that I could try SID on some of my favorite songs and sounds.

Eventually, I decided it would be more convenient to have a program that would do the music-to-data translation, and provide a way to store and retrieve songs from tape or disk, as well as play them back. The result of this exploration is "Songster."

The Program

Referring to the program listing, lines 100 through 280 provide screen formatting and a menu for the various functions available. The subroutine at lines 1000 through 1130 initializes variables and sets up the Sound Interface Device as described in the *Programmer's Reference Guide,* except that I changed the note storage arrays from floating-point to integer variables. Since integer variables require less memory, there is room for longer songs.

The HELP subroutine begins at line 1500 and gives a short explanation of how Songster is used. Line 1510 changes the screen display to upper/lowercase for ease of reading. Lines 1660 and 1670 halt the screen display until you press the space bar. Lines 1700 through 1790 give an example of how the screen will look when you begin entering notes. Line 1830 returns the display to uppercase/graphics mode.

Lines 5000 through 6170 are the ENTER/EDIT module. Line 5030 starts things off accepting notes for voice 1, and creates a loop for accepting notes for voices 2 and 3. Line 5090 tests to see if you've typed a negative number, indicating a rest. Line 6000 interprets a 0 to mean you want to change to the next voice (or return to the Command Menu if you've completed the third voice). Lines 6030 and 6040 test to see if you want to move notes up or down an octave, and then redisplay that information if you've made a change. Line 6050 changes the note-name normally used in musical annotation to an equivalent number. Lines 6060 through 6150 are borrowed directly from the *Programmer's Reference Guide,* and line 6170 returns action to the Command Menu. Lines 9000 through 9180 do the note-name to number conversion. Notice that if some invalid key is pressed, this module defaults to the bass note (C, for the current octave).

Lines 10000 through 10150 provide for loading a song from tape. Disk users should make the necessary changes to lines 10040 and 20040 as indicated in the REM statement.

Lines 15000 through 15030 format the screen for playing back songs, and accept a tempo number from the keyboard before the song is played. I left some space between lines 15030 and 15100, thinking that I might want to accept some waveform control inputs here. Lines 15100 through 15200 are also taken almost directly from the *Programmer's Reference Guide,* which explains these lines very well.

Now let's type the program in, so you can see how it works. After you SAVE a copy on tape, RUN it and look at the Command Menu displayed on your screen. The first choice is HELP, but rather than examine that first, let's do a simple example. Type E (without a carriage return) and notice that we go to the ENTER/EDIT module, where the program asks for a song title. Type JUNQUE or some other testing-title; note that we are entering data for voice 1. Songster is asking for *duration,* or how long the note should last. Answer as follows:

Program Prompt	**You Type**
DURATION?	0 (zero)
DATA FOR VOICE # 2	
DURATION?	4
OCTAVE = 4 : NOTE?	– (minus symbol)
OCTAVE = 3 : NOTE?	B– (B flat)
DURATION?	2

```
OCTAVE = 3 : NOTE?              + (plus symbol)

OCTAVE = 4 : NOTE?              F

                 DURATION?      4

OCTAVE = 4 : NOTE?              –

OCTAVE = 3 : NOTE?              F

                 DURATION?      2

OCTAVE = 3 : NOTE?              F

                 DURATION?      4

OCTAVE = 3 : NOTE?              +

OCTAVE = 4 : NOTE?              F

                 DURATION?
```

Now look at the screen. Notice that except when you're in the Command Menu, all your responses are followed by a carriage return. The responses to DURATION? are always a number (negative numbers for rests). NOTE questions are always answered with a plus (+), a minus (–), or a note letter-name (A through G).

The program is now waiting at the DURATION? question, so answer with a zero (0) and notice that we switched to voice 3. Answer this DURATION? question with a zero (0) also, and we return to the Command Menu. Typing a P will switch us to the PLAY module, and the song title will be displayed. The TEMPO is set to 80 by default, so just turn up the volume on your monitor and type a carriage return. You will hear our little song played, and at the conclusion we return to the Command Menu. Now select the PLAY module again, but this time answer the TEMPO question with a 40 and notice the difference in the speed at which notes are played. If you play the song again, you'll see that this TEMPO (40) will be retained until you change it again. If you now select the ENTER/EDIT module, you can enter information for voices 1 or 3, without changing or disturbing voice 2. You can also change voice 2 if you wish. You may want to try storing and retrieving examples from tape now, to complete testing the program.

If you want to try a song of your own choice now, type the Q selection in the Command Menu to stop the program, then type RUN in order to clear all variables and arrays previously used.

