All About the Commodore 64

V · O · L · U · M · E T · W · O

Everything you need to know to create sophisticated music, sprites, and detailed bitmapped screens on your Commodore 64—with both the convenience of BASIC and the speed of machine language.

Craig Chamberlain

A **COMPUTE! Books** Publication

$16.95

All About the Commodore 64

V · O · L · U · M · E T · W · O

Craig Chamberlain

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

Copyright 1985, COMPUTE! Publications, Inc. All rights reserved

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America

ISBN 0-942386-45-0

10 9 8 7 6 5 4 3 2 1

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies and is not associated with any manufacturer of personal computers. Commodore 64 is a trademark of Commodore Electronics Limited.

Contents

Acknowledgments vii
Foreword ix
Introduction xi

Part 1: Advanced BASIC 1
Chapter 1. Numbers and Mathematics 3
Numerical Values for Conditions 3
Precision and Scientific Notation 8
Integer Variables 11
Random Number Seeding and Sequencing 15

Chapter 2. More Functions 19
Print Formatting Functions 19
Transcendental Functions 22
User-Defined Functions with the DEF Statement 26

Chapter 3. File Input/Output 37
Introduction to I/O 37
The OPEN and CLOSE Statements 38
The Printer and the CMD Statement 45
Files on Tape 50
Disk Files 58
The Modem and Other RS-232 Devices 71

Chapter 4. The End of BASIC 81
Using Commands in Programs 81
The WAIT Statement 84
Machine Language and the SYS Statement 88

Part 2: Bitmapped Graphics 95
Chapter 5. The Bitmapped Graphics Utility 97
Introduction to the Bitmapped Graphics Utility 97
The New Statements 97
Graphics Modes and Point Plotting 106
Changing Colors 110
Line Drawing 112
Area Filling 115
Multicolor Modes 119

Chapter 6. Shapedit 123
Introduction to Shape Tables 123
The Shapedit Program 130
Using Shapes in BASIC Programs 146
Rotation 150

Drawing Pen Indirection 153
Mixing Text with Bitmapped Graphics 164
Suggested Applications 166
The Technical Side 168
EXTRACT and MERGE 178
BMG Statement Index 181

Part 3: Sidplayer 183

Chapter 7. Introduction to Music and the Sidplayer 185
Electronic Music 185
The Sidplayer 186
Let the Music Play 190
Fundamentals of Music Theory 206

Chapter 8. The Editor 227
Using the Editor 249
EDIT Music 249
Special Options 255
Function Key Summary 260
Entering a Melody 260
Error Checking 262
Saving the Music 263
Loading a Song 264
What's on the Disk? 264
Using All Three Voices 264
Keyboard Note Entry 266
Complete Joystick Editing 267
Leaving the Editor: QUIT 268

Chapter 9. Making Music 271
Waveforms 271
Envelopes 282
Envelope Example 287
The Filters 290
Three Demonstrations 296
Repetition 298

Chapter 10. Sophisticated Sounds 305
Advanced Music Theory 305
Portamento and Vibrato 314
Detuning 320
Transposing 322
Synchronization 324
Ring Modulation 333
Advanced Techniques 337

Chapter 11. Music and Your BASIC Program 351
Merging Sidplayer with BASIC Programs 351
Utility Programs 364
Sidplayer Editor Command Index 375

Part 4: Sprite Control System 377

Chapter 12. Introduction to the Sprite Control System 379
Principles of Animation 379
Character Graphics 380
Bitmapped Graphics 380
Sprite Graphics 381
The Sprite Control System 381
Before You See It Work 382
On with the Show 389
Getting Fancy 393
Hi, Guy 398

Chapter 13. Defining Sprite Shapes 403
Joystick Editing 410
The Sprite Editor Commands and Features 410
Sprites and More Sprites 417

Chapter 14. Using the Control System in BASIC 419
Sophisticated 419
Variable Assignments 420
Load Machine Language Routines 420
Load Definitions 420
Install Sprite Control System 421
Define Block 421
Assign Block to Sprite 422
Set Sprite Color 423
Set Sprite Size 423
Set Sprite Position 423
Set Sprite Direction 425
Select Wraparound or Bounce 426
Set Sprite Speed 427
Set Sprite Boundaries 428
Remove Sprite Control System 429
Review 429
Errors 430
Multiple Sprites 430

Chapter 15. SCS Advanced Features 433
Available Definition Blocks 433
Automatic Shape Changing 434
Multicolor Mode 437

Priority 437
Joystick Control 438
Chase Mode 440
Synchronized Sprite Motion 441
Enable/Disable 443

Chapter 16. Miscellaneous Topics 445
The Merge Utility 445
The Extract Utility 447
Sprite-to-Sprite Collisions 449
Sprite-to-Screen-Data Collisions 450
POKEing and PEEKing 451
Reading Definitions from DATA 451
Protecting Free Memory 452
Relocating Screen Memory 452
Relocating the Sprite Control System 456
Using the Control System with the Bitmapped Graphics Extensions 457
Using the Sprite Control System with Sidplayer 459
Using All Three Utilities Together 460
Final Comments 462
Sprite Control System Command Index 463

Appendices 465
A. A Beginner's Guide to Typing In Programs 467
B. How to Type In Programs 469
C. Using the Machine Language Editor: MLX 471
D. The Automatic Proofreader 478
E. Program Merge Utility 482

Index 487

Acknowledgments

Over a year in development, this book represents the talents of many people. The author wishes to thank the following people for their assistance in designing and debugging the programs:

Bitmapped Graphics: Sheldon Leemon, Mark Davids, Bob Retelle, and Jerry Brady.

Sidplayer: Harry Bratt, Robert Higgins, Jerry Brady, Chuck Lever, Bob Retelle, Steve Maggs, Sharon Aker, Marc Sugiyama, Mark Freitas, and Jim Ockers.

Sprite Control System: Mark Thomas, Mark Davids, and Bob Retelle.

Special thanks are owed to Sheldon Leemon, author of *Mapping the Commodore 64*, published by COMPUTE! Publications, Inc. Sheldon wrote the BASIC interface for the bitmapped graphics routines, and generously offered technical assistance whenever it was needed. This book might not have been possible had it not been for Sheldon's help.

Also deserving special recognition are Harry Bratt, author of the Sidplayer Editor, and Mark Davids and Bob Retelle, for carefully reviewing the text.

Finally, the people at COMPUTE! Publications are to be thanked for their help and patience, which has been greatly appreciated by the author.

Foreword

Bitmap graphics...music and sound...sprites—these are some of the most powerful features of your Commodore 64. You know what the computer can do if you've seen high-quality commercial software. The 64 can paint any picture, play almost any kind of music, animate an endless variety of shapes. Using those same techniques in your own programs, however, isn't that simple. If you know machine language, you can do it a bit more easily. That still may not solve the problem though, for BASIC is often the language of convenience. And if you know only BASIC, what can you do?

All About the Commodore 64, Volume Two is the answer. It contains three major machine language utilities which add new BASIC commands for bitmap graphics creation, allow you to play the most involved music, and give you control over almost every aspect of sprites. Each utility is supported by an editor, as well as other secondary programs. Examples show you precisely what can be done and how to do it.

With bitmapped graphics you can draw, fill, and color on the bitmap screen. With shape tables, you can draw detailed shapes. By saving them to tape or disk, you can call them instantly to the screen. "Shapedit," a shape editor, makes it a snap to produce shapes. You can quickly change or copy any-

"Sidplayer" and its accompanying Editor allow you to create music of your own, or type sheet music into your 64. This editor is one of the most sophisticated available: You can use the keyboard or joystick to add or delete notes, specify the key, select the tempo, control volume, and number measures. Advanced features allow you to repeat sections of a musical piece, filter sound, and use the 64's SID chip's synchronization and ring modulation. Almost anything that can be played, from a symphony to a pop tune, can be reproduced on your 64. And though there are detailed explanations, you don't even need to know how to read music to use this versatile tool.

Sprites, those shapes you can sculpt and move across the screen, can be easily controlled with the Sprite Control System. With the complete package, including a sprite editor, you'll quickly design and create countless sprites, then move

them on the screen at incredible speeds. The computer will move them for you, or you can control them with the joystick. In a matter of minutes you can have sprites bouncing, disappearing, reappearing, crawling, and flying through your own programs. Even though you may know how to program just in BASIC, you can create impressive arcade-quality games with the Control System.

Each of these utilities can be used separately, or in any combination. They let you control, through your own BASIC programs, some of the most advanced features available on any home computer.

There's even more information in *All About the Commodore 64, Volume Two.* An entire section is devoted to advanced BASIC programming techniques. It picks up where Volume One left off. You'll learn about advanced functions, file input/output, and the little-known, but powerful, WAIT statement.

All the programs in Volume Two are ready to type in. Error-checking programs make typing them in easy.

If you prefer, you can purchase all the programs from Parts 2, 3, and 4 on disk for $12.95 plus shipping by calling toll free 1-800-334-0868.

With the information and utilities included in this book, you can turn your 64 into the computer you've always hoped it would become. Powerful features become convenient enough to use. Everything is clearly documented and illustrated, from the simplest technique to the most advanced. With *All About the Commodore 64, Volume Two,* you can have a new computer. It may look like your old 64, but it's not. It's a machine that's opened up hundreds of new doors into personal computing.

Introduction

Welcome to the second volume of COMPUTE! Books' comprehensive series, *All About the Commodore 64*. The first volume showed you everything you needed to write BASIC programs on your 64. But there's more built into your computer than just BASIC. There's a dedicated graphics chip which features superior bitmapped graphics and eight sprites, as well as a sound chip with capabilities comparable to those in many commercial synthesizers. No other computer on the market gives you such graphics and sound for such a low price.

Unfortunately, these features are not easily accessible from BASIC. Commodore 64 BASIC does not offer commands for high-resolution plotting, for playing music, or for animating sprites. These have to be done with POKE statements, but POKEs can be unwieldy and slow. In fact, BASIC programs which use POKEs just do not run fast enough to play three voices simultaneously or to control eight sprites at the same time.

One solution is to write the programs in machine language (ML), the "native" language of the Commodore 64. Machine language is so fast that it can clear the bitmap screen in a fraction of a second; it can even move sprites faster than the screen can display them. But to use ML you must first learn how to write it. Learning a new language can be time-consuming.

All About the Commodore 64, Volume Two offers an alternative. It describes three major utilities, written in machine language, which can be merged with your BASIC programs to do fast drawing, play music, or animate sprites. You can keep the convenience of BASIC while adding the speed of ML—all without having to plug a special cartridge into your computer.

This book is divided into four parts. Part 1 explains advanced features of BASIC; the remaining three each cover one utility, showing how to use it and suggesting some practical applications. The concise, but clear, approach of Volume One, including the use of summaries at the end of each section in Part 1, has been retained for this book. These two books supplement each other. What you didn't find in Volume One, you'll find here. Although you don't need Volume One to

understand this book, both are necessary to completely explore the capabilities of your computer.

Advanced BASIC

Part 1 picks up where *All About the Commodore 64, Volume One* left off. The advanced features of BASIC are explored, defined, and illustrated.

Chapters 1 and 2 cover subjects such as user-defined functions and how to simplify IF-THEN statements. Chapter 3 discusses peripherals, such as printers and modems. The advanced topic of tape and disk files is presented with numerous examples. A simple terminal program is even included for those of you who have a modem. Finally, Chapter 4 covers several miscellaneous items (how to call machine language from a BASIC program, for instance).

Since the topics in this section are somewhat advanced and may not be immediately applicable, you might want to skip over these chapters and move to the graphics and sound utilities first. However, there is a wealth of information included in this section—it's nice to know that it's there when you need it.

Bitmapped Graphics

This collection of machine language routines lets you work with high-resolution and multicolor graphics. To make them as accessible as possible, they're written in the form of statements which can be added to BASIC. Even more important, the routines don't take up a single byte of BASIC's free memory.

New BASIC statements like GRAPHICS and DRAW are used to clear the bitmap screen and do high-resolution plotting, while others let you quickly draw lines and fill in areas. The SHAPE statement lets you draw a complete object at any place on the screen. There's even a program—"Shapedit"—that helps you create a library of shapes.

Sidplayer

The SID (Sound Interface Device) chip built into your 64 is a rather complex integrated circuit. However, "Sidplayer," the music editing program included here, has been carefully designed so that everyone can use it. No particular musical knowledge is required; in fact, it's so easy to use that you can

enter a song without knowing how to read sheet music.

As well as detailing Sidplayer's advanced features, this section offers a discussion of the fundamentals of music theory.

Sidplayer is capable of emulating a wide variety of instruments, and any music you create with it can be merged with other BASIC programs. The music will play while the program is doing something else. Several demonstration tunes are even provided.

Sprite Control System

How would you like to see eight sprites go whizzing across the screen, each with a different direction, speed, shape, size, and color? What if you could control them with one or even two joysticks? And how about if they changed shape whenever they changed direction?

All of these things are possible with the Sprite Control System—and you don't need a single POKE statement! Even better, the animation takes place independently of BASIC. This means that sprites can move and change shape while a BASIC program is displaying a score or generating sound effects.

You'll find this utility especially useful for your own projects, such as action games. Programs include a Sprite Editor and plenty of demonstrations.

Each of these three utilities—Bitmapped Graphics, Sidplayer, and the Sprite Control System—can make your Commodore 64 do impressive things. That's not all, though—you'll also see ways to use all three together to create even more impressive results with your computer.

You Paid for It

All About the Commodore 64, Volume Two has something for everyone. You'll finally be able to make use of all the special features within your machine. They've always been there; it's just that they haven't been easy to get to. You paid for them—isn't it about time you used them?

Part 1
Advanced BASIC

Numbers and Mathematics

Numerical Values for Conditions

In *All About the Commodore 64, Volume One*, you saw how the AND and OR operators work on true/false values and on numbers. Actually, these operators work only on numbers; true and false have corresponding numerical values. Consider the following statement:

```
PRINT 5=5
```

Surprisingly enough, this BASIC statement will make your computer print a minus one (−1). Another (and somewhat clearer) way of writing this is:

```
PRINT (5=5)
```

Note that the equal sign is not assigning anything. Rather, it's being used as a relational operator, just like the greater than (>) and less than (<) symbols. The values to the equal sign's left and right are compared for equality. If the two values *are* equal—if the relation is true—BASIC associates a −1 with this condition.

However, if the values are *not* equal—if the relation is false—zero is associated with such a condition. For example, the following statement will print a zero:

```
PRINT 5=4
```

The next example shows both uses of the equal sign. The first equal sign is being used to assign the variable A, as in a LET statement. The second is comparing variables B and C for equality.

```
A=B=C
```

When this statement is executed, B and C will be compared first, and the result (0 or −1) will then be assigned to A. Variables B and C will not be changed.

The fact that true and false have numerical values can sometimes be used to simplify IF-THEN statements. These statements work by performing a comparison, and then using the result to decide whether to execute the commands after the THEN or to skip to the next line.

```
IF 5=5 THEN PRINT "TRUE"
```

The condition 5=5 is evaluated. Since both values are equal, the result is −1, and the instruction after THEN is executed. But look at this line:

IF 5=4 THEN PRINT "TRUE"

This time the result is 0, and the part after THEN does not execute.

Actually, the instructions following THEN will be executed whenever the result is any nonzero value, not just −1. In the first line 310 below, the PRINT statement after the THEN will execute if C has a value other than 0. Execution will skip to the next line only when C *equals* 0.

```
310 IF C THEN PRINT "C HAS NONZERO VALUE"
```

This line is just a shortened form of:

```
310 IF C<>0 THEN PRINT "C HAS NONZERO VALUE"
```

Thus, whenever a conditional statement checks to see if a value does not equal zero, the *<>0* portion can be eliminated.

This has a number of applications. For instance, here are two ways of checking for a sprite collision. (Variable SN is the number of the sprite being checked.)

IF (PEEK(53278)AND2↑SN)<>0 THEN PRINT "COLLIDED!"
IF PEEK(53278)AND2↑SN THEN PRINT "COLLIDED!"

Both lines produce the same results, but the second is a little shorter.

Not only can lines be shortened because conditions have numerical values, but they can be eliminated altogether. Consider a program which has to keep incrementing a color number. When the color reaches 16, it has to be set back to 1.

```
260 C=C+1 : IF C=16 THEN C=1
270 POKE 53280,C
```

Every time these lines are executed, C is incremented. The conditional statement insures that the value of C is always kept in the range from 1 to 15.

Lines 260 and 270 could be combined if it wasn't for the IF-THEN statement at the end of 260. This statement makes it

impossible to add other instructions to the end of the line, because they may not always be executed. However, the lines could be combined if the IF-THEN was eliminated. The following shows how it can be done, not in two lines as above, but in only one line.

```
26Ø C=C+1:C=C+15*(C=16):POKE 5328Ø,C
```

The logic may be hard to follow, so let's run through a short explanation. Whenever C has a value from 1 to 15, the statement C=C+15*(C=16) does not change the variable's value. For all these values, (C=16) is false, or 0. Since 15 times 0 is still 0, the statement effectively becomes C=C+0; the value of C is left unchanged. But when C is 16, (C=16) is true, or −1. Multiplying 15 by −1, however, gives −15, which when added to the value of C (16) yields 1. Notice that an addition operation has to be used because the value being added is negative. But the end result is subtraction. The process is outlined below.

C has the value 16
C=C+15*(C=16)
C=C+15*(−1)
C=C+(−15)
C=C−15
C=16−15
C=1

The example could be compacted even more by combining the C=C+1 with the C=C+15*(C=16). The following line produces the same result as the previous two examples.

```
26Ø C=C+1+15*(C=15):POKE 5328Ø,C
```

This statement behaves just like C=C+1 when C ranges from 1 to 14. When C is 15, however, adding 1 to it would make it 16, so the 15 is subtracted instead.

Here's a slight variation on this technique; it's taken from the "Shapedit" program found later in the book. The variable X represents a coordinate that must be kept in the range 0–39. If X is larger than 39, it must be set back to 0. If it's less than 0, it must wrap around to 39.

```
24Ø IF X<Ø THEN X=39
25Ø IF X>39 THEN X=Ø
```

Instead, one line was saved by writing the code this way:

```
240 IF X<0 OR X>39 THEN X=-39*(X<0)
```

If X is less than 0, the value −39*(X<0) is 39, and the variable is assigned the number 39. When X is greater than 39, the value −39*(X<0) is 0, and the variable is set to 0.

One other application of this number technique is to use it with the ON-GOTO statement. Consider the following program segment, which waits until the joystick is pushed, and then jumps to line 300.

```
320 JS=PEEK(56320)AND15:IF JS=15 GOTO 320
330 GOTO 300
```

Having a GOTO on a line all by itself seems wasteful. Line 330 is necessary, but anything put on the same line, after the GOTO, would never be executed. By taking advantage of the number trick, you can revise the lines and end up with:

```
320 JS=PEEK(56320)AND15:ON (JS=15)+2 GOTO 320,300
```

The possible values of (JS=15) are −1 and 0, which when added to 2 give the values 1 and 2, just right for using ON-GOTO. Here is another way of writing the line so that the numbers after GOTO can be transposed:

```
320 JS=PEEK(56320)AND15:ON 1-(JS=15) GOTO 300,320
```

Subtracting the numbers −1 and 0 (possible values of (JS=15)) from 1 gives 2 and 1, respectively.

These tricks and techniques with numerical values for conditions are not necessarily faster, because they often require the use of multiplication, a slow operation. They may also seem to be rather complicated. However, they can be used to reduce excessive IF-THEN and GOTO statements, which contribute to confusing program logic. It's up to you whether you want to use these techniques in your own programs, but at least you'll be able to recognize them in other programs.

Summary

- The values *true* and *false* have corresponding numerical values, −1 and 0.

- The statement PRINT A=5 will print a −1 if A equals 5, a 0 if A does not equal 5.
- In the statement PRINT A=5, the equal sign is used as a relational operator. The variable A is not being assigned and will keep its previous value.
- BASIC uses the numerical values in determining whether to execute instructions after THEN in an IF-THEN statement.
- In the statement IF A=5 THEN PRINT "TRUE", BASIC will execute instructions after THEN if the value of (A=5) is −1, and will skip to the next line if the value is 0.
- Actually, BASIC will execute instructions after THEN if the part between IF and THEN is any nonzero value. The statement IF A THEN PRINT means "IF A is not zero THEN PRINT" and will print a message if A has any value other than 0.
- The statement IF A THEN PRINT is identical to IF A<>0 THEN PRINT. Whenever <>*0* appears in a condition, it can be removed to shorten the line without affecting the logic.
- One disadvantage in using IF-THEN statements is that two lines cannot be combined when the first one contains such a statement. To do so would affect the program's logic. However, the fact that the values true and false have numerical values can be used to get around this problem.
- The statement C=C+1+15*(C=15) will increment the variable C and reset it to 1 if its new value would be 16. The end result is that C will always have a value from 1 to 15. This statement is comparable to C=C+1:IF C=16 THEN C=1, but is shorter and can be followed by more statements.
- When a GOTO statement is found on a line by itself, it can usually be combined with the previous line.
- If a GOTO cannot be combined because the previous statement is an IF-THEN, the ON-GOTO statement can be used instead. Each of the three following examples accomplishes the same thing.

```
350 JS=PEEK(56320)AND15:IF JS=15 GOTO 350
360 GOTO 300

350 JS=PEEK(56320)AND15:ON (JS=15)+2 GOTO 350,300

350 JS=PEEK(56320)AND15:ON 1-(JS=15) GOTO 300,350
```

- The number techniques introduced in this section are mainly used to shorten a program. They do not necessarily make the program any simpler.

Precision and Scientific Notation

Your Commodore 64 always performs mathematical calculations with a high degree of accuracy. If it tries to tell you that 2+2 is not 4, something is wrong with your computer. But there's a limit to how precise an answer can be. Commodore 64 BASIC uses a maximum of nine digits in representing numbers. If an answer needs more than nine, the least significant digits are dropped.

Here's an example.

PRINT 1234567 + 0.9876543

The answer should be 1234567.9876543, but that takes 14 digits. The most significant digits are the ones to the left, so only the first 9 are used. BASIC responds with the answer 1234567.99. Notice that the ninth digit is rounded up because the tenth digit would have been 7. Numbers are rounded up if the missing digit is greater than or equal to 5; otherwise they're rounded down.

Because BASIC can manage only nine digits, there are some numbers that BASIC can only approximate. The fraction 1/7 is an example. The decimal representation of 1/7 contains a sequence of digits which repeat forever.

1/7 = 0.142857142857142857...

But if you PRINT this fraction, you'll get only the first nine digits, the last a 3, rounded up from ...28.

PRINT 1/7
.142857143

This inability to exactly represent some numbers doesn't apply just to numbers with too many digits. The following program looks simple enough, but it will not work as you might expect.

```
10 FOR K=1 TO 10 STEP 0.01
20 PRINT K
30 NEXT K
```

The program starts printing the numbers correctly (1, 1.01, 1.02, and so on), but after 1.22 things begin to get

strange. The number 1.22999999 appears for 1.23. This continues for the rest of the program. To see why, enter the following statement.

PRINT 1.01 − 1

You'll see .0100000002. The number 0.01 is another of those values that BASIC cannot precisely represent. This can lead to some really frustrating problems if you're not aware of the limitations. For instance, enter this next line.

PRINT 1.01 − .01

BASIC prints 1. But the printed number may sometimes be slightly different from the machine's internal representation of the number. In the next example, the part after the THEN will *not* be executed.

IF 1.01 − .01 = 1 THEN PRINT "YES"

Because of the problems in computing .01, 1.01−.01 may print as 1 but yet not exactly equal 1.

Sometimes you can use programming and a little mathematics to get around these problems. The example that stepped in 0.01 increments could be rewritten like this:

```
10 FOR K=100 TO 1000
20 PRINT K/100
30 NEXT K
```

Other problems may not be as easy to solve, and will require more creative solutions. The IF-THEN statement above needs to be rewritten so that it's something like this:

N=1.01 − .01 : IF INT(N*100)=100 THEN PRINT "YES"

One other possibility must be considered. If BASIC can handle a maximum of only nine digits, what happens when it comes across a number which has more than nine digits to one side of the decimal? Can BASIC process numbers such as 1,234,567,890 (1.23456789 billion)? Yes, but it has to display them using a form of representation known as *scientific notation.*

PRINT 1234567890
1.23456789E+09

Any number can be represented as a number (the *mantissa*) times 10 raised to a power (the *exponent*). The *E* means *times*

10 to the ..., so *1.23456789E+09* means *1.23456789 times 10 to the ninth power.* This is commonly written as:

1.23456789×10^9

For practical purposes, *E+09* means *move the decimal point nine places to the right to get the correct value.* If 1.23456789 is adjusted, 1234567890 results. (Sometimes it's necessary to add one or more zero digits to move the decimal point completely.)

Scientific notation is convenient when representing numbers with a lot of digits, especially those which contain several zeros. The numbers 10000000000 and 1000000000 look very much the same (especially without the normal comma notation), but there's a difference of 9000000000 between them. Representing the numbers as 1E+10 and 1E+09 eliminates the need to count the zeros, and reduces the chance of error.

The use of scientific notation is not confined to the displaying of numbers, as in the following:

```
PRINT 1E8,1E9
 100000000       1E+09
```

You can also express numbers in scientific notation in response to INPUT statements, include them as values in DATA statements, and use them in most other places where numbers are used. The absolute maximum value that BASIC can represent is 1.70141183E+38. If a number exceeds this limit, BASIC will stop and print the OVERFLOW error.

Scientific notation works in the other direction as well. Values less than 0.01 are always represented using scientific notation.

```
PRINT 0.00001089
 1.089E-05
```

The minus sign means that the decimal point should be moved the designated number of places to the *left.* It does *not* mean that the number is negative.

There is no underflow error. Numbers less than 2.93873588E−39 are converted to zero.

```
PRINT 3.14E-40
 0
```

Summary

- Commodore 64 BASIC uses a maximum of nine digits to represent any number. If a number is made up of more than that, the nine most significant digits (leftmost digits) are used. The remaining digits are dropped.
- When digits have to be dropped, the nine-digit number is rounded up or down (depending on the tenth digit). If the tenth digit is 5 or more, the number is rounded up; otherwise, it's rounded down.
- Values need not have more than nine digits for the computer to have trouble representing them. Examples showed how the 64 has difficulty handling the number 0.01.
- The computer's internal representation of a number may be slightly different from the printed form. The result of 1.01 − .01 prints as 1, but is not equal to 1 in IF-THEN comparisons.
- Sometimes it's possible to rewrite a line or lines to reduce the problems associated with precision.
- Scientific notation can be used to represent numbers with more than nine digits to one side of the decimal point. Any number can be represented as a number from 0 to 9 times 10 raised to a power. The number 1234567890 is represented as 1.23456789E+09 in scientific notation. The *1.23456789* is called the *mantissa*, the *E* means *times 10 to the ...*, and the *+09* is called the *exponent*.
- In practical terms, *E+n*, where *n* is a number, means that the decimal point should be moved *n* places to the right, with zeros added if necessary.
- Scientific notation makes it easier to read numbers with lots of digits and reduces the chance of error.
- If a value exceeds 1.70141183E+38, BASIC will stop with an OVERFLOW error.
- All numbers of magnitude less than 0.01 are represented in scientific notation. The number 0.00123 becomes 1.23E−03, with *E−03* meaning that the decimal point should be moved to the left three places.
- Numbers less than 2.93873588E−39 are converted to 0. No error occurs when this happens.

Integer Variables

In *All About the Commodore 64, Volume One,* the concept of *type* was introduced. The variable type helps describe what

kind of information is being expressed by the variable. The types described in Volume One were *numbers* and *character strings.* The TYPE MISMATCH error occurred whenever these were improperly mixed, as in LET A="CHRIS".

There's a third type of variable that's not often used. An *integer variable* is denoted by a percent symbol (%) after the variable name, just as a string variable is indicated by a dollar sign ($). Integer variables are treated like other types of variables, so they can be used with such statements as PRINT, INPUT, READ, and so on. The difference lies in their possible values.

The numeric variables that have been used up until now are also known as *floating-point variables.* They can have fractional values, which are written in decimal form with digits to the right of the decimal. Integer variables cannot.

Integer numbers include the counting numbers (1, 2, 3, and so on), their negative counterparts (−1, −2, −3, ...), and 0. Fractional values cannot be represented by integers.

Integer variables can have integer values ranging from −32768 to +32767. Values outside this range cause an ILLEGAL QUANTITY error.

```
LET A%=5:PRINT A%      5
N%=-3:PRINT N%         -3
Q%=100000              ILLEGAL QUANTITY
```

If you try to assign a decimal value to an integer variable, the value is converted to an integer before it's assigned.

```
A%=5.5:PRINT A%        5
N%=6.9:PRINT N%        6
```

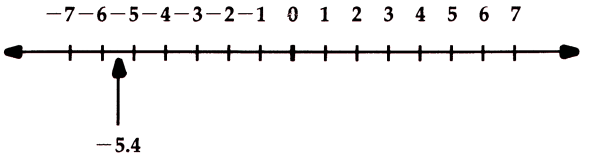
This conversion uses the same process as that used by the integer function (INT(X)). The INT function takes a decimal value and returns the largest integer that's less than or equal to the argument (the value within parentheses). For positive numbers, this has the effect of dropping all digits to the right of the decimal point. But for negative numbers, it works a little differently.

```
PRINT INT(-5)          -5
PRINT INT(-5.4)        -6
```

In the first example, the argument is already an integer, so no conversion is necessary. The second example returns −6 instead of −5, because −5 is actually *larger* than −5.4. You can more easily see this on a number line, as shown by Figure

1-1. Notice that the largest integer *less than* −5.4 is not −5, but −6.

Figure 1-1. A Number Line



Here's another example:

```
A%=-3 : PRINT A%          -3
N%=-3.427 : PRINT N%      -4
```

Integer and floating-point variables can be used together, since the variable named A% is completely different from the variable named A. Floating-point variable A does not change when integer variable A% is assigned, and vice versa. Integer variables can even take the values previously assigned to floating-point variables, although the values are converted into integer form. The process can be reversed as well. The next few lines illustrate all these points. Notice that the last line (IF N=N% THEN PRINT...) shows that N and N% are *not* equal.

```
Q=3.14159 : PI%=Q : PRINT PI%                    3
R%=32 : S=R% : PRINT S                           32
N=2 : N%=3 : IF N=N% THEN PRINT "YES"  (nothing printed)
```

The only exception to using integer and floating-point variables together is that the former can't be used as counting variables in FOR-NEXT loops. For some strange reason, Commodore 64 BASIC stops and displays a SYNTAX ERROR if you try to use an integer variable in this way.

```
FOR K%=1 TO 10 : REM WRONG
```

In most computer languages, integer variables are available because they take less memory and can be processed faster. This isn't true, however, of integer variables in Commodore 64 BASIC. All math operations are computed using floating-point numbers, so if an integer variable is used in a math operation, it's first converted to floating point (the machine's internal representation for integer and floating-point numbers is different). Then the operation is performed, and

the result converted back to integer form. Look at the following example; three conversions have to be done.

LET C%=A%+B%

1. Convert A% to floating point
2. Convert B% to floating point
   Perform the operation
3. Convert result to integer

Conversions take time. This means that integer variables actually compute slower than floating-point variables. Also, because each floating-point *and* integer variable take up seven bytes of free memory, using integer variables does not save memory.

All of this leads to a very good question. If integer variables have a limited range, cannot have decimal values, compute slower than floating-point variables, and take just as much memory, when would you ever want to use them? The answer is never, unless they're being used as an array. Each entry in an integer array takes only two bytes, compared with five bytes per entry in a floating-point array.

DIM A(100)	101 entries * 5 + 7-byte array overhead = 512 bytes
DIM A%(100)	101 entries * 2 + 7-byte array overhead = 209 bytes

If your program is using several very large arrays and free memory is a concern, it may be worthwhile to use integer arrays. Or, if you want to use a certain variable name (such as X), but the name is already in use for a floating-point variable, you can still use the name with an integer variable (X%). With these exceptions, you might as well not bother to use integer variables.

Summary

- Numeric variables used up to now are more accurately called *floating-point* variables.
- *Type* describes what kind of information is being conveyed by a variable. The two types of variables used thus far have been floating-point and string variables.
- Integer numbers include the counting numbers 1, 2, 3, ..., their negative counterparts, and 0. A fraction cannot be expressed by an integer.

- Integer variables are a third type of variable. They're distinguished from floating-point variables by a percent sign (%) after their name.
- Integer variables can have integer values from −32768 to +32767. If an integer variable is assigned a value outside this range, the ILLEGAL QUANTITY error will occur.
- When an integer variable is assigned a decimal value, the value is first converted to an integer using the same procedure as that used by the INT function. For positive numbers, this has the effect of dropping all digits to the right of the decimal point.
- Integer and floating-point variable names are completely separate. N and N% are two different variables, and what happens to one will not affect the other.
- Integer variables can be used wherever floating-point variables can be used, except that they cannot be used as counting variables of FOR-NEXT loops. A SYNTAX ERROR would result.
- Integer variables take just as much memory as floating-point variables and actually compute slower.
- The only practical application of the integer variable type is for an array, the one time when integers take less memory than floating-point numbers. Each entry in an integer array takes two bytes, as opposed to five bytes per entry in a floating-point array.

Random Number Seeding and Sequencing

The random function was introduced in Volume One as a means of adding a certain amount of unpredictability to a program. RND(0) returns a decimal number between 0 and 1, and returns a different number each time it's called. By multiplying the value by 10 and taking the integer, a random number from 0 to 9 can be obtained. This is useful in such applications as games. There are some applications of random numbers, though, where you will not want to use RND(0). Although this function returns numbers between 0 and 1, it will return only selected values. After a number of calls, RND(0) will repeat some values, yet not return others, no matter how many times the function is used. The following program line graphically demonstrates this.

```
10 Z=RND(0)*1000:POKE 55296+Z,15:POKE 1024+Z,160:G
   OTO 10
```

The plotting appears random at first, but it soon begins to form a pattern of diagonal lines. Since not all of the screen is covered, you can easily tell that the numbers generated by RND(0) are not very *dense*. In other words, there are plenty of values missing.

Fortunately, there's a solution to this. Change the argument of the RND function from 0 to 1, clear the screen, and run the program again. This time the plotting does not form any discernable pattern, and if you let the program run long enough, the whole screen is eventually filled.

Commodore 64 BASIC actually supports two kinds of random functions. If the argument is 0, the computer PEEKs some constantly changing hardware locations to construct the random number. This would be acceptable, except that the contents of the hardware locations do not change enough. One is a clock/timer that normally isn't even running!

Using the RND function with a nonzero argument calculates a value based on a repeating series. This process forms a very dense set of numbers, yet it's not without flaws either. RND(1) will return the same sequence of numbers every time you turn the computer on. Try it by first turning your computer off and back on. Then enter this line:

```
PRINT RND(1);RND(1);RND(1)
```

Note the three values returned. Now turn the computer off again, then back on, and retype the line. After pressing RETURN, you'll see the same three values as before.

Using arguments such as 8 or 1.5 in place of 1 doesn't make any difference. The numbers may be dense, but they will always occur in the same order. This certainly limits the usefulness of the RND function for applications like games.

To get a dense set of numbers that will be different every time you turn on the computer, you have to "seed" the random number generator. This is usually done at the beginning of a program by calling the random function with a negative argument. For every negative argument, a different sequence is selected. For example, every time you enter the following line, you'll get the same results.

```
N=RND(-1):PRINT RND(1);RND(1);RND(1)
```

Try changing the seed value to −2. Though a different set of numbers will be printed, they'll still be the same each time. You can imagine that the computer holds a very long sequence

of numbers, and each seed starts the random function at a different point in the sequence.

To get truly random numbers, you need to randomly seed the random function. Sound confusing? It's not. Instead of using a constant negative number for the seed value at the beginning of the program, just use a changing value, such as:

```
100 N=RND(-TI)
```

Do this only once, when the program initializes, and then use RND(1) to get the random numbers.

This should be the perfect solution. You get a dense set of numbers, and the set is different every time. Random numbers produced by this technique are suitable for both games and statistical applications.

Summary

- The random function, RND, returns a random decimal number between 0 and 1.
- When RND is called with an argument of 0, the value returned is based on a hardware timer. However, only a few of the possible numbers are ever returned. In other words, the numbers are not very *dense.*
- Calling RND with an argument greater than 0, such as 1, makes the function return a value based on a repeating series. Values returned by RND(1) *are* dense. The only problem with RND(1) is that it returns values in the same order every time the computer is turned on.
- The random number generator can be "seeded" by calling RND with a negative argument. For each negative number, or seed, the values returned by RND(1) start at a different place in the series.
- To get truly random numbers (dense and different each time the computer is turned on), the random number generator should be seeded with a random value as part of a program's initialization. Using RND(−TI) should be satisfactory.

More Functions

Print Formatting Functions

The screen on the Commodore 64 is divided into 25 rows of 40 columns each. These rows are more specifically called *physical* lines. Another type of row is known as a *logical* line. You may have noticed while editing a line in BASIC, that when the typing wraps around to the left edge of the screen, the rows below scroll down. This makes room for typing on a second row. Together, the two rows form one logical line.

One logical line can be up to two physical lines long, which is why BASIC lines on the 64 cannot exceed 80 characters. If the typing wraps around to a third row, a new logical line is used.

Logical lines control the cursor movement when the RETURN key is pressed. Hitting RETURN moves the cursor to the beginning of the next logical line. This means that the cursor may move down one or two physical lines. Normally, the cursor just moves to the next row; if it's on the first row of a two-row logical line, however, it skips the second and goes to the first row of the next logical line.

Logical lines also affect screen scrolling. The screen may scroll by one or two rows. The number of rows in the logical line currently scrolling off the screen determines how far the screen scrolls.

The POS function returns the current horizontal position of the cursor within a logical line. POS indicates this position with a column number from 0 to 79. Values 0–39 correspond to the 40 columns of the first row, while those from 40 to 79 indicate that the cursor is on the second row of a logical line.

POS requires an argument. It's a dummy argument (it doesn't affect the value returned) and is usually set to 0.

```
PRINT POS(0)
PRINT "GRAPHICS" POS(0) "SOUND"
```

In the first example, BASIC printed a zero—that's because the cursor moved to column 0 when the RETURN key was pressed to enter the line. Eight was returned in the second example because the cursor was in column 8 after printing the word *GRAPHICS.*

Once in a while you'll find a situation in which POS may come in handy, but generally it's a relatively useless function.

Two functions which can be used more often are TAB and SPC. Actually, they are not true functions. They do not return values, and they can be used only in PRINT statements. TAB and SPC are used to change the position of the cursor. Here's an example:

PRINT TAB(20) "HOWDY"

The only thing that was printed was the word *HOWDY,* but its first letter was in column 20. TAB is like a special instruction to the PRINT statement which makes the cursor move to a certain column. Commas in PRINT statements allow only very limited formatting of displayed information, because you have no choice about where the first character appears. With TAB, however, you can select your own columns. Program 2-1, "Parts," shows a simple example of how you can use TAB to format a screen display.

Program 2-1. Parts

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
10 PRINT " PART" TAB(6) "NAME" TAB(16) " QUANTITY"
                                              :rem 126
20 FOR K=1 TO 3:READ PN,N$,Q                  :rem 231
30 PRINT PN TAB(6) N$ TAB(16) Q                :rem 62
40 NEXT                                       :rem 163
50 DATA 33,WING NUT,5,10,BOLT,3,47,1/4 SCREW,16
                                               :rem 85
```

TAB works by comparing the number in parentheses with the current position (the value returned by POS). If the cursor has to be moved to the right, the necessary number of spaces are printed. TAB cannot move the cursor to the left, which would cause overwriting of something already on the screen. In fact, if you try to use TAB(12) when the cursor is at column 20, the cursor won't move. You can see this for yourself by entering this PRINT statement:

PRINT TAB(20) 6 TAB(12) 4

Another use of TAB is to center or right justify text lines. These lines center each word printed.

```
10 READ S$:IF S$<>"END" THEN PRINT TAB(20-LEN(S$)/
   2) S$:GOTO 10
20 DATA BITMAPPED,GRAPHICS,MUSIC,SYNTHESIS,SPRITE
   {SPACE}ANIMATION,END
```

By changing the expression for the TAB value from 20−LEN(S$)/2 to 39−LEN(S$) the words will be right justified instead.

TAB is used to *absolutely position* the cursor. Regardless of where the cursor is, TAB moves it to the requested column, as long as it's not trying to move to the left. The SPC function works a little differently. It moves the cursor the designated number of spaces to the right from its current position. It allows *relative positioning* of the cursor. If the cursor is at column 8, for instance, SPC(10) moves it to column 18. If it's at column 12, SPC(10) moves it to column 22. This is useful when you want to print several leading spaces. A statement starting with PRINT SPC(6) is easier to read than PRINT "   ", where it can be hard to count the number of spaces. SPC can also be used to print patterns, as the example below illustrates.

```
10 FOR K=1 TO 9:PRINT SPC(K) MID$("COMMODORE",K,1)
   :NEXT K
```

Remember, TAB and SPC are not true functions, and can be used only with the PRINT statement. That means, though it would be handy, you cannot do something like this:

```
10 BL$=SPC(10) : REM WRONG
```

Summary

- A physical line is one of the 25 rows on the screen.
- A logical line can consist of one or two contiguous rows. A logical line becomes two rows long when typing at the right edge of the screen wraps around to the next row at the left edge.
- BASIC program lines are one logical line long, which is why they cannot be longer than 80 characters.
- When RETURN is pressed, the cursor moves to the beginning of the next logical line.
- Logical lines also affect screen scrolling.
- The POS function returns the current position (columns 0–79) of the cursor within a logical line. Columns 40–79 indicate the second row of a logical line. Unfortunately, POS has few applications.
- TAB and SPC are special kinds of functions which can be used only in a PRINT statement. They control cursor

positioning, more like commas and semicolons in PRINT statements than like true functions. TAB and SPC do not return values.

- TAB advances the cursor to a designated column, as long as this does not require moving the cursor leftward. TAB allows absolute positioning of the cursor. Its applications include columnar printing and the centering and right justification of text.
- SPC moves the cursor to the right a designated number of spaces. Thus, SPC allows relative positioning of the cursor. SPC is used to print leading spaces and patterns.

Transcendental Functions

This section presents a group of functions that are used mainly in mathematics-oriented applications.

The **SQR** function returns the square root of its argument. The square root of number *N* is the number that, when multiplied by itself, produces *N*. Thus, the square root of 9 is 3, because 3*3 is 9; the square root of 16 is 4. The square root of 12 is somewhere between 3 and 4, but the decimal representation for this number continues indefinitely. With nine digits the value can be approximated as 3.46410162. Square roots of negative numbers are undefined, so using a negative argument with SQR causes an ILLEGAL QUANTITY error.

Finding the square root of a number is the same as raising the number to the power of 1/2. Square roots can also be calculated by using the exponentiation operator—the up-arrow key on your Commodore 64 (↑)—with the value 0.5. That's why both the following lines produce the same result.

```
PRINT SQR(12)
PRINT 12↑.5
```

The only advantages to using SQR instead of the exponentiation operator are that SQR is easier to read, and it executes just a little faster.

It might be a good idea to take a moment and see how complex mathematical formulas are converted into BASIC expressions. BASIC, of course, uses a slightly different notation than normal mathematics.

A common formula that uses the SQR function is the solution to the quadratic equation $ax^2+bx+c=0$. The solution is written as:

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

The first difficulty is the ± operation, which is not available in BASIC. The equation has to be written twice, once using + and once using −.

The long horizontal bar represents division, and means that the entire top part is to be divided by the bottom part. To keep the precedence correct, parentheses have to be used. Here's how the equations might be written in BASIC.

```
X=(-B+SQR(B↑2-4*A*C))/2*A
X=(-B-SQR(B↑2-4*A*C))/2*A
```

These expressions, however, are wrong. A common mistake is to forget about precedence in the denominator. The quantity 2*A should be placed in parentheses so that the top part is divided by the value of 2*A. As the above expressions are written, the top part is first divided by 2 and then multiplied by A. That will give an incorrect result. This is the right way to note the formula:

```
X=(-B+SQR(B↑2-4*A*C))/(2*A)
X=(-B-SQR(B↑2-4*A*C))/(2*A)
```

These expressions are correct, but inefficient. The exponentiation operator executes very slowly. When possible, use multiplication instead. In this example, the

computer can calculate B*B almost seven times faster than B↑2. Change the expressions to read:

```
X=(-B+SQR(B*B-4*A*C))/(2*A)
X=(-B-SQR(B*B-4*A*C))/(2*A)
```

For further optimization, the part SQR(B*B−4*A*C), called the *discriminant*, should be calculated only once, like this.

```
D=SQR(B*B-4*A*C)
X=(-B+D)/(2*A)
X=(-B-D)/(2*A)
```

One more thing can be done to simplify the second expression above. Replace (−B+D) with the equivalent (D−B).

Converting formulas into BASIC expressions is actually quite easy. Just remember the rules of precedence, and look for ways to optimize the code.

EXP and **LOG** are two more functions based on exponents. EXP takes the natural number *e*, a constant equal to 2.71828183, and raises it to the power specified by the argument. This power cannot exceed 88.0296919, or the OVERFLOW error occurs. Arguments less than −88.0296919 return 0.

Like the square root of 12, e is an irrational number (it cannot be written as a fraction) and can only be approximated. The nine-digit approximation of e is 2.71828183, which is produced by EXP(1). This means that EXP can also be done using exponentiation. Both the following statements produce the same result.

```
PRINT EXP(8)
PRINT E↑8     (assuming E equals 2.71828183)
```

Again, EXP is a little easier to work with than exponentiation, but in this case it's also much faster; it can be almost twice as fast as raising e to a power.

The LOG function is the inverse of EXP. Again, both of the next two statements return the same value, in this case 8.

```
PRINT LOG(EXP(8))
PRINT EXP(LOG(8))
```

Note that LOG computes natural logarithms, logarithms to the base e, instead of the base 10 often found on pocket

calculators. Also, due to the nature of logarithms, the argument must be greater than zero, or the ILLEGAL QUANTITY error occurs.

The remaining functions have to do with trigonometry. **SIN** and **COS** are used to calculate the sine and cosine of an angle specified in radians. Radians are often specified in terms of the irrational number pi, written π, which is approximately equal to 3.14159265. Since this value is frequently used, it's available as a special keyboard character. Type SHIFT–up-arrow to get the π symbol.

```
PRINT π
```

If you prefer working with degrees instead of radians, you can convert one to the other. To convert *n* degrees to radians, multiply *n* by π/180. To convert *n* radians to degrees, multiply *n* by 180/π.

The SIN and COS functions take the argument in radians, subtract any extra multiples of 2*π (or 360 degrees), calculate the sine or cosine value, and return a decimal number from −1 to 1.

The **TAN** function calculates the tangent of an angle given in radians. The tangent of an angle is equal to the sine of the angle divided by the cosine. Thus, TAN is not really necessary, but it *is* more convenient. However, because it is a fraction with COS in the denominator, some restrictions must be placed on TAN's argument. The function is undefined for the angles π/2, 3*π/2, and all equivalent angles. These angles have a cosine of zero, and since tangent is sine/cosine, this would lead to division by zero (undefined in normal mathematics). You'll see the DIVISION BY ZERO error when TAN is given an illegal angle.

The last function is **ATN** (arctangent), the inverse of TAN. ATN returns the angle in radians which has the given tangent as the argument. Where TAN(π/4) is 1, ATN(1) is π/4, or 0.785398163. ATN will always return values from −π/2 to π/2.

Other trigonometric functions can be calculated using the transcendental functions already described. See Appendix H of the *Commodore 64 User's Guide* (the manual that came with your computer) for a listing of these derived mathematical functions.

Summary

- The SQR function returns the square root of its argument. The argument must be greater than or equal to zero, or an ILLEGAL QUANTITY error occurs. The function is used instead of raising a number to the power 0.5 because it's easier to read and executes slightly faster.
- The EXP function raises the natural number *e*, approximately equal to 2.71828183, to the power indicated by its argument. If the power is greater than 88.0296919, the OVERFLOW error occurs. If the power is less than −88.0296919, EXP will return 0. EXP is used instead of exponentiation because it's easier to read and it executes significantly faster than raising e to a power.
- The function LOG returns the natural logarithm, the log to the base e, of its argument. The argument must be greater than zero to prevent the ILLEGAL QUANTITY error.
- The π symbol can be used to represent the value 3.14159265, an approximation of the mathematical constant pi.
- The SIN and COS functions return the sine and cosine, respectively, of the angle specified (in radians) as the argument. Extra multiples of 2*π (360 degrees) are removed from the angle. The value returned is a decimal number from −1 to 1.
- It's possible to convert radians to degrees, and vice versa. To convert *n* degrees to radians, multiply *n* by π/180. To convert *n* radians to degrees, multiply *n* by 180/π.
- The TAN function returns the tangent of the angle (expressed in radians) given by the argument. TAN is not defined for the angles π/2, 3*π/2, and equivalent angles. Using any of these causes the DIVISION BY ZERO error.
- The ATN function is used to calculate the arctangent (inverse tangent). It returns the angle in radians which has the tangent specified as the argument. ATN will always return values from −π/2 to π/2.
- Formulas can be used to obtain other trigonometric functions, such as the remaining inverses and the hyperbolics.

User-Defined Functions with the DEF Statement

Sometimes when writing a program, you may find yourself typing the same expression again and again. Consider a program that's working with dollar and cent values which has to

round numbers to the nearest cent several times. The process of rounding is done by the following expression:

INT(N*100+.5)/100

Given a number *N*, this formula will preserve all digits to the left of the decimal point (dollars) and the first two digits to the right of the decimal point (cents). The remaining digits will be dropped because they represent a value less than one cent. However, if what's dropped is half a cent or more, the number is rounded up. Try the short program below to check that the formula really does work.