Songster

```
100 GOSUB 1000:REM INITIALIZE
110 REM ****** COMMAND SELECTION ******
120 GOSUB 1400
130 PRINT"{3 SPACES}COMMAND MENU ........."
140 PRINT"{8 SPACES}H...HELP/INSTRUCTIONS"
150 PRINT"{8 SPACES}E...ENTER/EDIT SONG"
160 PRINT"{8 SPACES}L...LOAD SONG-FROM TAPE"
170 PRINT"{8 SPACES}P...PLAY SONG"
180 PRINT"{8 SPACES}S...SAVE SONG-ON TAPE"
190 PRINT"{8 SPACES}Q...QUIT/END PROGRAM"
200 PRINT"{3 SPACES}COMMAND?"
210 GET I$:IF I$=""THEN210
220 IF LEFT$(I$,1)="H" THEN 1500
230 IF LEFT$(I$,1)="E" THEN 5000
240 IF LEFT$(I$,1)="L" THEN 10000
250 IF LEFT$(I$,1)="P" THEN 15000
260 IF LEFT$(I$,1)="S" THEN 20000
270 IF LEFT$(I$,1)="Q" THEN 999
280 GOTO 120
999 GOSUB 1400:PRINT".....END":END
1000 REM **** INITIALIZE ****
1010 S=54272:FORL=STOS+24:POKEL,0:NEXT
1020 DIMH%(2,600),L%(2,600),C%(2,600)
1030 DIMFQ(11)
1040 V(0)=17:V(1)=65:V(2)=33
1060 FOR I=0TO11:READ FQ(I):NEXT
1070 TEMPO=80:OCT%=4
1080 RETURN
1100 REM ---- DATA-TOP OCTAVE
1110 DATA 34334,36376,38539,40830
1120 DATA 43258,45830,48556,51443
1130 DATA 54502,57743,61176,64814
1400 REM ---- SCREEN CLEAR ROUTINE
1410 PRINT"{CLR}":FOR I=1 TO 8:PRINTCHR$(13);:NEXT
1420 RETURN
1500 REM **** HELP/INSTRUCTIONS ****
1510 GOSUB 1400:PRINTCHR$(14)
1520 PRINT"TO WRITE MUSIC, YOU TYPE THE DURATION,
     {2 SPACES}OCTAVE, & LETTER OF EACH NOTE
1530 PRINT"FOR EACH OF THE 3 VOICES."
1540 PRINT"{2 SPACES}DURATION OF NOTES IS IN 1/16T
     HS, SO 8={2 SPACES}8/16THS, OR A HALF-NOTE."
1550 PRINT"{2 SPACES}AN ANSWER OF '0' MEANS NO MOR
     E ENTRIES{2 SPACES}FOR THIS VOICE; A MINUS"
```

```
1560 PRINT"{2 SPACES}NUMBER MEANS A REST; -4=-4/16
     THS OR A{3 SPACES}QUARTER REST"
1570 PRINT
1580 PRINT"{2 SPACES}NOTES ARE ENTERED BY TYPING T
     HE LETTER{2 SPACES}-NAME (A THRU G).
     {2 SPACES}THE"
1590 PRINT"{2 SPACES}NAME FOLLOWED BY '+' MEANS SH
     ARP, '-'{3 SPACES}MEANS{2 SPACES}FLAT.
     {2 SPACES}SO G+ MEANS"
1600 PRINT"{2 SPACES}G SHARP AND G- MEANS G FLAT."
1610 PRINT
1620 PRINT"{2 SPACES}IF YOU ANSWER THE 'NOTE' QUES
     TION WITH{2 SPACES}A '+' BUT NO LETTER, THE"
1630 PRINT"{2 SPACES}OCTAVE WILL INCREASE (HIGHER)
     ; A'-'{5 SPACES}(ONLY) WILL LOWER THE"
1640 PRINT"{2 SPACES}OCTAVE."
1650 PRINT"{10 SPACES}..................."
1660 PRINT "{4 SPACES}PRESS THE SPACE BAR FOR MORE
      HELP."
1670 GET A$: IF A$<>" " GOTO 1670
1680 GOSUB 1400
1690 PRINT"{2 SPACES}AN{SHIFT-SPACE}EXAMPLE
     {SHIFT-SPACE}SCREEN MIGHT{SHIFT-SPACE}LOOK
     {SHIFT-SPACE}LIKE:"
1700 PRINT
1710 PRINT"{9 SPACES}SCREEN{15 SPACES}MEANS"
1720 PRINT"
     ***********************
     {2 SPACES}*************"
1730 PRINT"{4 SPACES}DURATION?{2 SPACES}8
     {10 SPACES}(1/2 NOTE)"
1740 PRINT"OCTAVE=4{6 SPACES}NOTE?{2 SPACES}+
     {4 SPACES}(UP AN OCTAVE)"
1750 PRINT"OCTAVE=5{6 SPACES}NOTE?{2 SPACES}C"
1760 PRINT"{4 SPACES}DURATION?{2 SPACES}-1
     {9 SPACES}(1/16 REST)"
1770 PRINT"{4 SPACES}DURATION?{2 SPACES}4
     {10 SPACES}(1/4 NOTE)"
1780 PRINT"OCTAVE=5{6 SPACES}NOTE?{2 SPACES}F+
     {3 SPACES}(F SHARP)"
1790 PRINT"{4 SPACES}DURATION?{2 SPACES}0
     {10 SPACES}(END VOICE)"
1800 PRINT:PRINT"{3 SPACES}PRESS{SHIFT-SPACE}THE
     {SHIFT-SPACE}SPACE{SHIFT-SPACE}BAR
     {SHIFT-SPACE}TO{SHIFT-SPACE}RETURN
     {SHIFT-SPACE}TO{SHIFT-SPACE}THE{SHIFT-SPACE}M
     ENU"
1810 PRINT
1820 GET A$:IF A$<>" " THEN 1820
```

```
1830 PRINTCHR$(142):GOTO 110
5000 REM **** ENTER/EDIT SONG ****
5010 GOSUB 1400:REM - CLR SCREEN
5020 PRINTTAB(10);"ENTER/EDIT SONG":PRINT:INPUT "
     {5 SPACES}SONG TITLE ";F$
5030 FOR K=0 TO 2
5040 PRINT:PRINT"{4 SPACES}DATA FOR VOICE #";K+1
5050 PRINT
5060 I=0
5070 N$="":NT=0:DUR%=0
5080 INPUT "{10 SPACES}DURATION";DUR%
5090 IF DUR%<0 THEN 6060:REM=A REST
6000 IF DUR%=0 THEN 6150:REM=NEXT VOICE
6010 PRINT "{4 SPACES}OCTAVE = ";OCT%;
6020 INPUT " :{3 SPACES}NOTE{4 SPACES}";N$
6030 IF LEFT$(N$,1)="+" THEN OCT%=OCT%+1:GOTO 6010
6040 IF LEFT$(N$,1)="-" THEN OCT%=OCT%-1:GOTO 6010
6050 GOSUB 9000:REM-CHANGE NAME/NOTE #
6060 WA=V(K):IF DUR%<0THENDUR%=-DUR%:WA=1
6070 FR=FQ(N)
6080 IF OCT%=7 THEN 6100
6090 FOR J=6 TO OCT%STEP-1:FR=FR/2:NEXT
6100 HF%=FR/256:LF%=FR-256*HF%
6110 IF DUR%=1 THEN H%(K,I)=HF%:L%(K,I)=LF%:C%(K,I
     )=WA:I=I+1:GOTO 5070
6120 FOR J=1 TO DUR%-1:H%(K,I)=HF%:L%(K,I)=LF%:C%(
     K,I)=WA:I=I+1:NEXT
6130 H%(K,I)=HF%:L%(K,I)=LF%:C%(K,I)=WA-1
6140 I=I+1:GOTO 5070
6150 IFI>IM THEN IM=I
6160 NEXT K
6170 GOTO 110
9000 REM ---- CHANGE NOTE LETTER TO #
9010 IF N$="C" THEN N=0:RETURN
9020 IF N$="C+" THEN N=1:RETURN
9030 IF N$="D-" THEN N=1:RETURN
9040 IF N$="D" THEN N=2:RETURN
9050 IF N$="D+" THEN N=3:RETURN
9060 IF N$="E-" THEN N=3:RETURN
9070 IF N$="E" THEN N=4:RETURN
9080 IF N$="F" THEN N=5:RETURN
9090 IF N$="F+" THEN N=6:RETURN
9100 IF N$="G-" THEN N=6:RETURN
9110 IF N$="G" THEN N=7:RETURN
9120 IF N$="G+" THEN N=8:RETURN
9130 IF N$="A-" THEN N=8:RETURN
9140 IF N$="A" THEN N=9:RETURN
9150 IF N$="A+" THEN N=10:RETURN
9160 IF N$="B-" THEN N=10:RETURN
```

```
9170 IF N$="B" THEN N=11:RETURN
9180 RETURN
10000 REM **** LOAD SONG-TAPE ****
10010 GOSUB 1400
10020 PRINTTAB(10);"LOAD SONG FROM TAPE":PRINT
10030 INPUT"{9 SPACES}TITLE ";F$
10040 OPEN 1,1,0,F$:REM DISK USERS OPEN1,8,2,F$
10050 INPUT# 1,IM
10060 FOR K=0 TO 2
10070 I=0
10100 INPUT# 1,L%(K,I),H%(K,I),C%(K,I)
10110 IF L%(K,I)=0 THEN 10130
10120 I=I+1:GOTO 10100
10130 NEXT K
10140 CLOSE 1
10150 GOTO 110
15000 REM **** PLAY SONG ****
15010 GOSUB 1400
15020 PRINT "{11 SPACES}PLAY ";F$:PRINT
15030 INPUT "{5 SPACES}TEMPO (NORM=80)";TEMPO
15100 POKES+10,8:POKES+22,128:POKES+23,244
15110 POKES+5,0:POKES+6,240
15120 POKES+12,85:POKES+13,133
15130 POKES+19,10:POKES+20,197
15140 POKES+24,31
15150 FOR I=0 TO IM
15160 POKES,L%(0,I):POKES+7,L%(1,I):POKES+14,L%(2,I)
15170 POKES+1,H%(0,I):POKES+8,H%(1,I):POKES+15,H%(
      2,I)
15180 POKES+4,C%(0,I):POKES+11,C%(1,I):POKES+18,C%
      (2,I)
15190 FOR T=1 TO TEMPO:NEXT:NEXT
15200 FOR T=1 TO 200:NEXT:POKES+24,0
15210 GOTO 110
20000 REM **** SAVE SONG - TAPE ****
20010 GOSUB 1400
20020 PRINTTAB(10);"SAVE ";F$:PRINT
20030 C$=","
20040 OPEN1,1,2,F$:REM DISK USER OPEN 1,8,2,F$+",S
      ,W"
20050 PRINT# 1,IM
20060 FOR K=0 TO 2
20070 I=0
20100 PRINT# 1,L%(K,I)C$H%(K,I)C$C%(K,I)
20110 IF L%(K,I)=0 THEN 20130
20120 I=I+1:GOTO 20100
20130 NEXT K
20140 CLOSE 1
20150 GOTO 110
```

Chapter Six

MusicMaster

Chris Metcalf and Marc Sugiyama

This excellent program simulates a realtime, full-function, synthesizer control panel for Commodore 64 sound and music. Your keyboard becomes the connection between you and the sounds you hear. The screen displays a double piano keyboard and the status of the other elements of the sounds you are creating.

The functions of "MusicMaster" include: slide, one-key access to all the primary chords, timbre, envelope, duration, octave, maintain, polyphony, waveform, and others. All available immediately and automatically from the keyboard.

The power and versatility of the 64's "music synthesizer on a chip" offer the programmer-musician extraordinary control over sound: its shape, color, even interactions between sounds (modulation). There is much freedom, but this also means that there are many aspects of each sound for the programmer to control. MusicMaster automates this control: for example, you can play chords as easily as single notes. Above all, you'll learn the meaning of the various sound registers — because you'll hear the effect as you change the registers. Now you can begin to fully explore the amazing sounds of the 64.

Enter the "MusicMaster" program into your Commodore 64 as you would enter any other BASIC program. MusicMaster includes two short machine language subroutines in DATA statements, so be certain that all those numbers are entered correctly. After you have entered and saved the program, run it. Be sure that the volume of your television or audio output device is turned up enough so that you can hear the computer.

Shortly before the message PLEASE STAND BY has left the screen, the computer will display the instructions. Across the top of the screen, you will find a row of indicators. The first item on this row is the OCTAVE, which has a range from 1 to 8. This is followed by the VOICE number, which indicates the particular *timbre* of your output. After this is a series of letters which indicate the current mode of operation. These modes will be described below. The last indicator is the VOLUME, with a range of 0 to 15.

Chapter Six

The Double Keyboard

Under the indicator line are the two musical keyboards. They indicate where on the computer's keyboard the musical keyboards can be found. The lower keyboard is a continuation of the upper keyboard; thus the lower set of keys plays the higher notes.

Below the keyboards is a description of the functions assigned to the programmable function keys. The left column describes the unshifted function keys, and the right column describes the shifted function keys.

f1 and **f3**: These keys allow you to change the volume of the music. Pressing f1 will increase the volume one step, and pressing f3 will decrease the volume one step. Notice how the VOLUME indicator changes as you press either one of these keys. Remember that the volume ranges from 0 to 15; 0 is completely silent, and 15 is the maximum volume.

f4: Pressing f4 will change the status of the Maintain mode, indicated by the M in the indicator row. When this mode is in operation, the M will be in reverse field. When this mode is activated, the computer does not release the tones after the keys have been pressed. Instead, the tones continue until other keys are pressed. To silence all the voices, press the space bar.

f6: This key changes the status of the Multivoice mode. This mode is indicated by the V in the indicator row. A reverse field V indicates that the mode is in operation. The Multivoice mode enables more than one voice to be played at the same time. The program powers on with this mode activated. If this mode is not activated, then one tone follows the next on the same voice and chords cannot be played. This has some disadvantages, but it is useful in conjunction with the Slide mode. With this mode, you can have up to three simultaneous voices.

f7 and **f5**: Pressing these keys changes the status of the Slide and Chord modes. They will be described below.

f2: This key allows you to define your own waveforms.

Making Music

Once the program is ready, press the following key sequence: QWERTYUI. You should hear a C major scale. If you do not, check the program for typing errors. Now try this key sequence: IOP@*(up arrow)(RUN/STOP)Z. This time you should hear the same scale, but one octave higher.

Pressing the sequence ZXCVBNM, produces another scale one octave higher than the last. Now try pressing the keys QET all at once to get a C major chord. Each note of this chord is assigned one voice. Since there are only three voices, the computer can accept only three keys at one time as input.

If you want to change octaves, press the control key and a number from 1 to 8, 1 being the lowest octave and 8 the highest. Some of the voices do not work well in very low octaves. Pressing the Commodore key and a number will change the VOICE number. This, too, has a range from 1 to 8.

The Slide mode is very interesting. A reverse-field S on the status row indicates that the Slide mode is active. The Slide mode will work regardless of the Multivoice and Maintain modes. When in this mode, the computer steps smoothly through the tones rather than moving by half tones as a piano would. This can produce an intriguing, eerie effect with the Maintain mode activated. For example, enter the Slide mode, make sure that the Maintain and Multivoice modes are activated, and press the following key sequence: QETIP*ZCB,. As always, you can silence the voices by pressing the space bar.

Forming Chords

Another mode of operation is the Chord mode. This allows for *single key* control over different types of chords and their inversions. Once you activate the Chord mode, a second indicator row appears. On the left is the chord name, and on the right is the chord position — root, first inversion, or second inversion.

The root chord is a chord in which the lowest note is also the key of the chord. For example, the C major triad is formed using the notes C, E, and G. When the notes are in that order, CEG, the chord is a root chord. If the notes of the chord start on a different note than C, then we have the inversions of the chord. For example, E and G, with high C, is the first inversion, and G, with high C and E, is the second inversion.

To change the chord type, press the SHIFT key and a number from 1 to 9. The chords which are available correspond to the following numbers: (1) Major; (2) Minor; (3) Diminished; (4) Augmented; (5) Major Seventh; (6) Minor Seventh; (7) Dominant Seventh; (8) Major Sixth; (9) Minor Sixth.

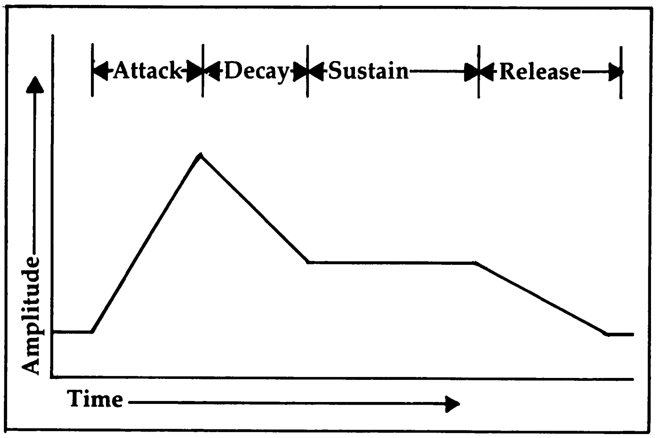
The inversions are selected by pressing the SHIFT key and the plus sign for root, the minus sign for the first inversion, and the pound sign for the second inversion.

In order to play a chord, you must first select the chord type and inversion that you want, and then press the note on the keyboard which corresponds to the lowest note of your chord. For example, if you want to play a B-flat minor second inversion chord, enter the chord mode, select the minor chord and the second inversion (by pressing SHIFT-2 and SHIFT- £) and press R, which corresponds to the note F on the musical keyboard. The chord that you will hear is comprised of the following notes: F, B-flat, and high D-flat. (Since the Slide mode can slide only one voice at a time, the Chord and Slide modes are incompatible, so turning on one automatically turns off the other.)

Attack, Decay, Sustain, Release

To define your own waveform, press f2. Once you are in this mode, the computer asks a series of questions that apply to the construction of a new waveform. The first question is which voice you wish to change. Pressing RETURN with no other input returns program control to the play mode. After this question, the computer displays the current Attack, Decay, Sustain, and Release values, and asks for new values. Pressing RETURN with no other input or giving a bad input returns you to the first question.

The Envelope



The attack rate is the time that it takes the sound to reach its highest volume level. The larger the number, the more time it takes. Decay is the time it takes the sound to drop to the Sustain volume level. Sustain is the volume level at which the sound remains until the Release is initiated. The Release rate is the time that it takes the sound to soften from the sustain level to silence (see the figure).

After these questions, the computer asks for the waveform type. You must enter the first letter of the type of waveform desired. If the pulse waveform is selected, then the pulse rate must be entered. The authors of the Commodore 64 manual have written the pulse value as two numbers, the LOW pulse and the HIGH pulse. To obtain a single value for the pulse rate, take the HIGH pulse times 256 and add it to the LOW pulse. Once these questions have been answered, the computer returns to the playing mode with the voice set to the one you have just modified.

Program Structure

The mechanics are fairly simple since most of the program is written in BASIC. The REMs identify the major sections of the program (see the table for a description of variables). However, some programming tricks are used. The POKE214,X command moves the cursor to line X on the screen. But a PRINT with no statement must follow this POKE or the cursor will not move to its new location. A POKE 788,52 disables the RUN/STOP key, but this can be annoying when listing programs. To reenable the RUN/STOP key, POKE 788,49. WAIT is also employed when waiting for input (WAIT 198,255).

The SYSS1 (to 49152) is a full keyboard scan routine for the Commodore 64. This routine is very useful because it allows the user to enter more than one key at a time.

The machine language routine returns the ASCII values of the keys being pressed to addresses 830, 831, and 832. (Due to a hardware problem involving the way the keyboard is wired, certain combinations of keys yield incorrect values.) The number of keys being pressed is stored in location 829. This routine could be used by games in which a multiple input is required.

A second machine language subroutine simply loads the values from 900-906 into the appropriate voice in the sound chip. Select the increment for voices 0, 1, and 2 (0, 7, or 14), POKE 251 with this value, then SYS(49408). The subroutine does not start the note, but leaves it to BASIC, via a POKE to the sound chip (SID), for the corresponding voice.

Chapter Six

Variables

A	miscellaneous
A$	miscellaneous
AD	attack/decay for define waveform routine
AD( )	table of attack/decay values
BF	constant pointer to buffer (198)
C$( )	table of chord names
C( )	table of chord note offsets
C1	chord number
C2	chord inversion
CH	chord mode flag
ER	INPUT routine error flag
ET	constant pointer for multikey input routine
FF	constant 255
FH( )	table of high bytes of frequencies
FL( )	table of low bytes of frequencies
HB	256 constant
I	miscellaneous
IK	constant for "inkey" or keyboard matrix value
IN	value for input from INPUT routine
IN$	input string from INPUT routine
J	miscellaneous
K( )	conversion table for ASCII values
LL	polyphonic flag
LN$	constant line
MN	multivoice flag
NH	constant high byte location 901
NL	constant low byte location 900
NM$( )	"root," "first," or "second" (for chord inversion display
OC	number of half steps offset (octave)
P	maintain mode flag
PH( )	table of pulse high bytes
PL( )	table of pulse low bytes
PU	pulse rate for define waveform routine
R	frequency number and miscellaneous
RA	slide mode register start pointer
RB	slide mode register end pointer
S	constant 54272
S1	constant 49152 (for multikey GET routine)
S2	constant 49403 (for music loader routine)
SL	slide mode flag
SP$	constant 39 spaces (for blanking)
SR	sustain/release value for define waveform routine

SR( )	table of sustain/release values
T	current base address of SID
T( )	table of last used base locations
V	computer voice number
VL	volume
VN	constant voice number location for music loader (251)
WF	waveform holder for define waveform routine
WV	current waveform
WV( )	table of waveform values

All variables beginning with "Z" are low numeric constants.