```
10 INPUT N:PRINT INT(N*100+.5)/100
20 GOTO 10
```

The value 1.234 becomes 1.23, but 1.235 will become 1.24.

Other computers may have a rounding process included in their BASIC. It's usually implemented in the form of a function, so that rounding a number can be as easy as saying PRINT ROUND(N). Since Commodore 64 BASIC has no rounding function, the formula will have to be placed in a program wherever a number has to be rounded. If rounding was necessary in several parts of the program, this could take a fair amount of typing. The formula is somewhat complicated, and every time you enter it, you increase the chance of error. One solution is to use a subroutine instead, something like the program segment which follows.

```
10 INPUT N:GOSUB 60:PRINT A
20 GOTO 10
60 A=INT(N*100+.5)/100:RETURN
```

For every place where a number has to be rounded, the statement GOSUB 60 can be used. This may be a little easier to type, and the risk of mistyping the formula is reduced because it's entered only once. But this method is still awkward to use.

The ideal solution would be to add the rounding process to BASIC as one of the standard functions. That's where the DEF statement comes in. DEF lets you create your own functions, which include arguments and are called like any other function. The availability of user-defined functions is one of the best features of Commodore 64 BASIC.

The syntax of the DEF statement consists of several parts:

The keyword DEF, the function name prefix FN, the function name, parentheses containing a variable, an equal sign, and finally an expression. The restrictions which apply to variable names apply to function names (only the first two characters are significant, the first character must be alphabetic, and the remaining characters are optional and must be alphanumeric). The expression should be defined in terms of the variable in the parentheses. This is rather complicated, so here's an example.

```
10 DEF FNR(N)=INT(N*100+.5)/100:REM DEFINES FUNCTI
   ON 'R'
```

To call the newly defined function, the function name prefix FN should be followed by the function name R, with parentheses containing an argument.

```
20 INPUT N:PRINT FNR(N):GOTO 20:REM CALLS FUNCTION
    'R'
```

When FNR(N) is executed, the value of argument *N* is substituted for every *N* in the definition. PRINT FNR(3) means PRINT INT(3*100+.5)/100. When you run the two-line program (lines 10 and 20 above), it gives the same results as the previous demonstrations. But the defined function is much more convenient.

There are several interesting things concerning the variable used in the function definition which you should know. First of all, the variable in the function call does not have to match the variable in the function definition. Line 20 could have been written as:

```
20 INPUT Q:PRINT FNR(Q):GOTO 20
```

You don't even have to use a variable. Make sure the two-line program from above (with either version of line 20) is loaded and run, hit the RUN/STOP–RESTORE key combination, then type this line in immediate mode (without a line number).

PRINT FNR(5.118)

You'll see 5.12 returned as the result.

The purpose of the variable in the definition is to show the relationship between the argument and the expression. The statement DEF FNR(N)=INT(N*100+.5)/100 really

means *define a function R which will take the argument, multiply it by 100, add 0.5, take the integer portion, divide by 100, and return the result as the value of the function.* The variable N has nothing to do with the definition, other than to show where the argument is in the expression. This is necessary in case more than one variable name appears.

```
10 DEF FNR(N)=INT(N*T+.5)/T
20 T=100
30 INPUT N:PRINT FNR(N):GOTO 30
```

The definition uses variable T, which has the value 100, to make the function execute a little faster. When executing function R, BASIC knows that the argument should be substituted for every N in the definition, because N, not T, appears as the argument.

Using a variable in a DEF statement, such as N, is a very unusual application. Variables are not used like this in any other aspect of BASIC. To illustrate the fact that N is independent of the defined function, here are two things you should know:

1. The value of N is not affected by the definition or call of a function defined in terms of N.
2. The value of N has no effect on the definition or call of a function defined in terms of N.

To demonstrate the first point, take a look at the next short routine. It shows how the value of N is preserved through a function definition and call.

```
10 N=193 : REM ARBITRARY CHOICE
20 DEF FNR(N)=INT(N*100+.5)/100
30 PRINT FNR(2.639)
40 PRINT N : REM WILL PRINT 193
```

To illustrate the second point, just try different values for N in line 10. You'll see that function R always returns the same value for the argument 2.639, regardless of the value of N.

You can even define a function that doesn't use its argument. This line defines function G which returns a random integer from 1 to 10.

```
10 DEF FNG(X)=INT(RND(1)*10)+1
```

The argument, indicated by X, is not used in the

expression. This means that whenever function G is called, the argument will not have any effect on the value returned by the function.

You can go further and use a dummy argument, such as 0, just to satisfy the function's syntax.

```
2Ø PRINT FNG(Ø)
```

Another possibility is to use two arguments in a function. Since the DEF statement syntax supports only one, variables will have to be used to pass the two values. The defined function in the following program ignores the argument from the function call and accepts the variables A and B instead. The function is named MN because it is a minimum function; it returns the smaller of the two arguments.

```
1Ø DEF FNMN(N)=-A*(A<B)-B*(A>=B)
2Ø A=3:B=4
3Ø PRINT FNMN(Ø)
```

When you run this, the number 3 prints. Change A so that it's equal to or greater than B, and the number 4 will print instead.

A=5:PRINT FNMN(0)

This program demonstrates the definition for a maximum function, one that returns the larger of the two arguments. Notice the three values that display. In each, the larger argument appears.

```
1Ø DEF FNMX(N)=-A*(A>B)-B*(A<=B)
2Ø A=3:B=4:PRINT FNMX(Ø)
3Ø A=5:PRINT FNMX(Ø)
4Ø B=-7:PRINT FNMX(Ø)
```

As you can see, using function calls can prevent repeated typings of an expression.

One application of user-defined functions is to calculate low and high bytes of memory addresses. A common method of storing data, from note information for a song to a sprite shape definition, is to place a sequence of bytes in memory locations. Whatever the information is, its first location is the *address* of the data. Locations can range from 0 to 65535; so can addresses.

This, however, presents a problem—there will be many

times when you'll have to store an address in a memory location. Locations hold values (bytes) which have a range only of 0–255. Since there's no way to convert an address (0–65535) into a byte (0–255), it's necessary to express the address in the form of *two* bytes. The high byte indicates the number of multiples of 256, and the low byte specifies the remainder. All memory locations can be expressed with this two-byte format. The table below shows some examples.

Address	Hi	Lo
0	0	0
1	0	1
2	0	2
254	0	254
255	0	255
256	1	0
257	1	1
511	1	255
512	2	0
513	2	1
1024	4	0
49152	192	0
49487	193	79
65535	255	255

To break any address into its two equivalent bytes, two functions are needed.

```
10 DEF FNH(N)=INT(N/256):DEF FNL(N)=N-256*INT(N/25
   6)
20 INPUT A:PRINT FNH(A);FNL(A):GOTO 20
```

Function H calculates the high byte and function L calculates the low byte. Check the values in the chart by running the program, and then enter a few of your own.

You'll find these two functions used quite often in programs in other parts of this book. Usually the low and high bytes will be POKEd into two consecutive memory locations, such as in:

POKE 251,FNL(BA):POKE 252,FNH(BA)

By convention, the low byte is always POKEd into the first location, and the high byte into the second. This format is important to remember.

A third function used in later programs is one that reconstructs an address stored in low byte/high byte form.

```
10 DEF FNDP(A)=PEEK(A)+256*PEEK(A+1)
```

The "double PEEK" function PEEKs the two locations and returns the address they contain. If you type in and run the line above, then enter PRINT FNDP(*n*), where *n* is the low byte location, the function will calculate the address and display it.

You may have noticed that part of the definition for function L is the same as that for function H. Can one defined function call another? Certainly. Take a look at this line.

```
10 DEF FNH(N)=INT(N/256):DEF FNL(N)=N-256*FNH(N)
```

The only thing you don't want a function definition to do is call itself. BASIC won't know how to deal with that and indicates its confusion by printing the OUT OF MEMORY error if the function is called.

```
10 DEF FNB(X)=3+FNB(X) : REM WRONG
```

If there are any SYNTAX ERRORs in the expression for the function definition, they won't be detected until the function is actually called. It's also possible for an error to occur within a defined function, such as ILLEGAL QUANTITY, DIVISION BY ZERO, or OVERFLOW. In all these cases, when BASIC stops and prints the error, the line number indicated is the line of the function call, *not* the definition.

Another error associated with user-defined functions is UNDEF'D FUNCTION. This happens whenever a program tries to call a function not yet defined. The most common causes are either forgetting to define the function or using different names for the function definition and call.

Once a function has been defined, it doesn't have to keep that definition. If a function name is already in use, but placed in another DEF statement, the old definition will be replaced with the new. This also means that it's okay for the same DEF statement to be executed more than once. No error will occur.

Defined functions are erased by the CLR statement.

The argument in a DEF statement must be a floating-point variable. Using a string variable or an integer variable leads to a SYNTAX ERROR.

Finally, like the GET and INPUT statements, the DEF statement can be executed only in a program. Attempting to define a function in the immediate mode only causes the ILLEGAL DIRECT error.

Summary

- Sometimes you'll find yourself entering the same expression again and again. Often the expression performs some process on a value. Repeatedly typing the same expression should be avoided, not only to eliminate extra work, but also to lessen the chance of error.
- The ideal way to handle such an expression is to implement it as a function. The value used by the expression could be the argument, and the number produced by the expression could be the value returned by the function.
- One of the best features of Commodore 64 BASIC is that it lets you create new functions through the DEF statement. Functions defined with DEF behave just like standard BASIC functions. They have a name, take an argument, and return a value.
- The syntax for the DEF statement is: DEF, the function name prefix FN, the function name, parentheses containing a floating-point variable for an argument, an equal sign, and an expression.
- Function names follow the same syntax rules as variable and array names. The first character must be alphabetic and the remaining characters, which are optional, must be alphanumeric. Only the first two characters are significant.
- The variable in parentheses is called the *argument variable.* The expression is defined in terms of this argument variable. Every occurrence of the argument variable name in the expression is used to represent the value passed as the argument.
- The argument variable is used only as a place holder, to show the relationship of the argument to the expression. Any variable name can be used, even one that does not appear elsewhere in the program.
- The DEF statement can be used only in program mode.
- To call a defined function, use the prefix FN followed by the function name, then parentheses containing an argument that is passed to the function.
- When the function is called, BASIC refers to the expression in the definition and substitutes the argument passed for every occurrence of the argument variable. BASIC then evaluates the expression and returns the result as the function value.

- The value of the variable used to represent the argument is not changed when DEF is executed or when the function is called; the definition and call of a function are not affected by the value of the variable.
- The argument of a call to a defined function can be a number, a variable, or an expression. It does not have to be a variable matching the argument variable in the corresponding DEF statement.
- The expression in the DEF statement does not have to contain the variable for the argument. The function can ignore the argument, and the argument will have no effect on the value returned. It will be a *dummy* argument.
- To pass more than one argument to a defined function, variables have to be used. A function can ignore the normal argument and return a value based on one or more variables. An example here was a minimum function which took two arguments and returned the lesser value. The definition for this function expected the two arguments to always be in the variables A and B.
- The expression for a definition can contain other variables, functions, or even other defined functions.
- The only thing that a function definition may not contain is a call to itself. That is a form of recursion, which BASIC cannot handle, and is reflected by an OUT OF MEMORY error.
- When a call is made to a function that has not yet been defined, the UNDEF'D FUNCTION error occurs. Other errors, such as SYNTAX and ILLEGAL QUANTITY, can also occur while a function is being called. Whenever such an error occurs, the line number displayed is the line of the call, not of the definition.
- A function can be redefined without an error. This means that the definition associated with a function name can be changed while the program is running, or the same DEF statement can be executed more than once.
- All defined functions are erased when CLR is performed.
- The first memory location for a sequence of bytes is called the *address* of the data. An address can range from 0 to 65535. An address can be expressed in a two-byte format, each byte in the range 0–255. The *high byte* indicates multiples of 256, the *low byte* the remainder, of the address.
- Being able to express an address in this form makes it possible to POKE the address into memory locations.

- When the low and high bytes of an address are POKEd into memory locations, two very important rules must be followed: First, the locations must be contiguous. Second, the low byte is always POKEd into the first location, and the high byte is always POKEd into the second location.
- An address can be reconstructed after it has been broken down into low and high bytes and POKEd into locations. To get the original value, PEEK the second location, multiply it by 256, and add the result to the value found in the first location.
- A common application of defined functions is to calculate low and high bytes, and to reconstruct addresses. In this book, functions L and H are always used to take an address and return the low and high byte, respectively. Function DP will always take a location number, PEEK the location and the one that follows, and return a reconstructed address.

Chapter 3

File Input/Output

Introduction to I/O

A computer is a communications device—it moves information from one place to another. The most common example is when the computer accepts information from a user through the keyboard, and presents information by displaying it on a screen. This process of communicating is referred to as input/output, and is often abbreviated as I/O.

The keyboard and screen are not the only parts of a computer system that can be involved in input/output. Other peripherals, such as a printer, tape recorder, disk drive, or modem, can also be used to transfer information. With a printer, you can get a permanent copy of a program listing or program results on paper. The Datassette recorder and the disk drive can be used to load and save programs and other kinds of information. By using a modem and a telephone, you can connect your computer to other similarly equipped computers.

Collectively, the keyboard, screen, printer, Datassette, disk drive, and modem are called *devices.* The transfer of information takes place between the computer and one of these devices. An important aspect of this information exchange is the direction in which the information is traveling. The direction describes whether information is being sent or being received, and is always described from the point of view of the computer. If the computer is receiving information from a device, the term *input* is used. When the computer is sending information to a device, it's termed *output.* Sometimes the terms *reading* and *writing* are used to mean *receiving* and *sending.*

With some devices, only one direction is possible. For example, the keyboard can be used only to input information. The keyboard is not capable of displaying information, and it would be ridiculous to try to make the computer send information to the keyboard. On the other hand, the printer is an output-only device. Information cannot be entered into the computer through the printer. Devices such as the Datassette, disk drive, and modem, however, support both input and output.

The purpose of this chapter is to show you how BASIC can be used to make the computer communicate with other

devices. BASIC's input statements are GET and INPUT, and its output statement is PRINT. With a few modifications, these same statements can be used with all the devices, not just the keyboard and screen. The next section introduces a new way to access the keyboard and screen, while the later sections show how these methods can be applied to the other devices.

Summary

- The process of the computer communicating is called *input/output,* abbreviated *I/O.* A typical example is when the computer receives information from the keyboard and transmits information to the screen.
- Those parts of the computer system which can be used for input or output—the keyboard, screen, printer, Datassette, disk drive, and modem—are called *devices.* The transfer of information takes place between the computer and a device.
- The direction of the transfer is described from the point of view of the computer. When the computer accepts information from a device, the term *input* is used. *Output* applies to information sent from the computer to a device.
- Some devices are capable of communicating in only one direction. The keyboard can be used only to input information, for instance, and the printer can be used only to output information. Other devices, including the Datassette, disk drive, and modem, can communicate in both directions.
- The rest of this chapter will show how to use variations of GET, INPUT, and PRINT to communicate with all devices, not just the keyboard and screen.

The OPEN and CLOSE Statements

To formally initiate communication with a device, you must create a *file* which is used for the actual input/output. Creating a file informs the computer that I/O is going to be performed. The file then acts as a sort of intermediary: Information to be sent to the device is first sent to the file; information coming from the device is first brought into the file to be retrieved by BASIC. Like a program, a file is more of a concept than a tangible reality.

For now, all that matters is that you realize a file is necessary in order to communicate with a device, and that the communication is done through the file. The details of how a

file works will become clearer as you explore its function and use.

A file is created by the OPEN statement, which has the following syntax:

OPEN ***file number, device number, command number, filename***

The **file number,** sometimes also called the *logical file number,* can range from 1 to 255. Numbers greater than 127 may have special significance and normally should not be used. The actual value is chosen by the programmer, but the most common is 1. The file number will later be used by the GET, INPUT, and PRINT statements.

Since it's possible for several devices to be active at the same time, the file number is needed to distinguish one currently active file from another. For example, a program could print a prompt on the screen, wait for a response on the keyboard, print a status message on the printer, and read in some information from the disk. For each device that's used, a new file must be created with a different file number. In the example above, the screen might have been opened as file 1, the keyboard as file 2, the printer as file 3, and the disk as file 4.

The **device number** is needed to specify which device is associated with the file. Each device is identified by its own number.

Number	Device
0	Keyboard
1	Datassette
2	RS-232 (modem)
3	Screen
4	Printer
5,6,7	Other devices
8	Disk drive

As long as the file is open, the device indicated by the device number will correspond with the file. All references to the file will be directed to the specified device.

The **command number** is also called the *secondary address.* It's not needed when using the keyboard or screen, and will be discussed in a later section.

The **filename** is needed when working with devices such as tape or disk drive. Like the command number, the filename will be dealt with later.

This leaves us with the simplest form of the OPEN statement:

OPEN ***file number, device number***

Here's an example of how the statement would look in a program.

```
10 OPEN 1,0
```

This creates a file, designated as file 1, which uses device 0, the keyboard. To input information from file 1, the INPUT statement can now be used. However, since there might be several files open, to various devices, the computer will need to know from which file the input is expected. That's why the file number must be included as part of the INPUT statement, as shown by the next line.

INPUT# ***file number, variable list***

In our example program, the INPUT statement would look like:

```
20 INPUT#1,N
```

The # symbol is considered a part of the keyword, so no space is allowed between the word *INPUT* and the symbol. A comma must be used after the file number. Using a semicolon causes a SYNTAX ERROR.

Add these two lines to lines 10 and 20 above, and run the short program to see how this special form of INPUT works.

```
30 PRINT
40 PRINT "YOU TYPED" N
```

Notice that there's no question mark printed as a prompt. The cursor just blinks, waiting for you to enter a number. When you type a number and press RETURN, the program echoes it back, as you'd expect. Now run it again, but this time enter some letters instead of a number. Instead of getting the REDO FROM START message, the program stops with a FILE DATA error. If you run the program once more and type two numbers separated by a comma, you'll not get the EXTRA IGNORED message—instead, just the first number appears. Also, if you examine the syntax for the new INPUT, you'll discover that no optional prompt message is allowed. As you can

see, INPUT used with a file is different from the INPUT with which you're already familiar.

The explanation of these differences is that the plain INPUT is a special form of INPUT#. Because INPUT is always used with the keyboard, features like prompt messages and error handling are helpful. However, since INPUT# may be used to fetch information from a device other than the keyboard, such as a data file on tape, features like prompt messages would not be appropriate or even desirable.

Let's try using the screen for output. Erase the old program by typing NEW, and enter this one.

```
10 OPEN 1,3 : REM 3 IS FOR SCREEN
20 PRINT#1,34
30 PRINT#1,"HAPPY BIRTHDAY"
40 PRINT#1,64,46
```

The # symbol is a part of the keyword, so this is one instance where the abbreviation for PRINT (?) cannot be used—you must type the full word PRINT#. As with INPUT#, a comma is needed after the file number. This first comma does not affect spacing. Run the program to see for yourself.

There's no difference between using PRINT and PRINT# when the file has been opened to the screen. If a program is going to be printing information only to this device, there's no sense in using PRINT# when the normal PRINT works just as well. The advantage in using the file method is that by changing the device number in the OPEN statement, all printing will be directed to a different device. By simply changing the 3 in the OPEN statement to 4, the messages will be printed on the printer instead of the screen (assuming a printer is connected). Use of the printer is discussed in greater detail in the next section.

Now let's consider using two open files at the same time. The next program uses file 1 for keyboard input, and file 2 for screen output.

```
10 OPEN 1,0 : REM KEYBOARD
20 OPEN 2,3 : REM SCREEN
30 PRINT#2,"WHAT IS YOUR NAME?";
40 INPUT#1,N$
50 PRINT#2
60 FOR K=1 TO 10
70 PRINT#2,N$
80 NEXT K
```

Because two files are open simultaneously, different file numbers *must* be used. If the second OPEN statement also used file 1, a FILE OPEN error would occur. Use this new line 20 with the program you just ran.

```
20 OPEN 1,3 : REM SCREEN
```

The error indicates that file 1 is already in use. A number (such as 1) cannot be used in another OPEN statement until the file currently using that number is finished with all input/output. At that time, the file should be *closed* with the CLOSE statement, which has a very simple syntax.

CLOSE ***file number***

The CLOSE statement formally terminates an input/output session with a device. The file created by the earlier OPEN no longer exists after CLOSE is executed, and the file number is free to use in opening another file. The following program shows how CLOSE is used.

```
10 OPEN 1,3 : REM SCREEN
20 PRINT#1,"WHAT IS YOUR NAME?";
30 CLOSE 1
40 OPEN 1,0 : REM KEYBOARD
50 INPUT#1,N$
60 CLOSE 1
70 OPEN 1,3 : REM SCREEN
80 PRINT#1
90 FOR K=1 TO 10:PRINT#1,N$:NEXT K
```

If you run this, you'll not see a FILE OPEN error. That's because the CLOSE statement closed file 1 after it was used.

At this point you may be wondering why it's even necessary to bother with a file at all. Instead of a file number after PRINT# and INPUT#, why not just use the device number? The answer lies in the fact that with some devices, other information besides a device number is needed. Input/output with the disk or cassette requires that a filename be specified. Entering the filename in an OPEN statement once, and then using the corresponding file number in each subsequent PRINT# or INPUT# statement is a lot easier than typing the filename every time those statements are used.

Another reason is that some devices (such as the disk drive) permit more than one file to be open at the same time. The computer could input information from two different disk

files. Since the device number for both these files would be 8, just using a device number in an INPUT statement would not be enough—the computer couldn't tell them apart. File structure using OPEN and CLOSE provides an organized means of communicating with *all* types of devices, and has proven to be a good method for general input/output.

There are a few other errors which can appear when using file input/output. You've already seen the FILE OPEN error when you tried to open an already opened file. The FILE NOT OPEN error occurs when you try to access a file that has *not* been opened. You'll see this error when you run this next program.

```
10 OPEN 1,3
20 PRINT#3 : REM FILE 3 NOT OPENED YET
```

The NOT OUTPUT FILE error happens when an attempt is made to output information to a device which can only handle input, such as the keyboard. Likewise, the NOT INPUT FILE error occurs when a program requests input from a device which is not capable of outputting information.

These kinds of errors can occur only when there is something wrong with a program's logic. They should not happen when the normal OPEN and CLOSE procedure is followed. The only two errors which denote some other type of problem are FILE DATA and DEVICE NOT PRESENT. The FILE DATA error (described as BAD DATA in Commodore literature) occurs when INPUT# is expecting a numeric value, but a string value is received instead. In some cases it's best to always use string variables with INPUT#, and the VAL function to determine if the value was a number. This prevents the program from crashing due to the FILE DATA error.

A DEVICE NOT PRESENT error appears when either the device is not connected to the computer or it's not turned on. A program has no way of preventing this type of error, so you just have to make sure that all peripherals are properly hooked up and have power before running a program that may use them.

A few more comments should be made about the CLOSE statement. It's good practice for a program to use CLOSE when it's finished sending or receiving information. This promotes good style and frees up a file for use later in the

program. When using tape or disk files, it's absolutely necessary to use CLOSE to terminate I/O.

A CLOSE of all files is performed as part of a CLR. Since CLR is part of the RUN, NEW, and LOAD commands, and also happens automatically when a program line is changed, any of these actions will close all open files.

Summary

- The OPEN statement is used to create a *file* through which input/output will be done. A file is the programming link between the computer and a device.
- The full syntax for the OPEN statement is the keyword OPEN, a file number, a comma, a device number, another comma, a command number, one more comma, and then a filename.
- The file number, also called the *logical file number,* is used to distinguish one active file from another. It's necessary because there may be several different files open at the same time.
- The file number is chosen by the programmer and may range from 1 to 255. File numbers greater than 127 are reserved for special purposes. File numbering usually starts at 1.
- The file number will later be used in the PRINT#, GET#, and INPUT# statements.
- The device number is used to identify the device associated with a file.
- A file can use only one device.
- The command number, also called the *secondary address,* and the filename are optional, and are covered in a later section.
- To make the INPUT statement read from a file instead of the keyboard, use INPUT#, the file number, a comma, and then a variable list, as in *INPUT#1,N.* No space is allowed between INPUT and the # symbol.
- The INPUT statement is actually a special form of INPUT# which is always set for keyboard input.
- Features like REDO FROM START and EXTRA IGNORED warning messages are not available with INPUT#. If INPUT# is expecting a numeric variable and gets characters, BASIC stops with a FILE DATA error. There's no warning if extra information is available. Also, INPUT# does not support an optional prompt string.

- These features are useful only when input is coming from the keyboard. Since INPUT# may get data from another device, it does not support these features.
- To make PRINT send output to a device other than the screen, use PRINT# followed by a file number, a comma, and then the information that is to be sent. No space is allowed between PRINT and #, and the ? abbreviation cannot be used.
- PRINT# to a file which has been opened to the screen is no different from using PRINT.
- Attempting to communicate with a file that's not been created by an OPEN causes the FILE NOT OPEN error. Attempting to open a file using a file number that is already in use causes the FILE OPEN error.
- When the computer is finished communicating with a device and the file is no longer needed, use the CLOSE statement to terminate the file. The syntax for this statement is the keyword CLOSE, followed by the file number.
- Once a file has been closed, it can be opened again for use with a different device.
- All files are automatically closed as part of CLR.
- The NOT OUTPUT FILE error occurs when you try to use PRINT# with a device that cannot handle output, such as the keyboard. The NOT INPUT FILE error occurs when you try to use INPUT# with a device that's not capable of generating input, such as the printer.
- The DEVICE NOT PRESENT error occurs when the computer tries to communicate with a peripheral that's either not hooked up or not turned on.

The Printer and the CMD Statement

A printer is an important part of a complete computer system because it can give you a permanent copy of information. Sending program output to a printer is as easy as using two statements.

```
10 OPEN 1,4 : REM PRINTER
20 PRINT#1,"MUSIC FILE LISTING"
```

All information being output by PRINT#1 will be sent to the printer. Of course, just plain PRINT will still print to the screen.

```
10 PRINT "TURN ON THE PRINTER"
20 PRINT "AND HIT ANY KEY"
30 GET K$:IF K$="" GOTO 30
40 OPEN 1,4 : REM PRINTER
50 PRINT#1,"MUSIC FILE LISTING"
60 PRINT "PRINTER IS WORKING"
```

You can also use a variable for the device number in the OPEN statement and let the person running the program decide where the information will be sent. This technique is demonstrated by the next program.

```
10 INPUT "OUTPUT DEVICE NUMBER";DN
20 OPEN 1,DN
30 PRINT#1,"MUSIC FILE LISTING"
```

The program user could enter either 4 for printer output or 3 for the screen. Using the OPEN statement like this means that you don't have to worry about where the printing will be sent. You can write the program just like any other, and as long as you use PRINT# wherever you would normally use PRINT, it will have the ability to send output to the printer, screen, or any other device.

There are just a few minor differences between printing on the screen and printing on paper. One obvious difference is that the printer will not know what to do with special CHR$ codes which move the cursor, change the current character color, or clear the screen. Printers usually do not have backspacing capability and cannot erase anything that's already been printed.

A more subtle difference has to do with the print formatting functions, POS and TAB. Both of these depend on the current location of the screen cursor. The POS function returns that value, and TAB uses it to determine how many spaces the cursor should be moved to the right. Since the screen cursor does not move when information is sent to the printer, these functions will not work correctly with printer output. This is not a major inconvenience, though, because the SPC function will still work and can be used to simulate TAB. The comma used for far spacing and printing in columns will also work differently for printer output.

The combination of OPEN and PRINT# can be very powerful. Nevertheless, there are some instances when it would be nice to send output to the printer which is not

program-generated. For example, it would be handy to be able to send a program listing to the printer. A program is much easier to read and debug when you can see it on paper, rather than just a screen at a time. For situations like this, the CMD statement is available.

CMD *file number*

The CMD statement causes all output normally sent to the screen to be sent to the designated file. This includes output from normal PRINT statements, INPUT prompt messages, disk and tape SEARCHING and LOADING messages, program listings, and even the READY. prompt. The file must, of course, be open when CMD is executed, or a FILE NOT OPEN error will occur. By using CMD, you can list a program on the printer by entering only three lines in immediate mode.

```
OPEN 1,4
CMD 1
LIST
```

This could also be used to print a disk directory. (Remember to load the directory before opening the file because LOAD closes all open files.) Any printing that defaults to the screen will now be sent to the file.

The syntax for CMD allows an optional comma and string to be placed after the file number. The string will be the first thing sent to the file when the CMD statement is executed. This is useful for such things as titling program listings.

```
LOAD "POEM"
OPEN 1,4:CMD 1,"POEM":LIST
```

CMD does not have to be used with just the printer. It can be used to send normal screen output to any device, such as the modem. Remember, CMD sends the output to a file; the device to which the information is sent depends on how the file was opened.

CMD stays in effect until the file is closed or an error occurs, in which case printing reverts back to the screen.

It's recommended that you send an empty print line before closing a CMD file, to help the computer terminate the I/O.

```
OPEN 1,4:CMD 1:REM START CMD
PRINT#1:CLOSE 1:REM STOP CMD
```

When CMD is used in a program, in effect it changes every PRINT into a PRINT#. This is useful when you have a

program which uses PRINT statements, but you want the output to be sent to a device different from the screen.

You might wonder why PRINT# has to be used at all if CMD is available. As it turns out, there are some reasons why it's better to use PRINT#. The major reason applies when you want your program to communicate with more than one device. If you use CMD to send output to one device and then switch to another, output may in some cases end up going to both. Also, CMD can sometimes turn itself off even when the file has not been closed.

PRINT# is more reliable and is usually the better choice. CMD is best used in the immediate mode when you want to do something that can't be done with PRINT#, such as listing a program to the printer. When using CMD in a program, make sure the program is sending output to only one device.

The remaining topics concerning the printer have to do with special modes of operation.

The file number in OPEN should normally be in the range 1–127. File numbers greater than 127 have a special meaning when used with the printer, because they instruct the computer to send a linefeed character after every carriage return. The linefeed character makes the printer advance the paper one line. Most printers do this automatically when they receive a carriage return. A printer which does not have this feature will keep printing on the same line. If your printer doesn't have the automatic linefeed feature, or if it does and you want the printer to use double-spacing, use a file number from 128 to 255.

If you're using a Commodore printer, the third parameter of the OPEN statement, the command number, can be used to select a character set. When no command is specified, the printer assumes that the command number is 0, which means that the uppercase and graphics character set will be used.

OPEN 1,4 : REM UPPERCASE/GRAPHICS

or

OPEN 1,4,0 : REM UPPERCASE/GRAPHICS

If you want the printer to use the uppercase/lowercase set, use 7 as the command number.

OPEN 1,4,7 : REM UPPERCASE/LOWERCASE

Remember, this applies only to Commodore printers.

Some printers support special operations modes. These may allow double-width printing, underlining, bold print, different type styles, dot graphics, and other features. They're invoked after a file has been opened by sending control codes to it. Control codes are sent as ASCII characters. For example, the statement PRINT#1,CHR$(14) makes a Commodore printer select the double-width mode, which means that characters are expanded to twice their normal width. Printing CHR$(15) turns the mode off. Other printers may use different numbers, so consult your manual for more information.

Sometimes printers require an *escape* character, abbreviated *ESC*, to be sent before the control code. The ESC code is character 27, so you would have to use PRINT#1,CHR$(27); CHR$(*control code*) to select the desired mode.

Summary

- To send program output to the printer, open a file to device 4 and use PRINT#.
- When sending output to the printer, the formatting functions POS and TAB, and comma spacing, will not work correctly.
- To send computer-generated output (such as a program listing) to the printer, open a file to device 4 and use the CMD statement. The statement's syntax is the keyword CMD, the file number, and an optional comma and string.
- The file must already be open, or the FILE NOT OPEN error will occur.
- If a string is included with the CMD statement, it will be sent to the file. A typical use of this feature is to title a program listing.
- When executed, CMD redirects all output that normally defaults to the screen to go to the file instead. This includes output from PRINT, as well as messages generated by the computer, such as the READY. prompt.
- CMD has no effect on printing sent to a file by PRINT#, even if the device associated with the file is the screen.
- When CMD is being used with a device other than the keyboard or screen, it's a good practice to send an empty print line to the file before closing it.
- CMD stays in effect until the file is closed or an error occurs. In either case, printing reverts back to the screen.
- It's best to use CMD only in situations where PRINT#

cannot be used, for it's not always reliable. This is especially true when the program uses multiple devices.

- File numbers greater than 127 make the computer send a linefeed after every carriage return. This is necessary if your printer does not automatically advance the paper. If your printer does, use a file number from 128 to 255 for double-spacing.
- When using a Commodore printer, the third parameter of the OPEN statement, the command number, can be used to select a character set. Use 0 for the uppercase/graphics set, which is the default set. Use 7 to switch to the uppercase/lowercase character set.
- Some printers support special modes of operation, such as double-width printing. These modes are enabled by sending control codes to the printer, either by printing a string or by using the CHR$ function. Sometimes a control code may have to be preceded by the ESC (escape) character, CHR$(27).

Files on Tape

Thus far you've probably used the Datassette only for storing and retrieving programs. This storage and retrieval is necessary because the computer can hold only one program in memory at a time—anything in memory is erased when the computer is turned off. It's much easier to load a program than it is to retype it every time you want to use it!

When a program is saved to tape, it's stored as a file consisting of program lines. That doesn't mean tape files can contain only program lines. They can contain other information, such as numbers and character strings, just as easily. Just as a program is lost when the computer is turned off, so is the data maintained by that program. You may want a program to save this data to tape for future reference.

An example is a word processing program which saves a document on tape for later changes and reprinting. Or a game might maintain a high-score file that would have to be periodically updated. A mailing list program is of little value if it can't save the names and addresses on tape or disk. A long and complex adventure game may have an option to stop play and continue later, in which case the current character status would have to be saved. These are all excellent applications of data files on tape.

The Datassette differs from other devices such as the keyboard and printer, since it can be used for both input *and* output. However, the Datassette can handle only one direction at a time, because the user controls the pressing of the RECORD button. So the direction has to be specified by the program when the file is opened. Another aspect of tape I/O is that filenames can be used. Both of these things can be handled by the OPEN statement.

OPEN *file number, device number, command number, filename*

This is the complete syntax of OPEN. The file number can range from 1 to 127. The device number for the Datassette is 1, which is the default value if none is specified. The command number tells the direction.

Number	Command
0	Input (read from Datassette)
1	Output (write to Datassette)

If no command number is given, the computer opens the file for input.

Finally, a filename, up to 16 characters long, can be specified. As when saving a program, a filename does not have to be specified. However, if none is given when opening a file for output, none can be specified when the file is later opened for input.

Here's an example of a file being opened for tape output.

OPEN 1,1,1,"TAPE FILE"

The file can later be opened for input by the following statement.

OPEN 1,1,0,"TAPE FILE"

Let's examine just how the information is stored and retrieved. The following program creates a tape file containing two items of information.

```
10 REM GAME HI SCORE DEMONSTRATION
20 N$="CHRIS":SC=12345
30 OPEN 1,1,1,"HI SCORE":REM TAPE OUTPUT
40 PRINT#1,N$:REM NAME
50 PRINT#1,SC:REM SCORE
60 CLOSE 1
70 END
```

When this program ends, a new file exists on tape. You can't load it as a program, but its contents can be retrieved by a program like this:

```
10 REM GAME HI SCORE DEMONSTRATION
20 OPEN 1,1,0,"HI SCORE":REM TAPE INPUT
30 INPUT#1,N$,SC:REM NAME AND SCORE
40 CLOSE 1
50 PRINT N$ " HAD A HIGH SCORE OF" SC
60 END
```

This is a rather simple example, but even so, it demonstrates a couple of important things. First, notice that the program which created the tape file used two PRINT# statements, instead of only one followed by a variable list. To understand why, let's consider exactly what's stored on the tape when PRINT# is used.

The first PRINT# writes the character string "CHRIS" to the file. Then, because there is no comma or semicolon at the end of the statement, a carriage return is written to the file. A carriage return is character 13, or CHR$(13). The second PRINT# writes the number *12345* to the file. Because BASIC prints positive numbers with one leading space and one trailing space, a total of seven characters will be written. The CHR$(13) for the second PRINT# would be the last character. The resulting file could be represented like this:

CHRISCHR$(13)(space)**12345**(space)CHR$(13)

Think of what happens when you use PRINT to print two values. The statement PRINT N$,SC will print the name CHRIS, then print a few spaces because of the comma, print the number 12345, and finally print the ending CHR$(13). Only one carriage return has been printed, but a lot of extra spaces have been inserted. This kind of file could be represented as follows:

CHRIS(several spaces)(another space)**12345**(space)CHR$(13)

Now think of what happens when INPUT is used to get two values. The statement INPUT N$,SC waits for the characters to be entered, then waits for either a comma or a carriage return, and finally waits for a number. (Remember that if you type only one value and press RETURN, the computer will print a ?? prompt and wait for the second value.) The same is true for PRINT# and INPUT#. The statement INPUT#1,N$,SC

expects either a comma or a CHR$(13) between the two values. The problem is that when the two values are printed with PRINT#1,N$,SC there won't be a comma or carriage return between the two values, just a whole lot of spaces. Here's what will happen:

1. INPUT#1,N$,SC will read all the characters up to the first CHR$(13) or comma. This means that N$ would be assigned the value "CHRIS 12345."
2. INPUT#1,N$,SC will then try to read a number to assign to SC. But since there's nothing more in the file, BASIC will stop with an error.

The description of the problem may be long, but the solution is simple. Either avoid using a variable list after PRINT# or put a CHR$(13) between each item. Each of the examples below sends the same characters to the file.

PRINT#1,N$:PRINT#1,SC (original example)

PRINT#1,N$;CHR$(13);SC
or

PRINT#1,N$ CHR$(13) SC

If you have a long list, it may be easier to assign the CHR$(13) to a string and use the string between each item, like this:

CR$=CHR$(13):PRINT#1,N$ CR$ SC CR$ A$ CR$ B$ CR$ C

These values can be retrieved by INPUT#1,N$,SC,A$,B$,C without any trouble.

The preceding examples have illustrated one other important point. The program reading the file must know how many values are to be read. An error such as STRING TOO LONG will occur when a program tries to read more data than is contained in a file. Furthermore, the program must know the order in which the values were written. If the first program had written the values with PRINT#1,SC:PRINT#1,N$ and the second program read them with INPUT#1,N$,SC (the variables are switched), there would definitely be a problem. The INPUT# would read the digits and assign " 12345 " to N$, and then stop with a FILE DATA error when it tried to read a number for SC and found characters instead.

In some cases, such as a game which keeps track of the top ten scores, the number of items never changes. However,

other files, like those maintained by a telephone directory program, may change size. How to know when to stop requesting input will be dealt with first.

There are three methods you can use to avoid running off the end of a file. One is to have the first item in the file, a number, indicate how many values follow. The example programs below use this technique. The first writes the names and scores to tape, while the second reads them from tape and displays them on the screen. Type in the first and run it; then enter the second and run it. (You need to rewind the tape to the beginning of the first file before pressing PLAY.)

```
100 OPEN 1,1,1,"HI SCORES"
110 PRINT#1,3:REM ITEM COUNT
120 PRINT#1,"BOB" CHR$(13) 14000
130 PRINT#1,"KEVIN" CHR$(13) 12000
140 PRINT#1,"ERIC" CHR$(13) 11000
150 CLOSE 1
160 END
```

```
100 OPEN 1,1,0,"HI SCORES"
110 INPUT#1,N:REM ITEM COUNT
120 FOR K=1 TO N
130 INPUT#1,N$,SC
140 PRINT N$,SC
150 NEXT K
160 CLOSE 1
170 END
```

(These programs only demonstrate the method, and are not intended to be practical applications.)

In some cases, a program may not know how many values will eventually be printed when the file is opened. Since no length number can be printed at the beginning of the file, the program must have another way to detect the file's end. This is similar to the problem of indicating the end of values read from DATA statements. You can use the same trick here as you would with READ and DATA—a flag. This write-and-read example uses *END* as a flag. Enter the first and run it; then type in and run the second.

```
100 OPEN 1,1,1,"HI SCORES"
110 PRINT#1,"MARK" CHR$(13) 504
120 PRINT#1,"JEAN" CHR$(13) 915
130 PRINT#1,"MARYBETH" CHR$(13) 412
140 PRINT#1,"KEN" CHR$(13) 755
150 PRINT#1,"END" CHR$(13) 0
160 CLOSE 1
170 END
```

```
100 OPEN 1,1,0,"HI SCORES"
110 INPUT#1,N$,SC
120 IF N$<>"END" THEN PRINT N$,SC:GOTO 110
130 CLOSE 1
140 END
```

The third method is to have the program monitor the value of ST, a reserved variable which holds status information. ST usually holds zero, but may contain another value when an error occurs or something important has happened. Bit 6 of ST will be set when the end of a file has been reached. The value associated with bit 6 is 64, so when a program sees that ST has the value 64 (or any nonzero value), it can stop requesting input. Take a look at this example for the technique's application. Enter and run these short programs as you did with the previous write/read samples.

```
100 OPEN 1,1,1,"HI SCORES"
110 PRINT#1,"MATTHEW" CHR$(13) 9
120 PRINT#1,"ANDREA" CHR$(13) 10
130 PRINT#1,"TODD" CHR$(13) 9
140 PRINT#1,"JEFFREY" CHR$(13) 8
150 CLOSE 1
160 END
```

```
100 OPEN 1,1,0,"HI SCORES"
110 INPUT#1,N$,SC
120 PRINT N$,SC
130 IF ST=0 GOTO 110
140 CLOSE 1
150 END
```

Notice that the end-of-file check was done *before* the printing in the previous method, but is done *after* the printing here.

Getting the correct values into the correct variables is simply a matter of taking care when writing the program.

Make sure that the variables in the PRINT# and INPUT# statements line up. The only thing that can cause trouble is when a null string is printed by PRINT#. The INPUT# statement will skip past a null string and read the next value instead.

For instance, the second program stops with an error at line 130.

```
100 N1$="GAIL":N3$="MAX":REM MISSING N2$
110 OPEN 1,1,1,"TAPE FILE"
120 PRINT#1,N1$
130 PRINT#1,N2$
140 PRINT#1,N3$
150 CLOSE 1
160 END
```

```
100 OPEN 1,1,0,"TAPE FILE"
110 INPUT#1,N1$:PRINT N1$
120 INPUT#1,N2$:PRINT N2$
130 INPUT#1,N3$:PRINT N3$
140 CLOSE 1
150 END
```

The program stopped because the null string was ignored—thus, N2$ was assigned the value intended for N3$. When INPUT# went looking for N3$, the file was empty. To avoid this potential pitfall, never print a null string. If you want INPUT# to retrieve a null string, have PRINT# print a space. INPUT# will not ignore the space, but when assigning the string variable, it will skip past all leading spaces. The effect is that the variable is assigned a null string.

An advanced feature of tape files is to use a command number of 2 in the OPEN statement. This has the same effect as using command 1, and prepares the file for output. The difference is that when the file is closed, it will be specially marked to indicate that it's the last file on the tape. This guarantees that when searching for a tape file, the computer will not look past the last file. Instead, the computer stops with a DEVICE NOT PRESENT error, which is equivalent to FILE NOT FOUND on the disk drive.

The direction of a tape file cannot be changed as long as it's open. If a tape file is opened for input and PRINT# is used, the NOT OUTPUT FILE error occurs. Likewise, the NOT

INPUT FILE error appears when INPUT# is used with a file that has been opened for output.

Before ending this section, it would be a good idea to stress the importance of using CLOSE. This statement is not so important when working with the keyboard or screen, but it's required when using tape or disk files. The Datassette can read only one file at a time. If a tape file opened for input is not closed, no other files can be opened until CLOSE is executed.

Much worse is when a CLOSE is accidentally omitted for a tape file that has been used for output. It's possible that some of the data will not be sent if the CLOSE is not performed. The file will be incomplete and the data lost—a program which later opens the file for input will be able to read only the first part, and may even crash because of an error.

In other words, a program should *always* perform CLOSE when communication with a tape file is completed.

Summary

- A tape file is not limited to storing only the lines of a program. Tape files can also contain data, in the form of numbers and strings, which can be used by a program.
- The Datassette can be used for both input and output, but can handle only one direction at a time. The direction (input or output) must be specified when the file is opened.
- Opening a tape file uses the full syntax of the OPEN statement: OPEN *file number, device number, command number, filename.*
- The device number for the Datassette is 1.
- A command number of 0 opens the file for input; 1 opens the file for output. Using 2 also opens the file for output, but marks the file as the last on the tape when it's closed. The command number is optional—by default, the file will be opened for input.
- A filename is optional but recommended. If none is specified when the file is created, none can later be used when the file is opened for reading.
- It's a good idea not to use a variable list after PRINT#. Commas in PRINT statements are used to format the screen display into columns, which is not necessary with tape files. Instead, output the items one at a time so that a carriage return will be printed after each. If you do use a variable list, print a CHR$(13) between each item.

- It is acceptable to use a variable list with INPUT#.
- A program should input variables in the same order they were written.
- The FILE DATA error occurs when a program tries to read character data into a numeric variable.
- If a program is expecting a string variable and a number is read instead, the string representation of the number, like that produced by the STR$ function, will be assigned to the variable.
- An error such as STRING TOO LONG appears when a program tries to read more information from a file than was written in the first place.
- There are three methods to avoid reading more data than exists in a file. One way is to use the first number to indicate how many items are in the file. This number can then be used to control a loop which reads the data.
- The other two methods work by detecting the end of the file while it's read. One way, often used with READ and DATA statements, is to use a flag, such as the word *END* or the number *−999*, as the last item written to the file.
- The third method is to stop when bit 6 of the status variable (ST) is set. ST has a value of 64 when the end of the file is reached, and has other nonzero values when I/O errors occur.
- INPUT# ignores null strings. If it reads a null string, it goes to the next item. To print a string so that it will be read as a null, print a single space.
- It's very important that CLOSE be executed when the computer is finished communicating with a tape file. An output file may not be written completely if this is not done.
- Only one file can be opened to the Datassette at a time. Once a file has been opened, its direction (input or output) cannot be changed.
- If a file is opened for input and PRINT# is used, the NOT OUTPUT FILE error occurs. If a file is opened for output and INPUT# is used, the NOT INPUT FILE error will occur.

Disk Files

The type of file just described for cassette is often referred to as a *sequential file*, one in which the information can be read back only in the order it was written. It's not possible to reread just one item in the file; you have to use OPEN again

and start at the beginning. With a sequential file it's also not possible to skip some items to read one later on. If a file has just been opened and you want to read only its last item, you have to read past all the other items first.

Sequential files are the simplest and most commonly used type of file. They are well-suited to the Datassette, because the computer cannot control the REWIND and F.FWD buttons. The nature of tape means that the Datassette can handle only sequential files.

But sequential files can also be used with the disk drive. There are a couple of differences in the OPEN statement, but otherwise the procedure is the same.

OPEN *file number, device number, secondary address, string*

Any **file number** from 1 to 127 is acceptable. The **device number** for the disk drive is usually 8, although it can be changed to 9 when a second drive is used. The command number, also called the **secondary address,** serves a different purpose when using disk files. It will be explained in a moment. For now, just use 2. The **string** is required, and indicates the filename, type of file, and direction. The filename can be up to 16 characters, and is sometimes followed by the type of file (indicated by an *S* representing sequential) and perhaps a letter *R* or *W*, reading or writing. Commas are necessary to separate the filename, the type, and the direction. You *must* specify the type of file if you're writing to the disk—if you're only reading the file, you can omit the *S* designation. If no direction is indicated, the computer will open the file for reading. Here are some examples.