MusicMaster

```
1000 GOTO1260
1010 :
1020 :
1030 REM SLIDE SUBROUTINE
1040 IFRA<0THENRA=R
1050 RB=R:T=S+V*Z7:POKEVN,V*Z7:POKENL,FL(RA):POKEN
     H,FH(RA):SYSS2:POKET+Z4,WV+Z1
1060 FORI=RATORBSTEPSGN(RB-RA)/2:POKET,FL(I):POKET
     +1,FH(I):NEXT
1070 IFPEEK(IK)=JANDPEEK(IK)-64THEN1070
1080 RA=RB:POKET+Z4,WV+P:V=V+MN*(Z1+Z3*(V=Z2)):RET
     URN
1090 :
1100 REM CHORD SUBROUTINE
1110 POKEBF,Z0:FORI=Z0TOZ2:A=R+C(C1,C2,I):POKEVN,I
     *Z7:POKENL,FL(A)
1120 POKENH,FH(A):SYS S2:NEXT:POKES+Z4,WV+Z1:POKES
     +11,WV+Z1:POKES+18,WV+Z1
1130 IFPEEK(IK)=JANDPEEK(IK)-64THEN1130
1140 POKES+Z4,WV+P:POKES+11,WV+P:POKES+18,WV+P:RET
     URN
1150 :
1160 REM POLYPHONIC SUBROUTINE
1170 A=PEEK(IK):SYS S1:J=PEEK(ET):IFJ=Z0ORA=ZSTHEN
     RETURN
1180 FORI=Z1TOJ:R=K(PEEK(ET+I))+OC:IFR=OCTHENNEXT:
     RETURN
1190 T(I)=V*Z7:POKEVN,T(I):POKENL,FL(R):POKENH,FH(
     R):SYS S2
1200 IFMNTHENV=V+Z1:IFV=Z3THENV=Z0
1210 NEXT:FORI=Z1TOJ:POKES+T(I)+Z4,WV+Z1:NEXT
1220 SYS S1:IFJ=PEEK(ET)ANDA=PEEK(IK)THEN1220
1230 FORI=Z1TOJ:POKES+T(I)+Z4,WV+P:NEXT:GOTO1170
1240 :
```

```
1250 :
1260 REM INITIALIZE VARIABLES
1270 PRINT"{CLR}"CHR$(142)CHR$(8);:POKE53280,0:POK
     E53281,0:POKE788,52
1280 FORI=1TO39:SP$=SP$+" ":LN$=LN$+"{T}":NEXT
1290 PRINT"{5}OCTAVE=5{2 SPACES}VOICE=1 :C:S:M:
     {RVS}V{OFF}:{RVS}P{OFF}: VOLUME=10{RIGHT}"LN$
1300 POKE214,23:PRINT:PRINT"MUSICMASTER ..........
     .. {8}COMPUTE! BOOKS{5}{HOME}{2 DOWN}
1310 A$="PLEASE STAND BY{5}":POKE214,21:PRINT:PR
     INTTAB(12)"{WHT}"A$:S=54272:GOSUB2390
1320 DIMFL(134),FH(134),K(255),C(8,2,2):OC=48:VL=1
     0:MN=1:LL=1:RA=-1
1330 Z1=1:Z2=2:Z3=3:Z4=4:Z7=7:ZS=64:FF=255:HB=256
1340 IK=197:BF=198:VN=251:NL=900:NH=901:ET=829:S1=
     49152:S2=49408:FORI=1TO41
1350 K(ASC(MID$("Q2W3ER5T6Y7UI9O0P@-*£↑{HOME}
     {STOP}ZSXDCVGBHNJM,L.:/",I)))=I:NEXT
1360 PRINTTAB(12)"{8}{UP}"A$:R=5.8:A=10787.4138:
     J=2↑(-1/12)
1370 FORI=94TO0STEP-1:FH(I)=INT(A*R/HB):FL(I)=A*R-
     HB*FH(I):A=A*J:NEXT
1380 PRINTTAB(12)"{UP}"A$:GOSUB2120
1390 :
1400 REM READ ALL DATA
1410 FORA=49152TO49294:READIN:POKEA,IN:NEXT
1420 FORA=49408TO49454:READIN:POKEA,IN:NEXT
1430 FORI=0TO8:FORJ=0TO2:READC(I,J,0),C(I,J,1),C(I
     ,J,2):NEXT:READC$(I):NEXT
1440 READNM$(0),NM$(1),NM$(2):FORI=1TO8:READAD(I),
     SR(I),WV(I),PL(I),PH(I):NEXT
1450 PRINTTAB(9)"{DOWN}(USE CONTROL-X TO EXIT)":I=
     1:GOSUB1670
1460 :
1470 :
1480 REM NUCLEUS
1490 WAIT BF,FF:J=PEEK(IK):GETA$:R=K(ASC(A$))+OC:I
     FR=OCTHENGOSUB1610:GOTO1490
1500 IFSLTHENGOSUB1040:GOTO1490
1510 IFCHTHENGOSUB1110:GOTO1490
1520 IFLLTHENGOSUB1170:GOTO1490
1530 T=S+V*Z7:POKEVN,V*Z7:POKENL,FL(R):POKENH,FH(R
     ):SYS S2:POKET+Z4,WV+Z1
1540 IFMNTHENV=V+Z1:IFV=Z3THENV=Z0
1550 IFPEEK(IK)=JANDPEEK(IK)-64THEN1550
1560 POKET+Z4,WV+P:WAIT BF,FF:GETA$:J=PEEK(IK):R=K
     (ASC(A$))+OC:IFR<>OCTHEN1530
1570 GOSUB1610:GOTO1490
1580 :
```

```
159Ø :
16ØØ REM PARAMETER FUNCTIONS
161Ø IFCH=ØTHEN164Ø
162Ø FORI=ØTO2:IFA$=MID$("+-£",I+1,1)THENC2=I:PRI
     NT"{HOME}{DOWN}"TAB(23)NM$(I):RETURN
163Ø NEXT:A=ASC(A$):IFA>32ANDA<42THENC1=A-33:PRINT
     "{HOME}{DOWN}"TAB(11)C$(C1):RETURN
164Ø FORI=1TO8:IFA$<>MID$("{BLK}{WHT}{RED}{CYN}
     {PUR}{GRN}{BLU}{YEL}",I,1)THENNEXT:GOTO166Ø
165Ø OC=12*(I-Z1):PRINT"{HOME}"TAB(7)MID$(STR$(I),
     2):RETURN
166Ø FORI=1TO8:IFA$<>MID$("⁅1⁆⁅2⁆⁅3⁆⁅4⁆
     ⁅5⁆⁅6⁆⁅7⁆⁅8⁆",I,1)THENNEXT:GOTO169Ø
167Ø POKE9Ø2,PL(I):POKE9Ø3,PH(I):WV=WV(I):POKE9Ø4,
     WV:POKE9Ø5,AD(I):POKE9Ø6,SR(I)
168Ø PRINT"{HOME}"TAB(16)MID$(STR$(I),2):RETURN
169Ø IFA$<>"{F1}"ANDA$<>"{F3}"THEN174Ø
17ØØ VL=VL-(VL<15ANDA$="{F1}")+(VL>ØANDA$="{F3}"):
     POKES+24,VL
171Ø PRINT"{HOME}"TAB(37)RIGHT$("Ø"+MID$(STR$(VL),
     2),2):RETURN
172Ø :
173Ø REM STYLE FUNCTIONS
174Ø IFA$="{F4}"THENP=1-P:POKE1Ø47,13+128*P:GOTO23
     9Ø
175Ø IFA$="{F6}"THENMN=1-MN:POKE1Ø49,22+128*MN:GOT
     O239Ø
176Ø IFA$="{F8}"THENLL=1-LL:POKE1Ø51,16+128*LL:RET
     URN
177Ø IFA$="{F7}"THENSL=1-SL:RA=-1:POKE1Ø45,19+128*
     SL:CH=1:GOTO18ØØ
178Ø IFA$<>"{F5}"THEN182Ø
179Ø POKE1Ø45,19:SL=Ø
18ØØ CH=1-CH:POKE1Ø43,3+128*CH:IFCH=ØTHENPRINT"
     {HOME}{DOWN}"LN$:PRINTSP$:RETURN
181Ø PRINT"{HOME}{DOWN}"SP$"{RIGHT}{UP}CHORD TYPE:
     "C$(C1)TAB(23)NM$(C2)" INVERSION{RIGHT}"LN$:R
     ETURN
182Ø IFA$=" "THENGOSUB239Ø:RA=-1:POKEBF,ZØ:RETURN
183Ø IFA$="{X}"THENGOSUB239Ø:PRINT"{CLR}";:POKE788
     ,49:END
184Ø IFA$<>"{F2}"THENRETURN
185Ø :
186Ø :
187Ø REM DISPLAY WAVEFORM PARAMETERS
188Ø GOSUB228Ø:POKE214,13:PRINT
189Ø PRINT"VOICE TO BE DEFINED (1-8)";:J=1:GOSUB23
     1Ø
19ØØ IFIN<1ORIN>8THENGOSUB228Ø:GOTO221Ø
```

```
1910 I=IN:PRINTTAB(31)"ATT:"MID$(STR$(INT(AD(I)/16
     )),2)
1920 PRINTTAB(31)"DEC:"MID$(STR$(AD(I)AND15),2)
1930 PRINTTAB(31)"SUS:"MID$(STR$(INT(SR(I)/16)),2)
1940 PRINTTAB(31)"REL:"MID$(STR$(SR(I)AND15),2)
1950 PRINTTAB(31)"WVF:⁅8⁆"MID$("SAWTRIPULNSE",3*
     LOG(WV(I))/LOG(2)-11,3)"⁅5⁆
1960 IFWV(I)=64THENPRINTTAB(31)"PLS:"MID$(STR$(PH(
     I)*HB+PL(I)),2)
1970 :
1980 REM DEFINE A NEW WAVEFORM
1990 POKE214,14:PRINT:PRINT"ATTACK RATE (Ø-15)";:J
     =2:GOSUB231Ø:IFERTHEN188Ø
2000 AD=IN:PRINT"DECAY RATE (Ø-15)";:GOSUB231Ø:IFE
     RTHEN188Ø
2010 AD=AD*16ORIN:PRINT"SUSTAIN LEVEL (Ø-15)";:GOS
     UB231Ø:IFERTHEN188Ø
2020 SR=IN:PRINT"RELEASE RATE (Ø-15)";:GOSUB231Ø:I
     FERTHEN188Ø
2030 SR=SR*16ORIN:PRINT"⁅8⁆S⁅5⁆AW ⁅8⁆T⁅5⁆R
     IANGLE ⁅8⁆P⁅5⁆ULSE ⁅8⁆N⁅5⁆OISE";:J=1:
     GOSUB231Ø
2040 FORJ=1TO4:IFIN$<>MID$("STPN",J,1)THENNEXT:GOT
     O188Ø
2050 WF=2↑(J+3):IFWF<>64THEN2Ø7Ø
2060 PRINT"PULSE RATE (Ø-4Ø95)";:J=4:GOSUB231Ø:PU=
     IN:IFIN<ØORIN>4Ø95THEN188Ø
2070 WV(I)=WF:PL(I)=PU-HB*INT(PU/HB):PH(I)=INT(PU/
     HB):AD(I)=AD:SR(I)=SR
2080 GOSUB228Ø:GOSUB222Ø:GOTO167Ø
2090 :
2100 :
2110 REM DISPLAY KEYBOARDS
2120 POKES+24,VL:PRINT"{HOME}{3 DOWN}"TAB(9)"⁅M⁆
     {RVS} {RIGHT} {RIGHT} - {RIGHT} {RIGHT}
     {RIGHT} - {RIGHT} {RIGHT} - {RIGHT} {RIGHT}
     {RIGHT} "
2130 PRINT"{2 SPACES}LOW{4 SPACES}⁅M⁆{RVS} {OFF}
     2{RVS} {OFF}3{RVS} - {OFF}5{RVS} {OFF}6{RVS}
     {SPACE}{OFF}7{RVS} - {OFF}9{RVS} {OFF}Ø{RVS}
     {SPACE}- {OFF}-{RVS} {OFF}£{RVS} S "
2140 PRINT"KEYBOARD ⁅M⁆{RVS} - - - - - - - - - -
      - - - "
2150 PRINTTAB(9)"⁅M⁆{RVS}Q-W-E-R-T-Y-U-I-O-P-@-*
     -↑- "
2160 PRINTTAB(13)"{DOWN}⁅N⁆{RVS} {RIGHT} {RIGHT}
      - {RIGHT} {RIGHT} {RIGHT} - {RIGHT} {RIGHT}
     {SPACE}{OFF}⁅H⁆"
```

```
217Ø PRINT"{2 SPACES}HIGH{7 SPACES}ƐNƎ{RVS}
     {OFF}S{RVS} {OFF}D{RVS} - {OFF}G{RVS} {OFF}H
     {RVS} {OFF}J{RVS} - {OFF}L{RVS} {OFF}:{RVS}
     {OFF}ƐHƎ"
218Ø PRINT"KEYBOARD{5 SPACES}ƐMƎ{RVS} - - - - -
     {SPACE}- - - - {OFF}ƐHƎ"
219Ø PRINTTAB(13)"ƐNƎ{RVS}Z-X-C-V-B-N-M-,-.-/
     {OFF}ƐHƎ"
22ØØ :
221Ø REM DISPLAY FUNCTION MENU
222Ø POKE214,13:PRINT:PRINT"F1 -- LOUDER{5 SPACES}
     F2 -- DEFINE WAVEFORM
223Ø PRINT"{DOWN}F3 -- SOFTER{5 SPACES}F4 -- Ɛ8Ǝ
     MAINTAINƐ5Ǝ
224Ø PRINT"{DOWN}F5 -- Ɛ8ƎCHORDSƐ5Ǝ{5 SPACES}F
     6 -- Ɛ8ƎMULTIVOICEƐ5Ǝ
225Ø PRINT"{DOWN}F7 -- Ɛ8ƎSLIDESƐ5Ǝ{5 SPACES}F
     8 -- Ɛ8ƎPOLYPHONICƐ5Ǝ":RETURN
226Ø :
227Ø REM CLEAR DISPLAY AREA
228Ø POKE214,12:PRINT:FORJ=1TO11:PRINTSP$:NEXT:RET
     URN
229Ø :
23ØØ REM INPUT SUBROUTINE
231Ø IN$="":PRINT"? ";
232Ø PRINT"{RVS} {OFF}{LEFT}";:WAITBF,FF:GETA$:IFA
     $="{X}"THEN183Ø
233Ø A=ASC(A$):IFA=13THENPRINT" ":IN=VAL(IN$):ER=(
     IN<ØORIN>15)ORIN$="":RETURN
234Ø IFA=2ØANDLEN(IN$)THENPRINT" {2 LEFT} {LEFT}";
     :IN$=LEFT$(IN$,LEN(IN$)-1)
235Ø IF(AAND127)<35ORLEN(IN$)=JTHEN232Ø
236Ø PRINTA$;:IN$=IN$+A$:GOTO232Ø
237Ø :
238Ø REM CLEAR MUSIC CHIP
239Ø FORI=4TO18STEP7:POKES+I,Ø:NEXT:FORI=ØTO23:POK
     ES+I,Ø:NEXT:RETURN
24ØØ :
241Ø :
242Ø REM MULTI-INPUT ASSEMBLY CODE
243Ø DATA 12Ø,169,Ø,141,61,3,17Ø,169,254,133,252,1
     65,252,141,Ø,22Ø,173,1,22Ø
244Ø DATA 157,143,192,232,56,38,252,176,239,162,Ø,
     16Ø,Ø,189,143,192,42,176,29
245Ø DATA 72,132,253,138,1Ø,1Ø,1Ø,5,253,168,185,79
     ,192,238,61,3,172,61,3,153
246Ø DATA 61,3,1Ø4,192,3,24Ø,12,164,253,2ØØ,192,8,
     2Ø8,219,232,224,8,2Ø8,2Ø9,88
```

```
2470 DATA 96,17,135,134,133,136,29,13,20,0,69,83,9
     0,52,65,87,51,88,84,70,67,54
2480 DATA 68,82,53,86,85,72,66,56,71,89,55,78,79,7
     5,77,48,74,73,57,44,64,58,46
2490 DATA 45,76,80,43,47,94,61,1,19,59,42,92,3,81,
     2,32,50,4,95,49
2500 :
2510 REM MUSICLOADER ASSEMBLY CODE
2520 DATA 169,212,133,252,169,0,160,6,145,251,136,
     145,251,170,169,8
2530 DATA 136,145,251,138,145,251,136,192,1,208,24
     9,188,41,193,185
2540 DATA 132,3,145,251,232,224,6,208,243,96,2,3,0
     ,1,6,5
2550 :
2560 REM CHORD DATA
2570 DATA 0,4,7,0,3,8,0,5,9,"MAJOR{5 SPACES}",0,3,
     7,0,4,9,0,5,8,"MINOR{5 SPACES}"
2580 DATA 0,3,6,0,3,9,0,6,9,"DIMINISHED",0,4,8,0,4
     ,8,0,4,8,"AUGMENTED "
2590 DATA 0,4,11,0,4,11,0,4,11,"MAJOR 7TH ",0,3,10
     ,0,3,10,0,3,10,"MINOR 7TH "
2600 DATA 0,4,10,0,4,10,0,4,10,"DOMIN 7TH",4,7,9,4
     ,7,9,4,7,9,"MAJOR 6TH "
2610 DATA 3,7,9,3,7,9,3,7,9,"MINOR 6TH","
     {2 SPACES}ROOT"," FIRST","SECOND"
2620 :
2630 REM WAVEFORM PARAMETER DATA
2640 DATA 0,249,16,0,0, 0,249,32,0,0, 0,249,64,160
     ,15, 0,249,128,0,0
2650 DATA 0,240,16,0,0, 204,204,16,0,0, 0,252,64,2
     00,0, 192,240,32,0,0
```