```
OPEN 1,8,2,"DOODLE,R" : REM READING, NO TYPE NEEDED
OPEN 1,8,2,"POEM,S,W" : REM WRITING, TYPE SPECIFIED
OPEN 1,8,2,"GAME" : REM READING ASSUMED
```

With minor changes, the high-score demonstration from the "Files on Tape" section can be used with a disk file. Type in both programs below. Load and run the first (make sure there's a disk in the drive). This creates a sequential file on disk. Now load and run the second. It reads the file from disk and displays the items on the screen.

```
100 OPEN 1,8,2,"HI SCORES,S,W"
110 PRINT#1,"KEVIN" CHR$(13) 80
120 PRINT#1,"CHRISTINE" CHR$(13) 90
130 PRINT#1,"ALISDAIR" CHR$(13) 91
```

```
140 PRINT#1,"LISA" CHR$(13) 87
150 CLOSE 1
160 END
```

```
100 OPEN 1,8,2,"HI SCORES"
110 INPUT#1,N$,SC
120 PRINT N$,SC
130 IF ST=0 GOTO 110
140 CLOSE 1
150 END
```

Examine the directory of the disk you just used. You'll notice that beside the filename HI SCORES is a three-letter code, normally PRG for a program file, but now SEQ. This is to remind you that it's a sequential file, contains data, and should not be loaded as a program.

The method of reading and writing a sequential file is pretty much the same for both tape and disk. A major difference between these two devices, though, is in the number of files that can be open at one time. While the Datassette can read or write only one file at a time, the disk drive can handle several files simultaneously. Files 1 and 2 could be open for reading, while file 3 would be open for writing. The only restriction when using multiple disk files is that each file must have a different secondary address. Secondary address numbers can range from 2 to 14.

```
100 REM PAYROLL PROGRAM
110 OPEN 1,8,2,"EMPLOYEES,R"
120 OPEN 2,8,3,"HOURS,R"
130 OPEN 3,8,4,"PAYCHECKS,S,W"
```

This program might read an employee's name and Social Security number from the first file, get the hours and wage information from the second, do whatever processing is necessary, and then send the results to the third. This last file would be used later to print paychecks. The example may not be practical, but it does illustrate how several disk files can be managed. Most applications don't require as many files open at the same time.

Another capability of the disk drive is that it can allow two open files to read from the same disk file.

```
100 OPEN 1,8,2,"WEATHER DATA"
```

(later in the program, when file 1 may still be open)

```
650 OPEN 2,8,3,"WEATHER DATA"
```

This will not work when a disk file is written. Once a disk file has been opened for output, it cannot be opened by a different file number for either input *or* output until it's closed.

A complication when using multiple files is that the status variable (ST) must serve for all files. ST always reflects the status of the most recently accessed file. This means that if a program reads from file 1, then from file 2, ST indicates the status of file 2—file 1's status information is lost. The solution is to assign ST to a temporary variable before accessing file 2. This variable can then be used to determine the first file's status. This next program illustrates the procedure.

```
100 REM READ ONE FILE TO ANOTHER
110 OPEN 1,8,2,"SOURCE,R"
120 OPEN 2,8,3,"DESTINATION,S,W"
130 INPUT#1,S$
140 A=ST
150 PRINT#2,S$
160 IF A=0 GOTO 130:REM BASED ON FILE 1 STATUS
170 CLOSE 1
180 CLOSE 2
190 END
```

There's another problem that you'll encounter when a disk file must be periodically updated. A file which maintains high scores, for instance, may be read when a game is finished to compare the current score against the high score. If the player did well and the high score file has to be modified, the old file has to be erased and a new one written using the same filename. This poses no problem with tape files since the tape can be positioned to start at the beginning of the old file. The disk drive, however, won't let you write a file with a filename already in use. A different solution must be found.

Secondary addresses from 2 to 14 are available for normal disk file handling. Addresses 0 and 1 are reserved for LOAD and SAVE format files. The number *15* is used to send commands to the disk drive. One of these commands is called **Scratch,** and informs the drive that it should erase a file on the disk. For example, the file "POEM" could be erased by entering this line in immediate mode.

OPEN 1,8,15:PRINT#1,"S0:POEM":CLOSE 1

This is one case where no string has to be included as part of the OPEN statement. The secondary address (15) indicates that the *command channel* will be referenced, so no filename or type of file is needed. The command is sent as a one-letter code, followed by a zero for the drive number, a colon, and then the filename. You can also scratch files with this procedure from within a program.

If a program is going to open a disk file for output, and there's a chance the filename may already exist, the program should first send the Scratch command. (No error occurs if an attempt to scratch a nonexistent file is made.) Here's an example of scratching a file before writing a revised version of it to disk.

```
100 OPEN 1,8,15
110 PRINT#1,"S0:HI SCORES"
120 CLOSE 1
130 OPEN 1,8,2,"HI SCORES,S,W"
140 REM CONTINUE WITH REST OF PROGRAM
```

Perhaps you've used the *at (@)* character to save and replace a file. Assuming there's a file on disk named GAME, the statement SAVE "@0:GAME",8 scratches the old file GAME, then saves the current BASIC program using that filename. This replaces the file on the disk with the one in memory.

Do not use the @ character. This feature may not always work correctly. In some cases, the disk drive may even tamper with other files on the disk. Even though using the @ character to save and replace may seem very convenient, my advice is to avoid it. Use the Scratch (S0:) method instead.

Other useful disk drive commands are New, Rename, and Validate. **New** is used to format a disk and clear its directory. The full format of the command is "N0:*diskname, identification*", where the disk name is up to 16 characters and the disk identification 2 characters long. Make sure that no two disks are given the same disk identification. This precaution insures that there's no confusion between the source and destination disks when copying files from one to the other.

If the identification is omitted, all the files will be erased and the new disk name will be used, but the disk will *not* be reformatted. This can come in handy when you want to use the disk for a new project, but it's already been formatted. Formatting really needs to be done just once. Skipping the

process saves time. Take a look at the following lines; they show how to format a disk, as well as how to create a new directory only.

OPEN 1,8,15
PRINT#1,"N0:GAMES,01" (format disk and create new directory)
or

PRINT#1,"N0:GAMES" (create new directory only)
CLOSE 1

Rename lets you change the name of a file without disturbing its contents. Here's the proper syntax:

PRINT#1,"R0:*new filename = old filename*"

This example shows how a file named DOODLE can be renamed as POEM (assuming the OPEN has already been executed).

PRINT#1,"R0:POEM=DOODLE"

When using Rename, make sure the new filename is not already in use on the disk.

Validate is used to recalculate the number of free blocks on a disk. If some blocks have been allocated for files, but have not been used, they'll be made available after the Validate command is sent. This command also deletes any files which were never properly closed. Such files are marked in the directory with an asterisk (*) before the three-letter file type. Validate does not alter any properly closed files. If you validate periodically, you may be able to get a few more free blocks, letting you squeeze more on the disk.

The secondary address of 15 refers to the command channel when PRINT# is used. INPUT# can be used with this channel to get error messages. However, BASIC reports only a few I/O error messages, such as FILE NOT FOUND and DEVICE NOT PRESENT. When an error occurs, the disk drive can provide more information if you run the following short program.

```
10 OPEN 1,8,15
20 INPUT#1,A,B$,C,D
30 CLOSE 1
40 PRINT A;B$;C;D
```

Normally, the message 0 OK 0 0 will appear, indicating all is well.

If you tried to save a program using a filename already in

use, BASIC would not print an error message, but the drive light would flash. Run the above program, and you'll see the message 63 FILE EXISTS 0 0. Error number 63 tells you that an attempt was made to create a new disk file with an existing filename. Other error numbers and their descriptions can be found in the disk drive user's manual.

(The status numbers returned in variables C and D are for advanced applications, and can be ignored.)

There are a couple of differences between normal disk files (which use secondary addresses 2–14) and a file opened to the command/error channel (secondary address of 15). No direction is specified when the file is opened, and it's possible to use both PRINT# and INPUT# without having to close and reopen the file.

The other difference is that when the command/error channel is closed, the disk drive closes all the other disk files as well. BASIC may think that the files are still open, but an error will occur when the program tries to read from them. Thus, if a program is going to use the command/error channel while other disk files are open, it's best to open the channel during initialization and leave it open until the program ends. Many people set aside file 15 for this purpose.

Sometimes it would be handy if a program could read the disk directory. This is easy to do, but several things must be done differently. First, the OPEN statement should use "$0:" (or just "$") as the filename. Also, because of the structure of the directory file, INPUT# cannot be used. The file has to be read a character at a time. Fortunately, this can be done by the GET# statement.

GET#'s syntax is the keyword GET#, followed by a file number, a comma, and then a variable list. Something like GET#1,A$ retrieves the next character from the file and assigns it to A$. Only string variables should be used with GET# to avoid the FILE DATA error.

The following demonstration shows how to read a sequential file character by character.

```
10 OPEN 1,8,2,"HI SCORES"
20 GET#1,A$:PRINT A$;:IF ST=0 GOTO 20
30 CLOSE 1
```

The directory file contains characters for block lengths and filenames. The first two characters in the file can be ig-

nored. The first line in the file is the directory header, giving the disk name, identification, and DOS (Disk Operating System) version. Several lines follow, one for each entry. Again, the first two characters of each line can be ignored. The next two form the low-byte and high-byte number which indicates the file's block length. A few spaces precede the opening quotation mark, which begins the filename. After the filename and the closing quotation mark, there are several more spaces and then the three-letter file type. Each line ends with a CHR$(0), interpreted by GET# as a null string. This format repeats for each line in the directory. The block length for the last line shows the number of free blocks on the disk, with the filename BLOCKS FREE.

The end of the directory file is detected by checking ST after reading the block length. This program illustrates how it's done.

Program 3-1. Directory Reader

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
10 OPEN 1,8,0,"$":GET#1,S$,S$                    :rem 204
20 GET#1,S$,S$,L$,H$:IF ST THEN CLOSE 1:END
                                                 :rem 103
30 PRINT ASC(L$+CHR$(0))+256*ASC(H$+CHR$(0));
                                                 :rem 163
40 GET#1,S$:IF S$>"" THEN PRINT S$;:GOTO 40:rem 66
50 PRINT:GOTO 20                                 :rem 199
```

The directory file can only be read and cannot be opened for output.

A handy trick with reading the directory is to put a filename after the $0:, as in "$0:POEM". This makes the disk drive search the directory for the filename POEM and list it if it's on the disk. If there's no file named POEM, there'll be no directory lines read after the header. This is useful when you want a program to check if a file already exists. If the directory is opened with the filename after the $0: and no filename lines are read, the file isn't on the disk.

A variation on this technique is to use *wild cards* in the filename. If you use a question mark (?) as one of the characters in the filename, the disk drive will ignore that character position when searching for filenames. Let's say a disk contains the following files.

DOODLE
GAME1PLAYER
GAME2PLAYER
POEM
GMINUET
AMINUET
SONATA1
SONATA22
SONATINA1
SONATINA3
DMINUET

Using the string "$0:?MINUET" would read only the three minuet files, since the ? forces the disk to ignore the *G, A,* and *D* characters in the files. If you used the string "$0:GAME?PLAYER", both GAME1PLAYER and GAME2PLAYER would be read. If necessary, several question marks can be used in the same filename.

A second wild card is the asterisk (*). While the question mark can take the place of only one character, the asterisk can be substituted for several. An asterisk at the end of a filename makes the disk drive search for all filenames which begin with the letters up to the asterisk. All characters at the position of the asterisk and after are ignored.

In the example directory, if "$0:SON*" were used as the filename, all files starting with the letters *SON* would be returned in the directory, four altogether. With "$0:SONATA*", only the filenames SONATA1 and SONATA22 would be read.

A combination of the ? and * wild card characters can be used in the same filename.

When loading a program or opening a file for reading, wild cards within the filename make the computer access the *first file* on the disk which matches. This can come in handy if you can't remember or don't want to type the full name of a program. LOAD "PROG*",8, for instance, loads the first filename on the disk which has PROG as its first four characters.

Wild cards can be very powerful if you choose filenames to accommodate them. Just be careful that the filenames referenced with wild cards are the ones you wanted. This is especially true when using wild cards with the Scratch command. It's quite easy to erase too many files—more filenames may fit the wild card pattern than expected. In fact, it may be

best to avoid using wild cards with Scratch. At least check which files will be scratched beforehand by reading the directory with the wild card pattern.

As you've seen, sequential files are identified in the directory by the letters *SEQ*. The disk drive also supports another three-letter code, *USR*, for the same type of file. A USR file is handled just like a sequential file; the only difference is in the directory listing. Since some system utilities such as the Editor use SEQ files, you may want to use the USR type in your programs to help distinguish program data files from system files.

The full syntax for the string in the disk OPEN statement is:

filename, optional type, optional mode

The type can be SEQ, USR, PRG, or REL. These last two types will be discussed in a moment. If no type is specified, the default SEQ is used. (Remember that the type is *not* optional if you're writing. It must be included.) The mode tells the direction, and can be Read or Write. Default is Read.

Both the type and mode permit single-letter abbreviations; the type can be designated by S, U, P, or R, and the mode can be R or W. A file identified by the letters USR in the directory could be opened for output with:

OPEN 1,8,2,"HIGH SCORES,U,W"

From there, the handling would be just as if the file were of the type SEQ.

File type PRG is reserved for programs stored and retrieved using the SAVE and LOAD commands. It's possible to open these files and read them, but the reading has to be done one character at a time. Another problem is that due to the way the program is stored, the file may contain several nonprinting characters. The best way to examine a program file is to print the ASCII value of each character with something like this routine.

```
10 OPEN 1,8,2,"PROGRAM,P":REM DEFAULTS TO READING
20 GET#1,S$
30 PRINT ASC(S$+CHR$(0))
40 IF ST=0 GOTO 20
50 CLOSE 1
```

The +CHR$(0) in the ASC function (line 30 above) is needed because the file may contain several CHR$(0) characters. A CHR$(0) is interpreted by GET# as a null string, which

when used as the argument of the ASC function makes BASIC stop with an error. This can be averted by appending a CHR$(0) to the argument. If the string does contain characters, the extra CHR$(0) at the end won't affect the ASC function because ASC looks only at the first character. If the string is null, it becomes CHR$(0), which makes ASC return the 0, the correct value.

Reading a PRG file is useful only for advanced applications, such as programs which search for all variable names in a program, renumber a program, or compact a program by removing extra spaces and combining lines.

The one remaining type of file supported by the disk drive is the REL, or *relative,* file. Unlike a sequential file, this type of file makes it possible to back up to an earlier part of the file without having to use OPEN again, and to skip past items in the file without reading or writing them. A relative file is also known as a *random access* file because the reading or writing of items can be done arbitrarily, without having to follow any order.

Such a file structure is useful when a file contains several entries, only a few of which may have to be changed at any time. If it's necessary to read an account for a customer near the end of the file, and update the account of a customer near the beginning, these read and write operations can be performed quickly without disturbing the rest. REL files make it easy to handle large amounts of information. In fact, it's possible for a REL file to contain more data than could be held in memory at one time.

Unfortunately, Commodore 64 BASIC does not directly support relative files; it's a very complicated subject, far beyond even the nature of these discussions of advanced BASIC.

Summary

- In a sequential file, data must be read in the same order as it was written. It's not possible for reading to back up or skip ahead.
- With a few changes to the OPEN statement, sequential files can be used with disks. The syntax to open a disk file is OPEN *file number, device number, secondary address, string.*
- File numbers should range from 1 to 127. The device number for the disk drive is usually 8, but can be 9. Secondary ad-

dresses 0 and 1 are reserved for use by the LOAD and SAVE commands. The numbers 2–14 are used for data files. Each open file must have a different secondary address. A secondary address of 15 is used to access the command and error channel of the disk drive.

- The string consists of a filename, a file type, and a direction. The filename is required and can be up to 16 characters. The file type is optional when reading, mandatory when writing. File type can be SEQ, USR, PRG, or REL (abbreviated as S, U, P, and R, respectively). If no file type is specified, SEQ is assumed. The direction is optional, and can be reading or writing (or R or W). If no direction is given, the file is opened for reading.
- Unlike the Datassette, several different files can be opened to the disk drive simultaneously. Some can be set for input, others for output.
- Two or more files can be opened for input from the same disk file at the same time. However, once a disk file has been opened for output, it cannot be read or written by any other file until it is closed.
- The disk drive command and error channel is accessed by opening a file with a secondary address of 15 and no string, as in OPEN 1,8,15. No filename or direction has to be specified.
- While the file is open, PRINT# can be used to send commands to the drive, and INPUT# can be used to get status information. Unlike normal disk files, the file does not have to be closed and reopened to switch between input and output.
- The Scratch command is used to erase one or more files on a disk. The command string is "S0:*filename*". The *S* represents the Scratch command, the *0* is the drive number, and the filename indicates which file is to be erased. *If the filename contains wild card characters, more than one file may be erased.*
- Do not use the @ character in the filename of an OPEN or SAVE to replace and save a file on a disk. This feature is not reliable and can damage the contents of a disk.
- The New command is used to format a disk and clear the directory. The command string is "N0:*diskname,id*".
- The Rename command is used to change the name of a file. The command string is "R0:*new filename=old filename*". The contents of the file are not altered in any way.

- Validate is used to make previously allocated blocks not used by files available on the disk. Files that were never properly closed (indicated by an asterisk before the file type) will be erased. Properly closed files will be left alone. After the disk has been validated, there may be a few more free blocks.
- BASIC reports only the errors FILE NOT FOUND and DEVICE NOT PRESENT, and sometimes doesn't report an error at all. A flashing light on the disk drive may be the only clue that an error has occurred. More specific error messages are available from the disk drive. Assuming that file 1 has been opened to the error channel, the statement INPUT#1,A,B$,C,D will read the error number into A, the error message into B$, and other values for advanced applications into C and D. When no error has occurred, the error channel will return the values 0, OK, 0, 0.
- The GET# statement is used to input single characters from a file. The syntax is the keyword GET#, the file number, a comma, and a variable list. Only string variables should be used with GET#.
- The directory can be read by a program by using OPEN with the filename "$0:" and no file type or direction. The format of the directory file requires that it be read one character at a time, using GET#.
- The directory file will contain all the filenames on the disk when the string used with OPEN is $0:. To make the disk drive search for a specific filename in the directory, put the filename at the end of the string, as in "$0:DOODLE". The corresponding filename line will be read if the file exists. If not, the directory file will contain only the disk header and free blocks lines.
- The wild card characters (? and *) can be used in the filename after $0: to list all filenames which match a certain pattern.
- When wild cards are used with LOAD or OPEN for input, the computer will access the first file on the disk that has a filename matching the wild card pattern.
- The file type USR is identical to the file type SEQ. The different name can be useful to help distinguish between sequential files used for different purposes.
- The file type PRG is used for LOAD and SAVE type files, including BASIC and machine language programs. In ad-

vanced applications, a PRG file can be read or written one character at a time.
- Another type of file is the *relative* file, identified as REL. In a relative file, reading or writing can jump back or ahead. No particular order must be followed. Relative files, although supported by the disk drive, are not supported by Commodore 64 BASIC.

The Modem and Other RS-232 Devices

RS-232 refers to an agreement among manufacturers of electronic equipment (Electronic Industries Association) which defines how devices communicate. Its specifications include such things as power levels, which connector pins serve which functions, and so on. This insures that products from different manufacturers will be more or less compatible. Like ASCII, it's one of the few standards in the computer industry.

The RS-232 standard applies to *serial communication.* The usual method of transfer is parallel communication, in which information is sent a byte at a time. All eight bits of each byte are sent simultaneously. In serial communication, however, information is sent a bit at a time. To send a full byte, the eight bits have to be sent separately, one after another. Devices which communicate serially include modems and some printers.

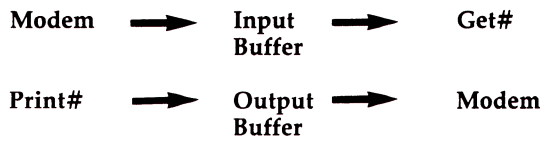
A *modem* converts bits into tones which can be sent over telephone lines to another computer. The receiving computer must also be connected to a modem, which then converts the tones back into bits. The general term for this is *telecommunications.* The process of changing bits into tones is called *modulation,* and the process of converting the tones back into bits is called *demodulation.* In fact, the word *modem* is derived from the words *MOdulate* and *DEModulate.*

In telecommunications, *transmit* is often used to mean *send.* The modem is different from most other devices because it can both transmit and receive information at the same time. But a computer can handle only one character at a time. If a character comes in while the computer is busy sending another, the computer must remember that incoming character and deal with it later. It's also possible for the computer to output characters faster than the modem can transmit them. To avoid losing incoming characters or to hold outgoing characters until the modem can take them, a *buffer* has to be

used. A buffer is a temporary holding place for characters. When using the modem, the computer needs two buffers, one for input and another for output.

To better understand the concept of a buffer, let's consider an example using an input buffer. Characters received are first stored in the input buffer, in the order in which they are received. Every time GET# executes, it retrieves the next character from the buffer. If the buffer is empty, GET# returns with the null string. The process could be illustrated like this:

Figure 3-1. Buffer



The following steps show how the method might work in practice.

1. The buffer starts out empty.

   _ _ _ _ _

2. The modem receives an *M* character and places it in the buffer.

   M _ _ _ _

3. The GET# statement retrieves the first character from the buffer (M). The buffer is emptied.

   _ _ _ _ _

4. The modem receives the character *A* and places it in the buffer.

   A _ _ _ _

5. The modem receives another letter, *R*, before the computer has retrieved the previous character. That's okay, since the *R* can be placed in the buffer after the *A*.

   A R _ _ _

6. GET# draws off the first character in the buffer (A). The *R* is now the first character in the buffer.

   R _ _ _ _

7. GET# gets the *R* and the buffer is once again empty.

   _ _ _ _ _

8. GET# tries to get another character, but since the buffer is now empty, GET# returns a null string. The program keeps looping until it gets something other than a null string.

9. The modem eventually receives a letter *K* and puts it in the buffer.

   K _ _ _ _

10. The next time GET# executes, it returns with *K* instead of a null string. The buffer is empty.

    _ _ _ _ _

Two important observations should be made. The modem received the characters in the order *M–A–R–K*, and the program (with GET#) retrieved the characters in that order. Even though the computer wasn't always able to retrieve each character as it came in, the buffer insured that no characters were lost.

The output buffer works in much the same way. Characters sent by PRINT# are held in the buffer until the modem is ready to transmit them.

Here's the syntax for the OPEN statement when used with an RS-232 device.

OPEN ***file number, device number, secondary address, string***

File numbers can range from 1 to 127. Numbers from 128 to 255 cause the computer to send a linefeed (CHR$(10)) after every carriage return. The linefeed character is required for printers which do not automatically advance the paper. The **device number** for an RS-232 device is 2. The **secondary address** is not used and should be 0. The **string** can be one or two characters. The first character is a control value, and the second is an optional command value.

The **control character** sets the number of stop bits, the word length, and the baud rate. Remember that each character is sent as a sequence of bits. Stop bits help the receiving modem detect the end of each character. The Commodore 64 can support either one or two stop bits, depending on the value of bit 7 in the control character.

Bit 7	Value	Effect
0	0	1 stop bit
1	128	2 stop bits

The *word length* determines how many bits are transmitted for each character, and can range from five to eight bits. Bits 6 and 5 of the control character establish the word length. For most purposes, eight bits will be just fine. When communicating with some modems or printers, especially older ones, fewer bits may be required. The device owner's manual should tell you if less than eight bits are to be used.

Bits 6	5	Value	Effect
0	0	0	8-bit word
0	1	32	7-bit word
1	0	64	6-bit word
1	1	96	5-bit word

The *baud rate* determines how quickly the bits are transmitted. The most common rate is 300 baud (roughly 300 bits per second), which is equivalent to about 30 characters per second. Faster baud rates (600 or 1200, for instance) require higher quality circuitry and make devices which are capable of such rates more expensive. Older printers and modems may even operate at 110 baud. Bits 3 through 0 of the control character should be set according to the baud rate of the device.

Bits 3	2	1	0	Value	Effect
0	0	0	1	1	50 baud
0	0	1	0	2	75 baud
0	0	1	1	3	110 baud
0	1	0	0	4	134.5 baud
0	1	0	1	5	150 baud
0	1	1	0	6	300 baud
0	1	1	1	7	600 baud
1	0	0	0	8	1200 baud

Here's the formula for calculating the value of the control character. Place the bit values from the tables above in the appropriate places in the formula.

Control character = CHR$(stop bits + word length + baud rate)

A typical setting is one stop bit, eight-bit word length, and 300 baud.

Control character = CHR$(0 + 0 + 6) = CHR$(6)

The result would be used in the OPEN statement like this:

```
OPEN 1,2,0,CHR$(6)
```

The **command character** sets the parity, duplex, and handshaking. *Parity* is used to check for errors in transmission. If there's considerable static on the telephone line, a bit may be accidentally switched in value, resulting in garbled communications. Parity checking can detect an error like this. When parity checking is selected, it can be odd or even. The correct setting depends on the setting of the device at the other end of the link. Bits 7 through 5 control the parity.

Bits 7	6	5	Value	Effect
0	0	0	0	No parity
0	0	1	32	Odd parity
0	1	1	96	Even parity

When using a word length of eight bits, parity checking is not available—no parity should be selected.

Duplex determines whether both modems can transmit at the same time or if they have to take turns. The usual value is full duplex. The other setting, half duplex, is rarely used. Again, this setting depends on the receiving modem. If half duplex is desired, bit 4 of the command character should be set (to 1). In most cases, especially with newer modems, full duplex is available, so bit 4 should be clear, or 0.

Bit 4	Value	Effect
0	0	Full duplex
1	16	Half duplex

Handshaking refers to how the transmitting and receiving devices recognize and establish communication between each other. Handshaking is controlled by bit 0, which for most applications should be clear, or 0.

As you can see, the command character is not as important as the control character. If the sum of the command values is zero, the command character does not have to be used. If a different value is needed, the command character is concatenated to the control character.

This next OPEN statement shows how that's done to select half duplex (bit value of 16).

```
Command character = CHR$(16)
OPEN 1,2,0,CHR$(6)+CHR$(16) : REM SELECT HALF DUPLEX
```

When the OPEN statement is executed, it creates input and output buffers of 256 bytes each. This allotment of 512 bytes comes from free memory. It forces an automatic CLR, which erases all previous variables and closes all other open files. Thus, if a program is going to use an RS-232 device, the OPEN should come before any variable assignments, array dimensions, function definitions, or other OPEN statements.

The 512 bytes are restored when the RS-232 file is closed. CLOSE causes another CLR, so the CLOSE should be the last thing in the program.

The variable ST has a different meaning when used with an RS-232 device. The last three bits are set to indicate errors.

Bit	Value (set)	Error when bit set
0	1	Parity error
1	2	Framing error
2	4	Receiver buffer overflowed

A parity error occurs when a bit has been switched in value, probably due to a bad connection. Don't bother to check this bit when using a word length of eight bits.

IF ST AND1 THEN parity error

(Note: Due to a quirk in BASIC, there must be a space between ST and AND.)

A framing error also indicates a problem in receiving a character.

IF ST AND2 THEN framing error

The buffer OVERFLOW error occurs when the input buffer is full (contains 256 bytes) and another character is received before the program retrieves a character. This can happen when the program handling the input and output is too slow for the baud rate used. A BASIC program can run only so fast. If the program has to do a lot of processing and a fast baud rate such as 600 or 1200 is used, this error may occur. The solution is to write the program in machine language.

IF ST AND4 THEN buffer overflowed

Another point about the ST variable is that it can be read only once after an input/output statement. After it's read, it's automatically set to zero. If you need to read it more than once, assign it to a temporary variable.

RS-232 communication may seem complicated in its description, but it can be very easy to implement. To show just

how easy it is, take a look at Program 3-2, "Modem." It's a simple program which makes your computer and modem act as a terminal, allowing you to communicate with another computer, perhaps one running a bulletin board system. The program is set for 300 baud and cannot handle faster rates. To use this program, you must have a modem connected to your Commodore 64 (and another computer with a modem to receive your transmission).

After typing in and running Program 3-2, make sure your modem is properly connected. Access the receiving computer as you would normally, by dialing its telephone number. Although the program establishes full duplex, you can switch your modem to half duplex (if it has that feature) to see what you type on the screen.

Program 3-2. Modem

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
10 OPEN 1,2,0,CHR$(6):POKE 56577,0          :rem 5
20 DIM T%(255),R%(255):FOR K=1 TO 255:T%(K)=K:R%(K
   )=K:NEXT                                  :rem 112
30 FOR K=65 TO 90:T%(K)=K+32:R%(K)=K+128:NEXT:FOR
   {SPACE}K=97 TO 122:R%(K)=K-32:NEXT        :rem 245
40 FOR K=193 TO 218:T%(K)=K-128:NEXT:T%(20)=127:R%
   (127)=20:PRINT "{CLR}{N}"                 :rem 234
50 GET A$:IF A$>"" THEN PRINT#1,CHR$(T%(ASC(A$)));
                                             :rem 122
60 GET#1,A$:IF A$>"" THEN PRINT CHR$(R%(ASC(A$)));
                                             :rem 121
70 GOTO 50                                    :rem 5
```

A few comments about this program might be helpful. Lowercase is obtained by using the second character set. Unfortunately, this means that some character translation has to be done to convert between ASCII codes and character set numbers. The translation process works like this:

Receive	Print on screen as
0–64	0–64 (punctuation)
65–90	193–218 (uppercase)
91–96	91–96
97–122	65–90 (lowercase)
123–126	123–126
127	148 (backspace)

Keyboard	Transmit
0–64	0–64 (punctuation)
65–90	97–122 (lowercase)
91–96	91–96
148	127 (backspace)
193–218	65–90 (uppercase)

The fastest way to translate is with arrays. Arrays T% and R% have been set up for this purpose (line 20 in Program 3-2).

There are a couple of restrictions in using an RS-232 device. First, only one such device can be used at a time. Second, an RS-232 device should not be used while another peripheral, such as the Datassette or disk drive, is in use.

The last topic concerns receiving a CHR$(0). If the modem receives a CHR$(0), it stores it in the buffer properly, but the GET# statement interprets it as a null string. This means a null string can indicate either that the input buffer is empty or that the character is a zero character. To determine which, check bit 3 of ST after the GET# is executed. If the bit is set (has a value of 8), the input buffer is empty. If the bit is clear (value of 0), the character is a CHR$(0).

IF ST AND8 THEN input buffer is empty
IF (ST AND8) = 0 THEN it's a CHR$(0)

The CHR$(0) normally is not used in ASCII communication and can just be ignored. It's important, though, in such activities as transferring a program file by modem, in which case every character is important.

Summary

- RS-232, like ASCII, is one of the few standards in the computer industry. It specifies how computers and devices communicate.
- The RS-232 convention applies to serial communication, where bits of a byte are sent one at a time. Parallel communication, the other method of data transmission, transfers all eight bits at the same time.
- The modem is used to convert bits into tones that can be carried by telephone lines to another computer which converts the tones back into bits. This process is called modulation and demodulation.
- The modem can transmit and receive at the same time. No direction has to be specified as part of the OPEN statement.

- A buffer is a temporary character-holding place. An input buffer insures that no incoming characters will be lost if some come in too fast for the computer to process. An output buffer holds characters until the modem is ready to transmit them.
- The syntax for OPEN to open an RS-232 file is the standard OPEN *file number, device number, secondary address, string.*
- File numbers from 1 to 127 are most frequently used. File numbers from 128 to 255 can be used with a serial printer which needs linefeed characters. The device number for an RS-232 device is 2. The secondary address is not used and should be set to 0. The string is one or two characters. The first character is a control character which sets the number of stop bits, the word length, and the baud rate. The second character is optional and is rarely used. It sets the parity, duplex, and handshaking.
- When an RS-232 OPEN is executed, 512 bytes are taken from free memory and are used for buffers. If there are not 512 bytes available, the end of the program is overwritten.
- The OPEN performs a CLR, which erases all variables, arrays, and defined functions, and closes all open files. An OPEN to an RS-232 device should be done as part of the program's initialization, before the assignments, dimensions, definitions, and OPENs. A CLR is also performed when a file to an RS-232 device is closed.
- Only one file to an RS-232 device can be open at any time. Communication with other devices is not allowed when an RS-232 file is open.
- With RS-232 communication, the status variable, ST, indicates errors.
- Translation is required to convert between ASCII and Commodore 64 character codes.

The End of BASIC

Using Commands in Programs

Normally, commands such as RUN, NEW, LIST, CONT, LOAD, and SAVE are used only in the immediate mode, without line numbers. But commands can also be used in programs, some with better results than others.

RUN. The RUN command can be used to make a program automatically rerun when it's done. Just place RUN where the END statement would normally go. RUN first performs a CLR, erasing all variables, arrays, and defined functions before starting execution at the program's first line. If a line number is placed after RUN, execution begins with that line.

NEW. Another way to end a program is to use NEW. The NEW command both ends a program and erases it. (This isn't recommended when you're still debugging a program, for you'll have to continually reload it.)

LIST. The LIST command is not very useful in a program. When executed, all or some of the program lines will be listed, depending on the optional line number range after the keyword. Execution then ends. If there are statements after the command, they won't be executed.

CONT. When used in the program mode, the CONT command puts BASIC in an infinite loop, and no processing is done. To break out of the loop, press the RUN/STOP key. CONT is best used only in the immediate mode.

LOAD. Using LOAD in program mode is somewhat complicated, so let's first examine what happens when programs are saved and loaded. A BASIC program is stored in memory as a sequence of bytes. The address of the program is usually 2049, but it can be different for nonstandard system configurations (such as those with special device handlers). Whatever the address, the SAVE command will start at that address and send the bytes which form the program to the disk or tape file. When LOAD is executed, the bytes in the file are read into memory starting at the address of BASIC memory.

LOAD's full syntax is:

LOAD, a *filename* (optional for tape), a *device number* (optional), and a *secondary address* (optional).

If no device number is given, device 1 (the Datassette) is assumed. If no secondary address is given, the default of zero is used.

A secondary address of 0 means that a relocating LOAD should be performed. The program will be loaded starting at the current address of BASIC's free memory. This insures that a file which was saved at one address can be loaded when BASIC memory starts at a different address. Relocating LOADs are used mainly when loading BASIC programs. The BASIC program that is loaded replaces the one currently in memory.

Another type of program is one written in machine language. In most cases, a machine language cannot be relocated. It must be loaded at the same address from which it was saved. A secondary address of 1 tells the computer to ignore the current address of BASIC memory and to use the address which was used for saving the file. Keep in mind that nonrelocating LOADs are most often used when loading machine language programs.

Here are the secondary addresses used with LOAD and their effects:

Secondary Address	**Effect**
0	Relocating LOAD
1	Nonrelocating LOAD

Let's take a look at what happens with LOAD in program mode. When executed, the command loads the requested program and performs a RUN. This is handy since it makes it possible to run a series of programs, one after another. This is called *chaining.*

There's one important difference between the normal RUN command and what's automatically performed after a LOAD, however. The automatic RUN after a LOAD does not include a CLR. None of the variables, arrays, and functions are erased. This doesn't mean that their values are preserved. All function definitions are overwritten when the new program is loaded. Calling a defined function when no corresponding DEF has been executed will not cause an UNDEF'D FUNCTION error, but it will probably force some sort of error because the definition is no longer intact. Numeric variables and arrays will be preserved if the new program is shorter than the old one; however, string variables

may or may not be preserved, depending on how they were assigned.

It would be nice if it were possible to pass variables between chained BASIC programs, but there are so many restrictions that in most cases it isn't worth the trouble. Just have the chained programs start with a CLR statement and there shouldn't be any problems.

The problem of passing variables doesn't apply when LOAD is used with a machine language program. In these situations, the original program is not replaced, so the variables, arrays, and function definitions should not be disturbed. This is often used when a BASIC program loads and executes a machine language routine or program. In the lines below, the SYS statement, introduced in a later section, starts the execution of the machine language.

```
10 IF A=0 THEN A=1:LOAD "EDIT.C64",8,1:REM LOADS M
   ACHINE CODE
20 SYS 49152:REM EXECUTES MACHINE CODE
```

Since you started the program with the normal RUN command, the variable A is 0, and the condition is true. Variable A is set to 1, and the machine language program is loaded from disk. After the LOAD is complete, BASIC automatically does another RUN, this time *without* clearing the variables. Variable A keeps the value of 1, so this second time the section after the THEN is not executed. Execution falls through to line 20, the SYS statement, which starts the machine language program.

SAVE. Last but not least, the SAVE command, when used in a program, saves the current BASIC program to tape or disk. This could be useful if you want a program to make a copy of itself when it runs.

Summary

- RUN can be used as a substitute for the END statement to make a program rerun when it ends.
- NEW, when used in a program, ends and erases the program.
- If LIST is included in a program, execution stops after the LIST is executed, even if there are lines yet unexecuted.
- CONT causes BASIC processing to enter an infinite loop

which can be stopped by pressing the RUN/STOP key.

- The full syntax for LOAD is: LOAD *filename, device number, secondary address.*
- The secondary address should be 0 for a relocating LOAD, which indicates that the program will start loading at the current address of BASIC memory, replacing the program already in memory. Zero is the default secondary address.
- Relocating LOADs are usually used with BASIC programs.
- The secondary address should be 1 for a nonrelocating LOAD. The program will be loaded at the same address from which it was saved. This usually leaves the current BASIC program intact.
- Nonrelocating LOADs are most often used for machine language programs.
- When LOAD executes in a program, the LOAD is performed and BASIC goes to the first line of the current BASIC program. This is like performing a RUN without a CLR.
- All variables, arrays, and function definitions are preserved during a nonrelocating LOAD. This fact can be used to make a BASIC program load and begin executing a machine language program.
- After a relocating LOAD, BASIC starts to execute the program which replaced the one in memory. String variables, arrays, and function definitions will still exist but their values will be incorrect because they were overwritten by the new program. Numeric variables will be preserved only if the new program is shorter than the old one. In most cases, it's best to simply have the new program start with a CLR.
- The process of having one program load and run another is called *chaining.* Commodore 64 BASIC allows any number of programs to be chained.
- A SAVE used in a program makes the program copy itself to tape or disk. Execution will continue with the first statement after the command SAVE.

The WAIT Statement

The WAIT statement is an advanced topic because it uses binary values and, compared with other statements, is seldom used. But WAIT does have a few applications and can help you tighten a program by saving lines.

WAIT makes BASIC stop all processing until a designated bit has a specified value.

As an example, let's use location 56320, the second joystick port. Bits 0–3 of this location report the direction of the joystick; bit 4 reflects the status of the joystick button. To monitor a specific bit, you can use the AND operator and the number which corresponds with the bit. The chart below gives the values associated with each bit.

Bit	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

The following line shows the relationship between bit 4 and the joystick button. *Note: Plug a joystick into port 2.*

```
10 PRINT PEEK(56320)AND16:GOTO 10
```

Normally, *16* is printed, which means that the bit is *set.* When the button is pressed, the bit is *cleared,* meaning a *0* is printed. The WAIT statement uses this process to decide when to halt execution, as well as when to continue. Make sure the above program line is still in memory, and enter this line in immediate mode:

WAIT 56320,16

It seems as if nothing happened. That's because the statement told BASIC to wait until bit 4 (value of 16) in location 56320 was set. Since you weren't pressing the joystick button when you entered the statement, the bit was already set, and no WAIT was executed.

Enter WAIT 56320,16 again, but this time press down on the button before hitting RETURN. The cursor will disappear, and nothing will be printed. In fact, BASIC will be concentrating so hard on waiting for bit 4 to become set that it will not notice if you press RUN/STOP. Now release the button. The READY. prompt will appear and BASIC will be functioning as normal.

The *16* in the last example is a *bit mask* which tells WAIT to freeze execution until bit 4 goes from clear to set.

Sometimes, however, you might want to reverse the

condition and have the WAIT statement delay BASIC until a selected bit goes from set to clear. Consider the following program, which waits until the button is pressed (bit 4 is cleared) and then prints a message.

```
10 IF PEEK(56320)AND16 GOTO 10:REM LOOP WHILE NOT
   {SPACE}PRESSED
20 PRINT "FIRE!"
```

To make WAIT halt execution until a bit is cleared, you need to add another bit mask to the statement. In the line below, the first *16* indicates that bit 4 of location 56320 is to be monitored; the second *16* says that the condition to be fulfilled is that the bit must go from set to clear, instead of the normal change from clear to set.

```
10 WAIT 56320,16,16:PRINT "FIRE!"
```

This does the same thing as the two-line program above, but in only one line.

Now let's extend the example so that it waits repeatedly for the joystick button to be pressed.

```
10 WAIT 56320,16,16:PRINT "FIRE!";:REM NOTE THE SE
   MICOLON
20 GOTO 10
```

When you run this program, you'll quickly notice a problem. Each press of the button is detected more than once. This problem, called *bounce,* can also happen with the keyboard. Sometimes you hit one key, but it's actually read more than once. To eliminate the bounce, make the program wait for the button to be released *before* relooping. Just substitute this line 20 for that above.

```
20 WAIT 56320,16:GOTO 10
```

In fact, the whole program could be written in one line:

```
10 WAIT 56320,16,16:PRINT "FIRE!";:WAIT 56320,16:G
   OTO 10
```

If the WAIT statement was not available, this program would have had to be written in three or four lines, depending on whether the ON-GOTO technique was used. Notice that in both the programs below the RUN/STOP key operates, unlike when you use the WAIT statement.

```
10 IF PEEK(56320)AND16 GOTO 10
20 PRINT "FIRE!";
30 IF (PEEK(56320)AND16)=0 GOTO 30
40 GOTO 10
```

or

```
10 IF PEEK(56320)AND16 GOTO 10
20 PRINT "FIRE!";
30 ON (PEEK(56320)AND16)/16+1 GOTO 30,10
```

WAIT can even monitor several bits at a time. Location 198 counts the number of keypresses (0–10) in the keyboard buffer. It takes four bits to represent a number in that range, so bits 0–3 have to be watched. The value 0 indicates that the buffer is empty, so when the first four bits in location 198 are all clear, no key has been pressed. To make a program wait until a key is pressed, you can use the standard method:

```
10 GET K$:IF K$="" GOTO 10
20 PRINT "GOT A KEY"
```

Or you can use WAIT to stop program execution until location 198 contains a nonzero value (when any bit from 0 to 3 is set). With WAIT, the program would look like:

```
10 WAIT 198,15:GET K$:PRINT "GOT A KEY"
```

The value 15 was calculated from 1+2+4+8, the values associated with bits 0–3.

Another common application for WAIT is to wait until the PLAY button on the Datassette is pressed. Bit 4 of location 1 goes from set to clear when a button on the Datassette is pressed.

```
10 PRINT PEEK(1)AND16:GOTO 10
```

Type in and run this program. You'll see *16* appear on the screen when the PLAY button is not pressed, *0* when it is.

When a program opens a tape file, the computer prints the message PRESS PLAY ON TAPE. However, the message does not appear if the PLAY button is already pressed. The computer just proceeds with the open processing. If you don't want the prompt message, perhaps because it might disrupt your screen display, put a WAIT statement before the OPEN. It would look like this:

```
10 WAIT 1,16,16:OPEN 1,1,0,"TAPE FILE"
```

The file TAPE FILE is still opened, but the prompt fails to appear.

Summary

- The WAIT statement makes BASIC halt execution until selected bits in a designated location are set or cleared.
- The syntax is WAIT, *location number, comma, mask,* and *optional comma and mask.*
- The bit set in the first mask tells which bit is to be monitored. For instance, if BASIC is to monitor bit 4, then 16 (that bit's value) would be used as the mask number.
- The second bit mask tells whether BASIC is to wait for the selected bit to go from clear to set, or from set to clear. If there is no second bit mask, BASIC will wait until the bit goes from clear to set. To reverse the condition, the value associated with the bit should be used for the second bit mask.
- More than one bit can be monitored in the same location. To make BASIC monitor both bits 4 and 5, the associated values of 16 and 32 would be added and used as the first mask number. BASIC would then wait until *either* bit 4 or bit 5 went from clear to set.
- The second bit mask can also be used with several bits. The statement WAIT 1,16+32,16 makes BASIC wait until either bit 4 of location 1 goes from set to clear, or bit 5 goes from clear to set.
- While BASIC is waiting, it does not check the RUN/STOP key, so RUN/STOP–RESTORE has to be used to stop execution.
- Here are several example applications of WAIT: Halt execution until the joystick button is pressed or released; force a program to wait until the PLAY button on the Datassette is pressed so that no PRESS PLAY ON TAPE message appears; and freeze a program until a key is pressed.

Machine Language and the SYS Statement

The SYS statement is used to execute machine language routines and programs. In order to understand what this means, let's take a moment to explain machine language.

Inside your Commodore 64, there are several computer chips. One of them, the VIC chip, handles the video display.

Another, the SID, is in charge of audio output. These are support chips, and give the Commodore 64 its excellent sound and graphics capabilities. But they don't give the 64 any computing ability. That's what the CPU (Central Processing Unit), the heart of the computer, does. It's the CPU which really runs the show, doing everything from reading the keyboard to processing all calculations and decisions and telling the VIC chip what to display on the screen. Even if the VIC and SID chips were not working, the computer could still run programs. You just wouldn't see or hear any output. The computer could not function, however, if there were no CPU.

All computers, from micros to mainframes, must have a CPU. Some even have more than one. In microcomputers, the CPU is called a *microprocessor.* The type of microprocessor used in the Commodore 64 is the 6510, which is practically identical to the 6502 found in many other home computers. Some other popular microprocessors are the Z80 and the 8088.

Machine language is the native language of the microprocessor. It's quite different from BASIC. Just as a BASIC program contains statements, a machine language program consists of *instructions* which are executed by the microprocessor. The difference is that each instruction can do only a very small task, so many are necessary to get anything done. It takes several instructions just to add two numbers together. However, machine language instructions can be executed very quickly. It's quite common for a machine language program to run a hundred times faster than a comparable BASIC program. A BASIC program is not machine language and cannot be directly executed by the microprocessor. Rather, the BASIC language itself is one huge machine language program. It's a collection of machine language routines, one for each operation, function, statement, and command in BASIC. For instance, there's a routine to add two numbers, a routine which calculates the absolute value of a number, and a routine to print something on the screen. There are also routines to find the value associated with a variable, search for a line number, check syntax, and so on, and so on.

When BASIC is told to run a program, it looks at the first statement, executes the necessary machine language routines to process the statement, and then moves on to the next statement. Thus, a BASIC program is really a kind of data file. It cannot stand on its own, but instead is treated as data for a

machine language program called BASIC. The end result may appear to the user as though BASIC statements have computing power, but the real processing is being done in machine language.

The process of reading data from a program file and selectively executing some routines is called *interpreting*. The interpreting process is what makes BASIC programs run slower than equivalent machine language programs. It takes time to read statements, look up values for variables, search for program lines, and check for correct syntax. An alternative is to convert a BASIC program to machine language with a process known as *compiling*. After a program has been written, it's compiled before it's executed. Compiled programs are actual machine language programs that can stand on their own, programs which don't need BASIC. This can significantly increase the speed of the program. However, program development is less interactive, because every time the program is changed in even the smallest way, it has to be recompiled.

The BASIC language, whether its programs are interpreted or compiled, is called a *high-level* language because each statement is rather general and can do a lot. If you think about it, the PRINT statement is extremely versatile. It can print numbers and characters, call functions and operations, and control cursor positioning. Machine language, on the other hand, is a *low-level* language, because each instruction is limited and specific. Most people without computer experience learn BASIC first. However, if you want to tap the full potential of your computer and want to know what computing is really about, machine language is the way to go.

The SYS statement instructs BASIC to stop executing and to transfer control to a machine language routine which starts at a specified address. Thus, SYS is to a machine language program what RUN is to a BASIC program. If you're using a machine language routine published in a book or magazine, the address will be given in the article or documentation. Care must be taken that the address is correct, because if the microprocessor is told to execute machine language instructions at memory locations where there are no instructions, it will "hang" and the computer will crash. This does no permanent damage to the computer, but it does mean that you'll have to turn the computer off and back on again to regain control. Anything in memory will be lost.

To demonstrate the speed of machine language and to show how the SYS statement is used, the doodling program from *All About the Commodore 64, Volume One* has been rewritten in machine language. Here's the original BASIC program. Plug a joystick into port 2. Push the joystick to draw in all eight directions, and press the button to change the color.

Program 4-1. BASIC Doodle

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
10 PRINT CHR$(147):POKE 53280,0:POKE 53281,0:S=500
   :C=1                                          :rem 181
20 DIM JS(15):FOR K=5 TO 14:READ JS(K):NEXT
                                                 :rem 126
30 POKE 55296+S,C:POKE 1024+S,81                 :rem 205
40 S=S+JS(PEEK(56320) AND 15)                     :rem 15
50 IF S<0 THEN S=S+1000                          :rem 177
60 IF S>999 THEN S=S-1000                         :rem 49
70 IF (PEEK(56320) AND 16)=0 THEN C=C+1:IF C>15 TH
   EN C=1                                         :rem 66
80 GOTO 30                                         :rem 4
90 DATA 41,-39,1,0,39,-41,-1,0,40,-40           :rem 239
```

Now, let's try the machine language version. The instructions have been stored as numbers in DATA statements, which is the most convenient form for short programs or routines. The instructions are read and POKEd into free memory, and then SYS is used to begin executing the code. Just lightly tap the joystick left or right at first, so you can see that the drawing is done a pixel at a time. You'll notice that the machine language version is so fast that even if you just lightly hit the joystick, most of the screen fills up.