Appendices

A Beginner's Guide to Typing In Programs

What Is a Program?

A computer cannot perform any task by itself. Like a car without gas, a computer has *potential,* but without a program, it isn't going anywhere. Most of the programs published in this book are written in a computer language called BASIC. BASIC is easy to learn and is built into all Commodore 64s.

BASIC Programs

Computers can be picky. Unlike the English language, which is full of ambiguities, BASIC usually has only one right way of stating something. Every letter, character, or number is significant. A common mistake is substituting a letter such as O for the numeral 0, a lowercase l for the numeral 1, or an uppercase B for the numeral 8. Also, you must enter all punctuation such as colons and commas just as they appear in the book. Spacing can be important. To be safe, type in the listings *exactly* as they appear.

Braces and Special Characters

The exception to this typing rule is when you see the braces, such as {DOWN}. Anything within a set of braces is a special character or characters that cannot easily be listed on a printer. When you come across such a special statement, refer to Appendix B "How to Type In Programs."

About DATA Statements

Some programs contain a section or sections of DATA statements. These lines provide information needed by the program. Some DATA statements contain actual programs (called machine language); others contain graphics codes. These lines are especially sensitive to errors.

If a single number in any one DATA statement is mistyped, your machine could lock up, or crash. The keyboard and STOP key may seem dead, and the screen may go blank. Don't panic — no damage is done. To regain control, you have to turn off your computer, then turn it back on. This will erase whatever program was in memory, *so always SAVE a copy of your program before you RUN it.* If your computer crashes, you can LOAD the program and look for your mistake.

Sometimes a mistyped DATA statement will cause an error message when the program is RUN. The error message may refer to the program line that READs the data. *The error is still in the DATA statements, though.*

Get to Know Your Machine

You should familiarize yourself with your computer before attempting to type in a program. Learn the statements you use to store and retrieve programs from tape or disk. You'll want to save a copy of your program, so that you won't have to type it in every time you want to use it. Learn to use your machine's editing functions. How do you change a line if you made a mistake? You can always retype the line, but you at least need to know how to backspace. Do you know how to enter reverse video, lowercase, and control characters? It's all explained in your computer's manuals.

A Quick Review

1) Type in the program a line at a time, in order. Press RETURN at the end of each line. Use backspace or the back arrow to correct mistakes.
2) Check the line you've typed against the line in the book. You can check the entire program again if you get an error when you RUN the program.

How to Type In Programs

To make it easy to know exactly what to type when entering one of these programs into your computer, we have established the following listing conventions.

Generally, Commodore 64 program listings will contain words within braces which spell out any special characters: {DOWN} would mean to press the cursor down key. {5 SPACES} would mean to press the space bar five times.

To indicate that a key should be *shifted* (hold down the SHIFT key while pressing the other key), the key would be underlined in our listings. For example, S would mean to type the S key while holding the SHIFT key. This would appear on your screen as a heart symbol. If you find an underlined key enclosed in braces (e.g., {10 N}), you should type the key as many times as indicated (in our example, you would enter ten shifted N's).

If a key is enclosed in special brackets, [<>], you should hold down the *Commodore key* while pressing the key inside the special brackets. (The Commodore key is the key in the lower-left corner of the keyboard.) Again, if the key is preceded by a number, you should press the key as many times as necessary.

Rarely, you'll see a solitary letter of the alphabet enclosed in braces. These characters can be entered by holding down the CTRL key while typing the letter in the braces. For example, {A} would indicate that you should press CTRL-A.

About the *quote mode*: you know that you can move the cursor around the screen with the CRSR keys. Sometimes a programmer will want to move the cursor under program control. That's why you see all the {LEFT}'s, {HOME}'s, and {BLU}'s in our programs. The only way the computer can tell the difference between direct and programmed cursor control is the quote mode.

Once you press the quote (the double quote, SHIFT-2), you are in the quote mode. If you type something and then try to change it by moving the cursor left, you'll only get a bunch of reverse-video lines. These are the symbols for cursor left. The only editing key that isn't programmable is the DEL key; you can still use DEL to back up and edit the line. Once you type another quote, you are out of quote mode.

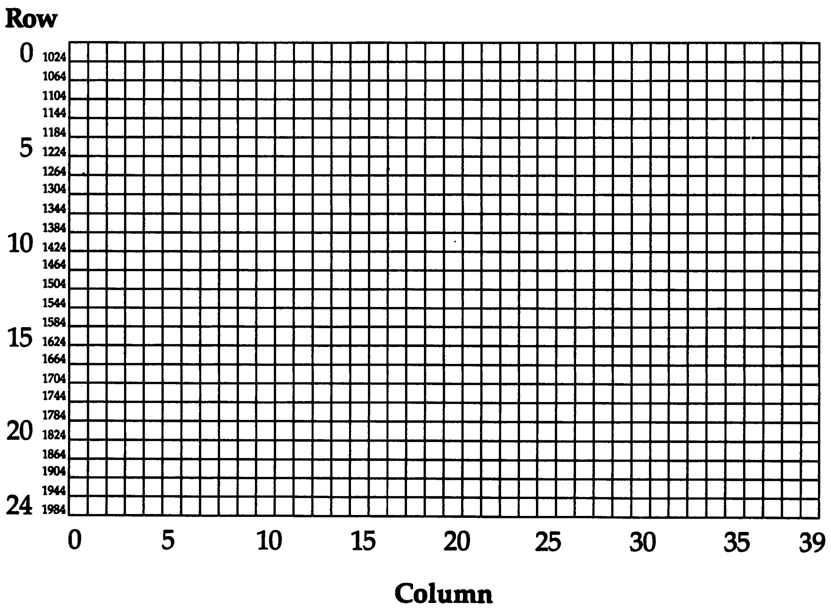
Appendix B

You also go into quote mode when you INSerT spaces into a line. In any case, the easiest way to get out of quote mode is to just press RETURN. You'll then be out of quote mode and you can cursor up to the mistyped line and fix it.

Use the following table when entering cursor and color control keys:

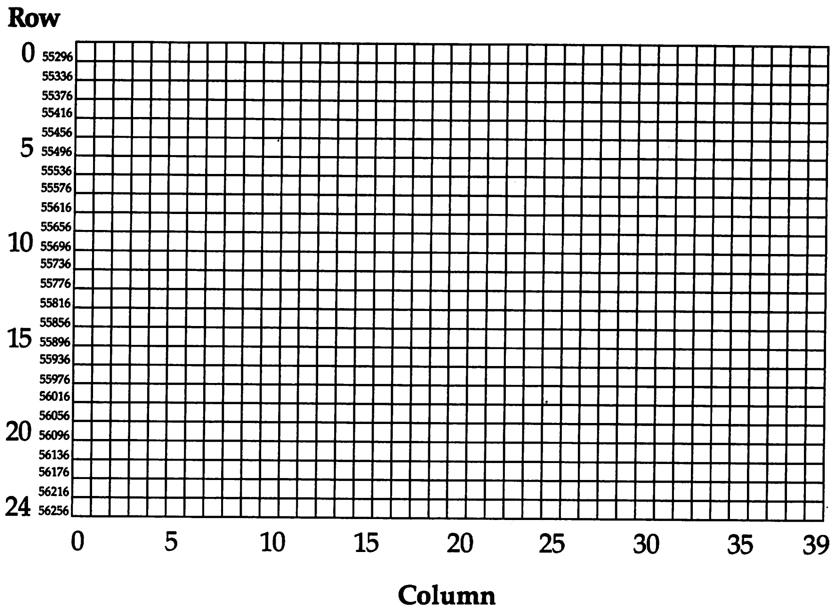
When You Read:	Press:	See:	When You Read:	Press:	See:
{CLR}	SHIFT CLR/HOME	■	{ORANGE}	COMMODORE f1	■
{HOME}	CLR/HOME	■	{BROWN}	COMMODORE f2	■
{UP}	SHIFT ↑CRSR↓	■	{LT/RED}	COMMODORE f3	■
{DOWN}	↑CRSR↓	■	{DK/GREY}	COMMODORE f4	■
{LEFT}	SHIFT ←CRSR→	■	{MED/GREY}	COMMODORE f5	■
{RIGHT}	←CRSR→	■	{LT/GREEN}	COMMODORE f6	■
{RVS}	CTRL 9	■	{LT/BLUE}	COMMODORE f7	■
{OFF}	CTRL 0	■	{LT/GREY}	COMMODORE f8	■
{BLACK}	CTRL 1	■	{F-1}	f1	■
{WHITE}	CTRL 2	■	{F-2}	f2	■
{RED}	CTRL 3	■	{F-3}	f3	■
{CYAN}	CTRL 4	■	{F-4}	f4	■
{PURPLE}	CTRL 5	■	{F-5}	f5	■
{GREEN}	CTRL 6	■	{F-6}	f6	■
{BLUE}	CTRL 7	■	{F-7}	f7	■
{YELLOW}	CTRL 8	■	{F-8}	f8	■
			£	£	■

Screen Location Table



Appendix D

Screen Color Memory Table



Screen Color Codes

Value To POKE For Each Color

Color	Low nybble color value	High nybble color value	Select multicolor color value
Black	0	0	8
White	1	16	9
Red	2	32	10
Cyan	3	48	11
Purple	4	64	12
Green	5	80	13
Blue	6	96	14
Yellow	7	112	15
Orange	8	128	-
Brown	9	144	-
Light Red	10	160	-
Dark Grey	11	176	-
Medium Grey	12	192	-
Light Green	13	208	-
Light Blue	14	224	-
Light Grey	15	240	-

Where To POKE Color Values For Each Mode

Mode*	Bit or bit-pair	Location	Color value
Regular text	0	53281	Low nybble
	1	Color memory	Low nybble
Multicolor	00	53281	Low nybble
text	01	53282	Low nybble
	10	53283	Low nybble
	11	Color memory	Select multicolor
Extended	00	53281	Low nybble
color text #	01	53282	Low nybble
	10	53283	Low nybble
	11	53284	Low nybble
Bitmapped	0	Screen memory	Low nybble ±
	1	Screen memory	High nybble ±
Multicolor	00	53281	Low nybble
bitmapped	01	Screen memory	High nybble ±
	10	Screen memory	Low nybble ±
	11	Color memory	Low nybble

* For all modes, the screen border color is controlled by POKEing location 53280 with the low nybble color value.

# In extended color mode, bits 6 and 7 of each byte of screen memory serve as the bit-pair controlling background color. Because only bits 0-5 are available for character selection, only characters with screen codes 0-63 can be used in this mode.

± In the bitmapped modes, the high and low nybble color values are ORed together and POKEd into the *same location* in screen memory to control the colors of the corresponding *cell* in the bitmap. For example, to control the colors of cell 0 of the bitmap, OR the high and low nybble values and POKE the result into location 0 of screen memory.

Appendix F

ASCII Codes

ASCII	CHARACTER	ASCII	CHARACTER
5	WHITE	50	2
8	DISABLE	51	3
	SHIFT COMMODORE	52	4
9	ENABLE	53	5
	SHIFT COMMODORE	54	6
13	RETURN	55	7
14	LOWERCASE	56	8
17	CURSOR DOWN	57	9
18	REVERSE VIDEO	58	:
19	HOME	59	;
20	DELETE	60	<
28	RED	61	=
29	CURSOR RIGHT	62	>
30	GREEN	63	?
31	BLUE	64	@
32	SPACE	65	A
33	!	66	B
34	"	67	C
35	#	68	D
36	$	69	E
37	%	70	F
38	&	71	G
39	'	72	H
40	(	73	I
41	)	74	J
42	*	75	K
43	+	76	L
44	,	77	M
45	–	78	N
46	.	79	O
47	/	80	P
48	0	81	Q
49	1	82	R

Appendix F

ASCII	CHARACTER	ASCII	CHARACTER
83	S	120	♣
84	T	121	▕
85	U	122	♦
86	V	123	┼
87	W	124	▒
88	X	125	│
89	Y	126	π
90	Z	127	◥
91	[	129	ORANGE
92	£	133	f1
93	]	134	f3
94	↑	135	f5
95	←	136	f7
96	─	137	f2
97	♠	138	f4
98	│	139	f6
99	─	140	f8
100	▔	141	SHIFTED RETURN
101	▔	142	UPPERCASE
102	▁	144	BLACK
103	▏	145	CURSOR UP
104	▕	146	REVERSE VIDEO OFF
105	╮	147	CLEAR SCREEN
106	╰	148	INSERT
107	╯	149	BROWN
108	└	150	LIGHT RED
109	╲	151	GRAY 1
110	╱	152	GRAY 2
111	┌	153	LIGHT GREEN
112	┐	154	LIGHT BLUE
113	●	155	GRAY 3
114	▁	156	PURPLE
115	♥	157	CURSOR LEFT
116	▏	158	YELLOW
117	╭	159	CYAN
118	╳	160	SPACE
119	○	161	▌