Program 4-2. Machine Language Doodle

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
300 FOR K=4096 TO 4290:READ P:POKE K,P:NEXT
                                                 :rem 150
310 SYS 4096                                     :rem 102
320 END                                          :rem 108
800 DATA 169,147,32,210,255,169,0,141,32,208,141,3
    3,208,169,244,141,193,16,169                   :rem 9
801 DATA 1,141,194,16,169,1,141,192,16,169,0,24,10
    9,193,16,133,251,169,216,109                   :rem 3
802 DATA 194,16,133,252,173,192,16,160,0,145,251,1
    69,0,24,109,193,16,133,251                   :rem 155
```

```
803 DATA 169,4,109,194,16,133,252,169,81,145,251,1
    73,0,220,41,15,170,189,165              :rem 171
804 DATA 16,24,109,193,16,141,193,16,189,176,16,10
    9,194,16,141,194,16,173,194             :rem 237
805 DATA 16,16,17,173,193,16,24,105,232,141,193,16
    ,173,194,16,105,3,141,194,16             :rem 1
806 DATA 169,231,205,193,16,169,3,237,194,16,176,1
    7,173,193,16,56,233,232,141             :rem 234
807 DATA 193,16,173,194,16,233,3,141,194,16,173,0,
    220,41,16,208,15,238,192,16             :rem 210
808 DATA 169,15,205,192,16,176,5,169,1,141,192,16,
    76,28,16,41,217,1,0,39,215              :rem 169
809 DATA 255,0,40,216,0,0,255,0,0,0,255,255,0,0,25
    5,0,0,0,0                                :rem 28
```

BASIC always checks for the RUN/STOP key before it processes each statement. Since BASIC is not in control, RUN/STOP won't work, and you'll have to use the RUN/STOP–RESTORE combination to halt the program.

It's possible to pass information between BASIC and machine language programs. In many of the programs later in this book, you'll see a line like the following:

```
120 SA=780:SX=781:SY=782:SP=783
```

Locations 780–783 can be POKEd before a SYS call. This area of memory is called the *Register Storage Area* and is used to store the values held by the Accumulator, the X register, the Y register, and the status register. The Register Storage Area is used by the BASIC SYS command to load each of the registers from the corresponding storage address. The values POKEd can then be used by the machine language routine. That's what this next line does.

```
130 POKE SA,1:POKE SX,8:POKE SY,0:SYS 65466
```

When the machine language routine ends and control is returned to the BASIC interpreter, any new values held by the registers are stored in the appropriate memory locations. Program execution will continue with the first statement after SYS. This may be a PEEK to one of the locations to get information inserted by the machine language program or routine.

```
140 SYS 65493:IF PEEK(SP)AND1 GOTO 700
```

These variable names (SA, SX, SY, and SP) are used consistently in the later programs.

There's one other feature of BASIC related to machine

language. The USR function is an alternative method of executing machine language, and allows one floating-point value to be passed back and forth. However, this technique is less convenient than SYS, so it's not used very often. Calling the USR function without some special setup results in an ILLEGAL QUANTITY error.

The USR function is not related in any way to the USR disk file type.

Summary

- Of all the parts of a computer, the one that provides the real "brains" is the CPU, or central processing unit. In microcomputers, the CPU is called a *microprocessor.* The Commodore 64's microprocessor is the 6510, which for all practical purposes is identical to the 6502.
- Machine language is the language of the microprocessor. Machine language consists of *instructions* (rather than statements) which are directly executed by the microprocessor.
- Each machine language instruction is very simple and limited in what it can accomplish. But though it takes a number of them to get anything done, each instruction can be executed very quickly.
- The BASIC language on the Commodore 64 is one large machine language program.
- A BASIC program is not machine language and cannot be directly executed by the microprocessor. Rather, it's treated as data read by the BASIC interpreter program.
- The process of BASIC reading statements and executing the appropriate machine language routines is called *interpreting.* Interpreted BASIC programs are relatively slow because things like syntax checking are done as the program is running.
- A BASIC program can be *compiled* into machine language, which can then be directly executed by the microprocessor. The advantage of compiling is that things like syntax checking are done as part of the compiling step, so the program runs much faster. The disadvantage is that a program has to be recompiled every time it's changed.
- BASIC is a *high-level* language because each statement can do a lot. Machine language is a *low-level* language.
- The SYS statement is used to start the execution of machine

language at a specified address. If the address is wrong, the computer may crash.

- Information from a BASIC program can be made available to the machine language code by POKEing the information into locations 780 through 783 before the SYS is executed.
- If the machine language program ends and returns control to BASIC, execution will continue with the first statement after the SYS.
- Locations 780 to 783 can be PEEKed when the machine language program is finished to get values returned.
- The USR function offers a means of executing machine language and passing one floating-point value back and forth, but in most cases it's less convenient to use than SYS.

Part 2
Bitmapped Graphics

The Bitmapped Graphics Utility

Introduction to the Bitmapped Graphics Utility

Bitmapped graphics lets you cover the whole screen with graphics. There are no size limitations as there are with redefined characters and sprites. When redefining a character, you are confined to working in an 8 × 8 grid. Even with sprites, the 24 × 21 grid restricts the size of an object. But with bitmapped graphics, the whole screen is one big grid. You can control any point on the screen. You can draw lines to create things like line graphs and three-dimensional pictures. Or you can fill in areas to create pie charts or background scenes.

The Commodore 64's bitmapped graphics capabilities are very good, but difficult to use from BASIC. As a result, bitmapped graphics is often the most overlooked and underused feature of the computer.

Since the goal of this book is to help you get the most out of your computer, the first major utility is designed to open up the world of bitmapped graphics. The utility consists of a collection of machine language routines which make bitmapped graphics easier to use. The routines can plot points, draw lines, fill in areas, and draw whole shapes. Because the routines are written in machine language, they're fast. That leaves only one problem—how will the routines be called by a BASIC program?

One method would be to use the SYS statement. A slightly more elaborate method, and the one used for this utility, is to add new statements to BASIC. This method is a good choice because it offers the speed of machine language *and* the convenience of BASIC.

The New Statements

The new statements work just like standard BASIC statements. They can be used in program lines, they can be listed, and they can cause errors if their syntax is wrong or their values are out of range. A program containing these new statements

can be saved to tape or disk just like any other program.

Besides these statements, we've added a new function and a new command. Here's a list of the additions to BASIC:

Statements

GRAPHICS	DRAW	SETPEN
CLS	MODE	TEXT
FILL	LFILL	RFILL
SHAPE	PEN	

Function

LOOK

Command

KILL

To install the BASIC extensions, two files must be created. One file contains the machine language routines which perform all the graphics work. The other is a BASIC program that loads the machine language file and merges it with BASIC, actually adding the new statements.

First, enter and save either Program 5-1 (for disk) or Program 5-2 (for tape) to disk or tape. Make sure you use the filename BMGLOADER.

Next, type in the machine language file (Program 5-3), using the "Machine Language Editor: MLX" program found in Appendix C, and save it using the filename BMG.OBJ. If you're using tape, be sure to save the file immediately after the "BMGLOADER" program.

(Please read the article in Appendix C before typing in Program 5-3. You'll need to use MLX to enter other files in this book, so be sure to save a copy. If you have a copy of MLX from another COMPUTE! publication and you're using tape, change line 763 of your copy so that it matches line 763 as listed in this book. Tape users who have a copy of MLX which does not have a line 763 *must* use the version of MLX from this book.)

After loading and running MLX, you'll be asked for two prompts. Respond to these with:

Starting address: 51200
Ending address: 53245

Save this file as BMG.OBJ

Once you have these two files on disk or tape, you're ready to start using bitmapped graphics. Load and run the BMGLOADER program, which in turn loads and executes the

machine language file. After the BASIC extensions are installed, the READY. prompt will appear.

For a short demonstration of what the new graphics statements can do, enter and run Program 5-4. It draws a rectangular kaleidoscope. To stop the drawing, press the RUN/STOP key. To reset the screen to text mode, hit RESTORE while holding down RUN/STOP.

One very important word of caution: The bitmapped graphics extensions require memory from 50176 to 53247 (hexadecimal $C400 to $CFFF). The extensions cannot be used in conjunction with any other utility which uses these same memory locations, including the popular DOS wedge. Remove all such utilities before loading the bitmapped graphics extensions. The easiest way to do this is to turn the computer off and back on.

The bitmapped graphics extensions stay in effect until the computer is turned off, so there's no need to run the BMGLOADER program more than once as long as the computer is on. The next time you turn on the computer, though, you will have to run the BMGLOADER program before using any graphics statements. *Programs containing graphics statements will not list properly and will not work if the BMGLOADER program has not been run earlier.*

If you're finished using the bitmapped graphics BASIC extensions and want to use another utility that requires the same memory locations, use the KILL command. This removes the extensions and frees up the memory. The graphics statements can later be reinstalled by running BMGLOADER again.

All the program examples in this chapter and in Chapter 6 require that the BASIC extensions be active when the programs are typed in, run, or loaded. This is extremely important. You must run BMGLOADER each time you turn on your computer and after the KILL command is used if you wish to use any program containing the BASIC extensions.

Program 5-1. BMGLOADER Disk Version

```
10 IF A THEN SYS 52506:NEW
20 A=1:LOAD "BMG.OBJ",8,1
```

Program 5-2. BMGLOADER Tape Version

```
10 IF A THEN SYS 52506:NEW
20 A=1:LOAD "BMG.OBJ",1,1
```

Program 5-3. BMG.OBJ

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

```
51200 :072,173,252,204,010,010,209
51206 :010,173,253,204,144,002,024
51212 :009,002,010,010,072,173,032
51218 :002,221,009,003,141,002,140
51224 :221,173,000,221,041,252,164
51230 :013,250,204,141,000,221,091
51236 :104,141,024,208,173,017,191
51242 :208,009,032,141,017,208,145
51248 :162,001,142,255,204,189,233
51254 :193,204,157,250,255,189,022
51260 :195,204,157,254,255,202,047
51266 :016,241,162,002,189,197,105
51272 :204,157,009,205,202,016,097
51278 :247,104,141,254,204,170,174
51284 :074,168,189,200,204,141,036
51290 :004,205,185,208,204,141,013
51296 :007,205,173,022,208,041,240
51302 :239,162,001,160,255,144,039
51308 :005,009,016,162,003,200,247
51314 :141,022,208,142,003,205,067
51320 :140,006,205,162,003,138,006
51326 :157,021,205,202,016,249,208
51332 :096,072,138,072,152,072,222
51338 :165,001,009,002,133,001,193
51344 :169,200,072,169,157,072,215
51350 :008,072,072,072,108,020,246
51356 :003,165,001,041,253,133,240
51362 :001,104,168,104,170,104,045
51368 :064,169,000,072,173,252,130
51374 :204,133,076,169,000,133,121
51380 :075,162,002,157,000,205,013
51386 :157,015,205,202,016,247,004
51392 :168,162,031,032,225,200,242
51398 :160,064,032,219,200,104,209
51404 :141,033,208,174,253,204,193
51410 :134,076,162,003,032,225,074
51416 :200,160,232,136,145,075,140
51422 :208,251,096,145,075,200,173
51428 :208,251,230,076,202,208,123
51434 :246,096,173,017,208,041,247
51440 :032,240,237,173,002,221,121
51446 :009,003,141,002,221,173,027
51452 :000,221,009,003,141,000,114
51458 :221,076,129,255,169,128,212
51464 :142,000,205,170,169,000,182
51470 :042,141,002,205,140,001,033
51476 :205,138,048,035,045,003,238
51482 :205,141,255,204,240,022,069
51488 :170,189,008,205,041,015,148
51494 :202,208,004,010,010,010,226
51500 :010,141,008,205,173,006,075
51506 :205,029,212,204,141,005,078
51512 :205,169,000,133,082,173,050
```

```
51518 :001,205,133,079,173,254,139
51524 :204,074,208,019,173,000,234
51530 :205,133,080,173,002,205,104
51536 :133,081,144,023,006,080,035
51542 :038,081,076,105,201,170,245
51548 :173,000,205,006,079,010,053
51554 :202,208,250,133,080,038,241
51560 :081,165,081,041,001,133,094
51566 :081,240,006,165,080,201,115
51572 :064,176,006,165,079,201,039
51578 :200,144,001,096,170,041,006
51584 :248,010,010,010,133,075,102
51590 :138,074,074,074,133,076,191
51596 :074,074,024,101,076,133,110
51602 :076,138,069,080,041,007,045
51608 :069,080,101,075,133,075,173
51614 :165,081,101,076,109,252,174
51620 :204,133,076,165,001,133,108
51626 :085,045,251,204,133,001,121
51632 :165,080,041,007,109,004,070
51638 :205,168,185,215,204,164,043
51644 :082,240,086,016,021,133,254
51650 :077,160,000,177,075,070,241
51656 :077,144,003,074,016,249,251
51662 :166,085,134,001,045,003,128
51668 :205,096,073,255,133,077,027
51674 :045,012,205,133,078,160,083
51680 :000,177,075,205,012,205,130
51686 :208,034,165,100,240,017,226
51692 :016,001,136,024,109,000,010
51698 :205,141,000,205,152,109,030
```

```
51704 :002,205,141,002,205,173,208
51710 :005,205,172,007,205,145,225
51716 :075,136,016,251,048,034,052
51722 :037,077,197,078,056,208,151
51728 :097,165,077,073,255,133,048
51734 :077,073,255,045,005,205,170
51740 :133,078,172,007,205,177,032
51746 :075,037,077,005,078,145,195
51752 :075,136,016,245,138,174,056
51758 :255,204,024,240,063,041,105
51764 :248,133,075,010,038,076,120
51770 :010,038,076,024,101,075,126
51776 :133,075,165,076,041,003,045
51782 :105,000,133,076,165,081,118
51788 :074,165,080,106,074,074,137
51794 :168,165,076,224,003,208,158
51800 :010,009,216,133,076,173,193
51806 :008,205,024,144,013,013,245
51812 :253,204,133,076,177,075,250
51818 :061,229,204,013,008,205,058
51824 :145,075,165,085,133,001,204
51830 :096,133,086,038,077,152,188
51836 :160,001,056,237,001,205,016
51842 :176,005,073,255,105,001,233
51848 :136,133,076,208,001,136,058
51854 :132,088,138,056,237,000,025
51860 :205,133,075,165,077,041,076
51866 :001,237,002,205,133,077,041
51872 :160,000,132,078,132,089,239
51878 :010,144,015,152,229,075,023
51884 :133,075,152,229,077,133,203
```

```
51890 :077,198,089,136,048,005,219
51896 :005,075,240,001,200,132,069
51902 :087,162,001,160,000,132,220
51908 :091,165,075,197,076,165,197
51914 :077,233,000,144,016,202,106
51920 :200,165,077,074,165,075,196
51926 :176,003,208,007,096,240,176
51932 :004,132,091,181,075,133,068
51938 :090,185,075,000,010,153,227
51944 :075,000,072,185,077,000,129
51950 :042,153,077,000,104,056,158
51956 :245,075,133,092,185,077,027
51962 :000,245,077,133,093,022,052
51968 :075,054,077,185,075,000,210
51974 :133,094,056,245,075,133,230
51980 :095,185,077,000,133,096,086
51986 :245,077,133,097,134,098,034
51992 :132,099,162,000,165,093,163
51998 :048,024,232,164,099,185,014
52004 :000,205,024,121,087,000,217
52010 :153,000,205,152,208,008,000
52016 :173,002,205,101,089,141,247
52022 :002,205,165,092,024,117,147
52028 :094,133,092,165,093,117,242
52034 :096,133,093,166,098,189,073
52040 :000,205,024,117,087,157,150
52046 :000,205,138,208,008,173,042
52052 :002,205,101,089,141,002,112
52058 :205,032,057,201,176,012,005
52064 :165,086,208,009,198,090,084
52070 :208,178,198,091,240,174,167
```

```
52076 :096,133,082,174,000,205,030
52082 :134,101,174,002,205,134,096
52088 :102,172,254,204,190,232,250
52094 :204,134,100,041,001,240,078
52100 :031,238,000,205,208,003,049
52106 :238,002,205,032,061,201,109
52112 :144,243,165,086,041,002,057
52118 :240,033,165,101,141,000,062
52124 :205,165,102,141,002,205,208
52130 :169,000,056,229,100,133,081
52136 :100,173,000,205,208,003,089
52142 :206,002,205,206,000,205,230
52148 :032,061,201,144,240,165,255
52154 :101,141,000,205,165,102,132
52160 :141,002,205,024,144,158,098
52166 :133,107,169,000,141,025,005
52172 :205,141,012,205,141,018,158
52178 :205,141,019,205,141,020,173
52184 :205,173,013,205,133,105,026
52190 :173,014,205,133,106,208,037
52196 :110,074,176,003,073,255,151
52202 :136,109,015,205,141,015,087
52208 :205,152,109,016,205,141,044
52214 :016,205,024,144,088,074,029
52220 :176,002,073,255,109,017,116
52226 :205,141,017,205,024,144,226
52232 :074,072,101,107,041,007,154
52238 :170,189,242,204,016,001,068
52244 :136,024,109,015,205,141,138
52250 :015,205,152,109,016,205,216
52256 :141,016,205,189,240,204,003
```

```
52262 :024,109,017,205,141,017,039
52268 :205,168,104,041,024,074,148
52274 :074,074,170,240,013,173,026
52280 :025,205,016,008,041,015,110
52286 :141,025,205,157,008,205,035
52292 :173,016,205,074,189,021,234
52298 :205,174,015,205,032,008,201
52304 :201,176,037,160,000,177,063
52310 :105,230,105,208,002,230,198
52316 :106,074,144,133,074,144,255
52322 :152,074,144,163,074,144,081
52328 :016,074,144,020,074,144,064
52334 :032,162,002,074,144,045,057
52340 :074,144,066,024,096,009,017
52346 :128,141,025,205,144,211,208
52352 :074,170,240,006,173,006,029
52358 :205,029,211,204,141,012,168
52364 :205,144,196,170,173,016,020
52370 :205,074,138,174,015,205,189
52376 :172,017,205,032,119,202,131
52382 :144,179,096,208,011,189,217
52388 :015,205,157,018,205,202,198
52394 :016,247,144,165,189,018,181
52400 :205,157,015,205,202,016,208
52406 :247,144,154,157,015,205,080
52412 :202,016,250,144,146,168,090
52418 :200,133,200,013,010,006,244
52424 :000,008,008,008,009,009,242
52430 :011,011,000,001,003,007,239
52436 :085,170,255,127,191,223,239
52442 :239,247,251,253,254,063,245
```



```
52448 :015,207,000,243,240,252,157
52454 :015,240,007,003,003,003,245
52460 :001,001,000,000,255,255,236
52466 :000,001,001,001,000,255,244
52472 :255,255,000,253,224,196,151
52478 :000,000,000,000,000,000,254
52484 :000,000,000,000,000,000,004
52490 :000,000,000,000,000,000,010
52496 :000,000,000,000,000,000,016
52502 :000,000,000,000,173,004,199
52508 :003,201,233,208,007,173,085
52514 :005,003,201,206,240,027,204
52520 :162,011,189,000,003,157,050
52526 :167,002,189,068,205,157,066
52532 :000,003,202,016,241,169,171
52538 :235,141,024,003,169,207,069
52544 :141,025,003,096,221,206,244
52550 :131,164,233,206,119,207,106
52556 :172,207,206,207,071,082,253
52562 :065,080,072,073,067,211,138
52568 :068,082,065,215,082,070,158
52574 :073,076,204,076,070,073,154
52580 :076,204,070,073,076,204,035
52586 :084,069,088,212,067,076,190
52592 :211,077,079,068,197,083,059
52598 :069,084,080,069,206,083,197
52604 :072,065,080,197,080,069,175
52610 :206,075,073,076,204,076,072
52616 :079,079,203,000,199,205,133
52622 :000,206,003,206,006,206,001
52628 :009,206,235,200,214,205,193
```

```
52634 :208,205,158,206,093,206,206
52640 :178,206,198,206,032,115,071
52646 :000,032,250,174,032,235,121
52652 :183,142,232,206,032,247,190
52658 :174,032,227,205,032,006,086
52664 :201,176,055,168,076,162,254
52670 :179,032,158,183,138,201,057
52676 :008,176,043,096,032,169,208
52682 :200,032,191,205,076,000,138
52688 :200,032,191,205,076,080,224
52694 :200,240,005,032,158,183,008
52700 :138,044,169,000,076,171,050
52706 :200,165,021,074,208,010,136
52712 :172,232,206,138,166,020,142
52718 :096,076,008,175,076,072,229
52724 :178,032,121,000,240,244,035
52730 :201,164,240,240,076,241,132
52736 :183,160,000,044,160,001,036
52742 :044,160,002,044,160,003,163
52748 :140,231,206,201,164,240,170
52754 :020,032,235,183,142,232,094
52760 :206,174,255,204,032,245,116
52766 :205,032,227,205,032,008,227
52772 :201,176,203,032,121,000,001
52778 :240,194,201,164,208,191,216
52784 :032,115,000,032,235,183,133
52790 :142,232,206,162,000,236,008
52796 :231,206,208,010,032,227,206
52802 :205,032,119,202,176,170,202
52808 :144,221,032,245,205,138,033
52814 :240,006,173,006,205,029,225
52820 :211,204,141,012,205,174,007
52826 :231,206,208,226,032,138,107
52832 :173,032,247,183,140,013,116
52838 :205,141,014,205,032,121,052
52844 :000,240,040,032,253,174,079
52850 :032,138,173,032,247,183,151
52856 :166,020,032,121,000,240,187
52862 :022,142,015,205,165,021,184
52868 :141,016,205,032,241,183,182
52874 :142,017,205,032,121,000,143
52880 :240,005,032,241,183,138,215
52886 :044,169,000,032,198,203,028
52892 :176,168,096,032,158,183,201
52898 :138,041,003,133,020,032,017
52904 :241,183,138,166,020,240,132
52910 :003,157,008,205,096,032,163
52916 :158,183,138,041,003,133,068
52922 :020,032,241,183,138,041,073
52928 :003,166,020,157,021,205,252
52934 :096,162,011,189,167,002,057
52940 :157,000,003,202,016,247,061
52946 :169,071,141,024,003,169,019
52952 :254,141,025,003,096,138,105
52958 :072,032,236,200,104,170,012
52964 :108,167,002,000,000,173,166
52970 :171,002,133,085,173,172,202
52976 :002,133,086,032,084,000,065
52982 :162,000,160,004,132,015,207
52988 :189,000,002,133,008,201,017
52994 :034,240,079,036,015,112,006
53000 :038,201,065,144,034,201,179
```

```
53006 :091,176,030,132,113,160,204
53012 :096,132,011,160,255,134,040
53018 :122,202,200,232,189,000,203
53024 :002,056,249,080,205,240,096
53030 :245,201,128,208,048,005,105
53036 :011,164,113,232,200,153,149
53042 :251,001,185,251,001,240,211
53048 :054,056,233,058,240,004,189
53054 :201,073,208,002,133,015,182
53060 :056,233,085,208,179,133,194
53066 :008,189,000,002,240,223,224
53072 :197,008,240,219,200,153,073
53078 :251,001,232,208,240,166,160
53084 :122,230,011,200,185,079,151
53090 :205,016,250,185,080,205,015
53096 :208,180,189,000,002,016,187
53102 :190,153,253,001,169,255,107
53108 :133,122,096,016,042,201,214
53114 :255,240,038,036,015,048,242
53120 :034,201,224,144,036,056,055
53126 :233,223,170,132,073,160,101
53132 :255,202,240,008,200,185,206
53138 :080,205,016,250,048,245,222
53144 :200,185,080,205,048,008,110
53150 :032,071,171,208,245,076,193
53156 :243,166,076,239,166,108,138
53162 :173,002,032,115,000,201,181
53168 :224,144,021,032,185,207,221
53174 :076,174,167,233,224,010,042
53180 :168,185,141,205,072,185,120
53186 :140,205,072,076,115,000,034
```

```
53192 :032,226,207,108,175,002,182
53198 :169,000,133,013,032,115,156
53204 :000,201,236,208,003,076,168
53210 :164,205,032,226,207,108,136
53216 :177,002,165,122,208,002,132
53222 :198,123,198,122,096,072,015
53228 :138,072,152,072,032,188,122
53234 :246,032,225,255,240,003,219
53240 :076,076,254,076,105,254,065
```

Program 5-4. BMG Demonstration

```
1Ø GRAPHICS 6:POKE 53280,Ø:FOR I=1 TO 12:FOR J=12
   {SPACE}TO Ø STEP -1:K=I+J
2Ø SETPEN 1,RND(Ø)*16:DRAW 6+K,J,1 TO 6+K,24-J:DRA
   W 32-K,J TO 32-K,24-J
3Ø DRAW 7+J,25-K TO 31-J,25-K:DRAW 7+J,K-1 TO 31-J
   ,K-1:NEXT:NEXT:GOTO 1Ø
```

Graphics Modes and Point Plotting

There are two major problems when using bitmapped graphics from BASIC—clearing the bitmap screen takes too long, and plotting of individual points is too slow.

To clear the screen in BASIC, the 8000 memory locations for the bitmap have to be POKEd to zero. Plotting in BASIC is slow because it involves some very complicated calculations.

To solve the first problem, the GRAPHICS statement is available.

GRAPHICS *mode number*

This statement turns on the bitmapping mode of the VIC-II chip, clears the bitmap screen, and sets the background to black. The purpose of the mode number will be explained in a moment.

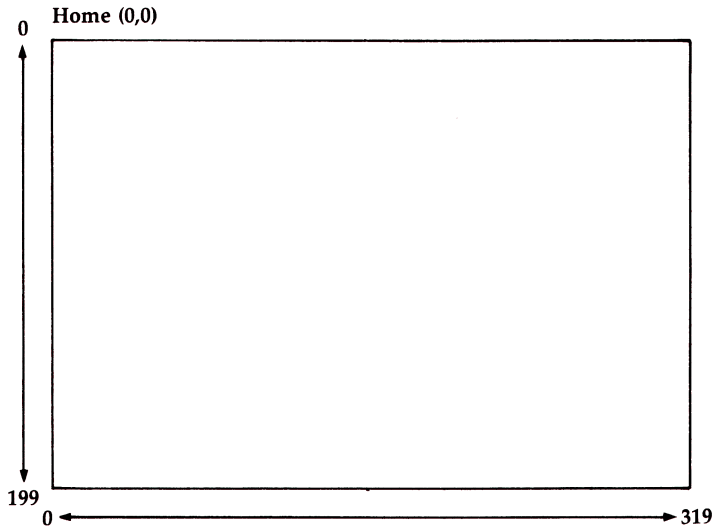
The screen is now all set for high-resolution plotting. That is done by the DRAW statement.

DRAW *X coordinate, Y coordinate*

Plotting is done by using the standard coordinate system of X and Y coordinates. The columns (the X coordinate) are numbered from 0 to 319, starting at the left edge of the screen. The rows (the Y coordinate) are numbered from 0 to 199. Unlike standard mathematics, the numbering of rows starts at the top and works down. The point is plotted where the column and row intersect. Take a look at Figure 5-1 for an illustration of this coordinate system.

The position 0,0 is at the intersection of the first column and the first row, which is the upper-left corner of the screen. This particular position is of special importance and is called the *home* position.

Figure 5-1. Bitmap Coordinate System



To see the GRAPHICS and DRAW statements in action, enter and run the program below.

```
10 GRAPHICS 0
20 DRAW 5,5
30 DRAW 35,5
40 DRAW 35,20
50 DRAW 5,20
60 GOTO 60
```

Four points are plotted, at the four corners of a rectangle. The infinite loop at line 60 is needed to prevent the program from ending. The screen automatically reverts back to text mode when the READY. prompt is printed. Since the screen switches to the text mode whenever READY. appears, the graphics statements cannot be used in the immediate mode. *Graphics statements should be used only in a program.*

As shown by the example, the DRAW statement is a fast and convenient way to plot points. The two major problems of using bitmapped graphics from BASIC have now been solved.

Unfortunately, the previous program reveals a new problem. The points are very small. To make the points more visible, they have to be made larger. That's where the *mode number* comes in. The term *resolution* is used to describe how many points are displayed on a bitmap screen. The more

points that are displayed, the higher the resolution and the smaller each point is.

Mode 0, used in the demonstration above, supports the highest resolution (320 × 200 points). Mode 2 cuts that resolution in half in each direction (only 160 × 100 points), but each point is four times the size as before. Change the mode number in the demonstration from 0 to 2, and run it again to see this effect.

There are a total of four different resolutions available. Here's a list showing the number of columns and rows in each mode.

MODE	RESOLUTION
0	320 × 200
2	160 × 100
4	80 × 50
6	40 × 25

There are four other modes, numbered 1, 3, 5, and 7, which are reserved for special purposes. These will be explained later. Using a mode number outside the range 0–7 will result in an ILLEGAL QUANTITY error.

The following program may not be very exciting, but it does show how plotting with the DRAW statement is much faster than using the old method of POKEing data to create something on the bitmap screen.

```
10 GRAPHICS 4
20 DRAW RND(1)*80,RND(1)*50:GOTO 20
```

These next two examples use trigonometric functions to draw a sine curve and a circle. Change the values for RX and RY in the second program to make the circle into an oval.

```
10 GRAPHICS 0
20 FOR X=0 TO 319
30 DRAW X,100+20*SIN(X/10)
40 NEXT X
```

```
10 GRAPHICS 0
20 RX=55:RY=40
30 FOR A=0 TO 2*↑ STEP ↑/180
40 DRAW 160+RX*COS(A),100+RY*SIN(A)
50 NEXT A
```

If you make the values for RX and RY too large, the coordinates for plotting will be off the screen. The program will stop, the screen will change back to text mode, and an ILLEGAL QUANTITY error message will be printed. This error indicates that the coordinates used in a DRAW statement are out of range for the current graphics mode. For example, trying to plot a point at −1,−1 causes this error.

A useful analogy to help understand bitmapped graphics is to imagine that the drawing is done by a pen. The coordinates position the pen on the screen, and the DRAW statement tells the pen to press down and make a dot. The color of the ink is light green. A pen with a fine point draws thin lines and is used for detailed drawing; this is similar to a high-resolution screen mode. A dull point creates broad strokes which cover more area, something similar to low-resolution modes.

In modes 0, 2, 4, and 6, there are actually two pens available. Pen 1 is used for plotting. It's the default pen and has been used in all the previous examples. Pen 0 is used for *unplotting.* This erases points plotted by pen 1. When a point is erased, the color of the background shows through.

The DRAW statement supports an optional third number that's used to change the current pen number.

DRAW ***X coordinate, Y coordinate, pen number***

When the GRAPHICS statement is used to switch from text to graphics mode, the current pen number is set to 1. The statement DRAW 5,6,1 is thus the same as DRAW 5,6. Both statements plot a point at position 5,6. To erase the point, just use pen 0. DRAW 5,6,0 erases the previously drawn point. Once you've switched to pen 0, all future drawing is done with that pen, at least until you change it back to pen 1.

The following program plots three points, waits for a keypress, erases the points, waits for another keypress, and then plots one more point. Type it in and try it out.

```
100 GRAPHICS 4
110 DRAW 3,4:REM DEFAULTS TO PEN 1
120 DRAW 7,9
130 DRAW 4,20
140 WAIT 198,15:GET K$:REM WAIT FOR KEYPRESS
150 DRAW 3,4,0:REM NOW IN PEN 0
160 DRAW 7,9:REM STILL PEN 0
```

```
170 DRAW 4,20
180 WAIT 198,15:GET K$
190 DRAW 5,11,1:REM BACK TO PEN 1
200 GOTO 200
```

The classic example of plotting and erasing is the bouncing ball program. A point is plotted at one position, then it's erased and plotted at an adjacent position. The direction of travel is reversed whenever the ball hits the edge of the screen. This next example does just that.

```
10 GRAPHICS 4
20 X=INT(RND(1)*78)+1:Y=INT(RND(1)*48)+1:DX=1:DY=1
30 DRAW X,Y,0:X=X+DX:Y=Y+DY:DRAW X,Y,1
40 IF X=0 OR X=79 THEN DX=-DX
50 IF Y=0 OR Y=49 THEN DY=-DY
60 GOTO 30
```

Changing Colors

Thus far, plotting has always been done in light green (color 13). With a normal pen, you can change the color by replacing the ink with a different refill. Likewise, the color associated with a graphics drawing pen can be changed to any of the 16 colors available on the Commodore 64. The format for altering colors is:

SETPEN ***pen number, color number***

When using modes 0, 2, 4, and 6, only pen 1 should be used with SETPEN. The color number can range from 0 to 15. (Refer to the *Commodore 64 User's Guide* for the color values.) Of course, if the pen is changed to color 0 (black), the plotting won't be visible because it then matches the background color.

The next program is a modified version of the bouncing ball program. The points are never erased, so the ball leaves a trail. Pressing any key makes the program restart with a different color for pen 1.

```
10 GRAPHICS 4
20 X=INT(RND(1)*78)+1:Y=INT(RND(1)*48)+1:DX=1:DY=1
30 SETPEN 1,RND(1)*15+1
40 X=X+DX:Y=Y+DY:DRAW X,Y
50 IF X=0 OR X=79 THEN DX=-DX
60 IF Y=0 OR Y=49 THEN DY=-DY
70 GET K$:IF K$="" GOTO 40
80 GOTO 10
```

Because of the GRAPHICS statement, the program clears the screen before using a new color for pen 1. Pen 1 can be used to draw in more than one color on the same screen, provided that points plotted in different colors are separated from one another. For example, let's say that some points are plotted while pen 1 has the color red, then SETPEN is used to change the pen color to blue. Points plotted now will be drawn in blue, and the earlier points will retain their red color. However, if some blue points are plotted too close to the red points, the color of the latter will be affected. Try changing the GOTO 10 in the previous program to GOTO 30.

Every time you press a key, the drawing continues as before, but a different color is used. All appears as it should, except at those places where two different colored lines cross each other. You'll see that sometimes the color of one line interferes with the color of the other. That's the restriction in using modes 0, 2, and 4. Drawing can be done in 16 different colors on the same screen, but colors have to be kept apart from each other. The problem doesn't occur in mode 6.

Pen 0 should not be used with the SETPEN statement. Remember, pen 0 is used for erasing, not for plotting, so the idea of an ink color doesn't apply. The only color associated with pen 0 is the background color, which is set to black (color 0) by the GRAPHICS statement when the screen is cleared. To clear the screen and set the background to a different color, use the CLS statement.

CLS ***color number***

When CLS is executed, the screen clears and the background is set to the specified color. The current graphics mode, however, stays the same. The statement CLS 2, for instance, clears the bitmap and sets the background to red without affecting the current resolution.

```
10 GRAPHICS 4
20 CLS 2
30 FOR K=5 TO 40
40 DRAW K,K
50 NEXT K
60 GOTO 60
```

The number after the keyword CLS is optional. If no number is specified, zero is assumed, so CLS alone sets the background to black.

You can even change the graphics mode from within a program, mixing modes on the screen. The MODE statement format is:

MODE ***mode number***

This changes the graphics mode *without* clearing the screen. You can then draw in one resolution, change the mode, and continue drawing in another resolution.

```
10 GRAPHICS 6:FOR X=1 TO 36 STEP 6:Y=RND(1)*21:GOS
   UB 60:NEXT
20 MODE 4:FOR Y=1 TO 46 STEP 3:X=RND(1)*76:GOSUB 6
   0:NEXT
30 MODE 2:FOR K=1 TO 32:X=RND(1)*156:Y=RND(1)*96:G
   OSUB 60:NEXT
40 WAIT 198,15:GET K$:GOTO 10
60 DRAW X+1,Y+1:DRAW X,Y:DRAW X+2,Y:DRAW X+2,Y+2:D
   RAW X,Y+2:RETURN
```

The MODE statement cannot be used in place of the GRAPHICS statement. MODE depends on the GRAPHICS statement having been executed beforehand. Statements like DRAW, CLS, and MODE should be used only when the screen is already in a graphics mode. The only graphics statement that should be used when the screen is in the text mode is GRAPHICS.

To make the screen go back to text mode while a program is running, use the TEXT statement.

TEXT

When executed, TEXT turns off the bitmap mode, sets the border and background colors to their default values, and clears the screen. After the screen has been changed to the text mode by TEXT, it can be returned to the graphics mode by the GRAPHICS statement.

Line Drawing

Given a starting point and an ending point, the DRAW statement draws a straight line by plotting all the points between them. All you have to do is use the keyword TO.

```
10 GRAPHICS 2
20 DRAW 10,10 TO 70,40
30 GOTO 30
```

In the example above, the starting point is 10,10 and the

destination point is 70,40. The DRAW statement first plots a point at 10,10. The drawing then continues in the direction of the point 70,40, and stops when that point is reached. The effect is one of moving the pen to the new position without ever lifting it off the screen.

Depending on the angle at which the line is drawn, the line may appear slightly jagged. (Try 50,40 as the destination point.) With some angles, the computer can only approximate a straight line. When the line is not vertical or horizontal or does not fall on a perfect diagonal, a jagged appearance results. Change the graphics mode to 4 and run the program one more time to get a better look at what's happening.

In extreme cases, a line may appear to be drawn in long segments. The line in the next example is broken into three parts because there's only one row between the two endpoints.

```
10 GRAPHICS 4
20 DRAW 10,10 TO 60,12
30 GOTO 30
```

Using a higher resolution often helps to minimize the problem.

The next program draws lines at all sorts of angles, then erases them for a shooting star effect.

```
10 GRAPHICS 0:POKE 53280,0:SETPEN 1,1
20 X1=RND(1)*320:Y1=RND(1)*200
30 X2=RND(1)*320:Y2=RND(1)*200
40 DRAW X1,Y1,1 TO X2,Y2
50 DRAW X1,Y1,0 TO X2,Y2
60 GOTO 20
```

You cannot change the current drawing pen when drawing a line this way. The point after the keyword TO consists only of the X and Y coordinates. A third number for the drawing pen is not allowed. Remember, once the pen is placed on the screen by DRAW, it stays on the screen while it draws a line.

On the other hand, a nice feature of the DRAW statement is that destination points can be chained together. A statement like DRAW X,Y TO X1,Y1 TO X2,Y2 is perfectly legal. The statement draws a line from X,Y to X1,Y1, and then draws a line from X1,Y1 to X2,Y2. Try this:

```
10 GRAPHICS 6
20 DRAW 10,10 TO 30,10 TO 30,20 TO 10,20 TO 10,10
```

```
30 GOTO 30
```

Here's another demonstration. This program uses trigonometry to create a picture.

```
10 GRAPHICS 0:POKE 53280,0:SETPEN 1,15:CX=160:CY=1
   00
20 FOR A=0 TO ↑/2 STEP ↑/180:Y=65*SIN(5*A)*SIN(A):
   X=150*SIN(A)*COS(A)
30 DRAW X+CX,Y+CY TO CX,CY TO CX-X,Y+CY:NEXT
40 FOR A=15*↑/36 TO ↑/2 STEP ↑/90:Y=-30*COS(4*A)*S
   IN(A):X=40*COS(2*A)*COS(A)
50 DRAW X+CX,Y+CY TO CX,CY TO CX-X,Y+CY:NEXT
60 DRAW CX-30,CY-70 TO CX,CY-30 TO CX+30,CY-70
70 GOTO 70
```

When DRAW is used with a destination point, the starting point is optional. If a point has already been plotted at X,Y, the statement DRAW TO X1,Y1 will draw from X,Y to X1,Y1. In effect, the last position of the pen has been remembered. This allows chaining of more destination points than can fit on one BASIC line.

```
10 GRAPHICS 2
20 DRAW 80,10 TO 111,82 TO 30,38
30 DRAWTO 130,38 TO 49,82 TO 80,10
40 GOTO 40
```

The pen position is remembered after the DRAW statement has been executed. That's how the DRAW statement on line 30 can pick up the drawing where the DRAW of line 20 left off. Notice that when the starting point is omitted, it's okay to contract the keywords *DRAW* and *TO* to form *DRAWTO.*

Another application of DRAWTO is to use it in a loop. The same DRAWTO statement can be used to draw a long chain of lines when it's executed repeatedly.

```
100 PRINT CHR$(147);CHR$(155):POKE 53280,0:POKE 53
    281,0
110 PRINT "SPIRALS":PRINT "BY CRAIG CHAMBERLAIN":P
    RINT
200 X=0.085:Y=0.105:M=1000:N=↑/180:R=360:DIM S(R),
    C(R)
210 FOR K=0 TO 90:A=M*SIN(K*N):B=A*X:C=A*Y:S(K)=B:
    S(180-K)=B
220 S(180+K)=-B:S(R-K)=-B:C(270+K)=C:C(90+K)=-C:C(
    90-K)=C:C(270-K)=-C:NEXT
230 X=160:Y=100:K=RND(-RND(0)):M=0:N=0:H=100
```

```
300 GRAPHICS 0:A=H*RND(1):B=H*RND(1):SETPEN 1,RND(
    1)*15+1:DRAW X,Y
320 FOR K=1 TO 180:N=N+A:IF N>R THEN N=N-R
330 M=M+B:IF M>R THEN M=M-R
340 O=S(N)/H:DRAWTO C(M)*O+160,S(M)*O+H:NEXT
360 FOR K=1 TO 1000:NEXT:GOTO 300
```

Remember, the pen color cannot be changed while the drawing pen is drawing lines. In the following program, the effect of the SETPEN statement is not seen until DRAW is used to plot a starting point.

```
10 GRAPHICS 6
20 DRAW 10,10 TO 20,20:REM DISPLAYED IN DEFAULT LI
   GHT GREEN
30 SETPEN 1,8
40 DRAWTO 30,10:REM STILL IN GREEN
50 DRAW 30,5 TO 10,5:REM NOW IN ORANGE
60 GOTO 60
```

Whenever the screen is cleared with a CLS, the pen is set to the home position (0,0). If DRAWTO is used after CLS and no previous point has been plotted, the drawing will start from the home position.

Area Filling

The next logical step after plotting points to draw a line is to plot points that fill in an area. That is done by the statement FILL.

The syntax for the FILL statement works the same as DRAW's.

FILL ***X coordinate, Y coordinate***

In fact, FILL X,Y plots one point at X,Y, just like DRAW X,Y. The difference is when a destination point is used. The statement FILL X,Y TO X1,Y1, after plotting an initial point at X,Y, starts drawing a line to X1,Y1. But as each point of the line is plotted, all points on the same row, to the left and right of the line, are also plotted.

```
10 GRAPHICS 6
20 DRAW 10,10 TO 30,10 TO 30,20 TO 10,20 TO 10,10
30 FILL 20,10 TO 20,20
40 GOTO 40
```

With FILL, the drawing pen behaves more like a paintbrush that takes a swipe to the left and right on each row

that's filled. The brush fills in the background and stops when it comes to a point that's already been plotted. This is why the fill stayed inside the rectangle; the brush stopped when it hit the left or right side of the shape.

FILL is normally used to fill in an area that's been outlined by DRAW. If nothing has been previously drawn to stop a fill, the filling continues until it reaches the edge of the screen. There it stops, without causing a range error. Remove line 20 of the previous example and run it again to see this happen.

Notice that the fill does not begin at the starting point. Rather, it begins on the next row.

The FILL statement is handy because it can cover large areas quickly. FILL can also be used to erase a large area. There's an optional third number after the X and Y coordinates for the destination point in a FILL statement. This number identifies the drawing pen that's to be covered up when the fill is performed. When no pen number is given, pen 0 is assumed. Since pen 0 corresponds to the background, this means that the area to be filled in is the background, and the filling should stop as soon as it hits anything else (pen 1 or the screen edge). Here are some examples.

FILL X,Y,1 TO X1,Y1

or

FILL X,Y,1 TO X1,Y1,0

Both of these statements indicate that pen 1 should be used to fill in the background.

However, this statement specifies that pen 0 should be used to erase all points plotted in pen 1.

FILL X,Y,0 TO X1,Y1,1

Keep in mind the fact that the pen number in a destination point has a different meaning than the pen number for a starting point. In a starting point, the pen number is used to change the current pen, to determine whether the pen will be used for drawing (pen 1) or for erasing (pen 0). If the number is omitted, the current pen is used.

The pen number in the destination point specifies which points are to be covered up by the fill. If no number is given,

it's assumed that the background is to be filled in. For a demonstration, the following program draws several concentric ovals, fills them, and then erases them.

```
300 GRAPHICS 0:POKE 53280,0
305 FOR R=90 TO 10 STEP -10:READ A(R/10):SETPEN 1,
    A(R/10)
310 A=160:B=100:PH=0:Y=0:X=R
320 IY=PH+Y+Y+1:XY=IY-X-X+1
330 DRAW A+X,B+Y:DRAW A-X,B+Y
340 DRAW A+X,B-Y:DRAW A-X,B-Y
350 DRAW A+Y,B+X:DRAW A-Y,B+X
360 DRAW A+Y,B-X:DRAW A-Y,B-X
370 PH=IY:Y=Y+1:IF ABS(XY)<ABS(IY) THEN PH=XY:X=X-
    1
380 IF X>=Y GOTO 320
390 NEXT R
400 FOR R=10 TO 90 STEP 10:SETPEN 1,A(R/10)
410 FILL A,B-R,1 TO A,B+R,0:FILL A,B+R+1,0 TO A,B-
    R-1,1:NEXT R
420 END
800 DATA 5,8,4,6,2,7,14,13,1
```

The DRAW and FILL statements are closely related. When the FILL statement fills in an area, it is still drawing a line. The filling is done just to each side of the line.

Filling can be limited, however, to only one side of the line. The statements LFILL and RFILL work like FILL except that they fill only to the left or the right. In some applications, they can be more convenient than using FILL. The next program shows an easier way to fill in a box.