Appendix F

ASCII	CHARACTER	ASCII	CHARACTER
162	▄	200	▕
163	□	201	╮
164	□	202	╰
165	□	203	╯
166	▒	204	□
167	□	205	╲
168	▄	206	╱
169	◤	207	□
170	▐	208	□
171	⊞	209	●
172	▗	210	□
173	┗	211	♥
174	┓	212	▏
175	▂	213	╭
176	┏	214	╳
177	┻	215	○
178	┳	216	♣
179	┫	217	▕
180	▏	218	♦
181	▌	219	┼
182	▐	220	▌
183	▔	221	│
184	▀	222	π
185	▂	223	◥
186	□	224	SPACE
187	▖	225	▌
188	▝	226	▄
189	┛	227	□
190	▘	228	□
191	▚	229	□
192	─	230	▒
193	♠	231	□
194	│	232	▄
195	─	233	▧
196	▔	234	▐
197	▔	235	⊞
198	▁	236	▗
199	▏	237	┗

Appendix F

ASCII	CHARACTER
238	
239	
240	
241	
242	
243	
244	
245	
246	
247	
248	
249	
250	
251	
252	
253	
254	
255	

0-4, 6, 7, 10-12, 15, 16, 21-27, 128, 130-132, and 143 are not used.

Appendix G

Screen Codes

POKE	Uppercase and Full Graphics Set	Lower- and Uppercase	POKE	Uppercase and Full Graphics Set	Lower- and Uppercase
0	@	@	31	←	←
1	A	a	32	-space-	
2	B	b	33	!	!
3	C	c	34	"	"
4	D	d	35	#	#
5	E	e	36	$	$
6	F	f	37	%	%
7	G	g	38	&	&
8	H	h	39	'	'
9	I	i	40	(	(
10	J	j	41	)	)
11	K	k	42	*	*
12	L	l	43	+	+
13	M	m	44	,	,
14	N	n	45	–	–
15	O	o	46	.	.
16	P	p	47	/	/
17	Q	q	48	0	0
18	R	r	49	1	1
19	S	s	50	2	2
20	T	t	51	3	3
21	U	u	52	4	4
22	V	v	53	5	5
23	W	w	54	6	6
24	X	x	55	7	7
25	Y	y	56	8	8
26	Z	z	57	9	9
27	[	[	58	:	:
28			59	;	;
29	]	]	60	<	<
30	↑	↑	61	=	=

Appendix G

POKE	Uppercase and Full Graphics Set	Lower- and Uppercase	POKE	Uppercase and Full Graphics Set	Lower- and Uppercase
62	>	>	99	▔	▔
63	?	?	100	▁	▁
64	─	─	101	▏	▏
65	♠	A	102	▒	▒
66	│	B	103	▕	▕
67	─	C	104	▄	▄
68	▔	D	105	◤	▨
69	▔	E	106	▐	▐
70	▁	F	107	├	├
71	▏	G	108	▗	▗
72	▕	H	109	└	└
73	╮	I	110	┐	┐
74	╰	J	111	▂	▂
75	╯	K	112	┌	┌
76	⌞	L	113	┴	┴
77	╲	M	114	┬	┬
78	╱	N	115	┤	┤
79	⌜	O	116	▎	▎
80	⌝	P	117	▌	▌
81	●	Q	118	▐	▐
82	▁	R	119	▔	▔
83	♥	S	120	▀	▀
84	▏	T	121	▄	▄
85	╭	U	122	┘	✓
86	╳	V	123	▖	▖
87	○	W	124	▝	▝
88	♣	X	125	┘	┘
89	▕	Y	126	▘	▘
90	♦	Z	127	▚	▚
91	┼	┼	128 – 255 reverse video of 0-127		
92	▒	▒			
93	│	│			
94	π	▒			
95	◥	▨			
96	-space-				
97	▌	▌			
98	▄	▄			

Appendix H

Commodore 64 Keycodes

Key	Keycode	Key	Keycode
A	10	6	19
B	28	7	24
C	20	8	27
D	18	9	32
E	14	0	35
F	21	+	40
G	26	–	43
H	29	£	48
I	33	CLR/HOME	51
J	34	INST/DEL	0
K	37	←	57
L	42	@	46
M	36	*	49
N	39	↑	54
O	38	:	45
P	41	;	50
Q	62	=	53
R	17	RETURN	1
S	13	,	47
T	22	.	44
U	30	/	55
V	31	CRSR↑↓	7
W	9	CRSR ⇄	2
X	23	f1	4
Y	25	f3	5
Z	12	f5	6
1	56	f7	3
2	59	SPACE	60
3	8	RUN/STOP	63
4	11	NO KEY	
5	16	PRESSED	64

The keycode is the number found at location 197 for the current key being pressed. Try this one-line program:

```
10 PRINT PEEK (197): GOTO 10
```

Using the Machine Language Editor: MLX

Charles Brannon

Remember the last time you typed in a long machine language program? You typed in hundreds of DATA statements, numbers, and commas. Even then, you couldn't be sure if you'd typed it in right. So you went back, proofread, tried to run the program, crashed, went back and proofread again, corrected a few typing errors, ran again, crashed, rechecked your typing .... Frustrating, wasn't it?

Until now, though, that has been the best way to enter machine language into your machine. Unless you happen to own an assembler and are willing to wrangle with machine language on the assembly level, it is much easier to enter a BASIC program that reads the DATA statements and POKEs the numbers into memory.

Some of these "BASIC loaders" will use a checksum to see if you've typed the numbers correctly. The simplest checksum is just the sum of all the numbers in the DATA statements. If you make an error, your checksum will not match up. Some programmers have made your task easier by creating checksums every ten lines, so you can zero in on your errors.

But wait! MLX comes to the rescue! MLX is a great way to enter all those long machine language programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to BASIC DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255. It will prevent you from entering the wrong numbers on the wrong line. In short, MLX will make proofreading obsolete!

Boot Disks

In addition, MLX will generate a ready-to-use tape or disk file. You can then use the LOAD command to read the program into the computer, just like with any program. Specifically, you enter:

LOAD "program",1,1 (for tape)

or

LOAD "program",8,1 (for disk)

To start the program, you need to enter a SYS command that transfers control from BASIC to machine language. The starting SYS will always be given in the appropriate article.

Using MLX

Type in and save MLX (you'll want to use it in the future). When you're ready to type in the ML program, RUN it. The program will ask you for two numbers: the start address and the ending address. These numbers should be 36864 and 39947 respectively, for "Hi-Res Sketchpad." For "Ultrafont," the starting address is 49152; the ending address, 52139.

The prompt is the current line you are entering from the listing. Each line is six numbers plus a checksum. If you enter any of the six numbers wrong, or enter the checksum wrong, the 64 will ring the buzzer and prompt you to reenter the line. If you enter it correctly, a pleasant bell tone will sound and you go on and enter the next line.

A Special Editor

You are not using the normal Commodore 64 editor with MLX. For example, it will only accept numbers as input. If you need to make a correction, press the <INST/DEL> key; the entire number is deleted. You can press it as many times as necessary back to the start of the line. If you enter three-digit numbers as listed, the computer will automatically print the comma and go on to accept the next number. If you enter less than three digits, you can press either the comma, space bar, or RETURN key to advance to the next number. The checksum will automatically appear in inverse video; don't worry — it's highlighted for emphasis.

When testing it, I've found it to be extremely easy to enter long listings. With the audio cues provided, you don't even have to look at the screen if you're a touch-typist.

Done at Last!

When you get through typing, assuming you type it all in one session, you can then save the completed and bug-free program to tape or disk. Follow the screen instructions. If you get any errors while writing, you probably have a bad disk, or the disk was full, or you've made a typo when entering the MLX program. (Sorry, it can't check itself!)

Command Control

What if you don't want to enter the whole program in one sitting? MLX lets you enter as much as you want, save the whole schmeer, and then reload the file from tape or disk when you want to continue. MLX recognizes these few commands:

SHIFT-S: Save

SHIFT-L: Load

SHIFT-N: New Address

SHIFT-D: Display

Hold down SHIFT while you press the appropriate key. You will jump out of the line you've been typing, so I recommend you do it at a new prompt. Use the Save command to save what you've been working on. It will write the tape or disk file as if you've finished, but the tape or disk won't work, of course, until you finish the typing. Remember what address you stop on. The next time you RUN MLX, answer all the prompts as you did before, then insert the disk or tape. When you get to the entry prompt, press SHIFT-L to reload the file into memory. You'll then use the New Address command to resume typing.

New Address and Display

After you press SHIFT-N, enter the address where you previously stopped. The prompt will change, and you can then continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing, or else the checksum won't match up. You can use the Display command to display a section of your typing. After you press SHIFT-D, enter two addresses within the line number range of the listing. You can abort the listing by pressing any key.

Tricky Stuff

The special commands may seem a little confusing, but as you work with MLX, they will become valuable. For example, what if you forgot where you stopped typing? Use the Display command to scan memory from the beginning to the end of the program. When you see a bunch of 170's, stop the listing (press a key) and continue typing where the 170's start. Some programs contain many sections of 170's. To avoid typing them, you can use the New Address command to skip over the blocks of 170's. Be careful, though; you don't want to skip over anything you *should* type.

You can use the Save and Load commands to make copies of the completed game. Use the Load command to reload the tape or disk, then insert a new tape or disk and use the Save command to create a new copy.