```
10 GRAPHICS 6
20 DRAW 10,20 TO 10,10 TO 30,10
30 LFILLTO 30,20
40 GOTO 40
```

As a final demonstration of line drawing and area filling, here's an example which uses the DRAW, FILL, LFILL, and RFILL statements to draw the flags of various nations. (Since "Flags" is a longer program than the previous examples, we've included rem statements at the end of each line. Don't type in these rems; they're used with "The Automatic Proofreader." Refer to the Automatic Proofreader article in Appendix D, and have a copy of the Proofreader program saved and loaded before entering the following.)

Program 5-5. Flags

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " FLAGS OF NATIONS":PRIN
    T " BY BOB RETELLE":PRINT                    :rem 130
110 FOR K=1 TO 7:READ S$:PRINT " " S$:NEXT:PRINT
                                                 :rem 155
120 PRINT " PRESS ANY KEY":PRINT " TO DISPLAY THE
    {SPACE}NEXT FLAG":GOSUB 600                  :rem 16
300 REM SWEDEN                                   :rem 61
302 GRAPHICS 0:CLS 7:POKE 53280,0:SETPEN 1,14
                                                 :rem 251
304 LFILL 0,76 TO 90,76 TO 90,0:RFILL 319,76 TO 15
    0,76 TO 150,0                                :rem 48
306 LFILL 0,122 TO 90,122 TO 90,199:RFILL 319,122
    {SPACE}TO 150,122 TO 150,199                 :rem 184
308 GOSUB 600                                    :rem 177
310 REM FRANCE                                   :rem 39
312 CLS 1:SETPEN 1,6:LFILL 0,0 TO 106,0 TO 106,199
                                                 :rem 29
314 SETPEN 1,2:RFILL 319,0 TO 213,0 TO 213,199:GOS
    UB 600                                       :rem 143
320 REM HUNGARY                                  :rem 151
322 CLS 1:SETPEN 1,2:LFILL 0,66 TO 319,66 TO 319,0
                                                 :rem 43
324 SETPEN 1,5:LFILL 0,134 TO 319,134 TO 319,199:G
    OSUB 600                                     :rem 254
330 REM CZECHOSLOVAKIA                           :rem 144
332 CLS 6:SETPEN 1,1:RFILL 319,95 TO 140,95 TO 0,0
                                                 :rem 50
334 SETPEN 1,2:RFILL 319,96 TO 140,96 TO 0,199:GOS
    UB 600                                       :rem 168
340 REM SWITZERLAND                              :rem 210
342 CLS 2:SETPEN 1,1:DRAW 80,88 TO 80,112        :rem 123
344 DRAW 180,166 TO 180,113:DRAW 240,112 TO 240,88
    :DRAW 180,87 TO 180,32                       :rem 36
346 RFILLTO 140,32 TO 140,87:FILLTO 140,112:RFILLT
    O 140,166:GOSUB 600                          :rem 15
350 REM JAPAN                                    :rem 230
352 CLS 1:SETPEN 1,2:CX=160:CY=100:RX=80:RY=60:DRA
    W CX+RX-1,CY                                 :rem 161
354 FOR A=0 TO 2*↑ STEP ↑/30:DRAWTO CX+RX*COS(A),C
    Y+RY*SIN(A):NEXT                             :rem 213
356 FILL CX,CY-RY TO CX,CY+RY-1:GOSUB 600        :rem 138
360 REM GREAT BRITAIN                            :rem 249
362 CLS 1:SETPEN 1,2:DRAW 136,0 TO 136,88:DRAW 136
    ,112 TO 136,199                              :rem 75
364 LFILLTO 184,199 TO 184,112:FILLTO 184,88:LFILL
    TO 184,0                                     :rem 165
```

```
370 DRAW 319,3 TO 216,80 TO 200,80:RFILLTO 307,0
                                              :rem 169
372 DRAW 319,196 TO 216,120 TO 200,120:RFILLTO 307
    ,199                                      :rem 225
374 DRAW 0,196 TO 104,120 TO 120,120:LFILLTO 12,19
    9                                          :rem 54
376 DRAW 0,3 TO 104,80 TO 120,80:LFILLTO 12,0
                                               :rem 2
380 SETPEN 1,6:DRAW 200,0 TO 200,60:LFILLTO 280,0:
    RFILL 319,20 TO 240,80                    :rem 11
382 RFILL 319,179 TO 240,120:DRAW 200,199 TO 200,1
    40:LFILLTO 280,199                        :rem 240
384 DRAW 120,199 TO 120,140:RFILLTO 40,199:LFILL 0
    ,180 TO 80,120                             :rem 27
386 LFILL 0,20 TO 80,80:DRAW 120,0 TO 120,60:RFILL
    TO 40,0:GOSUB 600:END                      :rem 7
600 WAIT 198,15:GET K$:RETURN                 :rem 162
800 DATA SWEDEN,FRANCE,HUNGARY,CZECHOSLOVAKIA,SWIT
    ZERLAND,JAPAN,GREAT BRITAIN               :rem 160
```

Multicolor Modes

What happens when two lines of different colors cross each other? This presents a problem. In modes 0, 2, and 4, when pen 1 is used to plot points in several colors, if the points in one color come too close to the points in another color, the colors interfere with each other. The following program illustrates this problem.

```
10 GRAPHICS 4
20 DRAW 25,5:D=15
30 FOR Y=10 TO 40 STEP 5
40 DRAWTO 40+D,Y
50 D=-D:NEXT
60 SETPEN 1,2:DRAW 43,5 TO 43,40
70 GOTO 70
```

The cause of the problem is that no matter which bitmap resolution is used, the color resolution is always 40 × 25. The bitmap portion of the screen may allow 320 × 200 points, but the color portion is always divided into 40 × 25 "color squares." Each color square is the same size as a text character, and all the points in a square have to share the same color. If the first color square on the screen is red, then all points plotted in that area will appear red. Points drawn in the next color square could have a different color, but again they

would all have to be the same color. This next program gives a better view of what actually happens when two colors come too close to each other.

```
10 GRAPHICS 4
20 DRAW 20,40 TO 20,10 TO 60,10:LFILLTO 60,40
30 SETPEN 1,2:DRAW 19,5 TO 59,45
40 GOTO 40
```

The parts of the line outside the box appear in the correct resolution. Inside the box, though, you get a perfect picture of the color squares. As the points forming the line were plotted, the color squares for those points were changed to red, the new pen color. The problem is that the color squares were also used by other points not in the line, and those points changed to red as well.

To help get around this limitation, the VIC-II chip supports a multicolor bitmap mode. This mode allows three colors in each color square, not just one. Now, when three points are plotted in the same color square, each can have a different color.

The multicolor mode has been implemented in the BASIC extensions as graphics modes 1, 3, 5, and 7. They are practically the same as modes 0, 2, 4, and 6, except that there are now three drawing pens (pens 1, 2, and 3) for plotting, not just one. You can use pen 1 to draw a line in one color, then use pen 2 to draw a line with a different color crossing the first line. There will be no conflict at the intersection.

```
10 GRAPHICS 5
20 DRAW 25,5:D=15
30 FOR Y=10 TO 40 STEP 5
40 DRAWTO 40+D,Y
50 D=-D:NEXT
60 SETPEN 2,2:DRAW 43,5,2 TO 43,40
70 GOTO 70
```

The program with the line passing through the box has been modified to use all three drawing pens.

```
10 GRAPHICS 5
20 DRAW 20,40 TO 20,10 TO 60,10:LFILLTO 60,40
30 SETPEN 2,2:DRAW 19,5,2 TO 59,45
40 GOTO 40
```

As shown by the last example, the default colors for pens 2 and 3 are light red (color 10) and blue (color 6), respectively. These colors were chosen because they have different luminance values, so they'll appear as different shades of gray on a black-and-white television. Pen 1's default remains light green (color 13).

The colors for pens 1, 2, and 3 are set back to the default values every time the GRAPHICS statement is executed.

The multicolor modes 1, 3, 5, and 7 are identical in resolution to modes 0, 2, 4, and 6, with one exception. The VIC-II chip does not support a horizontal resolution of 320 points in multicolor mode. Therefore, mode 1 has a resolution of 200 points vertically, but only 160 across.

Mode	Resolution	Mode	Resolution
0	320 × 200	1	160 × 200
2	160 × 100	3	160 × 100
4	80 × 50	5	80 × 50
6	40 × 25	7	40 × 25

It's not possible to mix normal and multicolor modes on the same screen. The MODE statement can be used to switch to any mode, but it's recommended that you switch only among modes 0, 2, 4, and 6, or modes 1, 3, 5, and 7. If you cross between a normal mode and a multicolor mode, the picture will appear to be distorted.

Since modes 3, 5, and 7 can do everything that modes 2, 4, and 6 can do, with the added convenience of two more drawing pens, you might wonder why modes 2, 4, and 6 are even needed. The fact that normal and multicolor modes cannot be mixed explains why the multicolor modes are not used exclusively. For most applications, the convenience of three drawing pens makes the multicolor modes preferable. However, if you want to use the highest resolution, 320 × 200, the multicolor modes cannot be used. If you want to mix other resolutions with the 320 × 200 mode, you need to use modes 2, 4, and 6.

Multicolor modes are used when you need up to three colors next to each other. The following program, which draws some three-dimensional cubes, could not be written in a normal graphics mode.

```
10 GRAPHICS 1:POKE 53280,0
20 FOR X=8 TO 140 STEP 30:FOR Y=20 TO 170 STEP 37:
```

```
   A=INT(RND(Ø)*15)+1
3Ø B=INT(RND(Ø)*15)+1:IF B=A GOTO 3Ø
4Ø C=INT(RND(Ø)*15)+1:IF C=A OR C=B GOTO 4Ø
5Ø SETPEN 1,A:DRAW X,Y,1 TO X,Y+2Ø TO X+15,Y+2Ø:LF
   ILLTO X+15,Y:SETPEN 2,B
6Ø DRAW X+16,Y-1,2 TO X+21,Y-11:LFILL X+21,Y-9 TO
   {SPACE}X+21,Y+9 TO X+16,Y+19
7Ø SETPEN 3,C:DRAW X+2Ø,Y-11,3 TO X+5,Y-11:RFILLTO
    X+1,Y-1
8Ø NEXT:NEXT:WAIT 198,15:GET K$:GOTO 1Ø
```

This program demonstrates drawing and filling in three drawing pens. One other possibility yet to be considered is filling in an area that has already been filled by another pen. If an area has been filled using pen 1 and you want to fill the inside of the area using pen 2, the optional third number after the destination point in the FILL statement has to be used. The following program draws some concentric diamond shapes to show how the pen number is used.

```
1Ø GRAPHICS 3
2Ø DRAW 8Ø,Ø TO 159,49 TO 8Ø,99:RFILLTO Ø,49 TO 79
   ,1
3Ø DRAW 8Ø,2Ø,2 TO 139,49 TO 8Ø,79:RFILLTO 2Ø,49,1
    TO 8Ø,2Ø,1
4Ø DRAW 8Ø,3Ø,3 TO 119,49 TO 8Ø,69:RFILLTO 4Ø,49,2
    TO 8Ø,3Ø,2
5Ø DRAW 8Ø,45,Ø TO 99,49 TO 8Ø,54:RFILLTO 6Ø,49,3
   {SPACE}TO 8Ø,45,3
6Ø GOTO 6Ø
```

The first diamond is drawn normally. No third number is used in the destination points because it's the background that's being filled. On the second diamond, the drawing pen is changed to pen 2. Because this diamond is drawn inside the first one, the destination points in the RFILL statement indicate that points plotted in pen 1 are to be covered by the filling. Notice that the third number has to be used in both destination points. For the third diamond, pen 3 is used, and the filling covers up pen 2. The last diamond is drawn by erasing part of the previous diamond. Points plotted in pen 3 are erased by pen 0.

One other difference between the multicolor modes and the normal modes is that in multicolor modes, the background color can be changed by POKEing location 53281. This allows you to change the background color without clearing the screen.

Shapedit

Introduction to Shape Tables

Statements like GRAPHICS and DRAW certainly make bitmapped graphics easy to use. In fact, bitmapped graphics would hardly even be feasible without them. These statements become less convenient, however, when they're used to draw detailed shapes. Drawing a rectangle or a box is easy enough, but drawing something like the outline of a person may require much more complicated plotting. It can take a lot of work to get all the coordinates in DRAW statements set properly. Also, the speed advantage of the machine language diminishes when several statements have to be executed.

The best way to handle shape drawing is to use a graphics utility commonly known as a *shape table*. A shape table is a sequence of instructions which tell the drawing pen where to plot points. Each instruction moves the pen one position, such as up, down, left, or right. After the pen moves to the new position, a point is plotted. Several instructions in the proper order can make the drawing pen trace a path to draw a shape.

To complete the collection of machine language graphics routines, we've provided a routine which uses shape tables to draw. Instructions in this implementation can move the pen in eight directions and plot points using pens 0 to 3. There are also advanced instructions to change the pen color, move the pen to another place on the screen, draw lines, fill areas, and so on.

The SHAPE statement is used to draw a shape. The syntax for this statement is:

SHAPE ***address, X coordinate, Y coordinate***

A shape table is stored as bytes in memory. The first number after the keyword SHAPE is the address of the shape table. The X and Y coordinates are the starting position of the pen. When the SHAPE statement is executed, the pen starts at the designated position and moves according to the instructions in the shape table.

The significance of this is that a single BASIC statement can draw an entire shape. And because only one statement is involved, the drawing of individual points is done much faster. By using the SHAPE statement, you can do things with

bitmapped graphics that otherwise cannot be done.

Let's take a look at just what the SHAPE statement can do. The following demonstrations illustrate just some of the things SHAPE is capable of. *Remember that "BMG.OBJ" (Program 5-3) must first be loaded into memory.* Use the loader program (Program 5-1 for disk, or Program 5-2 for tape) to load up the BMG BASIC extensions program.

The three example programs have many lines in common. To reduce typing, enter Program 6-1 and save it using the filename SHP.BAS.

These lines do not form a program and will not work properly if you try to run them. Rather, they are subroutines which must be contained in every program that uses shapes, whether those programs are demonstrations from this book or programs you later write yourself. The lines have been numbered starting at 56500 so that they'll be out of the way of normal program lines.

Program 6-1. SHP.BAS

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
56500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".SHP":GOSUB 59000                    :rem 50
56510 POKE SA,0:POKE SX,LA-256*INT(LA/256):POKE SY
      ,INT(LA/256)                             :rem 76
56520 SYS 65493:IF PEEK(SP)AND1 GOTO 59100 :rem 42
56530 NS=PEEK(LA):DIM AS(NS):LA=LA+1       :rem 107
56540 FOR K=0 TO NS:AS(K)=LA+2:LA=LA+2+PEEK(LA)+25
      6*PEEK(LA+1):NEXT:RETURN             :rem 120
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                                   :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                                 :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END              :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                           :rem 36
59120 PRINT ST:END                         :rem 70
```

Now that you've created the "SHP.BAS" file, you're ready to enter the demonstrations. As you type in each example program, start with the SHP.BAS file already in memory. In other words, instead of typing NEW before entering each program, enter LOAD"SHP.BAS",8 for disk, or LOAD"SHP.BAS" for tape. Then begin typing the lines in the

demonstration program. The result merges the SHP.BAS lines with those from the demonstration file to create a complete program.

One more thing must be done before a demonstration will work. Shape tables are stored as data files on tape or disk. Use the "MLX" program from Appendix C (this is the same program you used to create BMG.OBJ in Chapter 5) to enter the shape file which corresponds with the demonstration program. If you're using tape, save the shape file *immediately after* the demonstration program.

All demonstration programs as listed are set for use with the disk drive. To make a demonstration work with the Datassette, change the assignment of variable DN (the device number) from 8 to 1 in line 110.

Program 6-2 is the first demonstration. Merge it with the SHP.BAS (Program 6-1) and save the program with the filename MUNCHKINS.

Program 6-2. MUNCHKINS

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " MUNCHKINS":PRINT " BY
    {SPACE}CRAIG CHAMBERLAIN":PRINT        :rem 176
110 DN=8:SA=780:SX=781:SY=782:SP=783       :rem 133
120 F$="MUNCHKIN":LA=PEEK(49)+256*PEEK(50)+1000:GO
    SUB 56500:REM LOAD SHAPES               :rem 55
300 GRAPHICS 3:POKE 53280,0                :rem 222
310 M=AS(0):SHAPE M,110,10:SHAPE M,70,20:SHAPE M,4
    0,35                                   :rem 171
320 MODE 1:SHAPE M,30,35:SHAPE M,110,65:SHAPE M,35
    ,125                                   :rem 163
330 MODE 5:SHAPE M,45,25                    :rem 16
340 WAIT 198,15:GET K$                     :rem 137
350 GRAPHICS 7:SHAPE M,15,3                   :rem 9
360 WAIT 198,15:GET K$                     :rem 139
370 END                                    :rem 113
```

Program 6-3 is the shape file for the "MUNCHKINS" program. Save this file using the filename MUNCHKIN.SHP. All shape files (shape filenames always end with .SHP) must be entered and saved using the MLX program found in Appendix C. Each of these files requires a starting and ending address, as well as a filename, when prompted by MLX. Be sure to read the accompanying article in Appendix C before going on.

Program 6-3. MUNCHKIN.SHP

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49295
Filename: MUNCHKIN.SHP

```
49152 :000,139,000,087,083,083,136
49158 :083,083,083,083,099,115,040
49164 :115,099,115,067,115,115,126
49170 :115,115,091,115,115,115,172
49176 :107,083,083,083,083,083,034
49182 :083,083,083,083,083,083,016
49188 :067,067,083,099,099,075,014
49194 :083,107,083,083,091,115,092
49200 :115,115,115,115,115,115,226
49206 :115,115,115,115,115,115,232
49212 :115,115,115,115,039,163,210
49218 :063,171,163,171,171,171,208
49224 :171,179,179,191,083,083,190
49230 :083,119,211,063,227,235,248
49236 :227,235,227,235,235,235,198
49242 :243,191,083,083,083,103,108
49248 :147,063,163,163,171,163,198
49254 :163,171,163,155,155,191,076
49260 :083,083,083,135,211,063,254
49266 :227,219,227,235,227,219,188
49272 :219,227,227,235,191,083,022
49278 :083,083,083,071,147,063,144
49284 :163,163,163,155,155,155,062
49290 :163,155,155,255,013,013,124
```

This first demonstration uses several SHAPE statements to draw the same shape at different positions and in different resolutions. Notice that the legs of the Munchkin shape consist of many points that would require a lot of DRAW statements if a shape table wasn't used. Also notice that the shapes are drawn very quickly.

In the second demonstration, the illusion of more than four shapes is created by drawing the shapes with different color combinations. Type in and save Program 6-4 using the filename BROTHERHOOD (don't forget to first load SHP.BAS before typing what you see below). You'll also need to enter (with MLX) the shape file (Program 6-5).

Program 6-4. BROTHERHOOD

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " BROTHERHOOD":PRINT " B
    Y MARK DAVIDS":PRINT                    :rem 170
110 DN=8:SA=780:SX=781:SY=782:SP=783        :rem 133
120 F$="FAMILY":LA=PEEK(49)+256*PEEK(50)+1000:GOSU
    B 56500:REM LOAD SHAPES                 :rem 156
300 GRAPHICS 3:POKE 53280,11:POKE 53281,12 :rem 11
310 FOR K=20 TO 80 STEP 30:GOSUB 600:SHAPE AS(4*RN
    D(0)),10,K                              :rem 182
320 FOR J=1 TO 10:GOSUB 600:SHAPE AS(4*RND(0)):NEX
    T J,K                                   :rem 21
330 WAIT 198,15:GET K$:CLS 12:GOTO 310      :rem 14
600 N=3*RND(0):SETPEN 1,9+N:SETPEN 3,7*RND(0)+1:ON
     N GOTO 620,630                         :rem 64
610 SETPEN 2,7*INT(2*RND(0)):RETURN         :rem 60
620 N=INT(3*RND(0)):SETPEN 2,-7*(N=1)-9*(N=2):RETU
    RN                                      :rem 219
630 SETPEN 2,0:RETURN                       :rem 16
```

Program 6-5. FAMILY.SHP

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49445
Filename: FAMILY.SHP

```
49152 :003,070,000,083,115,139,154
49158 :139,131,139,131,147,022,203
49164 :031,163,155,163,155,099,010
49170 :083,008,179,033,031,000,096
49176 :001,115,115,123,067,083,016
49182 :083,083,099,115,115,123,136
49188 :179,139,147,147,147,155,182
49194 :016,002,021,179,029,095,128
49200 :227,018,031,061,031,163,067
49206 :179,195,049,031,012,195,203
49212 :013,095,227,037,031,155,106
49218 :179,203,033,031,030,017,047
49224 :255,046,000,083,091,091,126
49230 :155,147,147,147,147,075,128
49236 :075,075,016,021,067,123,205
49242 :083,083,067,115,115,179,220
49248 :139,147,147,155,012,021,205
49254 :155,147,147,227,243,243,240
49260 :227,211,211,227,227,099,030
49266 :051,115,195,195,022,049,229
49272 :255,103,000,083,083,067,199
```

```
49278 :067,083,067,083,067,211,192
49284 :083,067,091,067,075,067,070
49290 :115,115,115,099,083,083,236
49296 :075,131,179,179,179,171,034
49302 :163,163,147,014,017,163,049
49308 :163,163,179,227,083,099,046
49314 :083,099,083,099,099,083,196
49320 :012,025,227,243,243,243,137
49326 :227,211,211,211,219,243,216
49332 :243,243,243,243,227,211,054
49338 :211,211,211,211,219,243,212
49344 :243,243,243,243,243,243,114
49350 :227,211,211,211,211,211,200
49356 :211,211,227,024,031,018,158
49362 :099,029,031,163,179,179,122
49368 :179,006,067,115,115,025,211
49374 :095,026,025,255,061,000,172
49380 :083,219,211,203,203,211,078
49386 :067,123,083,083,067,115,004
49392 :115,179,139,147,147,155,098
49398 :000,013,227,219,219,211,111
49404 :075,008,243,243,243,227,011
49410 :211,211,211,243,243,243,084
49416 :243,219,211,211,163,179,210
49422 :179,163,147,147,163,163,208
49428 :163,163,227,051,243,131,230
49434 :131,131,131,147,018,033,105
49440 :255,013,013,013,013,013,096
```

The final demonstration draws a background scene and then uses fast shape-table plotting to make some tulips grow. As before, make sure both the following programs are on the same disk. If you're using tape, place the shape file (Program 6-7), immediately after Program 6-6. Remember to place the lines from SHP.BAS, Program 6-1, in memory before starting to type Program 6-6.

Program 6-6. TULIPS

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " TULIPS":PRINT " BY MAR
    K DAVIDS":PRINT                          :rem 75
110 DN=8:SA=780:SX=781:SY=782:SP=783        :rem 133
120 F$="TULIPS":LA=PEEK(49)+256*PEEK(50)+1000:GOSU
    B 565000:REM LOAD SHAPES                 :rem 187
200 DIM X(14),Y(14),TX(14),TY(14)            :rem 210
210 FOR C=1 TO 14:TX(C)=10*C+INT(5*RND(0)):TY(C)=7
    0+INT(10*RND(0)):NEXT                    :rem 210
```

```
220 FOR C=1 TO 14:R=C+INT((15-C)*RND(0)):X(C)=TX(R
    ):Y(C)=TY(R)                                  :rem 195
230 TX(R)=TX(C):TY(R)=TY(C):NEXT                  :rem 226
300 GRAPHICS 1:POKE 53280,6:SETPEN 1,14:LFILL 0,90
     TO 159,90 TO 159,0                           :rem 148
310 SETPEN 2,9:DRAW 0,75,2 TO 50,35 TO 100,60 TO 1
    20,40 TO 159,70                               :rem 133
320 FILL 0,90 TO 50,90,1 TO 50,36,1:FILL 120,61 TO
     120,41,1                                      :rem 32
330 MODE 5:SETPEN 1,5:DRAW 0,22,1 TO 79,22:rem 179
340 MODE 3:FOR C=1 TO 75:DRAW 150*RND(0),43:NEXT
                                                  :rem 105
400 FOR K=0 TO 9:FOR C=1 TO 14:SHAPE AS(K),X(C),Y(
    C):NEXT:NEXT                                  :rem 231
410 WAIT 198,15:GET K$                            :rem 135
420 END                                           :rem 109
```

Program 6-7. TULIPS.SHP

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49283
Filename: TULIPS.SHP

```
49152 :010,007,000,067,115,075,018
49158 :067,091,099,255,006,000,012
49164 :067,123,067,075,091,255,178
49170 :008,000,067,033,099,067,036
49176 :099,091,083,255,013,000,053
49182 :067,004,009,075,115,099,143
49188 :006,017,067,067,006,021,220
49194 :255,014,000,067,008,017,147
49200 :075,107,018,001,083,115,191
49206 :004,017,067,067,255,011,219
49212 :000,067,008,017,067,115,078
49218 :067,014,025,067,067,255,049
49224 :008,000,081,067,115,067,154
49230 :083,083,099,255,007,000,093
49236 :067,089,119,187,147,147,072
49242 :255,016,000,119,000,105,073
49248 :147,179,147,147,026,109,083
49254 :024,097,179,147,179,179,139
49260 :255,015,000,067,004,105,042
49266 :119,163,163,131,131,155,208
49272 :139,155,139,163,163,255,110
49278 :001,000,255,013,013,013,165
```

The Shapedit Program

Now that you've seen a small sampling of what the SHAPE statement can do, you ought to try creating a few shapes of your own. You'll be glad to know that making a shape is as easy as using a joystick. "Shapedit," Program 6-8, simplifies the task of defining a shape table. This program makes it easy to create and edit up to 100 shapes in any graphics mode. It also has options to save shape files to tape or disk. Shapedit has been carefully designed to make shape creation as simple as possible.

Shapedit uses graphics statements, so be sure that the BASIC extensions have been installed (by running "BMGLOADER") before you enter, save, load, or run the program.

It's set to use the disk drive. Again, to make it work with the Datassette, change the statement DN=8 in the first line to DN=1.

Save the program using the filename SHAPEDIT before you run it. This is a wise precaution whenever you run a program for the first time, in case the computer crashes due to a typing mistake.

Plug a joystick into port 2 and type RUN. The program takes a moment to initialize and then displays a black screen with a five-line text window at the bottom. The first line reports status information, including the current drawing pen, the current origin, and the current pen position. The next four rows have items which are accessible by moving the joystick. Items are selected by pressing the joystick button. The screen above the text window is graphics mode 7.

Program 6-8. Shapedit

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D. This program requires that BMG.OBJ (Program 5-3) be loaded into your computer before it is entered, saved, loaded, or run.

```
100 DN=8:X=0:Y=0:XX=0:YY=0:JS=56320:SA=780:SX=781:
    SY=782:SP=783:GOTO 800                    :rem 82
110 POKE 214,19:PRINT:F$=STR$(X):PRINT TAB(31) RIG
    HT$(F$,LEN(F$)-1) ",";                    :rem 219
120 F$=STR$(Y):PRINT RIGHT$(F$,LEN(F$)-1) LEFT$(BL
    $,39-POS(0)):K=FRE(0):RETURN              :rem 13
130 SYS 49345:T1=MX(MD):T2=MY(MD)             :rem 164
140 SYS 49391,X,Y:GOSUB 110                   :rem 243
```

```
150 N=PEEK(JS)AND15:IF N=15 THEN ON ((PEEK(JS)AND1
    6)=Ø)+2 GOTO 185,150                    :rem 69
160 X=X+JX(N):IF X<Ø OR X>T1 THEN X=-T1*(X<Ø)
                                            :rem 238
170 Y=Y+JY(N):IF Y<Ø OR Y>T2 THEN Y=-T2*(Y<Ø)
                                            :rem 248
180 GOTO 140                                :rem 103
185 WAIT JS,16:RETURN                        :rem 29
190 POKE 214,20+(CN AND3):PRINT:PRINT TAB(U(CN/4))
     CHR$(N) CN$(CN);:RETURN                :rem 221
200 N=146:GOSUB 190:IF PF=Ø GOTO 230        :rem 166
210 PF=Ø:X=XX:Y=YY:GOSUB 130:XX=X:YY=Y:POKE 53269,
    Ø:GOSUB 740:POKE 198,Ø                  :rem 112
220 GOSUB 650:IF PEEK(SP)AND1ORFNDP(52495)>MX(MD)O
    RPEEK(52497)>MY(MD) GOTO 790             :rem 62
230 POKE 52479,DP:IF DP THEN POKE 53280,PEEK(52488
    +DP):GOTO 240                           :rem 193
235 POKE 53280,CL                           :rem 134
240 X=FNDP(52495):Y=PEEK(52497):GOSUB 110    :rem 74
245 SYS 49466:N=PEEK(SA):IF N=255 THEN POKE 53269,
    Ø:GET F$:GOTO 260                       :rem 184
250 POKE AS+SL-1,N:POKE AS+SL,255:SL=SL+1:FM=FM-1:
    IF FM>Ø GOTO 230                         :rem 47
255 F$="MEMORY FULL":GOTO 795               :rem 171
260 IF F$<>CHR$(136) GOTO 275               :rem 206
265 DP=DP+1:DP=DPANDPEEK(52483):POKE 214,19:PRINT:
    PRINT TAB(5) CHR$(48+DP)                 :rem 74
270 GOTO 230                                :rem 103
275 IF F$<>CHR$(133) GOTO 330               :rem 201
280 IF SL=1 GOTO 245                         :rem 10
285 SL=SL-1:FM=FM+1:POKE AS+SL-1,255:CLS CL:IF SL>
    1 GOTO 220                              :rem 173
290 PF=1:GOTO 320                           :rem 167
300 POKE AS+SL-1,N:POKE AS+SL,255:SL=SL+1:FM=FM-1:
    IF FM<1 GOTO 255                         :rem 49
310 IF CN=12 GOTO 521                        :rem 37
320 N=146:GOSUB 190                          :rem 15
330 CN=8:N=18:GOSUB 190:K=FRE(Ø):WAIT JS,16
                                            :rem 221
350 POKE 198,Ø:K=PEEK(JS)AND15:IF K<>15 THEN N=146
    :GOSUB 190:GOTO 380                     :rem 126
360 IF PEEK(JS)AND16 GOTO 350                :rem 70
365 WAIT JS,16:IF PF AND (CN=2 OR CN=9 OR CN=10 OR
     CN>11) GOTO 350                        :rem 161
370 ON CN+1 GOTO 400,410,440,450,460,470,480,350,2
    00,500,510,490,520                       :rem 33
375 N=U(CN-8):ON CN-12 GOTO 530,540,550:GOTO 560
                                             :rem 40
380 CN=CN+4*JX(K)+JY(K):IF JY(K)=-1 AND (CN AND3)=
    3 THEN CN=CN+4                          :rem 230
```

```
385 IF JY(K)=1 AND (CN AND3)=Ø THEN CN=CN-4
                                          :rem 243
390 CN=CN-2Ø*(CN<Ø)+2Ø*(CN>19):IF CN=7 GOTO 38Ø
                                          :rem 229
395 N=18:GOSUB 19Ø:FOR K=1 TO 5Ø:NEXT:GOTO 35Ø
                                           :rem 79
4ØØ GOSUB 77Ø:IF F$="" THEN GOSUB 76Ø:GOTO 32Ø
                                          :rem 186
4Ø1 POKE 53269,Ø:PF=1:GOSUB 67Ø:POKE SA,1:POKE SX,
    DN:POKE SY,Ø:SYS 65466                :rem 159
4Ø2 F$=F$+".SHP":GOSUB 6ØØ:POKE SA,Ø:POKE SX,FNL(B
    A):POKE SY,FNH(BA)                    :rem 189
4Ø3 POKE 648,188:POKE 214,19:PRINT        :rem 146
4Ø4 SYS 65493:N=(PEEK(SP)AND1)*PEEK(SA):POKE 648,1
    92:PRINT                              :rem 163
4Ø5 GOSUB 76Ø:CLS CL:SYS 49152:IF N THEN GOSUB 73Ø
    :GOTO 42Ø                              :rem 8
4Ø6 NS=PEEK(BA):AS=BA+1:FOR K=Ø TO NS:SL%(K)=FNDP(
    AS):AS=AS+2+SL%(K):NEXT               :rem 92
4Ø7 FM=MT-AS:IF FM<Ø THEN GOSUB 73Ø:F$="NOT ENOUGH
     MEMORY":GOTO 795                     :rem 171
4Ø8 SN=Ø:POKE 214,2Ø:PRINT:PRINT TAB(15) "Ø ":GOSU
    B 63Ø:GOTO 32Ø                         :rem 35
41Ø IF NS=Ø AND SL=1 GOTO 35Ø             :rem 227
411 GOSUB 77Ø:IF F$="" THEN GOSUB 76Ø:GOTO 32Ø
                                          :rem 188
412 GOSUB 64Ø:POKE BA,NS:AS=BA+1:FOR K=Ø TO NS:POK
    E AS,FNL(SL%(K))                       :rem 43
413 POKE AS+1,FNH(SL%(K)):AS=AS+2+SL%(K):NEXT:POKE
     53269,Ø:GOSUB 67Ø                    :rem 149
414 POKE SA,1:POKE SX,DN:POKE SY,Ø:SYS 65466:F$=F$
    +".SHP":GOSUB 6ØØ                      :rem 99
415 POKE SA,251:POKE 251,FNL(BA):POKE 252,FNH(BA)
                                          :rem 224
416 POKE SX,FNL(AS):POKE SY,FNH(AS):POKE 648,188:P
    OKE 214,19:PRINT                      :rem 157
417 SYS 65496:N=(PEEK(SP)AND1)*PEEK(SA):POKE 648,1
    92:PRINT                              :rem 17Ø
418 GOSUB 76Ø:SYS 49152:GOSUB 63Ø:IF N=Ø GOTO 32Ø
                                          :rem 157
42Ø IF N=4 THEN F$="FILE NOT FOUND":GOTO 795
                                          :rem 163
422 IF N=5 THEN F$="DEVICE NOT PRESENT":GOTO 795
                                          :rem 219
424 F$=STR$(ST):GOTO 795                  :rem 11Ø
44Ø SL%(SN)=SL:H=NS:GOSUB 66Ø:IF FM<SL%(N) THEN F$
    ="NOT ENOUGH MEMORY":GOTO 795         :rem 157
442 S=BA+3:IF N=Ø GOTO 445                :rem 1Ø1
443 IF N<=SN THEN FOR K=Ø TO N-1:S=S+SL%(K)+2:NEXT
    :GOTO 445                              :rem 61
```

```
444 S=MT+2:FOR K=N TO NS:S=S-2-SL%(K):NEXT:rem 231
445 L=SL%(N):D=AS+SL-1:GOSUB 620:SL=SL+L-1:FM=FM-L
    +1:POKE 53269,0:GOSUB 650                :rem 113
446 IF PEEK(SP)AND1 OR FNDP(52495)>MX(MD) OR PEEK(
    52497)>MY(MD) GOTO 790                   :rem 243
447 X=FNDP(52495):Y=PEEK(52497):GOSUB 110:GOTO 320
                                             :rem 91
450 GOSUB 780:IF K$<>"Y" GOTO 320            :rem 213
455 GOSUB 670:POKE 648,4:POKE 51578,200:POKE SX,23
    7:POKE SY,246:SYS 49888:END              :rem 111
460 SL%(SN)=SL:L=0:H=NS:N=SN:IF FM>2 AND SL%(NS)>1
     AND H<MX THEN H=NS+1                    :rem 86
462 GOSUB 720:PF=1:IF N=SN GOTO 320          :rem 182
464 T1=N:N=146:GOSUB 190:GOSUB 640           :rem 182
466 SN=T1:IF SN>NS THEN NS=SN:SL%(SN)=1:FM=FM-3:PO
    KE AS+SL+AB+2,255                        :rem 239
468 GOSUB 630:GOTO 330                       :rem 196
470 N=CL:L=0:H=14:GOSUB 720:CL=N:CLS CL:POKE 53269
    ,0:PF=1                                  :rem 42
475 POKE 53287,-(CL<>1):GOTO 320             :rem 196
480 N=MD:L=0:H=7:GOSUB 720:MD=N:MODE MD:IF XX>MX(M
    D) THEN XX=MX(MD)                        :rem 27
482 IF YY>MY(MD) THEN YY=MY(MD)              :rem 75
484 DP=DPANDPEEK(52483):X=XX:Y=YY:GOSUB 760:POKE 5
    2479,DP:PF=1:GOTO 320                    :rem 38
490 GOSUB 780:IF K$<>"Y" GOTO 320            :rem 217
492 FM=FM+SL-1:SL=1:POKE AS,255:CLS CL:PF=1:IF SN<
    NS OR NS=0 GOTO 320                      :rem 227
494 NS=NS-1:FM=FM+3:SN=NS:POKE 214,20:PRINT:PRINT
    {SPACE}TAB(14) SN "{LEFT} ":GOSUB 630    :rem 176
496 GOTO 320                                 :rem 113
500 H=15:GOSUB 660:POKE 52505,N+128:N=16*N+7:GOTO
    {SPACE}300                               :rem 140
510 H=3:GOSUB 660:POKE 52492,0:IF N THEN POKE 5249
    2,PEEK(52486)ORPEEK(52435+N)             :rem 40
515 N=N*64+15:GOTO 300                       :rem 159
520 TX=X:TY=Y:GOSUB 130:TX=X-TX:TY=Y-TY      :rem 78
521 IF TX=0 GOTO 524                         :rem 20
522 N=ABS(TX):IF N>64 THEN N=64              :rem 218
523 T1=TX:TX=SGN(TX)*(ABS(TX)-N):N=(N-1)*4-2*(T1>0
    ):GOTO 300                               :rem 204
524 IF TY=0 THEN POKE 52495,FNL(X):POKE 52496,FNH(
    X):POKE 52497,Y:GOTO 320                 :rem 133
525 N=ABS(TY):IF N>32 THEN N=32              :rem 212
526 T1=TY:TY=SGN(TY)*(ABS(TY)-N):N=(N-1)*8-4*(T1>0
    )+1:GOTO 300                             :rem 53
530 POKE 52498,FNL(X):POKE 52499,FNH(X):POKE 52500
    ,Y:GOTO 300                              :rem 159
540 X=FNDP(52498):Y=PEEK(52500):GOTO 555      :rem 7
550 X=0:Y=0                                  :rem 95
```

```
555 SYS 49391,X,Y:POKE 52495,FNL(X):POKE 52496,FNH
    (X):POKE 52497,Y:GOTO 300                :rem 251
560 POKE SA,20-CN AND3:POKE SX,FNL(X):POKE SP,FNH(
    X):POKE SY,Y                             :rem 208
565 SYS 51831:GOTO 300                       :rem 167
600 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                     :rem 240
610 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                  :rem 158
620 POKE 251,FNL(S):POKE 252,FNH(S):POKE 253,FNL(D
    ):POKE 254,FNH(D)                         :rem 2
625 POKE SX,FNL(L):POKE SY,FNH(L):SYS 49777:RETURN
                                             :rem 63
630 SL=SL%(SN):AS=BA+3:IF SN THEN FOR K=0 TO SN-1:
    AS=AS+SL%(K)+2:NEXT                      :rem 241
632 AB=0:IF SN=NS THEN RETURN                :rem 226
634 FOR K=SN+1 TO NS:AB=AB+SL%(K)+2:NEXT:S=AS+SL:L
    =AB:D=MT-L:GOTO 620                      :rem 121
640 SL%(SN)=SL:IF AB=0 THEN RETURN           :rem 244
645 L=AB:S=MT-L:D=AS+SL:GOTO 620             :rem 179
650 POKE 52493,FNL(AS):POKE 52494,FNH(AS):POKE 524
    95,FNL(XX):POKE 52496,FNH(XX)             :rem 27
652 POKE 52497,YY:POKE SA,0:POKE 52489,13:POKE 524
    90,10:POKE 52491,6                       :rem 221
654 SYS 52166:RETURN                         :rem 188
660 N=146:GOSUB 190:CN=7:N=18:GOSUB 190:PRINT "
    {OFF} 0 ";:N=0:L=0:GOTO 720              :rem 211
670 POKE 49321,0:FOR K=1 TO 100:NEXT:SYS 49322:RET
    URN                                       :rem 9
720 K=PEEK(JS)AND15:IF K<>15 THEN N=N+JX(K):GOTO 7
    26                                       :rem 228
722 IF PEEK(JS)AND16 GOTO 720                 :rem 73
724 RETURN                                   :rem 125
726 IF N<L THEN N=H                           :rem 6
727 IF N>H THEN N=L                           :rem 9
728 POKE 214,20+(CN AND3):PRINT:PRINT TAB(14) N "
    {LEFT} ";                                :rem 141
729 FOR K=1 TO 50:NEXT:GOTO 720              :rem 207
730 NS=0:SN=0:AS=BA+3:SL=1:POKE AS,255:FM=MT-AS:AB
    =0                                       :rem 126
732 POKE 214,20:PRINT:PRINT TAB(15) "0 ":RETURN
                                             :rem 154
740 POKE 214,19:PRINT:F$=STR$(XX):PRINT TAB(14) RI
    GHT$(F$,LEN(F$)-1) ",";                   :rem 61
742 F$=STR$(YY):PRINT RIGHT$(F$,LEN(F$)-1) LEFT$(B
    L$,22-POS(0));:RETURN                    :rem 131
750 POKE 214,19:PRINT:PRINT BL$:POKE 214,19:PRINT:
    RETURN                                   :rem 175
760 POKE 214,19:PRINT::PRINT " PEN:" CHR$(48+DP) "
     ORIGIN:":GOSUB 740                      :rem 58
```

```
762 PRINT "POSITION:":GOTO 110                    :rem 36
770 GOSUB 750:PRINT TAB(9) "FILENAME? ";:F$=""
                                                  :rem 6
772 K=FRE(0):WAIT 198,15:GET K$:IF K$<" " OR K$>"Z
    " OR LEN(F$)=12 GOTO 776                      :rem 14
774 F$=F$+K$:PRINT K$;:GOTO 772                   :rem 209
776 IF K$=CHR$(20) AND F$>"" THEN PRINT "{LEFT}
    {LEFT}";:F$=LEFT$(F$,LEN(F$)-1):GOTO 772
                                                  :rem 233
778 IF K$<>CHR$(13) GOTO 772                      :rem 173
779 RETURN                                        :rem 135
780 GOSUB 750:PRINT TAB(15) "CONFIRM?"            :rem 161
785 WAIT 198,15:GET K$:ON (K$="Y" OR K$="N")+2 GOT
    O 760,785                                     :rem 233
790 PF=1:F$="OUT OF RANGE"                        :rem 195
795 GOSUB 750:PRINT TAB(9) "ERROR: " F$:WAIT 198,1
    5:GET K$:GOSUB 760:GOTO 320                   :rem 230
800 POKE 648,192:PRINT "{CLR}":POKE 648,4:PRINT "
    {CLR}":POKE 53280,0:POKE 53281,0              :rem 103
805 PRINT " [8]SHAPEDIT":PRINT " BY CRAIG CHAMBERL
    AIN":POKE 648,192:REM V1.0                    :rem 83
810 MX=99:DIM U(11),CN$(19),JX(15),JY(15),MX(7),MY
    (7),SL%(MX):N=146                             :rem 169
815 FOR K=0 TO 11:READ U(K):NEXT:FOR CN=0 TO 19:RE
    AD CN$(CN):GOSUB 190:NEXT                     :rem 232
820 FOR K=5 TO 14:READ JX(K),JY(K):NEXT:FOR K=0 TO
     7:READ MX(K),MY(K):NEXT                      :rem 239
825 FOR K=49152 TO 49901:READ N:POKE K,N:NEXT:DEF
    {SPACE}FNDP(N)=PEEK(N)+256*PEEK(N+1)          :rem 235
830 DEF FNL(N)=N-256*INT(N/256):DEF FNH(N)=INT(N/2
    56):DP=1:CL=0:MD=7:PF=1                       :rem 196
835 BL$="{39 SPACES}":GOSUB 760                   :rem 42
840 POKE 214,21:PRINT:PRINT TAB(14) CL:PRINT TAB(1
    4) MD:PRINT TAB(14) 0;                        :rem 63
845 BA=FNDP(49)+500:MT=FNDP(51)-500:GOSUB 730:POKE
     56334,PEEK(56334)AND254                      :rem 151
850 POKE 1,PEEK(1)AND251:S=53248:D=S:L=2048:GOSUB
    {SPACE}620:POKE 1,PEEK(1)OR4                  :rem 62
855 POKE 56334,PEEK(56334)OR1:POKE 657,128:POKE 50
    168,254:POKE 51192,254                        :rem 135
860 POKE 53287,1:POKE 53271,0:POKE 53277,0:POKE SX
    ,233:POKE SY,194:SYS 49888                    :rem 204
865 GRAPHICS 7:SYS 49152:POKE 51578,160:GOTO 330
                                                  :rem 168
900 DATA 1,9,19,28,34,63,191,127,31,223,159,95
                                                  :rem 226
910 DATA LOAD,SAVE,APPEND,QUIT,SHAPE,CLS,MODE,S/F/
    A,EDIT,SETPEN                                 :rem 47
920 DATA FILLPEN,ERASE,MOVE,RMEM,RSET,HOME,DRAW,FI
    LL,LFILL,RFILL                                :rem 203
```

```
930 DATA 1,1,1,-1,1,0,0,0,-1,1,-1,-1,-1,0,0,0,0,1,
    0,-1                                    :rem 212
940 DATA 319,159,159,159,159,79,159,79,79,39,79,39
    ,39,19,39,19                             :rem 71
950 DATA 169,127,141,13,220,173,20,3,141,167,192,1
    73,21,3,141,168,192,169,59              :rem 168
951 DATA 141,17,208,32,36,192,169,209,141,169,192,
    169,1,141,26,208,96,169,251             :rem 240
952 DATA 141,18,208,169,106,141,20,3,169,192,141,2
    1,3,169,133,141,254,255,169             :rem 217
953 DATA 200,141,255,255,96,72,138,72,152,72,173,2
    5,208,141,25,208,162,11,202             :rem 211
954 DATA 208,253,162,27,160,4,173,22,208,41,239,23
    4,142,17,208,140,24,208,141             :rem 206
955 DATA 22,208,32,36,192,104,168,104,170,104,64,1
    73,25,208,141,25,208,169,59             :rem 222
956 DATA 141,17,208,169,24,141,24,208,173,254,204,
    74,173,22,208,41,239,144,2              :rem 168
957 DATA 9,16,141,22,208,173,169,192,240,23,141,18
    ,208,169,67,141,20,3,169,192             :rem 22
958 DATA 141,21,3,169,62,141,254,255,169,192,141,2
    55,255,76,0,0,0,169,0,141,26              :rem 6
959 DATA 208,173,167,192,141,20,3,173,168,192,141,
    21,3,169,129,141,13,220,96              :rem 174
960 DATA 169,128,133,251,169,255,133,252,169,0,160
    ,64,136,145,251,208,251,174             :rem 230
961 DATA 254,204,189,32,194,188,40,194,136,136,136
    ,145,251,208,249,96,162,2,181            :rem 88
962 DATA 251,157,110,194,202,16,248,48,18,32,253,1
    74,32,138,173,32,247,183,132             :rem 17
963 DATA 251,133,252,32,241,183,134,253,165,252,13
    3,254,165,251,174,254,204,240            :rem 56
964 DATA 9,188,47,194,10,136,208,252,38,254,24,105
    ,24,141,0,208,165,254,105,0              :rem 215
965 DATA 141,16,208,165,253,224,2,144,7,188,53,194
    ,10,136,208,252,105,50,141               :rem 163
966 DATA 1,208,169,1,141,21,208,96,169,255,141,31,
    194,165,198,240,3,169,255,96             :rem 39
967 DATA 173,0,220,41,15,174,31,194,16,19,162,7,22
    1,61,194,240,5,202,16,248,48              :rem 3
968 DATA 227,32,236,193,176,222,144,43,188,69,194,
    132,254,162,3,221,77,194,240             :rem 39
969 DATA 9,70,254,70,254,202,16,244,48,50,165,254,
    41,3,240,44,170,189,80,194               :rem 182
970 DATA 24,109,31,194,41,7,170,32,236,193,176,28,
    142,31,194,32,227,192,173,0              :rem 217
971 DATA 220,41,16,240,22,165,162,41,7,208,250,173
    ,0,220,41,15,201,15,208,234              :rem 177
972 DATA 173,0,220,41,16,208,142,174,255,204,173,2
    5,205,16,8,41,15,141,25,205              :rem 193
```

```
973 DATA 157,8,205,172,112,194,140,17,205,173,111,
    194,141,16,205,74,138,174,110          :rem 54
974 DATA 194,142,15,205,32,8,201,169,0,141,21,208,
    173,255,204,10,10,10,13,31            :rem 127
975 DATA 194,10,10,10,9,3,96,160,0,189,86,194,16,1
    ,136,24,109,15,205,133,251            :rem 158
976 DATA 152,109,16,205,133,252,189,84,194,24,109,
    17,205,133,253,172,254,204            :rem 179
977 DATA 208,4,165,252,240,7,165,251,217,94,194,17
    6,5,165,253,217,102,194,96            :rem 197
978 DATA 0,128,192,192,192,240,240,255,255,3,3,6,6
    ,12,12,24,24,1,1,1,2,2,3,3            :rem 122
979 DATA 1,1,2,2,3,3,14,6,7,5,13,9,11,10,120,52,39
    ,19,210,193,141,76,7,11,13            :rem 118
980 DATA 14,1,4,255,255,255,0,1,1,1,0,255,255,255,
    64,160,160,160,80,80,40,40            :rem 131
981 DATA 160,160,80,80,40,40,20,20,0,0,0,142,222,1
    94,140,223,194,165,253,56,229          :rem 25
982 DATA 251,170,165,254,229,252,236,222,194,237,2
    23,194,144,35,160,0,174,223           :rem 225
983 DATA 194,240,14,177,251,145,253,200,208,249,23
    0,252,230,254,202,208,242,174          :rem 57
984 DATA 222,194,240,8,177,251,145,253,200,202,208
    ,248,96,173,223,194,168,101           :rem 230
985 DATA 252,133,252,152,24,101,254,133,254,172,22
    2,194,240,9,136,177,251,145           :rem 218
986 DATA 253,192,0,208,247,174,223,194,240,16,198,
    252,198,254,136,177,251,145           :rem 251
987 DATA 253,192,0,208,247,202,208,240,96,0,0,120,
    142,40,3,140,41,3,88,96,169           :rem 207
988 DATA 255,201,127,96                   :rem 127
```

Once you have Shapedit entered, saved, and run, you can begin experimenting with it. To help you familiarize yourself with the program, take time to read through the following explanations, trying out each feature.

EDIT. To begin, press the joystick button while on EDIT. A white cursor appears, indicating you're in the positioning mode. The purpose of this mode is to establish the origin. Move the cursor by pushing the joystick. If you go past a screen boundary, the cursor wraps around to the other side.

Once you've moved the cursor to the place where you want the shape to start, press the button. The cursor disappears, and the border changes to match the color of the current drawing pen. You're now in the editing mode. The cursor reappears as soon as you push the stick in any direction. By pushing the stick in different directions, you can move the

cursor to any of the eight positions around the origin. Press and release the button to actually plot a point. The point changes to the current drawing-pen color, and the cursor again vanishes.

This procedure of pushing the joystick and pressing the button is repeated for each point. The next time you push the stick, the cursor hovers around the most recently plotted point. Several points can be plotted to draw a shape.

An alternative method of drawing is to hold the trigger down while pushing the stick. This allows the drawing of a chain of points and is faster than point-by-point plotting. The program won't let you draw out of bounds while in the editing mode. Instead of wrapping around, the cursor just stops at the screen edge.

To change the drawing pen, press the f7 key. The border and display pen number change to reflect the color of the new pen. Plotting is now done with pen 2. Press f7 again to change to pen 3. The next time you press f7, pen 0 is selected. Pen 0 is used for erasing.

If you make a mistake while plotting and you want to delete the last point, press f1. The screen clears and the shape is redrawn *without* the last point. To fix a mistake earlier in the shape, it's necessary to delete all the subsequent points in order to back up to the mistake.

Each point that's plotted in the editing mode is one instruction in the shape table. The more points in a shape, the more instructions in the corresponding table. This implementation also supports some special instructions, accessible from the menu, for things like line drawing and area filling.

To leave the editing mode and return to the menu, press any key other than f1 or f7. The cursor disappears, and the word *EDIT* is highlighted. Now that you're in the menu, any item can be selected with the joystick.

SETPEN. Move to the item marked SETPEN and press the joystick button. The prompt jumps to the item marked S/F/A, which stands for SETPEN/FILLPEN/APPEND. These three items are not followed by numbers, so they use S/F/A when a number has to be entered.

Push the stick left or right to change the number after S/F/A. SETPEN supports the standard color numbers from 0 to 15. When you've chosen the color you want, press the but-

ton. If you want to change the pen color to yellow (color 7), for instance, keep pushing the joystick until 7 appears, then press the button. The instruction is added to the shape table, and the prompt jumps back to EDIT.

Return to the editing mode by pressing the button when EDIT is highlighted. The program bypasses the positioning mode and goes right to the editing mode, using the same origin as before. Editing continues from the last point. Use f7 to change the current pen to the one which will have the new color, and plot a point. The point appears in the new color and the border color changes to match the current drawing pen.

As with the SETPEN statement, the SETPEN instruction must be used with a drawing pen other than pen 0. The only color associated with pen 0 is the color of the background.

CLS. If you want to change the background color, use the CLS command. Return to the menu, move to CLS, and press the button. The number after CLS is the current background color number, and can be changed by pushing the stick left or right. Press the button when you've chosen the new color. The screen will clear, and the background is set to the new color. As an example, the background color can be changed to red by changing the number after CLS to 2 and pressing the button.

The program does not let you change the background color to 15. That's the color used for text; using that color as the background would make the text disappear. CLS is a command used for editing purposes only, and is not an instruction that can be included in a shape table.

Now that the screen has been cleared, press the button while on EDIT. This time there is a positioning mode, and the cursor appears at the old origin. If you don't want to change the origin, just press the joystick button again. The shape is redrawn, and editing continues from the last point.

The CLS command has another application. Perhaps you're editing a shape, but can't complete it because it would go out of bounds. What you need to do is change the origin to allow more room for the drawing. Move to the CLS command and press the button twice to clear the screen without changing the background color. Now press the button on EDIT. You'll be in the positioning mode, and the cursor will appear at the old origin. Move the cursor to the new origin by

pushing the stick, and then press the button. The shape is redrawn, starting at the new position, and you're back in the editing mode.

Care must be taken when choosing a new origin, because having the origin too close to a screen edge can make the shape drawing go out of bounds. This can happen when you press the button to go from the positioning mode to the editing mode. If this happens, the message ERROR: OUT OF RANGE shows in the top line of the text window, and the program waits until you acknowledge the error by pressing any key. The next time you select EDIT, be sure to use a different origin.

MODE. Sometimes you may find that the screen is simply not big enough to draw a particular shape, and a higher resolution is needed. Use the MODE command to change the current graphics mode. As a demonstration, change the mode from 7 to 5, and go back to EDIT. Like clearing the screen, changing the mode causes a positioning mode. The cursor appears at the same coordinates as before, but it's smaller and displays in a different position on the screen because of the new resolution.

Once you choose the origin and press the button, the shape is drawn. However, since MODE does not clear the screen, the shape is drawn on top of the old one in the lower resolution. In other words, the MODE command lets you see the same shape in different resolutions. Sometimes you may want to clear the screen by using CLS to get rid of old shapes.

Be aware that if you draw a shape in a high-resolution mode, it may not fit on the screen in a low-resolution mode and may cause the OUT OF RANGE error. Also, all points drawn by the same drawing pen must have the same color within the same color square. This is not a concern in modes 6 and 7, because the points are the same size as the color squares. However, it can be a problem in the other modes. Furthermore, the problem may not be detected until you switch to a higher resolution mode.

If you're creating a shape that will be drawn in a high resolution, it's more convenient to do the editing in a low-resolution mode. Just periodically change to a higher resolution to check for color-square conflicts.

When you switch between normal and multicolor modes,

old shapes on the screen will not be displayed correctly. You may want to use CLS to clear the screen.

ERASE. If you've been following the examples, the current shape may be getting rather long and cluttered. To erase the shape altogether, use the ERASE command. When you press the button on ERASE, the message CONFIRM? asks you to verify that you want to erase the entire shape. Since a shape cannot be retrieved once it's been erased, this prompt prevents accidental erasing of a drawing. Press the Y key to erase the shape, or the N key to cancel the command. If you press Y, the screen clears and the shape table becomes empty.

MOVE. Shapes often develop a snakelike appearance because of the restriction that each point must be adjacent to the previous one. To break the chain of points, use the MOVE instruction. This changes the pen position by adding offsets to the current X and Y coordinates. It has the effect of lifting the drawing pen and setting it down at another position on the screen.

Select MOVE. The cursor appears, and you're now in a type of positioning mode. Using the joystick, move the cursor to the new pen position. Press the joystick button to set this position; the next time EDIT is chosen, the cursor hovers around that location. This is called *relative positioning*. The position of the pen after it's moved is dependent on the old position. If the shape is drawn starting at a different origin, the pen still moves to the same place, *relative to the rest of the shape.*

DRAW. With the MOVE instruction, it's possible to use line drawing and area filling. After displacing the cursor by MOVE, choose the DRAW instruction. A line is drawn from the most recently plotted point to the current position, and the DRAW instruction is added to the shape table. This is certainly preferable to plotting individual points in a straight line because each point requires an instruction. Not only would that take more time to do with Shapedit, but the shape table itself would grow tremendously.

FILL, LFILL, and RFILL. Instead of selecting DRAW for line drawing, you can choose FILL, LFILL, or RFILL for area filling. As the line is drawn from the last point to the current position, filling is done to the left and/or right of the line.

FILLPEN. The fill instructions cause the background to be filled in. If you want to fill on top of an area already filled,

you must first use the FILLPEN instruction. This instruction specifies the number of the pen that's to be covered up by the fill. It's comparable to the optional third number in the destination point of a FILL statement.

Let's say that you want to fill the background using pen 1, then fill inside that area using pen 2. The cover-up pen number is reset to zero (for the background) at the beginning of a shape, so the FILLPEN instruction does not have to be used for the first fill. Before the second fill instruction is added to the shape table, however, the instruction FILLPEN 1 should be selected. The following fill instruction covers up points drawn with pen 1, and stops only when any other pen is encountered.

Once the cover-up pen number is changed by FILLPEN, it stays in effect for the rest of the shape unless changed by another FILLPEN instruction.

HOME. Absolute positioning is done by using the HOME instruction. This instruction moves the pen to the upper-left corner of the screen, known as the home position. The previous position of the pen has no effect on the instruction; the pen will always be at 0,0 after the HOME instruction has been selected.

You can absolutely position the pen at any place on the screen by using the HOME and MOVE instructions together. After the pen has been moved to 0,0 by the HOME instruction, use MOVE to move the pen to the desired position. Since the movement will be done relative to a constant position, the pen will always move to the same position, regardless of the shape's origin.

RMEM and RSET (REMEMBER and RESET). This method of moving the pen position is useful when you want to repeatedly move the pen back to the same position. The first time that the pen is at this particular position, choose the RMEM instruction. Although nothing will appear to happen, the computer will remember the current position as a reference point for later. Continue editing the shape. When you want to make the pen jump back to the reference point established by the earlier RMEM, choose the RSET instruction.

Once a reference point has been established by RMEM, the pen can be reset to that position as often as necessary. There is no limit to the number of times that RSET can be used. However, the computer can remember only one ref-

erence point at a time. If RMEM is selected again, the new position will be remembered, and the old one forgotten.

The default reference point is the home position. If RSET is selected and RMEM has not been used earlier in the shape, the pen moves to 0,0.

SHAPE. As you can see, the Shapedit program has many instructions and commands to help you create a detailed, colorful shape. But sometimes a single shape is not enough. An application may need to use several different shapes. This command lets you edit multiple shapes. It should not be confused with the BASIC statement SHAPE which is used to draw shapes.

Assuming that shape 0 has already been defined, here's how you would create a second shape. Press the button on the item marked SHAPE, push the stick to change the current shape number from 0 to 1, and press the button again. Now press the button on EDIT. Because you're editing a new shape, you'll have to establish an origin, so you'll be placed in the positioning mode. Push the stick to move the cursor to the starting position and press the button one more time. You're now set to start defining shape 1.

Changing the shape number does not clear the screen, which is why shape 0 is still displayed, even though you're editing shape 1. If you want to get rid of the image of the old shape, use the CLS command to clear the screen. Sometimes, however, it can be convenient to see one shape while editing another.

You can stop editing shape 1 and return to shape 0 at any time by using the SHAPE command again. Since the shapes are independent, shape 0 still has its own definition. Editing one shape has no effect on the other. Even if you use the ERASE command to erase one shape, the other remains intact.

The SHAPE command lets you create more than two shapes, provided that you create them one at a time. When you first run the Shapedit program, the SHAPE command only lets you edit shape 0. But as soon as the shape table for shape 0 contains at least one instruction, SHAPE allows access to shape 1. Likewise, the SHAPE command does not let you edit shape 2 until shape 1 has been defined.

Shapedit can handle a maximum of 100 shapes, though it's doubtful that you'll ever need that many.

APPEND. Multiple shapes are useful by themselves, but

one other possibility you'll find use for is to combine them to create new shapes. The APPEND command can be used to add one shape to another.

To demonstrate how APPEND works, assume you're editing shape 3 and the shape you want to append is shape 5. Move to the item marked APPEND and press the button. You're prompted for a shape number. Push the joystick left or right, stop when the number 5 appears, and press the button again. The program examines the definition of shape 5 and adds it to the end of shape 3. Shape 3 now contains the instructions for shape 3 plus the instructions for shape 5. The screen is updated to show shape 5 starting where shape 3 ends.

Shape 3 is the shape being edited, so it's the only shape changed. Shape 5 still retains its previous definition. The APPEND command simply added a copy of shape 5 to the end of shape 3; it did not merge the two shapes into one.

SAVE. In order to use shapes in a BASIC program, they have to be stored in a file. Use the SAVE command to write a copy of a shape file to tape or disk.

When you press the joystick button while on SAVE, you're prompted for a filename. Enter a filename, from 1 to 12 characters long, and press RETURN. The text window blanks, the shape file is saved to tape or disk (depending on what DN is set to in line 100), and the screen is restored.

The .SHP extension is automatically added to the filename by the Shapedit program, and should not be entered. To cancel the SAVE command, press the RETURN key without typing in a filename.

If you're using tape, the standard prompt message will not appear on the screen. You'll have to remember to press both the PLAY and RECORD buttons on the Datassette.

If you're using disk, be sure that the filename you selected is not already in use. The program will not report an error if the filename already exists, but the drive light will flash. The program, however, will report the DEVICE NOT PRESENT error if the disk drive is not connected or not turned on.

LOAD. At a later time you may want to revise some shapes in a file or add some new ones. Or you may wish to examine the shapes used in the demonstration programs. Use the LOAD command to retrieve a set of shapes. Like the

SAVE command, the LOAD command needs a filename. After you enter the filename, the program erases the shapes currently in memory and loads the new file. Again, if you're using tape, no prompt message appears. Be sure to press only the PLAY button on the Datassette.

If you're using disk and the requested file does not exist, you'll see the FILE NOT FOUND error.

QUIT. When you're finished editing and have saved the

Shapedit Reference Chart

General Editing

EDIT

Positioning mode	Establish origin
Editing mode	Add new plotting instructions to shape table
f1	Delete one instruction
f7	Change current pen
Any other key	Return to menu

Commands

CLS	Clear screen
MODE	Change mode
ERASE	Erase current shape
SHAPE	Edit another shape
APPEND	Add copy of any shape to end of current shape
SAVE	Store set of shapes as disk or tape file
LOAD	Retrieve set of shapes from disk or tape file
QUIT	Exit program

Special Instructions

(Cannot be selected until origin of shape has been established by choosing EDIT)

SETPEN	Change color of next pen used for drawing
MOVE	Displace pen position
DRAW	Draw line from last plotted point to current position
FILL	Fill from last plotted point to current position
LFILL	Fill to left only
RFILL	Fill to right only
FILLPEN	Specify which pen will be covered up by fill
HOME	Move pen to 0,0
RMEM	Remember current position
RSET	Reset pen to most recently remembered position

shapes, you can exit the program by using the QUIT command. Press the button on QUIT and type Y to end the program. Type N if you accidentally chose the QUIT command.

The next time you run Shapedit, be sure that the BASIC extensions have been installed.

Using Shapes in BASIC Programs

The SHAPE statement makes it easy to draw shapes in a BASIC program. But before a program can use this statement, some preparation is necessary.

First of all you need to have all the lines which make up Program 6-1, SHP.BAS, included in your program. Keep the line numbers as listed in Program 6-1.

Take a look at the previous shape demonstration programs. You'll find that they all start by assigning the variables DN, SA, SX, SY, and SP. These variables are used by the BASIC subroutine at line 56500 which loads the shape file from tape or disk. Every program which uses shape tables must assign these five variables:

```
DN =  8 (disk)
DN =  1 (tape)
SA = 780
SX = 781
SY = 782
SP = 783
```

After these variables have been assigned, the LOAD address for the shape file must be calculated and assigned to the variable LA. The shape file is loaded at the beginning of BASIC free memory. This address can be found by PEEKing location 49 and adding that value to 256 times the value found in location 50.

BASIC uses free memory whenever a variable is assigned for the first time and whenever an array is dimensioned. Therefore, the calculation of the LOAD address should be done only after most of the variables have been assigned and after all arrays have been dimensioned. As a precaution, an extra 1000 bytes should be added to the LOAD address in case there are a few more variables assigned later. Here's the statement you can use to assign LA.

```
LA=PEEK(49)+256*PEEK(50)+1000
```

Next, assign the name of the shape file to the variable F$. The string should consist only of the main part of the filename and should not include the .SHP extension.

F$=*"filename"*

To complete the preparation, call the subroutine at line 56500.

GOSUB 56500

The subroutine loads the shape file into memory and then determines how many shapes have been loaded. The number of the last shape is assigned to the variable NS. For example, if the file contained four shapes, numbered 0 to 3, the variable NS would be assigned the value 3.

The subroutine then dimensions an array named AS, using the value of NS. A loop from 0 to NS is executed to store the address of each shape in the array. Once that's been done, the subroutine ends and returns to the main program.

The program is now ready to use the SHAPE statement. Remember that this statement needs the address of the shape that you want to display. The address can be found by using the corresponding shape number as the subscript of the array AS. The numbers used for the X and Y coordinates depend on where you want to display the shape and are comparable to the origin used in the Shapedit program.

SHAPE AS*(shape number), X coordinate, Y coordinate*

The following two lines select graphics mode 5 and draw shape 0 at position 40,30.

```
GRAPHICS 5
SHAPE AS(0),40,30
```

Any of the shapes in the file can be drawn by changing the subscript of AS. Just make sure that the subscript never exceeds the value of NS or the program will stop with a BAD SUBSCRIPT error.

Another restriction is that a program can load only one shape file. If you attempt to load a second set of shapes, you'll get a REDIM'D ARRAY error, because array AS has already been dimensioned. A shape file merging utility is provided later so that you can combine two or more shape files into one file.

One last restriction is that, if possible, you should avoid assigning string variables once the shapes have been loaded.

Every assignment to a string variable uses a small amount of free memory. After several strings have been assigned, BASIC may use the same free memory that's used to store the shape tables. If it's necessary to assign strings, periodically call the FRE function. This function reorganizes the free memory used by strings so that the shape tables will be left alone.

The first demonstration program showed that you can draw the same shape at different positions on the screen. You can also change the graphics mode to display the shape in different resolutions.

Another technique, demonstrated by the second program, is to use the SETPEN statement before drawing a shape. The same shape can be drawn in different color combinations. To use this technique, the definition of the shape cannot contain any SETPEN instructions. If the shape table does contain SETPEN instructions, they'll override any SETPEN statements executed before SHAPE.

The coordinates in a SHAPE statement do not have to be constant numbers; they can be variables. An interesting effect is created by using the SHAPE statement in a loop. Save Program 6-9, using the filename TOWERS, and the shape file, Program 6-10, using the filename TOWER.SHP.

Remember that BMG.OBJ must be in memory before entering Program 6-9. Also load SHP.BAS (Program 6-1) before typing in Program 6-9.

Program 6-9. TOWERS

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " TOWER GRAPHS":PRINT "
    {SPACE}BY CRAIG CHAMBERLAIN":PRINT      :rem 86
110 DN=8:SA=780:SX=781:SY=782:SP=783       :rem 133
120 F$="TOWER":LA=PEEK(49)+256*PEEK(50)+1000:GOSUB
     56500:REM LOAD SHAPES                 :rem 107
300 GRAPHICS 1:POKE 53280,0:SETPEN 1,1:S=AS(0)
                                           :rem 82
310 FOR X=20 TO 100 STEP 40                :rem 70
320 READ C,H:SETPEN 2,C                    :rem 18
330 FOR Y=130 TO H STEP -1:SHAPE S,X,Y:NEXT Y,X
                                          :rem 137
340 WAIT 198,15:GET K$                    :rem 137
350 END                                   :rem 111
800 DATA 10,50,5,40,14,30                 :rem 181
```

Program 6-10. TOWER.SHP

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49259
Filename: TOWER.SHP

```
49152 :000,100,000,083,075,075,077
49158 :075,075,075,075,075,075,200
49164 :075,091,091,091,091,091,030
49170 :091,091,091,091,091,091,052
49176 :091,091,091,091,091,107,074
49182 :107,107,107,107,107,107,160
49188 :107,107,123,123,123,123,230
49194 :123,123,123,123,123,123,012
49200 :123,123,123,123,123,131,026
49206 :139,139,139,139,139,139,120
49212 :139,075,155,155,155,155,126
49218 :155,155,155,155,155,155,228
49224 :155,155,155,155,155,099,178
49230 :163,171,171,171,171,171,072
49236 :171,171,115,179,187,187,070
49242 :187,187,187,187,187,187,188
49248 :187,187,187,187,187,187,194
49254 :255,013,013,013,013,013,166
```

The syntax for the SHAPE statement also allows that no coordinates need to be specified.

SHAPE ***address of shape***

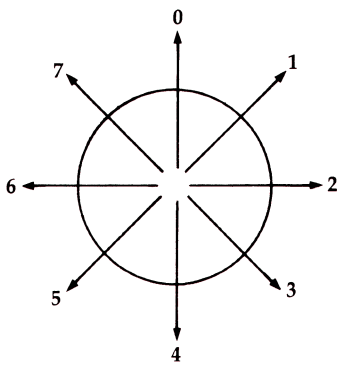
The coordinates are optional. A SHAPE statement without coordinates starts drawing a shape at the position where the most recent shape ended. That's how the figures in the second demonstration were drawn one after another. The position was established by the first shape of each row. The remaining shapes in each row were drawn by using SHAPE with no coordinates.

Drawing shapes with the SHAPE statement is really very simple. Just follow the procedure you've seen. An easy way to get started is to take one of the demonstration programs and modify it to suit your own needs. Change the filename assigned to F$ and write new lines to display the shapes.

Rotation

The plotting process used by shape tables is fairly straightforward. Each instruction moves the pen to one of eight adjacent positions and plots a point. These directions are numbered, starting at 0 for up, and moving clockwise around an imaginary circle. Thus, 1 is up and to the right, 2 is right only, and so on, up to 7, which indicates up and to the left. The directional numbers are illustrated by Figure 6-1. Each instruction consists of a direction number from 0 to 7, and a pen number from 0 to 3.

Figure 6-1. Directional Numbers



What if every time an instruction was processed, a constant number was added to the direction number? How would this affect the display? Well, with a constant of 2, the direction 0 (up) would become 2, which is to the right. The direction 2 (right) would be converted into 4, which is down. And the direction 7 would become direction 9, but since there is no direction 9, it would wrap around to direction 1.

Consider that the eight directions could also be treated like angles, with a measure of 45 degrees between each. Thus, if a constant of 2 was added to each angle as a shape was drawn, it would appear to be rotated by 90 degrees. The shape would still start at the same position on the screen and be the same size; it would just look as if it had been spun around at a right angle.

For added flexibility in shape drawing, the syntax of the SHAPE statement supports an optional rotation number. This

number ranges from 0 to 7 and must always be the last number in the statement.

SHAPE ***address, angle number***
SHAPE ***address, X coordinate, Y coordinate, angle number***

The following chart shows the effects of the different angle numbers.

Number	Angle of Rotation
0	0 degrees (no rotation)
1	45 degrees
2	90 degrees
3	135 degrees
4	180 degrees
5	225 degrees
6	270 degrees
7	315 degrees

Rotation is done clockwise. When no angle number is given, the value 0 is assumed, and the shape is not rotated. Negative angle numbers are not allowed, but comparable positive numbers can always be used. For example, rotating a shape by 270 degrees is the same as rotating it by −90 degrees.

The following demonstration, Program 6-11, takes the Munchkin shape and draws it at every angle. Remember that BMG.OBJ must be in memory and Program 6-1, SHP.BAS, should first be loaded so that it becomes part of "TWIST." Program 6-3, MUNCHKIN.SHP, must also be on the same disk. (If you're using tape, make sure that MUNCHKIN.SHP is saved immediately after Program 6-11. Since you already have Program 6-3 on tape from the earlier example, just resave it, using MLX, to this new area on the tape. Refer to Appendix C for information on using MLX to copy tape files.)

Program 6-11. TWIST

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " MUNCHKIN TWIST":PRINT
    {SPACE}" BY CRAIG CHAMBERLAIN":PRINT  :rem 248
110 DN=8:SA=780:SX=781:SY=782:SP=783      :rem 133
120 F$="MUNCHKIN":LA=PEEK(49)+256*PEEK(50)+1000:GO
    SUB 56500:REM LOAD SHAPES              :rem 55
300 GRAPHICS 1:POKE 53280,0               :rem 220
310 FOR A=0 TO 7                           :rem 3
320 SHAPE AS(0),80+20*COS(A*↑/4),100+30*SIN(A*↑/4)
    ,A                                     :rem 83
```

```
330 NEXT A                                          :rem 22
340 WAIT 198,15:GET K$                             :rem 137
350 END                                            :rem 111
```

When you run this program, you'll notice that the shape deteriorates when it's rotated to one of the diagonal angles. That's why these four angles are not frequently used. But the four main angles (0, 90, 180, and 270 degrees) can be useful, often in some surprising ways. The next program rotates a small shape to draw a continuous border around the screen.

Type in Program 6-12, "BRAID," after loading SHP.BAS so that the latter becomes part of Program 6-12. Then use MLX to enter Program 6-13. Be sure to use the filename BRAID.SHP for this.

Program 6-12. BRAID

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " BRAID":PRINT " BY CRAI
    G CHAMBERLAIN":PRINT                            :rem 98
110 DN=8:SA=780:SX=781:SY=782:SP=783               :rem 133
120 F$="BRAID":LA=PEEK(49)+256*PEEK(50)+1000:GOSUB
     56500:REM LOAD SHAPES                          :rem 60
300 GRAPHICS 2:S=AS(0)                             :rem 245
310 SHAPE S,20,15                                  :rem 120
320 FOR K=1 TO 11:SHAPE S,0:NEXT                    :rem 13
330 FOR K=1 TO 7:SHAPE S,2:NEXT                    :rem 229
340 FOR K=1 TO 12:SHAPE S,4:NEXT                    :rem 20
350 FOR K=1 TO 7:SHAPE S,6:NEXT                    :rem 235
360 SHAPE S,70,50                                  :rem 129
370 WAIT 198,15:GET K$                             :rem 140
380 END                                            :rem 114
```

Program 6-13. BRAID.SHP

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49205
Filename: BRAID.SHP

```
49152 :000,051,000,083,083,083,044
49158 :083,083,083,083,083,067,232
49164 :067,067,067,067,067,115,206
49170 :115,115,115,099,099,083,132
49176 :083,099,099,115,115,115,138
49182 :115,067,067,067,067,067,224
49188 :067,083,083,083,083,083,006
```

```
49194 :Ø83,Ø83,Ø83,Ø99,Ø99,Ø99,Ø76
4920Ø :Ø99,Ø99,Ø99,Ø99,Ø99,255,Ø3Ø
```

Rotation will not work correctly with shapes that contain a MOVE instruction. This instruction does not take into account the rotation angle when it moves the drawing pen. For instance, if a MOVE instruction moves the pen up a certain number of positions, it moves the pen up that many positions even if the shape is drawn upside down. Since MOVE is used with DRAW, FILL, LFILL, and RFILL, these instructions cannot be used either.

RMEM and RSET *do* work with rotation, so DRAW can be used if the pen is moved by RSET instead of MOVE. Unfortunately, this method does not work with any of the fill instructions. These always fill to the left and/or right, no matter how the shape is rotated.

Drawing Pen Indirection

Each shape table instruction contains direction and drawing pen information for one point. You've just seen how experimenting with the direction number can produce some interesting and useful effects. What about the drawing pen number?

Indeed, something special *can* be done with the drawing pen number. You know that the SHAPE statement can quickly draw shapes. It can quickly undraw shapes too, by using the PEN statement, which has a format of:

PEN ***pen number, pen number***

This statement indirects drawing done by the first pen number so that the drawing is actually done by the second pen number. For instance, the statement PEN 1,2 means that all shape table instructions using pen 1 will instead be drawn by pen 2. The term *indirection* is used because of this extra step in determining the pen number. The pen number in the instruction indirectly determines the pen number used for plotting.

Consider what would happen if the three drawing pens were set to 0 before a shape was drawn. Every point in the shape would be drawn with pen 0, the erasing pen. What does this get you? It means that you can draw a shape at any position on the screen, and then by setting the pens to 0 and drawing the shape at the same position, you can erase that

shape without disturbing anything else on the screen. Run Program 6-14 for a demonstration.

Again, insure that the bitmapped graphics BASIC extensions are enabled, and SHP.BAS in memory before typing any of the following listing. You also need to have MUNCHKIN.SHP on the same disk, or if you're using tape, immediately after Program 6-14.

Program 6-14. ANTIMUNCH

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " ANTIMUNCH":PRINT " BY
    {SPACE}CRAIG CHAMBERLAIN":PRINT        :rem 167
110 DN=8:SA=780:SX=781:SY=782:SP=783       :rem 133
120 F$="MUNCHKIN":LA=PEEK(49)+256*PEEK(50)+1000:GO
    SUB 56500:REM LOAD SHAPES               :rem 55
300 GRAPHICS 1:S=AS(0)                     :rem 244
310 FOR X=15 TO 135 STEP 30                 :rem 81
320 FOR Y=10 TO 180 STEP 20                 :rem 77
330 SHAPE S,X,Y                             :rem 99
340 NEXT:NEXT                               :rem 79
400 PEN 1,0:PEN 2,0:PEN 3,0                 :rem 91
410 FOR X=15 TO 135 STEP 30                 :rem 82
420 FOR Y=10 TO 180 STEP 20                 :rem 78
430 SHAPE S,X,Y                            :rem 100
440 NEXT:NEXT                               :rem 80
450 END                                    :rem 112
```

Applications of the PEN statement are not limited to just erasing shapes, however. Remember the demonstration of the rising tower graphs? The towers had to be separated to avoid problems with color squares. By using indirection, the program could have drawn the towers in overlapping pairs. To see this illustrated, check that Program 6-10, "TOWER.SHP," is on the same disk; save it right after Program 6-15 if you have a Datassette. As always, the bitmapped graphics BASIC extensions must be operating, and SHP.BAS should be in memory before you start typing.

Program 6-15. TOWER II

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " TOWERS II":PRINT " BY
    {SPACE}CRAIG CHAMBERLAIN":PRINT        :rem 118
110 DN=8:SA=780:SX=781:SY=782:SP=783       :rem 133
120 F$="TOWER":LA=PEEK(49)+256*PEEK(50)+1000:GOSUB
     56500:REM LOAD SHAPES                 :rem 107
```

```
300 GRAPHICS 1:POKE 53280,0:SETPEN 1,1:S=AS(0)
                                              :rem 82
310 FOR X=20 TO 100 STEP 40                   :rem 70
320 READ C,H:SETPEN 2,C                       :rem 18
330 FOR Y=130 TO H STEP -1:SHAPE S,X,Y:NEXT Y,X
                                             :rem 137
340 PEN 2,3                                   :rem 11
350 FOR X=30 TO 110 STEP 40                   :rem 76
360 READ C,H:SETPEN 3,C                       :rem 23
370 FOR Y=150 TO H STEP -1:SHAPE S,X,Y:NEXT Y,X
                                             :rem 143
380 WAIT 198,15:GET K$                       :rem 141
390 END                                      :rem 115
800 DATA 10,50,5,40,14,30                    :rem 181
810 DATA 6,90,4,80,8,70                      :rem 105
```

Here's another program which makes extensive use of the PEN statement. The program draws a weather map, complete with suns and clouds. Indirection is used to create the effect of flashing thunderbolts and falling rain. Start with SHP.BAS in memory.

Program 6-16. WEATHER

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " MICHIGAN LOWER PENINSU
    LA"                                       :rem 143
105 PRINT " WEATHER MAP DEMONSTRATION":PRINT " BY
    {SPACE}MARK DAVIDS":PRINT                 :rem 143
110 DN=8:SA=780:SX=781:SY=782:SP=783          :rem 133
120 F$="WEATHER":LA=PEEK(49)+256*PEEK(50)+1000:GOS
    UB 56500:REM LOAD SHAPES                  :rem 234
200 GRAPHICS 1:SETPEN 3,9:POKE 53280,0:REM DRAW MI
    CHIGAN LOWER PENINSULA                     :rem 64
205 DRAW 25,195,3 TO 35,169 TO 37,151 TO 39,127 TO
     35,109 TO 33,87 TO 36,69                 :rem 238
210 DRAWTO 44,39 TO 46,54 TO 48,54 TO 50,39 TO 61,
    34 TO 59,28 TO 64,21 TO 67,17             :rem 206
215 DRAWTO 79,23 TO 91,33 TO 99,44 TO 105,50 TO 10
    7,57 TO 102,55 TO 106,62                  :rem 233
220 DRAWTO 108,84 TO 103,101 TO 95,109 TO 88,121 T
    O 86,131 TO 90,139 TO 95,138              :rem 181
225 DRAWTO 97,134 TO 100,127 TO 106,120 TO 112,116
     TO 112,118 TO 119,139                    :rem 144
230 DRAWTO 121,164 TO 118,166 TO 115,174 TO 107,19
    9 TO 65,199 TO 65,195                     :rem 137
235 DRAWTO 25,195:DRAW 104,55 TO 105,55:DRAW 106,5
    6                                         :rem 149
```

```
240 FILL 44,42 TO 43,54:FILL 110,117 TO 110,139:FI
    LL 67,18 TO 66,199                          :rem 77
250 DIM X(6),Y(6),T(6):FOR K=1 TO 6:READ X(K),Y(K)
    :NEXT                                       :rem 250
300 FOR DAY=0 TO 6:MODE 3:SETPEN 1,7:SHAPE AS(5+DA
    Y),36,7:SHAPE AS(12):MODE 1                 :rem 22
310 FOR P=1 TO 6:C=INT(5*RND(0))+1:T(P)=C:ON C GOS
    UB 330,340,350,360,370                      :rem 25
320 NEXT:SETPEN 1,12:GOTO 400                   :rem 164
330 REM CLOUD                                   :rem 241
335 SETPEN 1,1:SHAPE AS(0),X(P),Y(P):RETURN:rem 29
340 REM CLOUD AND SUN                           :rem 187
345 SETPEN 1,12:SETPEN 2,7:SHAPE AS(0),X(P),Y(P):S
    HAPE AS(1),X(P),Y(P):RETURN                 :rem 250
350 REM CLOUD AND RAIN                          :rem 240
355 SETPEN 1,12:SHAPE AS(0),X(P),Y(P):SHAPE AS(2),
    X(P),Y(P):RETURN                            :rem 94
360 REM CLOUD AND LIGHTNING                     :rem 107
365 SETPEN 1,0:SETPEN 2,7:SHAPE AS(0),X(P),Y(P):SH
    APE AS(3),X(P),Y(P):RETURN                  :rem 203
370 REM SUNNY                                   :rem 27
375 SETPEN 1,7:SHAPE AS(4),X(P),Y(P):RETURN:rem 43
400 FOR T=1 TO 32:FOR Q=1 TO 6                  :rem 254
410 IF T(Q)=3 AND T/4=INT(T/4) THEN:PEN 1,3:SHAPE
    {SPACE}AS(2),X(Q),Y(Q)                      :rem 47
420 IF T(Q)=3 AND T/4>INT(T/4) THEN:PEN 1,1:SHAPE
    {SPACE}AS(2),X(Q),Y(Q)                      :rem 47
430 IF T(Q)=4 AND T/5=INT(T/5) THEN:PEN 2,3:SHAPE
    {SPACE}AS(3),X(Q),Y(Q)                      :rem 54
440 IF T(Q)=4 AND T/5>INT(T/5) THEN:PEN 2,2:SHAPE
    {SPACE}AS(3),X(Q),Y(Q)                      :rem 55
450 NEXT:NEXT                                   :rem 81
500 MODE 3:PEN 1,0:SHAPE AS(5+DAY),36,7:SHAPE AS(1
    2):MODE 1                                   :rem 224
510 PEN 1,3:PEN 2,3                             :rem 183
520 FOR Q=1 TO 6:FOR T=0 TO 4:SHAPE AS(T),X(Q),Y(Q
    ):NEXT:NEXT                                 :rem 242
530 NEXT DAY:GOTO 300                           :rem 187
800 DATA 60,44,38,81,88,74,55,120,37,169,95,159
                                                :rem 39
```

Program 6-17. WEATHER.SHP

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 50081
Filename: WEATHER.SHP

```
49152 :012,054,000,083,059,031,238
49158 :067,056,031,067,058,031,060
49164 :123,016,031,075,083,083,167
49170 :083,024,005,002,115,020,011
49176 :031,067,018,031,067,012,250
49182 :031,067,083,083,083,012,133
49188 :053,083,054,031,107,008,116
49194 :036,031,091,018,031,107,100
49200 :115,115,115,022,001,083,243
49206 :010,031,255,026,000,026,146
49212 :017,147,147,179,187,147,116
49218 :147,147,139,179,179,179,012
49224 :179,139,147,147,179,147,242
49230 :147,187,179,179,171,163,080
49236 :255,032,000,006,037,091,249
49242 :099,083,067,006,083,099,015
49248 :083,067,006,083,099,083,005
49254 :067,029,107,099,115,067,074
49260 :004,115,099,115,067,004,000
49266 :115,099,115,067,255,026,023
49272 :000,042,021,179,179,155,184
49278 :179,163,179,163,179,163,128
49284 :147,147,147,163,179,163,054
49290 :179,163,179,163,179,163,140
49296 :179,163,255,090,000,083,146
49302 :083,083,099,115,115,014,147
49308 :001,083,099,099,099,083,108
49314 :067,091,067,123,083,067,148
49320 :115,075,115,115,067,083,226
49326 :083,075,067,075,067,075,104
49332 :067,012,009,001,099,031,143
49338 :149,031,004,089,002,115,064
49344 :067,123,067,123,067,026,153
49350 :061,083,083,083,099,115,210
49356 :115,004,013,099,099,091,113
49362 :099,091,099,012,099,099,197
49368 :000,049,115,099,107,099,173
49374 :107,099,006,073,099,067,161
49380 :099,099,099,083,083,083,006
49386 :083,067,067,067,255,064,069
49392 :000,034,063,006,083,010,180
49398 :031,017,031,008,031,067,175
49404 :115,075,010,031,002,013,242
49410 :099,099,002,005,063,002,016
49416 :083,033,031,083,037,031,050
49422 :010,031,033,031,083,037,239
49428 :031,002,063,002,083,033,234
49434 :031,018,037,031,033,031,207
49440 :115,037,031,008,017,031,015
49446 :067,067,037,031,002,025,011
49452 :091,123,014,029,255,073,117
49458 :000,034,063,002,083,033,009
49464 :031,010,021,031,010,017,176
49470 :031,037,031,115,033,031,084
```

```
49476 :004,013,031,004,009,031,160
49482 :037,031,022,063,002,075,048
49488 :017,031,075,010,031,037,025
49494 :031,008,031,025,031,010,222
49500 :083,021,031,002,005,063,041
49506 :002,083,033,031,018,037,046
49512 :031,033,031,115,037,031,126
49518 :008,017,031,067,067,037,081
49524 :031,002,025,091,123,014,146
49530 :029,255,070,000,022,063,049
49536 :010,083,033,031,083,037,149
49542 :031,004,033,115,018,031,110
49548 :002,037,063,002,083,033,104
49554 :031,083,037,031,010,031,113
49560 :033,031,083,037,031,002,113
49566 :063,002,083,033,031,018,132
49572 :031,013,115,008,031,067,173
49578 :029,031,014,031,002,063,084
49584 :006,083,010,031,017,031,098
49590 :008,031,067,115,075,010,232
49596 :031,002,013,099,099,002,178
49602 :005,255,131,000,063,002,138
49608 :083,033,031,083,037,031,242
49614 :006,009,031,006,013,031,046
49620 :033,031,083,037,031,008,179
49626 :017,031,107,018,013,063,211
49632 :002,083,033,031,018,031,166
49638 :013,115,008,031,067,029,237
49644 :031,014,031,002,063,002,123
49650 :083,063,033,031,010,031,237
49656 :006,013,031,008,021,031,102
```



```
49662 :083,006,009,031,008,017,152
49668 :031,107,029,031,018,063,027
49674 :002,083,033,031,018,037,214
49680 :031,033,031,115,037,031,038
49686 :008,017,031,067,067,037,249
49692 :031,002,025,091,123,014,058
49698 :029,063,002,083,033,031,019
49704 :018,031,013,115,008,031,000
49710 :067,029,031,014,031,002,220
49716 :063,006,083,010,031,017,006
49722 :031,008,031,067,115,075,129
49728 :010,031,002,013,099,099,062
49734 :002,005,255,097,000,010,183
49740 :063,010,083,033,031,083,123
49746 :037,031,004,033,115,018,064
49752 :031,002,037,063,002,083,050
49758 :033,031,083,037,031,017,070
49764 :083,083,009,083,037,031,170
49770 :083,033,031,002,037,063,099
49776 :002,083,033,031,083,037,125
49782 :031,010,031,033,031,083,081
49788 :037,031,002,063,002,083,086
49794 :033,031,014,031,037,031,051
49800 :083,008,115,025,031,010,152
49806 :083,099,004,005,031,115,223
49812 :091,010,005,063,006,083,150
49818 :010,031,017,031,008,031,026
49824 :067,115,075,010,031,002,204
49830 :013,099,099,002,005,255,127
49836 :060,000,034,063,002,083,158
49842 :033,031,018,031,013,115,163
```

```
49848 :008,031,067,029,031,018,112
49854 :063,002,083,033,031,014,160
49860 :031,037,031,083,008,115,245
49866 :025,031,010,083,099,004,198
49872 :005,031,115,091,010,005,209
49878 :063,002,083,018,031,008,163
49884 :067,025,031,004,031,018,140
49890 :031,004,099,021,031,010,166
49896 :005,255,104,000,010,063,157
49902 :006,083,010,031,017,031,160
49908 :008,031,067,115,075,010,038
49914 :031,002,013,099,099,002,240
49920 :005,063,002,083,017,031,201
49926 :006,009,031,083,107,083,069
49932 :083,029,031,083,017,031,030
49938 :012,001,099,021,031,001,183
49944 :083,083,010,005,063,010,022
49950 :083,033,031,083,037,031,072
49956 :004,033,115,018,031,002,239
49962 :037,063,002,083,033,031,035
49968 :083,037,031,010,031,033,017
49974 :031,083,037,031,002,063,045
49980 :002,083,033,031,014,031,254
49986 :037,031,083,008,115,025,109
49992 :031,010,083,099,004,005,048
49998 :031,115,091,010,005,255,073
50004 :076,000,063,002,083,063,115
50010 :033,031,010,031,006,013,214
50016 :031,008,021,031,083,006,020
50022 :009,031,008,017,031,107,049
50028 :029,031,018,063,002,083,078
```



```
50034 :017,031,006,009,031,083,035
50040 :107,083,083,029,031,083,024
50046 :017,031,012,001,099,021,051
50052 :031,001,083,083,010,005,089
50058 :063,010,083,017,031,115,201
50064 :067,067,115,099,006,005,247
50070 :083,021,031,017,083,067,196
50076 :067,083,099,002,029,255,179
```

Program 6-18. CHSETA.SHP

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 50447
Filename: CHSETA.SHP

```
49152 :058,003,000,063,022,255,145
49158 :013,000,006,083,001,067,176
49164 :017,031,083,021,031,005,200
49170 :099,006,255,018,000,006,146
49176 :009,067,067,067,083,099,160
49182 :099,002,083,067,067,083,175
49188 :099,099,002,021,255,025,025
49194 :000,067,026,031,107,033,050
49200 :031,115,037,031,012,025,043
49206 :063,067,037,031,083,033,112
49212 :031,191,115,026,031,002,200
49218 :029,255,022,000,002,075,193
49224 :014,031,067,083,123,008,142
```

```
49230 :031,067,115,075,014,031,155
49236 :115,123,115,037,099,083,144
49242 :010,255,022,000,002,083,206
49248 :018,033,031,115,012,029,078
49254 :031,009,067,067,083,099,202
49260 :006,029,083,067,083,099,219
49266 :002,255,031,000,014,083,243
49272 :115,115,067,115,067,083,170
49278 :067,083,067,067,115,075,088
49284 :083,083,099,083,107,115,190
49290 :099,099,083,083,083,107,180
49296 :115,091,083,002,255,011,189
49302 :000,010,009,067,067,067,114
49308 :083,099,099,006,021,255,207
49314 :017,000,010,083,123,017,156
49320 :031,075,083,083,107,115,150
49326 :029,031,075,099,083,002,237
49332 :255,019,000,002,083,083,110
49338 :083,033,031,115,115,091,142
49344 :006,001,099,021,031,115,209
49350 :115,010,005,255,023,000,094
49356 :017,083,026,031,009,115,229
49362 :115,099,008,031,067,115,133
49368 :021,099,083,067,010,031,015
49374 :099,083,006,005,255,015,173
49380 :000,010,075,025,031,083,196
49386 :029,031,002,009,083,016,148
49392 :031,022,021,255,009,000,066
49398 :002,091,083,067,067,083,127
49404 :099,006,255,008,000,002,110
49410 :017,083,018,031,002,021,174
```

```
49416 :255,007,000,006,083,067,170
49422 :083,099,006,255,013,000,214
49428 :002,083,022,000,033,031,191
49434 :115,012,029,031,022,005,240
49440 :255,024,000,006,075,025,161
49446 :031,010,031,037,031,008,186
49452 :031,123,017,031,014,083,087
49458 :021,031,004,009,107,075,041
49464 :010,021,255,017,000,002,105
49470 :083,018,031,004,001,115,058
49476 :025,031,107,075,083,029,162
49482 :031,010,005,255,025,000,144
49488 :063,002,083,018,031,191,212
49494 :010,067,010,017,031,083,048
49500 :008,021,031,010,025,115,046
49506 :008,031,099,115,022,029,146
49512 :255,021,000,063,002,075,008
49518 :083,099,010,031,075,115,011
49524 :067,115,067,115,075,083,126
49530 :075,016,031,022,037,255,046
49536 :020,000,063,002,075,018,050
49542 :031,107,033,031,115,008,203
49548 :021,031,083,006,009,031,065
49554 :029,031,010,255,025,000,240
49560 :063,002,075,083,099,010,228
49566 :031,017,031,091,099,000,171
49572 :009,115,008,031,009,031,111
49578 :018,031,012,099,018,029,121
49584 :255,025,000,063,002,075,084
49590 :017,031,075,010,031,008,098
49596 :099,029,031,031,010,031,163
```

```
49602 :017,031,004,031,010,099,130
49608 :099,002,005,255,020,000,069
49614 :063,063,006,083,014,025,204
49620 :031,067,012,029,031,091,217
49626 :010,033,115,012,031,022,185
49632 :037,255,029,000,063,002,098
49638 :075,067,005,091,033,031,020
49644 :010,031,037,031,115,115,063
49650 :006,001,083,067,123,075,085
49656 :012,115,002,005,083,083,036
49662 :010,021,255,022,000,063,113
49668 :006,083,083,083,067,083,153
49674 :025,031,008,031,099,115,063
49680 :091,014,031,067,099,099,161
49686 :002,013,255,014,000,063,113
49692 :006,083,083,067,115,001,127
49698 :067,083,067,115,010,029,149
49704 :255,016,000,063,006,083,207
49710 :107,083,075,067,115,001,238
49716 :067,083,067,115,010,029,167
49722 :255,018,000,063,014,083,235
49728 :008,017,031,010,017,031,178
49734 :083,008,021,031,010,021,244
49740 :031,002,255,012,000,063,183
49746 :002,075,018,031,009,067,028
49752 :016,031,022,029,255,019,204
49758 :000,063,002,083,010,017,013
49764 :031,012,002,017,031,083,020
49770 :010,021,031,008,021,031,228
49776 :014,255,023,000,063,010,221
49782 :083,083,001,067,006,009,111
```



```
49788 :031,115,004,013,031,006,068
49794 :009,067,008,031,099,115,203
49800 :022,029,255,024,000,022,232
49806 :083,012,031,033,031,010,086
49812 :031,013,031,115,099,006,187
49818 :031,009,031,016,001,099,085
49824 :021,031,022,005,255,028,010
49830 :000,063,002,083,017,031,106
49836 :006,009,031,083,107,083,235
49842 :083,029,031,083,017,031,196
49848 :012,001,099,021,031,001,093
49854 :083,083,010,005,255,024,138
49860 :000,063,002,083,033,031,152
49866 :014,031,037,031,012,031,102
49872 :075,017,031,091,083,075,068
49878 :083,005,099,099,002,005,251
49884 :255,023,000,063,002,075,126
49890 :017,031,075,014,000,031,138
49896 :091,115,008,001,099,029,063
49902 :031,010,031,067,083,002,206
49908 :005,255,026,000,063,002,083
49914 :083,063,033,031,010,031,245
49920 :006,013,031,008,021,031,110
49926 :083,006,009,031,008,017,160
49932 :031,107,029,031,018,255,227
49938 :018,000,063,002,083,033,217
49944 :031,018,031,013,115,008,240
49950 :031,067,029,031,014,031,233
49956 :002,255,016,000,063,002,118
49962 :083,033,031,018,031,013,251
49968 :115,008,031,067,029,031,073
```

```
49974 :018,255,022,000,063,002,158
49980 :075,017,031,075,014,031,047
49986 :012,099,029,031,014,031,026
49992 :067,115,075,115,115,010,057
49998 :013,255,021,000,063,002,176
50004 :083,033,031,083,037,031,126
50010 :017,083,083,009,083,037,146
50016 :031,083,033,031,002,037,057
50022 :255,020,000,063,002,083,013
50028 :018,031,008,067,025,031,032
50034 :004,031,018,031,004,099,045
50040 :021,031,010,005,255,015,201
50046 :000,063,002,075,083,099,192
50052 :010,031,033,031,083,029,093
50058 :031,002,005,255,026,000,201
50064 :063,002,083,033,031,083,183
50070 :037,031,006,033,083,004,088
50076 :013,031,099,010,017,031,101
50082 :037,115,123,067,091,091,174
50088 :002,255,012,000,063,002,246
50094 :083,033,031,083,037,031,216
50100 :010,031,002,255,026,000,248
50106 :063,002,083,033,031,010,152
50112 :021,031,010,017,031,037,083
50118 :031,115,033,031,004,013,169
50124 :031,004,009,031,037,031,091
50130 :022,255,027,000,063,002,067
50136 :083,033,031,018,037,031,193
50142 :033,031,115,037,031,008,221
50148 :017,031,067,067,037,031,222
50154 :002,025,091,123,014,029,006
50160 :255,021,000,063,002,075,144
50166 :017,031,075,010,031,037,191
50172 :031,008,031,025,031,010,132
50178 :083,021,031,002,005,255,143
50184 :021,000,063,063,002,083,240
50190 :033,031,014,031,091,115,073
50196 :091,115,099,008,031,067,175
50202 :067,029,031,018,255,023,193
50208 :000,063,002,075,017,031,220
50214 :075,010,031,037,031,123,089
50220 :107,115,025,031,010,083,159
50226 :013,031,005,099,002,255,199
50232 :025,000,063,002,083,033,006
50238 :031,014,031,037,031,083,033
50244 :008,115,025,031,010,083,084
50250 :099,004,005,031,115,091,163
50256 :010,005,255,021,000,063,178
50262 :006,083,010,031,017,031,008
50268 :008,031,067,115,075,010,142
50274 :031,002,013,099,099,002,088
50280 :005,255,016,000,063,010,197
50286 :083,033,031,083,037,031,152
50292 :004,033,115,018,031,002,063
50298 :037,255,017,000,063,002,240
50304 :083,033,031,083,037,031,170
50310 :010,031,033,031,083,037,103
50316 :031,002,255,022,000,063,001
50322 :006,001,067,017,031,083,095
50328 :029,031,010,031,025,031,053
50334 :083,021,031,000,013,115,165
50340 :115,014,255,026,000,063,125
```

```
50346 :002,083,033,031,083,037,183
50352 :031,006,009,031,006,013,016
50358 :031,033,031,083,037,031,172
50364 :008,017,031,107,018,013,126
50370 :255,025,000,063,002,083,110
50376 :067,091,033,031,115,099,124
50382 :091,010,031,075,067,115,083
50388 :037,031,083,067,123,115,156
50394 :115,014,013,255,024,000,127
50400 :063,010,083,017,031,115,031
50406 :067,067,115,099,006,005,077
50412 :083,021,031,017,083,067,026
50418 :067,083,099,002,029,255,009
50424 :018,000,063,002,083,018,176
50430 :033,031,016,031,014,107,230
50436 :008,021,031,091,014,031,200
50442 :002,255,108,161,236,100,104
```

Because the graphics statements may not be positioned immediately after the keyword THEN, a colon must be placed between the THEN and the graphics statement. This was necessary in the weather map program. Line 410 is one example.