One quirk about tapes made with the Save command: when you load them, the message "FOUND program" may appear twice. The tape will load just fine, however. Once the Hi-Res Sketchpad is loaded, type SYS 36864 <RETURN> to run the program. For Ultrafont, SYS 49152.

Programmers will find MLX to be an interesting program, in terms of protecting the user from mistakes. There is also some screen formatting. Most interesting is the use of ROM Kernal routines for LOADing and SAVEing blocks of memory. Just POKE the starting address (low byte/high byte) into 251 and 252 and POKE the ending address into 254 and 255. Any error code can be found in location 253 (an error would be a code less than ten).

I hope you will find MLX to be a true labor-saving program. Since it has been tested by entering actual programs, you can count on it as an aid for generating bug-free machine language.

Machine Language Editor

```
100 PRINT"{CLR}{RED}";CHR$(142);CHR$(8);:POKE53281
    ,1:POKE53280,1
101 POKE 788,52:REM DISABLE RUN/STOP
110 PRINT"{RVS}{40 SPACES}";
120 PRINT"{RVS}{15 SPACES}{RIGHT}{OFF}E*3£{RVS}
    {RIGHT} {RIGHT}{2 SPACES}E*3{OFF}E*3£
    {RVS}£{RVS}{13 SPACES}";
130 PRINT"{RVS}{15 SPACES}{RIGHT} EG3{RIGHT}
    {2 RIGHT} {OFF}£{RVS}£E*3{OFF}E*3{RVS}
    {13 SPACES}";
140 PRINT"{RVS}{40 SPACES}"
150 V=53248:POKE2040,13:POKE2041,13:FORI=832TO894:
    POKEI,255:NEXT:POKEV+27,3
160 POKEV+21,3:POKEV+39,2:POKEV+40,2:POKEV,144:POK
    EV+1,54:POKEV+2,192:POKEV+3,54
170 POKEV+29,3
```

```
180 FORI=ØTO23:READA:POKE679+I,A:POKEV+39,A:POKEV+
    4Ø,A:NEXT
185 DATA169,251,166,254,164,255,32,216,255,133,253
    ,96
187 DATA169,Ø,166,251,164,252,32,213,255,133,253,9
    6
19Ø POKEV+39,7:POKEV+4Ø,7
2ØØ PRINT"{2 DOWN}{PUR}{BLK}{3 SPACES}A FAILSAFE M
    ACHINE LANGUAGE EDITOR{5 DOWN}"
21Ø PRINT"E5]{2 UP}STARTING ADDRESS?{8 SPACES}
    {9 LEFT}";:INPUTS:F=1-F:C$=CHR$(31+119*F)
22Ø IFS<256OR(S>4Ø96ØANDS<49152)ORS>53247THENGOSUB
    3ØØØ:GOTO21Ø
225 PRINT:PRINT:PRINT
23Ø PRINT"E5]{2 UP}ENDING ADDRESS?{8 SPACES}
    {9 LEFT}";:INPUTE:F=1-F:C$=CHR$(31+119*F)
24Ø IFE<256OR(E>4Ø96ØANDE<49152)ORE>53247THENGOSUB
    3ØØØ:GOTO23Ø
25Ø IFE<STHENPRINTC$;"{RVS}ENDING < START
    {2 SPACES}":GOSUB1ØØØ:GOTO 23Ø
26Ø PRINT:PRINT:PRINT
3ØØ PRINT"{CLR}";CHR$(14):AD=S:POKEV+21,Ø
31Ø PRINTRIGHT$("ØØØØ"+MID$(STR$(AD),2),5);":";:FO
    RJ=1TO6
32Ø GOSUB57Ø:IFN=-1THENJ=J+N:GOTO32Ø
39Ø IFN=-211THEN 71Ø
4ØØ IFN=-2Ø4THEN 79Ø
41Ø IFN=-2Ø6THENPRINT:INPUT"{DOWN}ENTER NEW ADDRES
    S";ZZ
415 IFN=-2Ø6THENIFZZ<SORZZ>ETHENPRINT"{RVS}OUT OF
    {SPACE}RANGE":GOSUB1ØØØ:GOTO41Ø
417 IFN=-2Ø6THENAD=ZZ:PRINT:GOTO31Ø
42Ø IF N<>-196 THEN 48Ø
43Ø PRINT:INPUT"DISPLAY:FROM";F:PRINT,"TO";:INPUTT
44Ø IFF<SORF>EORT<SORT>ETHENPRINT"AT LEAST";S;"
    {LEFT}, NOT MORE THAN";E:GOTO43Ø
45Ø FORI=FTOTSTEP6:PRINT:PRINTRIGHT$("ØØØØ"+MID$(S
    TR$(I),2),5);":";
451 FORK=ØTO5:N=PEEK(I+K):PRINTRIGHT$("ØØ"+MID$(ST
    R$(N),2),3);",";
46Ø GETA$:IFA$>""THENPRINT:PRINT:GOTO31Ø
47Ø NEXTK:PRINTCHR$(2Ø);:NEXTI:PRINT:PRINT:GOTO31Ø
48Ø IFN<Ø THEN PRINT:GOTO31Ø
49Ø A(J)=N:NEXTJ
5ØØ CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT
51Ø PRINTCHR$(18);:GOSUB57Ø:PRINTCHR$(2Ø)
515 IFN=CKSUMTHEN53Ø
52Ø PRINT:PRINT"LINE ENTERED WRONG : RE-ENTER":PRI
    NT:GOSUB1ØØØ:GOTO31Ø
```

```
530 GOSUB2000
540 FORI=1TO6:POKEAD+I-1,A(I):NEXT:POKE54272,0:POK
    E54273,0
550 AD=AD+6:IF AD<E THEN 310
560 GOTO 710
570 N=0:Z=0
580 PRINT"{+}";
581 GETA$:IFA$=""THEN581
585 PRINTCHR$(20);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N670
590 IFA>128THENN=-A:RETURN
600 IFA<>20 THEN 630
610 GOSUB690:IFI=1ANDT=44THENN=-1:PRINT"{LEFT}
    {LEFT}";:GOTO690
620 GOTO570
630 IFA<48ORA>57THEN580
640 PRINTA$;:N=N*10+A-48
650 IFN>255 THEN A=20:GOSUB1000:GOTO600
660 Z=Z+1:IFZ<3THEN580
670 IFZ=0THENGOSUB1000:GOTO570
680 PRINT",";:RETURN
690 S%=PEEK(209)+256*PEEK(210)+PEEK(211)
691 FORI=1TO3:T=PEEK(S%-I)
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT
700 PRINTLEFT$("{3 LEFT}",I-1);:RETURN
710 PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}"
720 INPUT"{DOWN} FILENAME";F$
730 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"
740 GETA$:IFA$<>"T"ANDA$<>"D"THEN740
750 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$
760 OPEN 1,DV,1,F$:POKE252,S/256:POKE251,S-PEEK(25
    2)*256
765 POKE255,E/256:POKE254,E-PEEK(255)*256
770 POKE253,10:SYS 679:CLOSE1:IFPEEK(253)>9ORPEEK(
    253)=0THENPRINT"{DOWN}DONE.":END
780 PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN720
781 OPEN15,8,15:INPUT#15,DS,DS$:PRINTDS;DS$:CLOSE1
    5:GOTO720
790 PRINT"{CLR}{RVS}*** LOAD ***{2 DOWN}"
800 INPUT"{2 DOWN} FILENAME";F$
810 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"
820 GETA$:IFA$<>"T"ANDA$<>"D"THEN820
830 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$
840 OPEN 1,DV,0,F$:POKE252,S/256:POKE251,S-PEEK(25
    2)*256
850 POKE253,10:SYS 691:CLOSE1
```

```
860 IFPEEK(253)>9 OR PEEK(253)=0 THEN PRINT:PRINT:
    GOTO310
870 PRINT"{DOWN}ERROR ON LOAD.{2 SPACES}TRY AGAIN.
    {DOWN}":IFDV=1THEN800
880 OPEN15,8,15:INPUT#15,DS,DS$:PRINTDS;DS$:CLOSE1
    5:GOTO800
1000 REM BUZZER
1001 POKE54296,15:POKE54277,45:POKE54278,165
1002 POKE54276,33:POKE 54273,6:POKE54272,5
1003 FORT=1TO200:NEXT:POKE54276,32:POKE54273,0:POK
     E54272,0:RETURN
2000 REM BELL SOUND
2001 POKE54296,15:POKE54277,0:POKE54278,247
2002 POKE 54276,17:POKE54273,40:POKE54272,0
2003 FORT=1TO100:NEXT:POKE54276,16:RETURN
3000 PRINTC$;"{RVS}NOT ZERO PAGE OR ROM":GOTO1000
```

Index

COMPUTE!'s First Book of Commodore 64 Sound and Graphics

COMPUTE!'s First Book of Commodore 64 Sound and Graphics puts the TV screen and speakers under your control. Beginners will find good introductions to sound and graphics on the Commodore 64. More advanced users will find exciting, sophisticated techniques that will let you use the most advanced capabilities of the machine. And there are utilities that anyone can use — even if you've never programmed before.

Here is a sample of what you'll find in this book:

- Programs to help you add music to your programs.
- A sprite editor.
- Programs that let you change the characters printed on the screen to any shape you want, including pictures.
- A complete music synthesizer.
- A utility to create high-resolution, fine line graphics, like those you see in arcade games.
- A program that adds four new graphics commands to BASIC.
- Many clear, understandable explanations for how to program graphics.

ISBN 0-942386-21-3

$12.95