There's one limitation with the PEN statement: It does not affect the FILLPEN value. Shapes which contain a fill instruction may not work properly with indirection.

When you've finished using indirection and you want to return to normal shape drawing, the values set by the PEN statement should be restored. Drawing pen indirection is canceled whenever the GRAPHICS or MODE statement is executed.

The PEN statement works only with shape table drawing, and has no effect on the other graphics statements like DRAW and FILL.

Mixing Text with Bitmapped Graphics

The normal method of displaying information on the screen is to use text. Letters, digits, and other symbols are combined to form words and numbers.

An alternative means of displaying information is to use graphics. Characteristics like color, size, shape, and position can all be used to express information, without the viewer having to know a specific language. Graphics is a method of communication that should not be overlooked.

You may have noticed that the weather map demonstration used text along with graphics. The program used letters to display the day of the week. By themselves, the methods of text and graphics each have their own advantages. When used together, they open up an almost endless number of new possibilities.

To make it easy to mix text with bitmapped graphics, we've provided a shape file containing definitions for the most commonly used characters (Program 6-18). The shapes include letters, digits, and punctuation symbols. Save this file using the filename CHSETA.SHP.

Also add the following lines to the SHP.BAS file:

```
56600 SHAPE AS(1)-3,X,Y:IF S$="" THEN RETURN
56610 FOR I=1 TO LEN(S$):SHAPE AS(ASC(MID$(S$,I))-
      32):NEXT:RETURN
```

These revised SHP.BAS lines (and the remainder of SHP.BAS, of course) must be included in all programs which use text on the bitmap screen.

You're now ready to use text in a program. Printing text on the bitmap screen is easy when you follow three simple steps.

1. Assign the string that you want to print on the screen to the variable S$.

```
300 GRAPHICS 2
310 S$="HAPPY BIRTHDAY"
```

2. Choose the position of the first character in the string, and assign the coordinates to the variables X and Y.

```
320 X=4:Y=8
```

3. Finally, call the subroutine at line 56600.

```
330 GOSUB 56600
```

This subroutine sequentially draws the shapes that correspond to the characters in the string. If the string contains characters not in the shape file, or if the writing goes out of range, the program stops with an error.

Changing the resolution changes the size of the letters. The following program writes messages in four different resolutions. In order to type in and run this demonstration, you need to have the BASIC extensions installed, the CHSETA.SHP file on the same disk or saved on tape immediately after Program 6-19, and the revised SHP.BAS in memory.

Program 6-19. CHDEMO1

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " CHDEMO1":PRINT " BY CR
    AIG CHAMBERLAIN":PRINT                    :rem 225
110 DN=8:SA=780:SX=781:SY=782:SP=783          :rem 133
120 F$="CHSETA":LA=PEEK(49)+256*PEEK(50)+1000:GOSU
    B 56500:REM LOAD SHAPES                   :rem 146
300 GRAPHICS 0:POKE 53280,0                   :rem 219
310 S$="MODE ZERO":X=14:Y=10:GOSUB 56600       :rem 17
320 MODE 2:S$="MODE TWO":X=6:Y=12:GOSUB 56600
                                              :rem 48
```

```
330 MODE 4:S$="MODE FOUR":X=2:Y=13:GOSUB 56600
                                          :rem 114
340 MODE 6:S$="MODE":X=0:GOSUB 56600       :rem 3
350 S$="SIX":Y=20:GOSUB 56600             :rem 113
360 WAIT 198,15:GET K$                    :rem 139
370 END                                   :rem 113
```

The characters are drawn in pen 1. To write text in a different drawing pen, use the PEN statement.

This next program uses multicolor mode to draw the same message in two different pens. The second message is drawn one position down and one position to the right to create a shadowed lettering effect. This example requires the shape file (CHSETA.SHP) on disk or tape; you also need the revised SHP.BAS in memory before beginning to type it in.

Program 6-20. CHDEMO2

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " CHDEMO2":PRINT " BY CR
    AIG CHAMBERLAIN":PRINT                :rem 226
110 DN=8:SA=780:SX=781:SY=782:SP=783      :rem 133
120 F$="CHSETA":LA=PEEK(49)+256*PEEK(50)+1000:GOSU
    B 56500:REM LOAD SHAPES               :rem 146
300 GRAPHICS 3:POKE 53280,0:S$="COMMODORE 64":X=30
                                           :rem 81
310 FOR Y=30 TO 70 STEP 20                 :rem 28
320 READ C:SETPEN 1,C:GOSUB 56600:NEXT    :rem 209
330 X=31:PEN 1,2                           :rem 59
340 FOR Y=31 TO 71 STEP 20                 :rem 33
350 READ C:SETPEN 2,C:GOSUB 56600:NEXT    :rem 213
360 WAIT 198,15:GET K$                    :rem 139
370 END                                   :rem 113
800 DATA 2,6,5,10,14,13                    :rem 85
```

The character shapes can be used with other shapes in the same program. When using Shapedit, start by loading the CHSETA.SHP file into memory. Then add the new shapes to the file.

Suggested Applications

Redefined (custom) characters. A popular way to draw shapes is to use redefined characters, but this method has some drawbacks. The major restriction is that character definitions must fit within an 8 × 8 grid. To create a shape larger than 8 × 8 points, several characters must be used. Another

limitation concerns positioning. A character must be placed at one of the 40 × 25 positions on the screen.

Bitmapped graphics shapes have many advantages over shapes drawn by redefined characters. There's no size restriction, so a shape drawn by the SHAPE statement can be any size. This gives you the freedom to do some interesting things. For instance, the letters in the character-set shape file were designed for proportional printing. Some letters, such as *W*, are wider than other letters, such as *I*. If you added lowercase letters to the file, true descenders could be supported. This means that the tails on letters (*g*, for example) could extend below the other characters.

Shapes drawn with the SHAPE statement are also more convenient to use than those created with redefined characters. The Shapedit program simplifies the task of defining a shape. Storing shapes in memory is less complicated because you don't have to reserve memory for a character set. Finally, an entire shape can be drawn by just one statement.

Sprites. Another type of graphics supported by the VIC-II chip is sprite graphics. Sprites are used mainly for animation. A sprite image can be moved to any position on the screen by just a few POKE statements. This is one application where it's just not practical to use bitmapped graphics shapes. Moving a shape requires erasing it and redrawing it at the new position. Although the plotting done by the SHAPE statement is fast, it's not fast enough to do this effectively.

Animation, however, isn't restricted just to motion across the screen. It can also mean "changing in shape." The demonstration programs showed how the SHAPE statement can be used to make flowers grow or thunderbolts flash. And sprites do have several limitations. They have to be defined within a 24 × 21 grid, they can have only three colors (maximum), and there can be only eight of them on the screen at one time. In applications where motion is not required, bitmapped graphics shapes may be worth consideration.

Another possibility is to combine sprites with bitmapped graphics. Remember that sprites are completely independent of anything else displayed on the screen. Using sprites and bitmapped graphics on the same screen might allow you to do things which couldn't be done using sprites or bitmapped graphics alone.

General applications. Shapes have distinguishing characteristics, like size, color, and position. They can be full (filled in) or empty (outlined). Their edges can be smooth or pointed. If you have several shapes, they can be used in counting exercises. Shapes have hundreds of applications in educational software. Concepts such as larger/smaller, above/below, inside/outside, open/closed, rightside up/upside down, and alike/different can easily be presented.

Games offer many opportunities to use shapes as well. Board games often use several objects which could be drawn with bitmapped graphics shapes. Another type of game is the role-playing adventure game. Graphics statements like DRAW and FILL could be used to draw detailed background scenes. The SHAPE statement could then be used to draw the objects being carried by each player.

Cave dwellers drew pictures on cave walls long before words were ever written. Pictures and shapes have a power of communication that should not be ignored. A lot of creative thought and time was invested in the design and implementation of the shape table utility. Now it's your turn.

The Technical Side

A bitmap requires 8K of RAM, which is a significant amount of memory. In order to retain as much free memory for BASIC as possible, the bitmapped graphics routines were designed to use the 8K of RAM underneath the Kernal ROM. This area of memory behaves a little differently from normal RAM because ROM and RAM locations exist at the same addresses. If a location in this area is POKEd, the value will be stored in RAM, but if the location is PEEKed, the value will come from ROM. The nature of this memory means that it's not suitable for use by BASIC. It's ideal, however, for things like bitmaps.

Unfortunately, a few complications may arise when this area of memory is used. Since the PEEK function cannot be used to look at RAM locations, there's no way for a program to examine the memory used by the bitmap. Using sprites is a little different, because placing the bitmap under the Kernal requires switching the VIC-II chip from the normal bank 0 to bank 3. Also, the RAM under the Kernal may be needed by other utilities, such as disk drive handlers.

As an answer to the first problem, the LOOK function is available.

LOOK(*X coordinate, Y coordinate*)

This function returns the pen value found at the specified position. For example, if position 5,6 has been plotted with pen 1, the statement PRINT LOOK(5,6) prints the number 1. If the position has not been plotted and displays only the background, the function will return the number 0.

Here's a program that randomly draws a maze. The program uses the LOOK function to decide where to draw the paths. Make sure that the bitmapped graphics extensions are enabled before typing in the following program.

Program 6-21. MAZE

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 REM MAZE DRAWING DEMONSTRATION          :rem 149
110 REM BY CRAIG CHAMBERLAIN                :rem 141
200 GRAPHICS 6:POKE 53280,0:SETPEN 1,15:DRAW 2,22
    {SPACE}TO 2,2:LFILLTO 36,22 TO 36,2     :rem 188
210 DIM DX(3),DY(3):FOR K=0 TO 3:READ DX(K),DY(K):
    NEXT                                    :rem 245
220 T=2:H=17:V=10:X=3:DEF FNRY(Y)=3+INT(RND(1)*V)*
    T:Y=FNRY(0):DRAW X,Y,0                   :rem 12
300 FOR K=1 TO H*V-1                        :rem 252
310 IF LOOK(X+T,Y) GOTO 370                 :rem 216
320 IF LOOK(X,Y+T) GOTO 370                 :rem 217
330 IF LOOK(X-T,Y) GOTO 370                 :rem 220
340 IF LOOK(X,Y-T) GOTO 370                 :rem 221
350 X=3+INT(RND(1)*H)*T:Y=FNRY(0):IF LOOK(X,Y) GOT
    O 350                                   :rem 170
360 GOTO 310                                :rem 102
370 D=RND(1)*4:IF LOOK(X+DX(D),Y+DY(D))=0 GOTO 370
                                              :rem 4
380 DRAW X,Y:X=X+DX(D):Y=Y+DY(D):DRAWTO X,Y:NEXT:D
    RAW 2,FNRY(0):DRAW 36,FNRY(0)            :rem 83
390 WAIT 198,15:GET K$:END                  :rem 159
800 DATA 2,0,0,2,-2,0,0,-2                  :rem 200
```

It's important to note that the LOOK function returns only the pen number of the designated point and ignores the associated color. In Program 6-22, even though the barriers are drawn in different colors, they're all detected by the LOOK function as points drawn with pen 1. As with Program 6-21, all you need to do before typing it in and running it is to insure that the BASIC extensions are operating.

Program 6-22. PINBALL

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
10 GRAPHICS 6:POKE 53280,0:SETPEN 1,15:DRAW 0,0 TO
    39,0 TO 39,24 TO 0,24 TO 0,0            :rem 15
20 FOR K=1 TO 30:X=RND(1)*38+1:Y=RND(1)*23+1:SETPE
   N 1,9*RND(1)+2:DRAW X,Y:NEXT            :rem 153
30 MODE 4:DX=1:DY=1:SETPEN 1,1              :rem 220
40 X=INT(RND(1)*80):Y=INT(RND(1)*50):IF LOOK(X,Y)
   {SPACE}GOTO 40                           :rem 247
50 DRAW X,Y,0:X=X+DX:Y=Y+DY:DRAW X,Y,1       :rem 77
60 IF LOOK(X,Y+DY) THEN DY=-DY              :rem 243
70 IF LOOK(X+DX,Y) THEN DX=-DX              :rem 241
80 IF LOOK(X+DX,Y+DY) THEN DX=-DX:DY=-DY    :rem 152
90 GOTO 50                                    :rem 7
```

In modes 0, 2, 4, and 6, the LOOK function returns only the values 0 or 1, because these modes only support an erasing pen and one drawing pen. The multicolor modes support three drawing pens, so in modes 1, 3, 5, and 7, the LOOK function can return values from 0 to 3. In any mode, the ILLEGAL QUANTITY error will occur if the position is out of range.

Be aware that the LOOK function changes the current pen position, which can affect line drawing and area filling. For instance, let's say that you plot a point at 3,4 and then look at position 8,7. If the statement DRAWTO 5,6 were now executed, the line would be drawn from the position 8,7, rather than from the last plotted point at 3,4.

Switching the VIC-II chip to bank 3 affects the sprite pointers and the available definition blocks. The address of the sprite pointers can be calculated with the following formula:

Sprite pointer address = Screen memory address + 1016

The screen normally starts at address 1024. Adding 1024 and 1016 gives 2040, the first of the eight sprite pointers. When the GRAPHICS statement turns on the bitmapped graphics mode, the screen is moved to start at 50176. The sum of 50176 plus 1016 is 51192, so locations 51192 through 51199 act as the sprite pointers when bitmapped graphics is used. (See Chapters 12–16 for more information on sprites.)

Bank 3 does not contain much extra memory for sprite definition blocks. Blocks 0–15 are available in normal RAM, and blocks 253 and 254 are available in RAM under the Kernal (you can POKE blocks 253 and 254 but you cannot

PEEK them). These are the only blocks available from BASIC when using the bitmapped graphics routines.

To calculate the beginning address of a definition block in bank 3, use this formula:

Block address = 49152 + 64 * Block number

For most applications, 18 definition blocks should be sufficient. If you need more, you'll have to relocate the bitmap. It's also necessary to move the bitmap when the RAM under the Kernal is needed by another utility. Relocating the bitmap is as easy as four POKE statements. The difficulty lies in choosing where you want to relocate the bitmap.

The memory on the Commodore 64 is divided into four banks of 16K each. A bitmap requires 8K, or half a bank, so it would seem that there is a total of eight different places in memory where a bitmap can be placed. Actually, three of these places are not available, either because they conflict with character-set images, or they are needed by the system.

Besides the 8K for the bitmap, 1K for screen memory must also be allotted. Screen memory is used to provide color information in the bitmap mode. Since the VIC-II chip can access only one bank at a time, the screen memory must be placed somewhere in the other half of the bank that contains the bitmap. Depending on which bank is being used, there may be from one to eight different places to put the screen memory.

Bank	Address	Supports Bitmap?	Possible Locations for Screen Memory
0	0	No	
0	8192	Yes	1024, 2048, 3072
1	16384	Yes	24576, 25600, 26624, 27648, 28672, 29696, 30720, 31744
1	24576	Yes	16384, 17408, 18432, 19456, 20480, 21504, 22528, 23552
2	32768	No	
2	40960	Yes	32768, 33792, 34816, 35840
3	49152	No	
3	57344	Yes	49152, 50176

In bank 3, the bitmap may be placed only in the second half of the bank, starting at location 57344. The screen can be placed either at address 49152 or at address 50176. The default arrangement of the bitmapped graphics routines is to put

the bitmap at 57344 and screen memory at 50176. As discussed earlier, this arrangement allows only 18 sprite definition blocks, but it does let you use the full 38K of BASIC free memory.

Bank 2 supports a bitmap only in the second half of the bank, at location 40960, which is under the BASIC ROM. There are four possible places to put the screen memory. All four are in the area of memory used by BASIC, so using bank 2 means that the amount of free memory will be reduced. The maximum amount of free memory can be obtained by placing screen memory at address 35840, which allows 33K of BASIC free memory. Unfortunately, this arrangement allows only three sprite definition blocks, blocks 253–255. By moving screen memory lower, to 32768, you can also use blocks 16–63.

There are no restrictions when using bank 1 for bitmapped graphics. Either half of the bank can support a bitmap, and a full eight positions are available for placing screen memory. The only problem with using this bank is that it uses a lot of BASIC free memory. A maximum of 21K free can be obtained by starting the bitmap at location 24576 and screen memory at address 23552. Sprite definition blocks 0–111 can be used in this configuration, but it further reduces the amount of free memory available for BASIC.

Since it limits BASIC free memory to only 6K, bank 0 should be used only as a last resort. Screen memory should be placed at location 1024, which is where it normally resides in the text mode. The other two places are right where the BASIC program is stored. Another problem with using bank 0 is that there's hardly any room for sprite definition blocks. To make bank 0 a little more useful, move the start of BASIC memory up to 16384 so that it comes after the bitmap. This allows 24K free. Enter the following line in the immediate mode to move the bottom of BASIC memory up to 16384.

POKE 44,64:POKE 64*256,0:NEW

This line should be entered immediately *after* you turn the computer on, but *before* you load any programs.

Once you've chosen the new locations for the bitmap and the screen memory (keeping in mind the fact that they must be in the same bank), you're all set to begin the relocation. Four POKE statements are all that's required. The first value to POKE specifies which bank is being used.

POKE 52474,3−*n* (where *n* is the bank number, 0–3)

The next POKE selects a mask value for ROM/RAM flipping.

POKE 52475,253 (for bank 3)

or

POKE 52475,254 (for bank 2)

or

POKE 52475,255 (for bank 1 or 0)

The location of the bitmap is determined by the third POKE value.

POKE 52476,*address of bitmap divided by 256*

The last POKE specifies the location of screen memory.

POKE 52477,*address of screen memory divided by 256*

Let's take a look at an example. Assume that you want to relocate the bitmap to bank 2. The bitmap will start at location 40960 and screen memory at address 35840. Here are the four POKE statements that would be executed.

POKE 52474,3−2	or	**POKE 52474,1**
POKE 52475,254		**POKE 52475,254**
POKE 52476,40960/256		**POKE 52476,160**
POKE 52477,35840/256		**POKE 52477,140**

Be sure that these POKE statements are executed *after* the BMG.OBJ file has been loaded by the BMGLOADER program.

The preceding POKE statements relocate the bitmap and screen memory, but they do not inform BASIC of the change. To protect your program and to make sure that it runs correctly, it's a good idea to lower the top of free memory. Determine the lowest address used by the graphics (whether it's the screen memory location or the bitmap), and use that value in the following line.

POKE 56,*address divided by 256*:CLR

If you're relocating to bank 2 and the lowest address used for graphics is that of the screen memory at location 35840, the line would look like this:

POKE 56,140:CLR

It's not necessary to lower the top of memory when you're using bank 0 and you've moved the bottom of BASIC memory up.

NMI interrupts don't work reliably when using a bitmap in bank 3. Since the modem uses NMI interrupts, the bitmap must be relocated to a bank other than 3 if a program is to receive characters from the modem while it's drawing bitmapped graphics.

One last item having to do with relocation is a revised formula for calculating the address of a sprite definition block. The formula has been updated to work for all banks.

Block address = Bank address + 64 * Block number

Bank	Address
0	0
1	16384
2	32768
3	49152

The bitmapped graphics routines use memory from location 51200 to 53247. They also share several zero page locations with BASIC, but do not use locations 2 or 251–254.

These bitmapped graphics extensions to BASIC may not work if other extensions have been installed. The bitmapped graphics routines are accessible by BASIC statements for convenience, but if you want, you can call the routines directly by using POKE and SYS statements. Instead of running the BMGLOADER program, enter the following lines to load the graphics routines.

LOAD "BMG.OBJ",8,1 (disk)
NEW

or

LOAD "BMG.OBJ",1,1 (tape)
NEW

Include this next line at the beginning of every program which uses bitmapped graphics.

SA=780:SX=781:SY=782:SP=783:POKE SP,0

You're now set to use bitmapped graphics. The following is a list of all the graphics statements and their equivalents using POKE and SYS.

POKE and SYS Equivalents

GRAPHICS N	SYS 51369	Clear screen
	POKE SA,N	Set mode number
	SYS 51200	Turn on bitmapped graphics
DRAW X,Y,P	POKE SA,P	Set pen number
	POKE SX,XAND255	X coordinate low byte
	POKE SP,-(X>=256)	X coordinate high byte
	POKE SY,Y	Y coordinate
	SYS 51464	Plot point
	IF PEEK(SP)AND1 THEN	Out of range error
SETPEN A,B	POKE 52488+A,B	Set pen color
CLS	SYS 51369	Clear screen (black)
CLS C	POKE SA,C	Set background color
	SYS 51371	Clear with color
MODE N	POKE SA,N	Set mode number
	SYS 51280	Change mode
TEXT	SYS 51436	Switch to text mode
DRAWTO X,Y	POKE SA,0	Set for line draw
	POKE SX,XAND255	X coordinate low byte
	POKE SP,-(X>=256)	X coordinate high byte
	POKE SY,Y	Y coordinate
	SYS 51831	Perform draw/fill
	IF PEEK(SP)AND1 THEN	Range error
FILLTO X,Y,F	POKE SA,3	Set for full fill
	POKE SX,XAND255	X coordinate low byte
	POKE SP,-(X>=256)	X coordinate high byte
	POKE SY,Y	Y coordinate
	POKE 52492,0	If F=0 (fill over background)
	or	
	POKE 52492,255	If F=1 in modes 0, 2, 4, 6
	or	
	POKE 52492,85	If F=1 in modes 1, 3, 5, 7
	or	
	POKE 52492,170	If F=2
	or	
	POKE 52492,255	If F=3
	SYS 51831	Perform draw/fill
	IF PEEK(SP)AND1 THEN	Range error
RFILLTO X,Y,F	POKE SA,1	Set for right fill
	POKE SX,XAND255	X coordinate low byte
	POKE SP,-(X>=256)	X coordinate high byte
	POKE SY,Y	Y coordinate
	POKE 52492,0	If F=0 (fill over background)

	or	
	POKE 52492,255	If F=1 in modes 0, 2, 4, 6
	or	
	POKE 52492,85	If F=1 in modes 1, 3, 5, 7
	or	
	POKE 52492,170	If F=2
	or	
	POKE 52492,255	If F=3
	SYS 51831	Perform draw/fill
	IF PEEK(SP)AND1 THEN	Range error
LFILLTO X,Y,F	POKE SA,2	Set for left fill
	POKE SX,XAND255	X coordinate low byte
	POKE SP,-(X>=256)	X coordinate high byte
	POKE SY,Y	Y coordinate
	POKE 52492,0	If F=0 (fill over background)
	or	
	POKE 52492,255	If F=1 in modes 0, 2, 4, 6
	or	
	POKE 52492,85	If F=1 in modes 1, 3, 5, 7
	or	
	POKE 52492,170	If F=2
	or	
	POKE 52492,255	If F=3
	SYS 51831	Perform draw/fill
	IF PEEK(SP)AND1 THEN	Range error
SHAPE A,X,Y,R	POKE 52493,A-256*INT(A/256)	Shape address low byte
	POKE 52494,A/256	Shape address high byte
	POKE 52495,XAND255	X coordinate low byte
	POKE 52496,-(X>=256)	X coordinate high byte
	POKE 52497,Y	Y coordinate
	POKE SA,R	Rotation value
	SYS 52166	Draw shape
	IF PEEK(SP)AND1 THEN	Range error
PEN A,B	POKE 52501+A,B	Set pen indirection
LOOK(X,Y)	POKE SX,XAND255	X coordinate low byte
	POKE SP,-(SX>=256)	X coordinate high byte
	POKE SY,Y	Y coordinate
	SYS 51462	Look at position
	IF PEEK(SP)AND1 THEN	Range error
	otherwise	
	Pen number = PEEK(SA)	

All the POKE statements listed for each graphics statement are required, and none should be omitted. For example, even though the pen number is optional in the DRAW statement, it must be specified by POKE SA,*pen number* when using the equivalent POKE and SYS statements.

The only exceptions to this rule have to do with the SHAPE statement. The X and Y coordinates do not have to be set. If you don't POKE the locations for the coordinates, the shape will be drawn starting at the end of the previous shape. Also, if the shape address has been set previously and you want to draw the same shape again, the address does not have to be set a second time.

Some interesting applications are possible if you eliminate the SYS 51369 in the code equivalent to the GRAPHICS statement. Not performing this SYS call means that the screen will not be cleared. This allows you to switch from bitmap mode to text and back to bitmap without losing the bitmap picture. By using relocation, together with the code equivalent to graphics statements, you can draw on two different bitmaps and flip between them. Program 6-23 shows how. (Make sure that the bitmapped graphics BASIC extensions are enabled before typing in this next program.)

Program 6-23. MODERNART

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 GOSUB 610:GRAPHICS 4:POKE 53280,0:SA=780:REM F
    IRST SCREEN                                :rem 62
110 SETPEN 1,5:LFILL 0,40 TO 40,0             :rem 195
120 SETPEN 1,4:DRAW 0,45 TO 60,0:LFILL 0,49 TO 20,
    49 TO 79,10 TO 79,0                        :rem 168
130 SETPEN 1,8:RFILL 79,20 TO 50,49            :rem 74
200 GOSUB 620:REM POINT TO SECOND SCREEN BUT DO NO
    T DISPLAY                                  :rem 246
210 SYS 51369:REM CLEAR SECOND SCREEN          :rem 155
220 SETPEN 1,6:LFILL 0,0 TO 10,0 TO 20,49:SETPEN 1
    ,7:DRAW 40,0 TO 30,49                      :rem 145
230 SETPEN 1,7:DRAW 40,0 TO 30,49:DRAWTO 40,49:LFI
    LLTO 50,0                                  :rem 216
240 SETPEN 1,2:RFILL 79,0 TO 60,0 TO 70,49:rem 123
250 GOSUB 600:REM WAIT FOR KEYPRESS            :rem 93
260 POKE SA,4:SYS 51200:REM NOW DISPLAY SECOND SCR
    EEN                                        :rem 144
270 GOSUB 600                                  :rem 175
300 GOSUB 610:REM FIRST SCREEN                  :rem 16
310 POKE SA,4:SYS 51200:GOSUB 600              :rem 56
```

```
320 GOSUB 620:REM SECOND SCREEN                    :rem 71
330 POKE SA,4:SYS 51200:GOSUB 600                  :rem 58
340 K=FRE(0):GOTO 300                             :rem 131
600 WAIT 198,15:GET K$:RETURN                     :rem 162
610 POKE 52474,0:POKE 52475,253:POKE 52476,224:POK
    E 52477,196:RETURN                            :rem 235
620 POKE 52474,2:POKE 52475,255:POKE 52476,96:POKE
     52477,92:RETURN                              :rem 146
```

Stop this program only when it's displaying the screen with the green, purple, and orange colors, or the bitmap will remain in lower memory.

When using two or more bitmap screens, only one of the screens can be in multicolor mode. Modes 1, 3, 5, and 7 use color memory starting at location 55296. This memory cannot be shared by different screens. Since this is also used by the text mode, you should not switch from text mode to a multicolor mode without first clearing the screen. Do this only with mode 0, 2, 4, or 6.

When using the POKE statement equivalent to SETPEN, the value for A must be in the range 1–3. The value for A in the POKE statement equivalent to PEN must be in the range 0–3.

When using the bitmapped graphics routines without adding statements to BASIC, the screen will not automatically revert back to the text mode when the READY. prompt is printed or an error occurs.

EXTRACT and MERGE

These last two programs are convenient utilities you can use to pull apart and put together various shape files. Both are easy to use and fairly self-explanatory. If you're using tape, make sure that the DN=8 is changed to DN=1 in line 110 of both programs.

"EXTRACT," Program 6-24, is used to extract a group of shapes from a shape file. When you run the program, it asks you for the name of the file from which the shapes will be extracted. Do not include the .SHP extension as part of the filename. The program loads the file and tells you how many shapes the file contains. You can then specify which shapes are to be extracted by entering the numbers for the first and last shapes. Finally, the program requests a filename for saving the extracted shapes. All the shapes in the specified range will

be saved. The original shape file, however, retains *all* the shapes. They are only extracted.

Program 6-24. EXTRACT

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SHAPE FILE EXTRACTION
    {SPACE}UTILITY"                              :rem 221
105 PRINT " BY CRAIG CHAMBERLAIN":PRINT          :rem 69
110 DN=8:SA=780:SX=781:SY=782:SP=783             :rem 133
120 DEF FNH(N)=INT(N/256):DEF FNL(N)=N-256*FNH(N):
    LA=PEEK(49)+256*PEEK(50)+1000                :rem 146
300 F$="":INPUT " LOAD FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 300                            :rem 238
310 PRINT:GOSUB 56500:PRINT " LOADED SHAPES 0 TO"
    {SPACE}NS:PRINT                              :rem 142
400 FS=-1:INPUT " NUMBER OF FIRST SHAPE";FS:IF FS<
    0 OR FS>NS GOTO 400                          :rem 181
410 LS=-1:INPUT " NUMBER OF LAST{2 SPACES}SHAPE";L
    S:IF LS<FS OR LS>NS GOTO 410                 :rem 228
420 PRINT                                        :rem 35
500 F$="":INPUT " SAVE FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 500                            :rem 1
510 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=F$
    +".SHP":GOSUB 59000                          :rem 200
520 A=AS(FS)-3:POKE A,LS-FS:POKE SA,251:POKE 251,F
    NL(A):POKE 252,FNH(A)                        :rem 42
530 B=AS(LS):B=B+PEEK(B-2)+256*PEEK(B-1):POKE SX,F
    NL(B):POKE SY,FNH(B)                         :rem 67
540 PRINT:SYS 65496:IF PEEK(SP)AND1 GOTO 59100
                                                 :rem 139
550 PRINT " SAVED" B-A "BYTES":END               :rem 106
56500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".SHP":GOSUB 59000                      :rem 50
56510 POKE SA,0:POKE SX,LA-256*INT(LA/256):POKE SY
      ,INT(LA/256)                               :rem 76
56520 SYS 65493:IF PEEK(SP)AND1 GOTO 59100       :rem 42
56530 NS=PEEK(LA):DIM AS(NS):LA=LA+1             :rem 107
56540 FOR K=0 TO NS:AS(K)=LA+2:LA=LA+2+PEEK(LA)+25
      6*PEEK(LA+1):NEXT:RETURN                   :rem 120
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                                     :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                                  :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END                    :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                 :rem 36
59120 PRINT ST:END                               :rem 70
```

"MERGE," Program 6-25, is used to combine shape files. You can load a shape file and append other shape files to create one larger file. The program asks for a filename, loads the file, and reports the number of shapes loaded. It then requests the name of the file to be appended. The shapes from that file will be placed after the shapes in the first file. This can be done several times to merge several files together. When you don't want to append any more files, press the RETURN key in response to the filename prompt. The program asks for a filename to save the file just created.

By loading a file and appending nothing, the MERGE program can be used to copy a shape file from one tape to another tape, or one disk to another disk.

Program 6-25. MERGE

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SHAPE FILE MERGE UTILI
    TY"                                          :rem 76
105 PRINT " BY CRAIG CHAMBERLAIN":PRINT          :rem 69
110 DN=8:SA=780:SX=781:SY=782:SP=783:BA=PEEK(49)+2
    56*PEEK(50)+1000                             :rem 27
300 F$="":INPUT " LOAD FILENAME";F$:IF LEN(F$)>12
    {SPACE}GOTO 300                              :rem 98
310 IF F$="" THEN END                            :rem 20
320 LA=BA:GOSUB 630:NA=PEEK(BA):PRINT " LOADED SHA
    PES 0 TO" NA                                 :rem 14
400 BB=PEEK(SX)+256*PEEK(SY)-1:PRINT            :rem 175
410 F$="":INPUT " APPEND FILENAME";F$:IF LEN(F$)>1
    2 GOTO 410                                   :rem 254
420 IF F$="" THEN PRINT:GOTO 500                 :rem 212
430 LA=BB:GOSUB 630:NB=PEEK(BB):PRINT " APPENDED S
    HAPES" NA+1 "TO" NA+NB+1                     :rem 193
440 POKE BB,255:NA=NA+NB+1:IF NA<256 GOTO 400
                                                 :rem 189
450 PRINT:PRINT " ERROR: TOO MANY SHAPES":END
                                                 :rem 241
500 F$="":INPUT " SAVE FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 500                             :rem 1
510 GOSUB 600:POKE SA,251:POKE 251,BA-256*INT(BA/2
    56):POKE 252,INT(BA/256)                     :rem 85
520 POKE BA,NA:BB=BB+1:POKE SX,BB-256*INT(BB/256):
    POKE SY,INT(BB/256)                          :rem 240
530 SYS 65496:IF PEEK(SP)AND1 GOTO 700           :rem 91
540 PRINT " SAVED" BB-BA "BYTES":END             :rem 237
600 PRINT:POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466
    :F$=F$+".SHP"                                :rem 215
```

```
610 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                        :rem 241
620 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                     :rem 159
630 GOSUB 600:POKE SA,0:POKE SX,LA-256*INT(LA/256)
    :POKE SY,INT(LA/256)                         :rem 52
640 SYS 65493:IF PEEK(SP)AND1 GOTO 700          :rem 90
650 RETURN                                      :rem 123
700 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRINT
     "FILE NOT FOUND":END                       :rem 96
710 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                :rem 188
720 PRINT ST:END                                :rem 222
```

BMG Statement Index

Bitmapped Graphics Utility

Statements

CLS 111
DRAW 106, 109
DRAWTO 114
FILL 115–16
GRAPHICS 106, 107
LFILL 117
MODE 112
PEN 153–54, 164
RFILL 117
SETPEN 110
SHAPE 123, 149
TEXT 112

Function

LOOK 168–70

Command

KILL 99

Part 3
Sidplayer

Chapter 7

Introduction to Music and the Sidplayer

Of all the special chips in the Commodore 64, none is more devoted to one specific purpose than the SID chip. This chip, more formally known as the Sound Interface Device, is solely responsible for producing music and sound effects. It has many features for controlling sound, and has been described as a complete synthesizer on a chip. The only problem is that the SID chip is very complicated. As a result, it's not fully understood by most people, and often not fully used.

Enter "Sidplayer," a music playing and editing utility for the Commodore 64. This music system is designed to make it easy to use the SID chip so that you can realize its full potential. Using Sidplayer, you can make your Commodore 64 produce music of extremely high quality.

To fully appreciate the capabilities of Sidplayer, it's first necessary to have an understanding of how electronic music works.

Electronic Music

When your ear hears a sound, it is actually detecting vibrations. The rate of vibration is called the *frequency* and determines the *pitch* of a sound. In a musical instrument, a metal string, reed, stretched membrane, or air in a tube is what vibrates. The player usually has a method for changing the pitch.

But sounds are not so simple. There are many different kinds of vibrations. When viewed with an oscilloscope, vibrations have another characteristic, called *waveform*. Square, triangle, and sawtooth are common waveforms. The waveform helps distinguish the sound produced by one instrument, such as a flute, from the sound produced by another instrument, such as a violin.

There's just one more essential characteristic remaining—volume. As a string is plucked or air is blown, the volume changes over a short period of time. This pattern of changing volume levels is called an *envelope*, and is usually divided into four parts called the *attack*, *decay*, *sustain*, and *release*. In the

first three parts, the volume rises to a peak level and then falls to a sustain level. When the note is released, the volume fades away to silence.

The frequency, waveform, and envelope are all essential parts of a note. A sequence of notes creates music. Electronic music is merely a method of producing these qualities of sound by electronic means. A device which does this is called a *synthesizer*. Theoretically, it's possible for a synthesizer to imitate any musical instrument or to produce sounds never heard before.

The Sidplayer

The SID chip contains three oscillators. Each oscillator acts as one "voice" and can produce a tone in a range of eight octaves, using one of four basic waveforms. The tone is passed through an envelope generator which regulates the volume of that voice. All three voices are then combined into one audio signal, which is controlled by the master volume and sent to the television or monitor speaker.

Figure 7-1. Producing Electronic Sounds

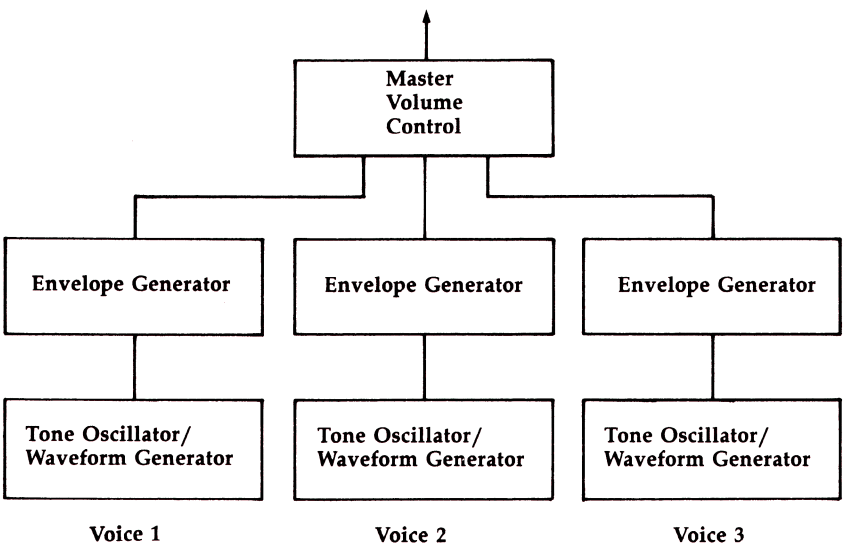


Figure 7-1 is a simplified description of the actual design of the SID chip, but it does serve our purposes for the moment. Advanced features not mentioned include a filter and options for synchronization and ring modulation.

Sidplayer fully supports all of the features built into the SID chip. As many as three voices can be played at the same time, each with its own pitch, waveform, and envelope. Sidplayer also supports the filter and all of the related filter controls, plus the synchronization and ring-modulation options.

To further extend the power of the SID chip, there are additional features provided by software control, such as vibrato, portamento, transposing, automatic filtering, and much more.

The most important thing, however, is not that Sidplayer has all these features, but that it makes them so easy to use. The music system includes a carefully designed editing program which lets you edit all three voices. Notes can be entered from keyboard or by joystick and are played as they are entered for immediate feedback. Special commands are available to select things like waveform and envelope settings.

Finally, all music created with Sidplayer can be merged with your own BASIC programs. The music will even play while the program is running. This opens up many possibilities, including animated screen displays that change in time with the music.

Some demonstration songs have been provided so that you can hear just what Sidplayer can do. First, however, you'll need to enter two programs. Disk users should enter and save Program 7-1; tape users, Program 7-2. Use the filename PLAYER.

Program 7-3 is a machine language listing that requires the "Machine Language Editor: MLX" program found in Appendix C (this is the same program used to enter "BMG.OBJ" in Chapter 5). Note the starting and ending addresses listed before the program listing. Use these as your responses to MLX's prompts. Program 7-3 should be saved using the filename SID.OBJ. If you're using tape, be sure to save the file on the same tape, right after Program 7-2, and note the number on the tape counter where SID.OBJ begins. Tape users who use an MLX program from another COMPUTE! publication must be sure to change line 763 to match

the listing in this book. (Tape users who have a copy of MLX which does not have a line 763 *must* use the version of MLX from this book.)

Program 7-1. PLAYER (Disk)

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SIDPLAYER":PRINT " BY
    {SPACE}CRAIG CHAMBERLAIN":PRINT          :rem 173
110 PRINT " TUNING INSTRUMENTS...":PRINT     :rem 245
120 DN=8:SA=780:SX=781:SY=782:SP=783:POKE SA,1:POK
    E SX,DN:POKE SY,1:SYS 65466              :rem 104
130 F$="SID.OBJ":GOSUB 600:POKE SA,0:SYS 65493:IF
    {SPACE}PEEK(SP)AND1 GOTO 710              :rem 12
140 SS=49152:HK=49435:PL=49458:DP=49629:LA=PEEK(49
    )+256*PEEK(50)+100:DR=LA                 :rem 212
150 FOR K=LA TO LA+90:READ P:POKE K,P:NEXT:LA=LA+9
    1:HI=INT(LA/256):LO=LA-256*HI            :rem 140
160 GOTO 510                                 :rem 102
300 K=FRE(0):F$="":INPUT " YOUR REQUEST";F$:IF LEN
    (F$)>12 GOTO 300                         :rem 153
310 IF F$="" GOTO 500                        :rem 220
320 PRINT:POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466
    :F$=F$+".MUS":GOSUB 600                   :rem 48
330 POKE SA,0:POKE SX,LO:POKE SY,HI:SYS 65493:IF P
    EEK(SP)AND1 GOTO 700                      :rem 92
400 SYS HK:POKE SX,LO:POKE SY,HI:SYS PL:K=PEEK(SX)
    +256*PEEK(SY):F$=" "                     :rem 196
410 P=PEEK(K):K=K+1:IF P AND P<>13 THEN F$=F$+CHR$
    (P):GOTO 410                             :rem 123
420 PRINT F$:IF P THEN F$=" ":GOTO 410       :rem 200
430 WAIT 56320,16:POKE 198,0:POKE SS,7        :rem 56
440 IF PEEK(198) THEN POKE 198,0:POKE SS,0    :rem 6
450 IF PEEK(SS)AND7 GOTO 440                  :rem 31
460 POKE 54276,0:POKE 54283,0:POKE 54290,0:SYS DP:
    GOTO 300                                 :rem 148
500 PRINT CHR$(147):PRINT " SIDPLAYER":PRINT " BY
    {SPACE}CRAIG CHAMBERLAIN":PRINT          :rem 177
510 FOR K=1 TO 4:POKE 580+K,ASC(MID$(".MUS",K)):NE
    XT                                       :rem 131
520 OPEN 1,8,0,"$":GET #1,S$,S$:PRINT " ";:TB=1:K=
    0                                        :rem 120
530 SYS DR:TB=TB+13:PRINT TAB(TB);:IF TB=40 THEN T
    B=1:PRINT " ";                           :rem 152
540 IF ST=0 THEN K=K+1:GOTO 530              :rem 165
550 PRINT:CLOSE 1:SYS 65484:IF K=0 THEN PRINT:PRIN
    T " NO MUSIC FILES ON DISK"              :rem 237
560 PRINT:GOTO 300                            :rem 46
600 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                     :rem 240
```

```
610 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                    :rem 158
700 IF PEEK(SA)=4 THEN PRINT " I DON'T KNOW THAT S
    ONG":PRINT:GOTO 300                        :rem 186
710 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRINT
     "FILE NOT FOUND":END                      :rem 97
720 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                               :rem 189
730 PRINT ST:END                               :rem 223
800 DATA 162,1,32,198,255,32,207,255,32,207,255,32
    ,207,255,133,251,32,207,255                :rem 206
801 DATA 133,252,32,207,255,164,144,208,62,201,34,
    208,245,160,0,32,207,255,201               :rem 238
802 DATA 34,240,6,153,73,2,200,208,243,132,253,32,
    207,255,168,208,250,164,253                :rem 200
803 DATA 192,5,144,200,162,3,185,72,2,221,69,2,208
    ,190,136,202,16,244,132,253                :rem 195
804 DATA 160,0,185,73,2,32,210,255,200,196,253,208
    ,245,96                                    :rem 247
```

Program 7-2. PLAYER (Tape)

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SIDPLAYER":PRINT " BY
    {SPACE}CRAIG CHAMBERLAIN":PRINT            :rem 173
110 PRINT " TUNING INSTRUMENTS...":PRINT       :rem 245
120 DN=1:SA=780:SX=781:SY=782:SP=783:POKE SA,1:POK
    E SX,DN:POKE SY,1:SYS 65466                :rem 97
130 F$="SID.OBJ":GOSUB 600:POKE SA,0:SYS 65493:IF
    {SPACE}PEEK(SP)AND1 GOTO 700               :rem 11
140 SS=49152:HK=49435:PL=49458:DP=49629        :rem 202
150 LA=PEEK(49)+256*PEEK(50)+100:HI=INT(LA/256):LO
    =LA-256*HI                                 :rem 141
300 K=FRE(0):F$="":INPUT " YOUR REQUEST";F$:IF LEN
    (F$)>12 GOTO 300                           :rem 153
310 IF F$="" THEN PRINT:ON F+1 GOTO 300,400
                                               :rem 207
320 PRINT:F=1:POKE SA,1:POKE SX,DN:POKE SY,0:SYS 6
    5466:F$=F$+".MUS":GOSUB 600                :rem 30
330 POKE SA,0:POKE SX,LO:POKE SY,HI:SYS 65493:IF P
    EEK(SP)AND1 GOTO 700                       :rem 92
400 SYS HK:POKE SX,LO:POKE SY,HI:SYS PL:K=PEEK(SX)
    +256*PEEK(SY):F$=" "                       :rem 196
410 P=PEEK(K):K=K+1:IF P AND P<>13 THEN F$=F$+CHR$
    (P):GOTO 410                               :rem 123
420 PRINT F$:IF P THEN F$=" ":GOTO 410         :rem 200
430 WAIT 56320,16:POKE 198,0:POKE SS,7         :rem 56
440 IF PEEK(198) THEN POKE 198,0:POKE SS,0     :rem 6
450 IF PEEK(SS)AND7 GOTO 440                   :rem 31
```

```
460 POKE 54276,0:POKE 54283,0:POKE 54290,0:SYS DP:
    GOTO 300                                    :rem 148
600 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                        :rem 240
610 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                     :rem 158
700 PRINT " ERROR" ST:END                       :rem 170
```

Let the Music Play

You may want to enter any or all of the three demonstration songs which follow Program 7-3. The first, Program 7-4, is "Bach's Two-Part Invention in A Minor." This is the theme music used by Commodore in their television ads. Save the file using the filename COMMODORE.MUS. The .MUS extension is used to identify all music files. Use the starting and ending addresses noted before this listing, as well as the next two programs, as your responses to MLX's queries.

Program 7-5 is the "March of the Wooden Soldiers." Save this file with the filename WSOLDIER.MUS. The final demonstration, Program 7-6, is an original composition written using the Sidplayer Editor. The filename for this file is ETAL.MUS.

When you run "PLAYER" (whether tape or disk), it displays the message TUNING INSTRUMENTS while it initializes and loads the SID.OBJ file. If you're using disk, the program also lists a directory of all music files on the disk in three columns. In response to the prompt YOUR REQUEST?, you should type the name of the song you want to play. *Do not include the .MUS extension.* The song loads, the full title and credit information print, and the song starts playing. It will go something like this:

YOUR REQUEST? COMMODORE
TWO PART INVENTION #13
J.S. BACH
COURTESY CRAIG CHAMBERLAIN

When the song is finished, you'll be prompted for another selection. You can enter another song name to load and play another tune.

If you're using disk and the requested file is not on the disk in the drive, PLAYER tells you that it doesn't know that song, and asks for a new selection.

If you press only RETURN in response to the song name prompt and are using tape, the last song will be replayed. If you're using disk, PLAYER prints a new directory. This is useful if the old directory has scrolled off the screen or if you want to switch disks.

To make a song stop playing before its end, hit any key. PLAYER cancels the current song and prompts you for a new selection. To end the program, press RUN/STOP–RESTORE.

The demonstration tunes cover a variety of music, from classical to modern. Enjoy listening to the music produced by Sidplayer.

Program 7-3. SID.OBJ

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49349
Ending Address: 51118
Filename: SID.OBJ

```
49349 :076,086,194,076,015,197,073
49355 :236,193,000,000,213,192,013
49361 :213,192,213,192,096,001,092
49367 :002,004,000,007,014,001,243
49373 :000,255,000,002,004,005,231
49379 :007,009,011,030,024,139,191
49385 :126,250,006,172,243,230,236
49391 :143,248,046,134,142,150,078
49397 :159,168,179,189,200,212,072
49403 :225,238,253,140,120,100,047
49409 :080,060,040,020,000,000,201
49415 :002,003,005,007,008,010,042
49421 :012,013,015,017,018,000,088
49427 :224,000,004,008,012,249,004
49433 :000,245,120,162,002,189,231
49439 :019,003,157,204,192,189,027
49445 :202,192,157,019,003,202,044
49451 :208,241,142,000,192,088,146
49457 :096,169,000,141,000,192,135
49463 :134,251,132,252,160,196,156
49469 :153,000,192,136,208,250,232
49475 :141,023,212,141,021,212,049
49481 :141,022,212,169,008,141,254
49487 :029,192,141,024,212,169,078
49493 :144,141,030,192,169,012,005
49499 :141,031,192,169,212,133,201
49505 :254,162,002,169,001,157,074
49511 :033,192,169,004,157,039,185
```

```
49517 :192,169,004,157,111,192,166
49523 :189,020,193,157,138,192,236
49529 :189,217,192,133,253,169,250
49535 :000,160,002,145,253,169,088
49541 :008,157,014,192,157,099,248
49547 :192,200,145,253,200,145,250
49553 :253,169,064,157,017,192,229
49559 :145,253,169,032,157,020,159
49565 :192,200,145,253,169,245,081
49571 :157,023,192,200,145,253,109
49577 :202,016,184,165,251,024,243
49583 :105,006,133,253,169,000,073
49589 :170,168,101,252,133,254,235
49595 :157,126,192,157,132,192,119
49601 :165,253,157,123,192,157,216
49607 :129,192,024,113,251,133,017
49613 :253,165,254,200,113,251,161
49619 :200,232,224,003,208,224,022
49625 :166,253,168,096,120,173,169
49631 :205,192,141,020,003,173,189
49637 :206,192,141,021,003,088,112
49643 :096,173,013,220,173,000,142
49649 :192,048,087,009,128,168,105
49655 :041,007,240,085,216,140,208
49661 :000,192,169,212,133,252,187
49667 :088,162,000,173,000,192,106
49673 :061,214,192,240,013,142,103
49679 :032,192,032,197,192,173,065
49685 :000,192,041,120,208,050,120
49691 :232,224,003,208,230,202,102
49697 :173,000,192,061,214,192,097
```



```
49703 :240,025,189,217,192,133,011
49709 :251,189,005,192,160,000,074
49715 :145,251,189,008,192,200,012
49721 :145,251,189,120,192,160,090
49727 :004,145,251,202,016,220,133
49733 :173,000,192,041,127,044,134
49739 :169,008,141,000,192,108,181
49745 :205,192,108,207,192,189,150
49751 :217,192,133,251,222,033,111
49757 :192,240,079,189,036,192,253
49763 :048,238,208,014,189,039,067
49769 :192,221,033,192,144,006,125
49775 :189,017,192,157,120,192,210
49781 :189,042,192,240,086,010,108
49787 :189,005,192,176,049,125,091
49793 :045,192,157,005,192,168,120
49799 :189,008,192,125,048,192,121
49805 :157,008,192,072,152,221,175
49811 :084,192,104,253,087,192,035
49817 :144,051,189,084,192,157,202
49823 :005,192,189,087,192,157,213
49829 :008,192,169,000,157,042,221
49835 :192,024,144,031,240,117,151
49841 :253,045,192,157,005,192,253
49847 :189,008,192,253,048,192,041
49853 :157,008,192,189,084,192,243
49859 :221,005,192,189,087,192,057
49865 :253,008,192,176,205,144,155
49871 :101,189,051,192,240,096,052
49877 :160,000,222,054,192,208,025
49883 :049,189,057,192,029,060,027
```

```
49889 :192,208,027,189,069,192,078
49895 :157,063,192,157,054,192,022
49901 :189,051,192,010,189,066,166
49907 :192,144,004,073,255,105,248
49913 :000,157,051,192,208,016,105
49919 :189,063,192,157,054,192,078
49925 :152,056,253,051,192,157,098
49931 :051,192,201,000,016,001,216
49937 :136,024,125,057,192,157,196
49943 :057,192,072,152,125,060,169
49949 :192,157,060,192,168,104,134
49955 :024,144,002,240,085,125,143
49961 :084,192,157,005,192,152,055
49967 :125,087,192,157,008,192,040
49973 :189,090,192,240,023,024,043
49979 :125,011,192,157,011,192,235
49985 :160,002,145,251,189,093,137
49991 :192,125,014,192,157,014,253
49997 :192,200,145,251,160,000,001
50003 :189,105,192,240,034,016,091
50009 :001,200,024,109,026,192,129
50015 :072,041,007,141,026,192,062
50021 :141,021,212,104,106,074,247
50027 :074,024,121,018,193,024,049
50033 :109,027,192,141,027,192,033
50039 :141,022,212,108,207,192,233
50045 :189,123,192,133,253,189,180
50051 :126,192,133,254,208,006,026
50057 :108,209,192,032,200,192,046
50063 :173,000,192,061,214,192,207
50069 :240,242,160,000,177,253,197
```



```
50075 :133,255,200,177,253,168,061
50081 :165,253,024,105,002,133,075
50087 :253,157,123,192,165,254,031
50093 :105,000,133,254,157,126,180
50099 :192,165,255,041,003,208,019
50105 :210,189,084,192,157,005,254
50111 :192,189,087,192,157,008,248
50117 :192,152,157,002,192,041,165
50123 :056,074,074,074,125,072,166
50129 :192,133,253,152,041,192,148
50135 :010,042,042,168,185,219,113
50141 :192,133,254,189,002,192,159
50147 :041,007,240,104,208,002,061
50153 :240,146,168,185,222,192,106
50159 :101,254,024,125,075,192,242
50165 :016,005,024,105,012,230,125
50171 :253,201,012,144,004,233,074
50177 :012,198,253,133,254,168,251
50183 :165,255,072,185,242,192,094
50189 :133,255,185,230,192,164,148
50195 :253,136,048,006,070,255,019
50201 :106,136,016,250,024,125,170
50207 :078,192,157,084,192,165,131
50213 :255,125,081,192,157,087,166
50219 :192,104,133,255,041,028,028
50225 :240,182,189,045,192,029,158
50231 :048,192,240,022,189,005,239
50237 :192,221,084,192,189,008,179
50243 :192,253,087,192,169,254,190
50249 :106,157,042,192,144,017,219
50255 :240,087,157,042,192,189,218
```

```
50261 :084,192,157,005,192,189,136
50267 :087,192,157,008,192,189,148
50273 :036,192,010,208,066,189,030
50279 :090,192,240,019,189,096,161
50285 :192,157,011,192,160,002,055
50291 :145,251,189,099,192,157,124
50297 :014,192,200,145,251,189,088
50303 :102,192,240,015,164,253,069
50309 :024,121,254,192,164,254,118
50315 :024,121,006,193,024,144,139
50321 :008,189,105,192,240,017,128
50327 :189,108,192,141,027,192,232
50333 :141,022,212,169,000,141,074
50339 :026,192,141,021,212,165,152
50345 :255,168,041,032,133,255,029
50351 :152,041,064,157,036,192,049
50357 :152,041,028,074,074,168,206
50363 :173,031,192,136,240,028,219
50369 :173,030,192,208,001,056,085
50375 :136,240,006,106,176,057,152
50381 :136,208,250,164,255,133,071
50387 :255,240,007,074,176,045,240
50393 :101,255,176,044,157,033,215
50399 :192,189,111,192,010,010,159
50405 :029,114,192,010,029,117,208
50411 :192,010,157,017,192,056,091
50417 :189,002,192,041,007,208,112
50423 :003,126,036,192,189,017,042
50429 :192,105,000,157,120,192,251
50435 :108,209,192,169,016,044,229
50441 :169,024,141,000,192,096,119
```



```
50447 :152,072,165,255,074,176,141
50453 :113,074,074,176,031,074,051
50459 :176,015,157,099,192,160,058
50465 :003,145,251,104,157,096,021
50471 :192,136,145,251,096,074,165
50477 :144,002,009,248,157,081,174
50483 :192,104,157,078,192,096,102
50489 :074,144,011,074,104,176,128
50495 :006,157,108,192,141,022,177
50501 :212,096,168,240,034,104,155
50507 :136,240,035,136,240,036,130
50513 :136,240,037,136,240,050,152
50519 :136,240,051,136,240,060,182
50525 :136,240,061,136,240,089,227
50531 :136,240,090,136,240,091,008
50537 :108,211,192,104,141,030,123
50543 :192,096,141,031,192,096,091
50549 :157,039,192,096,157,135,125
50555 :192,165,253,157,129,192,187
50561 :165,254,157,132,192,096,101
50567 :176,085,141,001,192,096,058
50573 :157,090,192,010,144,001,223
50579 :136,152,157,093,192,096,205
50585 :157,105,192,096,168,208,055
50591 :004,157,051,192,096,157,048
50597 :066,192,188,051,192,208,038
50603 :015,157,051,192,152,157,127
50609 :057,192,157,060,192,169,236
50615 :001,157,054,192,096,157,072
50621 :069,192,096,157,102,192,229
50627 :096,074,144,002,200,024,223
```

```
50633 :072,041,007,121,024,193,147
50639 :157,072,192,104,074,074,112
50645 :074,024,121,025,193,157,039
50651 :075,192,096,074,144,008,040
50657 :157,048,192,104,157,045,160
50663 :192,096,104,074,176,092,197
50669 :074,176,044,074,176,005,018
50675 :074,160,240,208,006,010,173
50681 :010,010,010,160,015,133,075
50687 :255,152,160,005,176,011,246
50693 :061,020,192,005,255,157,183
50699 :020,192,145,251,096,061,008
50705 :023,192,005,255,157,023,160
50711 :192,200,145,251,096,074,213
50717 :176,044,074,176,082,133,202
50723 :255,189,138,192,221,021,027
50729 :193,240,066,254,138,192,100
50735 :168,165,253,153,141,192,095
50741 :165,254,153,153,192,164,110
50747 :255,185,181,192,240,042,130
50753 :133,254,185,165,192,133,103
50759 :253,096,176,070,074,176,148
50765 :052,168,165,253,153,165,009
50771 :192,165,254,153,181,192,196
50777 :189,138,192,221,021,193,019
50783 :240,013,254,138,192,168,076
50789 :169,000,153,153,192,096,096
50795 :169,048,044,169,040,141,206
50801 :000,192,096,010,010,010,175
50807 :010,077,028,192,041,240,195
50813 :077,028,192,144,107,077,238
```

```
50819 :029,192,041,015,077,029,002
50825 :192,141,029,192,141,024,088
50831 :212,096,074,074,176,095,102
50837 :074,168,240,023,136,240,006
50843 :045,136,240,057,136,240,241
50849 :060,136,240,063,173,029,094
50855 :192,041,127,144,098,009,010
50861 :128,176,094,172,029,192,196
50867 :176,007,200,152,041,015,002
50873 :208,007,096,152,041,015,192
50879 :240,007,136,140,029,192,167
50885 :140,024,212,096,189,214,048
50891 :192,073,255,045,028,192,220
50897 :144,026,029,214,192,176,222
50903 :021,152,042,157,114,192,125
50909 :096,152,042,157,117,192,209
50915 :096,173,028,192,041,247,236
50921 :144,002,009,008,141,028,053
50927 :192,141,023,212,096,074,209
50933 :176,030,074,176,008,208,149
50939 :002,169,008,157,111,192,122
50945 :096,010,010,010,010,077,214
50951 :029,192,041,112,077,029,231
50957 :192,141,029,192,141,024,220
50963 :212,096,074,168,208,021,030
50969 :189,135,192,240,005,222,240
50975 :135,192,240,010,189,129,158
50981 :192,133,253,189,132,192,104
50987 :133,254,096,136,208,032,134
50993 :189,138,192,221,020,193,234
50999 :240,018,222,138,192,168,009
```

```
51005 :136,185,153,192,240,007,206
51011 :133,254,185,141,192,133,081
51017 :253,096,169,032,141,000,252
51023 :192,096,173,000,192,093,057
51029 :214,192,141,000,192,136,192
51035 :152,145,251,200,145,251,211
51041 :096,142,203,192,140,204,050
51047 :192,138,024,105,106,141,041
51053 :198,192,152,105,000,141,129
51059 :199,192,138,024,105,035,040
51065 :141,201,192,152,105,003,147
51071 :141,202,192,134,251,132,155
51077 :252,169,236,133,253,169,065
51083 :193,133,254,162,005,160,022
51089 :000,177,253,145,251,200,147
51095 :208,249,230,252,230,254,038
51101 :202,208,242,177,253,145,104
51107 :251,200,192,118,208,247,099
51113 :096,130,128,128,128,130,141
```

Program 7-4. COMMODORE.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 50849
Filename: COMMODORE.MUS

```
49152 :084,003,042,000,224,002,099
49158 :001,039,001,006,006,160,219
49164 :001,236,030,001,024,000,048
49170 :038,001,024,155,024,158,162
49176 :024,145,024,159,024,155,043
49182 :024,159,024,146,038,014,179
49188 :020,145,020,147,020,093,225
49194 :020,147,030,002,038,006,029
49200 :024,158,038,001,024,155,192
49206 :024,158,024,145,024,159,076
49212 :024,155,024,159,024,146,080
49218 :038,014,020,145,020,158,205
49224 :016,000,030,003,024,000,145
49230 :038,001,024,147,024,145,201
49236 :024,147,024,158,024,145,094
49242 :024,155,024,157,038,014,246
49248 :020,156,020,158,020,146,104
49254 :084,148,030,004,038,007,157
49260 :024,148,038,001,024,146,233
49266 :024,159,024,146,024,157,136
49272 :024,159,024,154,024,156,149
49278 :038,014,020,155,020,157,018
49284 :020,145,084,147,030,005,051
49290 :038,007,024,147,038,001,137
49296 :024,145,024,158,024,145,152
49302 :038,014,020,156,084,146,096
49308 :038,007,024,146,038,001,154
49314 :024,159,024,157,024,159,197
49320 :038,014,020,155,084,145,112
49326 :030,006,038,007,024,145,168
49332 :038,001,024,158,024,156,069
49338 :024,158,038,014,020,154,082
49344 :020,159,020,145,020,000,044
49350 :016,000,001,047,030,007,043
```

```
49356 :001,006,001,071,024,000,051
49362 :038,001,024,157,024,145,087
49368 :024,147,024,146,024,157,226
49374 :024,146,024,148,038,014,104
49380 :020,147,020,149,020,159,231
49386 :020,149,030,008,038,006,229
49392 :024,145,038,001,024,157,117
49398 :024,145,024,147,024,146,244
49404 :024,157,024,146,024,148,007
49410 :038,014,020,147,020,145,130
49416 :020,149,020,147,030,009,127
49422 :038,001,024,137,024,150,132
49428 :024,147,024,150,024,145,022
49434 :024,147,024,158,024,145,036
49440 :038,014,020,146,020,084,098
49446 :020,150,020,137,030,010,149
49452 :038,001,024,151,024,149,175
49458 :024,146,024,149,024,159,064
49464 :024,146,024,157,024,159,078
49470 :038,014,020,145,020,147,190
49476 :020,149,020,151,030,011,193
49482 :038,001,024,150,024,084,139
49488 :024,082,024,084,024,159,221
49494 :024,082,024,092,024,158,234
49500 :038,014,020,157,038,004,107
49506 :084,149,024,149,024,147,163
49512 :024,145,024,147,030,012,230
49518 :038,014,020,158,038,004,126
49524 :084,084,024,084,038,001,175
49530 :024,146,024,159,024,146,133
49536 :038,014,020,157,038,004,143
```



```
49542 :084,147,024,147,038,001,063
49548 :024,145,024,158,024,145,148
49554 :030,013,038,007,024,092,094
49560 :038,001,024,149,024,084,216
49566 :024,147,024,082,024,084,031
49572 :024,159,024,082,038,014,249
49578 :020,147,020,000,016,000,117
49584 :001,047,030,014,001,039,052
49590 :001,006,024,000,038,002,253
49596 :024,149,024,215,024,149,005
49602 :024,147,024,149,024,081,131
49608 :024,147,024,149,024,147,203
49614 :024,081,024,147,024,158,152
49620 :024,000,020,000,030,015,045
49626 :024,000,001,047,001,006,041
49632 :024,148,024,150,024,148,230
49638 :024,146,024,148,024,159,243
49644 :024,146,024,148,024,146,236
49650 :024,159,024,146,024,157,008
49656 :024,000,020,000,030,016,082
49662 :024,000,001,047,001,006,077
49668 :024,147,024,149,024,147,007
49674 :024,145,024,147,024,158,020
49680 :024,145,024,082,024,145,204
49686 :024,158,024,145,024,092,233
49692 :024,000,020,000,030,017,119
49698 :024,000,001,047,001,006,113
49704 :024,146,024,148,024,146,040
49710 :024,159,024,146,024,093,004
49716 :024,159,024,146,024,159,076
49722 :024,093,024,159,024,155,025
```

```
49728 :024,000,020,000,030,018,156
49734 :024,000,001,047,001,006,149
49740 :024,155,024,158,024,145,094
49746 :024,159,024,155,024,159,115
49752 :024,146,038,010,020,145,215
49758 :020,158,020,093,020,155,048
49764 :030,019,038,005,024,158,118
49770 :038,002,024,145,024,147,230
49776 :024,145,024,158,024,145,120
49782 :024,092,024,158,024,145,073
49788 :024,158,024,092,024,158,092
49794 :024,090,024,145,024,159,084
49800 :024,158,030,020,038,005,155
49806 :024,093,038,002,024,159,226
49812 :024,146,024,159,024,093,106
49818 :024,159,024,154,024,156,183
49824 :024,093,024,156,024,154,123
49830 :024,156,024,167,024,156,205
49836 :024,155,024,154,030,021,068
49842 :038,005,024,153,038,002,182
49848 :024,155,024,158,024,155,212
49854 :024,153,024,155,024,166,224
49860 :024,153,024,090,024,153,152
49866 :024,166,024,153,024,100,181
49872 :024,153,024,167,024,166,254
49878 :030,022,038,010,020,101,179
49884 :020,159,020,093,020,155,175
49890 :024,000,038,004,024,155,215
49896 :024,158,024,145,024,159,254
49902 :024,155,024,159,024,146,002
49908 :030,023,024,145,024,158,136
```



```
49914 :024,145,024,147,024,146,248
49920 :024,159,024,146,024,148,013
49926 :024,147,024,145,024,147,005
49932 :024,149,024,148,024,147,016
49938 :024,146,024,145,030,024,155
49944 :024,159,024,145,024,146,034
49950 :024,147,024,148,024,146,031
49956 :024,085,024,146,024,151,234
49962 :024,146,024,145,024,150,043
49968 :024,148,024,146,024,159,061
49974 :024,146,030,025,024,093,140
49980 :024,159,024,145,024,158,082
49986 :024,155,024,158,024,159,098
49992 :024,093,024,158,024,155,038
49998 :024,153,024,155,016,166,104
50004 :016,000,001,047,001,079,228
50010 :001,059,166,010,001,002,073
50016 :001,051,166,014,010,050,132
50022 :001,002,010,000,001,135,251
50028 :001,059,166,094,001,002,175
50034 :166,190,001,002,166,156,027
50040 :001,002,166,012,001,002,048
50046 :166,010,001,002,001,079,129
50052 :006,160,001,071,118,005,237
50058 :134,002,030,001,038,010,097
50064 :020,174,038,002,016,166,048
50070 :020,101,024,166,024,163,136
50076 :024,166,024,153,024,167,202
50082 :024,163,024,167,024,154,206
50088 :030,002,038,010,020,153,165
50094 :020,166,020,101,020,163,152
```

```
50100 :038,005,024,166,038,002,197
50106 :024,163,024,166,024,153,228
50112 :024,167,024,163,024,167,249
50118 :024,154,030,003,038,010,201
50124 :020,153,020,166,020,153,224
50130 :020,166,038,005,024,154,105
50136 :038,002,024,166,024,164,122
50142 :024,166,024,162,024,164,018
50148 :024,174,024,161,030,004,133
50154 :038,010,020,175,020,162,147
50160 :020,165,084,167,038,005,207
50166 :024,167,038,002,024,165,154
50172 :024,163,024,165,024,161,045
50178 :024,163,024,173,024,175,073
50184 :030,005,038,010,020,174,029
50190 :020,161,038,002,024,162,165
50196 :024,164,024,175,024,162,081
50202 :038,010,020,173,020,175,206
50208 :038,002,024,161,024,163,188
50214 :024,174,024,161,030,006,201
50220 :038,010,020,172,020,170,218
50226 :038,005,024,173,038,002,074
50232 :024,165,024,164,024,165,110
50238 :038,005,024,161,038,002,074
50244 :024,165,024,153,024,155,101
50250 :024,154,024,165,024,154,107
50256 :024,156,030,007,038,010,089
50262 :020,155,020,153,020,167,109
50268 :020,165,038,005,024,153,241
50274 :038,002,024,165,024,153,248
50280 :024,155,024,154,024,165,138
50286 :024,154,024,156,030,008,250
50292 :038,010,020,155,020,153,000
50298 :016,000,024,000,038,002,202
50304 :024,157,024,155,024,157,157
50310 :024,153,024,155,024,165,167
50316 :024,167,030,009,038,010,162
50322 :020,166,020,153,020,155,168
50328 :020,157,038,002,024,092,229
50334 :024,158,024,154,024,092,122
50340 :024,166,024,154,024,100,144
50346 :024,166,030,010,038,010,192
50352 :020,165,020,167,020,154,210
50358 :020,092,038,002,024,155,001
50364 :024,157,024,153,024,155,213
50370 :024,165,024,153,024,163,235
50376 :024,165,030,011,038,010,222
50382 :020,100,020,166,020,167,187
50388 :020,090,024,000,038,002,130
50394 :024,155,024,153,024,155,241
50400 :024,166,024,153,024,155,002
50406 :024,157,030,012,038,005,240
50412 :024,092,038,002,024,154,058
50418 :024,167,024,154,024,165,032
50424 :024,167,024,154,024,092,221
50430 :038,005,024,155,038,002,004
50436 :024,153,024,166,024,153,036
50442 :024,100,024,166,084,153,049
50448 :030,013,038,004,024,153,022
50454 :038,002,024,167,024,153,174
50460 :024,166,038,010,020,167,197
50466 :020,175,038,005,024,163,203
```

```
50472 :038,002,024,155,024,167,194
50478 :024,165,024,163,024,175,109
50484 :024,173,024,175,030,014,236
50490 :038,010,020,171,020,163,224
50496 :020,165,020,231,020,097,105
50502 :020,000,024,000,038,002,154
50508 :024,157,024,156,024,155,104
50514 :030,015,038,010,020,154,093
50520 :020,162,020,164,020,230,192
50526 :020,175,020,000,024,000,077
50532 :038,002,024,156,024,155,243
50538 :024,154,030,016,038,010,122
50544 :020,153,020,161,020,163,137
50550 :020,100,020,174,020,000,196
50556 :024,000,038,002,024,155,111
50562 :024,090,024,089,030,017,148
50568 :038,010,020,167,020,175,054
50574 :020,162,020,164,020,109,125
50580 :020,000,024,000,038,002,232
50586 :024,154,024,153,024,167,188
50592 :030,018,038,010,020,153,173
50598 :020,166,020,101,020,163,144
50604 :038,005,024,166,038,002,189
50610 :024,163,024,166,024,153,220
50616 :024,167,024,163,024,167,241
50622 :024,154,030,019,038,005,204
50628 :024,153,038,002,024,155,080
50634 :024,158,024,155,024,153,228
50640 :024,155,024,166,024,153,242
50646 :024,100,024,166,024,153,193
50652 :024,166,024,100,024,166,212
```



```
50658 :024,098,024,100,030,020,010
50664 :038,010,020,163,020,101,072
50670 :020,167,020,101,020,163,217
50676 :020,175,020,109,020,171,247
50682 :030,021,038,004,020,174,025
50688 :020,161,020,163,020,161,033
50694 :020,174,020,161,020,106,251
50700 :020,000,030,022,024,000,108
50706 :038,002,024,167,024,101,118
50712 :024,163,024,162,024,167,076
50718 :024,101,024,162,001,136,222
50724 :038,010,020,161,020,163,192
50730 :020,109,020,163,030,023,151
50736 :020,174,020,100,020,175,045
50742 :020,101,020,161,020,166,030
50748 :020,162,020,231,030,024,035
50754 :020,101,020,164,020,162,041
50760 :020,175,020,109,020,174,078
50766 :020,170,020,171,030,025,002
50772 :020,172,020,106,020,171,081
50778 :020,163,038,004,012,174,245
50784 :016,000,001,079,084,087,107
50790 :079,032,080,065,082,084,012
50796 :032,073,078,086,069,078,012
50802 :084,073,079,078,032,035,239
50808 :049,051,013,074,046,083,180
50814 :046,032,066,065,067,072,218
50820 :013,067,079,085,082,084,030
50826 :069,083,089,032,067,082,048
50832 :065,073,071,032,067,072,012
50838 :065,077,066,069,082,076,073
50844 :065,073,078,013,013,000,142
```

Program 7-5. WSOLDIER.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 50465
Filename: WSOLDIER.MUS

```
49152 :248,001,084,001,116,001,195
49158 :006,128,001,039,001,028,209
49164 :001,000,001,244,001,072,075
49170 :038,001,166,010,022,005,004
49176 :054,002,004,000,004,000,088
49182 :004,000,004,000,004,000,042
49188 :004,000,004,000,004,000,048
49194 :004,000,008,000,012,000,066
49200 :020,000,001,015,001,022,107
49206 :030,001,024,157,024,157,191
49212 :020,155,020,157,020,157,077
49218 :030,002,024,157,024,157,204
49224 :020,156,020,157,020,157,090
49230 :030,003,024,157,024,157,217
49236 :020,155,020,157,020,159,103
49242 :030,004,020,158,024,157,227
49248 :024,092,024,091,024,155,250
49254 :020,154,030,005,024,157,236
49260 :024,157,020,155,020,157,129
49266 :020,157,030,006,024,157,252
49272 :024,157,020,154,020,159,142
49278 :020,159,030,007,024,159,013
49284 :024,159,020,158,020,000,001
49290 :020,154,020,000,020,157,253
49296 :020,157,020,157,001,047,034
49302 :001,018,030,008,024,157,132
49308 :024,157,020,158,020,158,181
49314 :020,158,020,158,024,158,188
49320 :024,157,024,156,024,155,196
49326 :024,154,030,009,024,155,058
49332 :024,156,024,157,020,158,207
49338 :024,156,024,158,020,157,213
49344 :024,155,024,157,020,156,216
49350 :020,155,020,154,030,010,075
49356 :024,157,024,157,020,157,231
49362 :020,157,020,157,020,157,229
49368 :024,157,024,156,024,155,244
49374 :024,154,024,153,030,011,106
49380 :024,154,024,155,024,156,253
49386 :020,157,024,155,024,157,003
49392 :020,156,024,154,024,156,006
49398 :020,155,020,154,020,153,000
49404 :030,012,024,153,024,153,136
49410 :020,153,024,153,024,153,017
49416 :020,153,024,153,024,153,023
49422 :020,153,020,154,020,153,022
49428 :030,013,024,165,024,165,185
49434 :020,165,020,092,020,157,244
49440 :024,165,024,165,020,165,083
49446 :020,092,020,157,024,165,004
49452 :024,165,020,165,020,092,018
49458 :020,157,020,092,020,157,004
49464 :020,157,020,157,030,014,198
49470 :001,018,030,015,001,038,165
```

```
49476 :024,157,024,157,020,145,083
49482 :020,145,020,145,024,145,061
49488 :024,145,020,159,020,159,095
49494 :020,159,030,016,024,157,236
49500 :024,157,020,145,020,145,091
49506 :020,145,020,157,024,157,109
49512 :024,092,024,091,024,155,002
49518 :020,154,030,017,024,157,000
49524 :024,157,020,145,020,145,115
49530 :020,145,030,018,004,145,228
49536 :004,159,004,094,020,158,055
49542 :020,158,020,158,030,019,027
49548 :004,158,004,159,004,145,102
49554 :020,145,020,000,020,159,254
49560 :020,000,020,145,020,145,246
49566 :020,145,030,020,004,145,010
49572 :004,159,004,094,020,158,091
49578 :020,158,020,158,030,021,065
49584 :004,158,004,159,004,145,138
49590 :020,149,020,000,004,149,012
49596 :004,084,004,148,004,147,067
49602 :004,082,004,146,004,081,003
49608 :004,145,004,159,004,094,098
49614 :004,158,004,157,004,092,113
49620 :004,156,004,155,004,090,113
49626 :004,154,004,089,004,153,114
49632 :004,167,004,102,004,166,159
49638 :020,165,020,000,020,159,102
49644 :020,000,020,145,004,145,058
49650 :004,145,004,145,020,145,193
49656 :001,079,004,000,001,079,156
```



```
49662 :001,007,001,004,001,144,156
49668 :001,132,001,152,038,001,073
49674 :166,012,006,128,022,005,093
49680 :054,002,004,153,004,153,130
49686 :004,153,020,153,020,153,013
49692 :020,153,004,153,004,153,003
49698 :004,153,020,153,020,153,025
49704 :020,153,004,153,004,153,015
49710 :004,153,024,153,024,153,045
49716 :024,153,024,153,024,153,071
49722 :024,153,024,153,024,153,077
49728 :020,153,020,153,020,153,071
49734 :001,015,054,004,030,001,175
49740 :001,246,024,000,024,000,115
49746 :020,153,020,153,020,153,089
49752 :001,047,030,002,001,242,155
49758 :030,003,001,230,024,153,023
49764 :024,153,020,153,020,153,111
49770 :020,153,020,153,016,153,109
49776 :020,000,001,047,001,015,196
49782 :030,008,001,214,024,000,139
49788 :024,000,020,153,020,153,238
49794 :020,153,020,153,020,153,137
49800 :024,000,024,000,024,000,208
49806 :030,009,024,000,024,000,229
49812 :024,000,020,153,024,000,113
49818 :024,000,020,153,024,000,119
49824 :024,000,020,153,020,153,018
49830 :020,153,001,047,030,010,171
49836 :030,011,001,210,030,012,210
49842 :024,153,024,153,020,153,193
```

```
49848 :024,153,024,153,020,153,199
49854 :024,153,024,153,020,153,205
49860 :020,153,020,153,030,013,073
49866 :024,000,024,000,020,153,167
49872 :020,000,020,000,024,000,016
49878 :024,000,020,153,020,000,175
49884 :020,000,024,000,024,000,032
49890 :020,153,020,153,020,153,233
49896 :020,153,020,153,020,153,239
49902 :020,153,030,014,001,242,186
49908 :001,242,001,226,001,242,189
49914 :001,242,001,226,030,015,253
49920 :001,242,001,242,030,016,020
49926 :024,000,024,000,020,153,227
49932 :020,153,020,153,020,153,019
49938 :020,153,020,000,020,000,231
49944 :030,017,001,242,030,018,106
49950 :001,242,030,019,001,226,037
49956 :030,020,001,242,030,021,124
49962 :004,153,004,153,004,153,001
49968 :020,153,020,000,030,021,036
49974 :054,022,004,153,001,015,047
49980 :020,153,020,000,020,153,170
49986 :020,000,020,153,004,153,160
49992 :004,153,004,153,020,153,047
49998 :001,079,001,079,001,071,054
50004 :001,044,001,236,001,072,183
50010 :038,001,166,181,054,002,020
50016 :004,000,004,000,004,000,108
50022 :004,000,004,000,004,000,114
50028 :004,000,004,000,004,000,120
```



```
50034 :008,000,012,000,020,000,154
50040 :001,015,001,054,030,001,222
50046 :024,000,024,000,020,153,091
50052 :020,000,020,165,030,002,113
50058 :024,000,024,000,020,154,104
50064 :020,000,020,165,030,003,126
50070 :024,000,024,000,020,153,115
50076 :020,000,020,165,030,004,139
50082 :020,000,020,165,020,000,131
50088 :020,162,030,005,024,000,153
50094 :024,000,020,153,020,000,135
50100 :020,165,030,006,024,000,169
50106 :024,000,020,165,020,000,159
50112 :020,167,030,007,024,000,184
50118 :024,000,020,154,020,000,160
50124 :020,162,020,000,020,165,079
50130 :020,000,020,173,001,047,215
50136 :001,050,030,008,024,000,073
50142 :024,000,024,000,024,000,038
50148 :024,000,024,000,024,000,044
50154 :024,000,024,000,024,000,050
50160 :024,000,024,000,024,000,056
50166 :024,000,024,000,024,000,062
50172 :008,000,008,000,008,000,020
50178 :008,000,008,000,008,000,026
50184 :030,014,001,050,030,015,148
50190 :024,000,024,000,020,153,235
50196 :020,000,020,000,024,000,084
50202 :024,000,020,165,020,000,255
50208 :020,000,030,016,024,000,122
50214 :024,000,020,153,020,000,255
```

```
50220 :020,000,020,000,020,165,013
50226 :020,000,020,000,030,017,137
50232 :004,000,004,000,004,000,068
50238 :020,165,020,000,020,000,031
50244 :030,018,004,000,004,000,124
50250 :004,000,020,164,020,000,026
50256 :020,000,030,019,004,000,153
50262 :004,000,004,000,020,165,023
50268 :020,000,020,165,020,000,061
50274 :020,153,020,000,020,000,055
50280 :030,020,004,000,004,000,162
50286 :004,000,020,164,020,000,062
50292 :020,000,030,021,004,000,191
50298 :004,000,004,000,020,165,059
50304 :020,000,038,000,004,173,107
50310 :004,173,004,173,004,173,153
50316 :004,173,004,173,004,173,159
50322 :004,173,004,173,004,173,165
50328 :004,173,004,173,004,173,171
50334 :004,173,004,173,004,173,177
50340 :004,173,004,173,004,173,183
50346 :004,173,004,173,004,173,189
50352 :038,001,020,161,020,000,160
50358 :020,165,020,000,020,161,056
50364 :004,000,004,000,004,000,200
50370 :020,169,001,079,077,065,093
50376 :082,067,072,032,079,070,090
50382 :032,084,072,069,032,087,070
50388 :079,079,068,069,078,032,105
50394 :083,079,076,068,073,069,154
50400 :082,083,013,066,089,032,077
50406 :074,069,083,083,069,076,172
50412 :013,070,073,070,069,044,063
50418 :032,084,085,066,065,044,106
50424 :032,065,078,068,032,080,091
50430 :069,082,067,085,083,083,211
50436 :073,079,078,013,067,079,137
50442 :085,082,084,069,083,089,246
50448 :032,082,079,066,069,082,170
50454 :084,032,072,073,071,071,169
50460 :073,078,083,013,000,253,016
```

Program 7-6. ETAL.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49793
Filename: ETAL.MUS

```
49152 :248,000,244,000,116,000,096
49158 :066,000,086,025,001,012,196
49164 :001,144,001,132,001,008,043
49170 :030,001,001,102,054,003,209
49176 :024,239,024,161,024,162,146
49182 :020,227,052,162,024,174,177
49188 :024,239,024,161,020,162,154
49194 :020,161,020,174,052,172,129
49200 :030,002,024,239,024,161,016
49206 :024,162,020,227,052,162,189
49212 :024,174,024,239,024,161,194
49218 :020,239,020,174,020,239,010
```

```
49224 :052,173,001,015,001,246,048
49230 :024,239,024,174,024,173,224
49236 :020,239,052,170,024,174,251
49242 :024,173,024,172,020,169,160
49248 :020,172,020,170,052,173,191
49254 :024,239,024,174,024,173,248
49260 :020,239,052,170,024,235,080
49266 :024,170,024,169,020,172,181
49272 :020,169,001,047,020,170,035
49278 :052,181,001,242,001,047,138
49284 :020,170,052,181,001,192,236
49290 :016,162,001,144,030,003,238
49296 :054,002,024,170,024,171,077
49302 :024,173,020,170,052,183,004
49308 :024,182,024,247,024,169,058
49314 :024,172,020,173,020,181,240
49320 :024,170,024,171,024,173,242
49326 :020,170,052,183,001,160,248
49332 :016,169,016,180,001,144,194
49338 :024,170,024,171,024,173,004
49344 :020,170,052,183,020,247,116
49350 :024,170,020,172,024,169,009
49356 :020,172,016,173,016,173,006
49362 :024,173,020,173,020,173,025
49368 :024,173,024,173,024,173,039
49374 :001,015,016,170,016,170,098
49380 :024,170,024,170,024,170,042
49386 :024,170,024,170,024,170,048
49392 :024,170,024,170,030,002,148
49398 :001,098,084,173,052,173,059
49404 :001,079,001,118,118,026,083
```

```
49410 :134,002,050,132,001,039,104
49416 :001,006,008,000,008,000,031
49422 :012,000,001,022,000,154,203
49428 :020,146,000,157,024,149,004
49434 :000,157,080,149,024,149,073
49440 :000,156,052,148,000,158,034
49446 :020,150,000,153,024,145,018
49452 :000,154,024,146,000,154,010
49458 :080,146,024,146,030,001,221
49464 :000,154,020,146,000,157,021
49470 :024,149,000,157,080,149,109
49476 :024,149,000,156,024,148,057
49482 :000,157,024,149,000,156,048
49488 :024,148,001,047,000,154,198
49494 :020,146,000,231,024,223,218
49500 :000,219,024,211,000,154,188
49506 :080,146,024,146,001,018,001
49512 :000,154,020,146,000,154,066
49518 :024,146,000,231,024,223,246
49524 :000,154,080,146,024,146,154
49530 :001,047,030,002,001,135,082
49536 :002,200,086,020,054,002,236
49542 :020,149,024,138,080,138,171
49548 :024,138,052,137,020,140,139
49554 :020,137,080,138,024,138,171
49560 :020,149,024,138,080,138,189
49566 :024,138,052,141,020,140,161
49572 :020,142,080,141,024,141,200
49578 :001,015,001,047,016,076,070
49584 :001,212,066,176,086,000,205
49590 :166,179,118,000,134,000,011
```

```
49596 :054,002,030,003,052,141,214
49602 :020,138,020,141,024,138,163
49608 :052,140,020,137,052,149,238
49614 :052,141,020,138,052,141,238
49620 :024,140,024,141,052,142,223
49626 :052,140,052,141,020,138,249
49632 :052,141,052,207,020,142,070
49638 :052,140,008,141,001,015,075
49644 :008,076,001,114,001,079,003
49650 :001,134,166,108,001,039,179
49656 :001,059,019,176,001,002,250
49662 :001,051,001,071,003,000,125
49668 :166,012,030,002,054,002,014
49674 :020,154,048,231,080,166,197
49680 :024,166,020,164,080,162,120
49686 :024,162,020,162,048,239,165
49692 :052,227,016,161,080,162,214
49698 :024,162,001,015,001,047,028
49704 :016,174,166,179,086,000,149
49710 :001,212,066,176,054,002,045
49716 :030,003,052,151,020,149,201
49722 :052,146,016,150,024,149,083
49728 :052,159,052,151,020,149,135
49734 :020,146,024,159,052,145,104
49740 :020,148,052,150,052,151,137
49746 :020,149,052,146,052,156,145
49752 :020,153,052,158,008,159,126
49758 :001,015,008,158,001,130,151
49764 :001,079,034,069,084,032,143
49770 :065,076,034,013,066,089,193
49776 :032,072,065,082,082,089,022
49782 :032,066,082,065,084,084,019
49788 :013,013,013,000,020,153,080
```

Fundamentals of Music Theory

Sidplayer is a lot more than just a music playing program. It's also a complete system for entering and editing music. The Player is accompanied by an Editor which is so easy to use that you don't even have to know how to read sheet music. You may want to skip this section for now and come back to it later. This section offers the rudiments of elementary music theory—it presents the fundamental concepts that will help you get started.
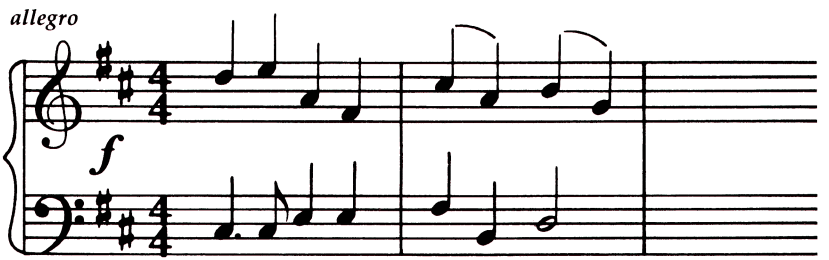
Notation. If a piece is good, the melody will stick in your mind, and you may find yourself humming the tune long after it's finished playing. Sometimes a song can be so good you

can't get it out of your mind. By hearing the song, you've learned it and can play it yourself. Songs such as those sung in native American ceremonies have been passed from generation to generation in just this way. As songs get longer and more complex, however, this method of communicating a song becomes less reliable. This is when it's necessary to make a permanent copy of the song on paper, which is the purpose of sheet music. Today, orchestras can faithfully reproduce the great symphonies of Beethoven. These symphonies have survived for nearly two centuries only because they were written down.

To express music on paper, a special form of notation has been developed. This notation is capable of describing every facet of a piece of music, from the order in which to play the notes to specifics, such as the style in which they are to be played.

Each group of five horizontal lines is called a staff. At the left edge of each staff is a clef symbol. The clef symbol for the top staff indicates that the staff is a *treble* clef. The bottom staff uses a different clef symbol and is called a *bass* clef. Together, the two staves form a *grand staff*, which is most often used for displaying notes. Figure 7-2 illustrates a grand staff.

Figure 7-2. Grand Staff



In the following text, the different characteristics of notes are introduced one at a time. As each characteristic is discussed, the method for expressing it in notation is also shown. Admittedly, music theory is an extremely complicated subject. What follows is only a simplified explanation of the essential concepts and isn't intended as a complete treatment. Once you understand what's presented here, however, you should be able to read a simple piece of sheet music.

Pitch. When an object is vibrating, its vibrations pass through the air and are detected by your ear as sound. Frequency is the measure of the number of vibrations per unit of time. The most common method of specifying a frequency is in terms of vibrations per second. Such a measurement is indicated by the unit *hertz*, abbreviated *Hz*.

The frequency of a sound is interpreted by your ear as a *pitch*. Faster rates of vibration produce higher pitches. Usually, the smaller an instrument, the higher the pitch it can produce. A piccolo can produce a very high pitch, whereas a tuba produces a very low pitch.

Although a wide range of frequencies can be detected by the human ear, only frequencies occurring at specific intervals are commonly used in music. Let's start with one of these pitches and label it C. This pitch has a frequency of 261.63 Hz, or 261.63 vibrations per second. The sequence of pitches continues, with pitches at the following intervals being named D, E, F, G, A, and B.

B 493.88 Hz
A 440.00 Hz
G 392.00 Hz
F 349.23 Hz
E 329.63 Hz
D 293.66 Hz
C 261.63 Hz (Start here)

When you listen to the sequence of pitches in order, they form what's called a *scale*, but the scale will seem incomplete. One final note, after the B, is needed to complete the scale. This note happens to be another C, related to the earlier C, but at a higher pitch. (The actual mathematical relationship is that the new C occurs at 523.25 Hz, exactly twice the frequency of the first.) It doesn't stop here, though. There's another D after the new C, and a second E after the new D, and so on. In fact, the scale repeats several times, both above and below the original C.

D 1174.70 Hz
C 1046.50 Hz
B 987.77 Hz
A 880.00 Hz
G 783.99 Hz
F 698.46 Hz
E 659.26 Hz

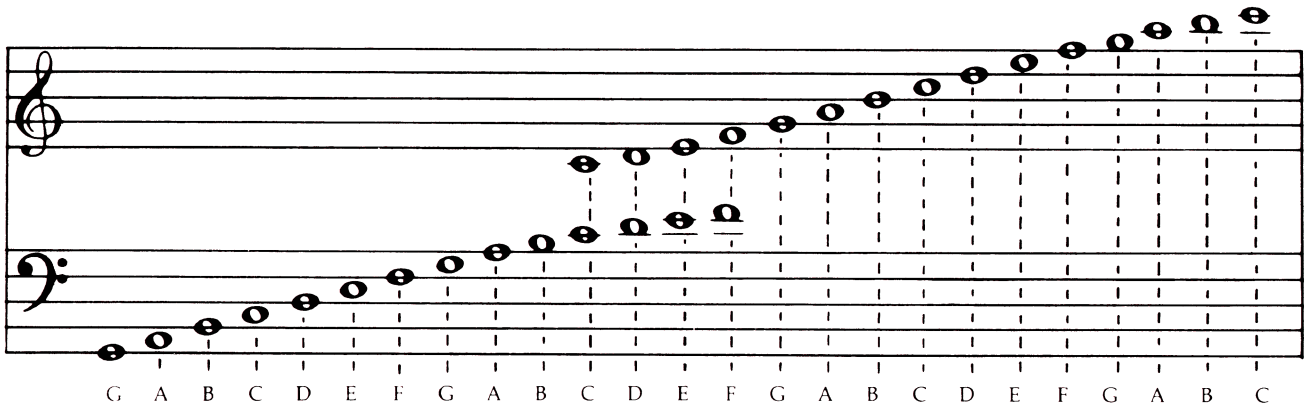
D 587.33 Hz
C 523.25 Hz
B 493.88 Hz
A 440.00 Hz
G 392.00 Hz
F 349.23 Hz
E 329.63 Hz
D 293.66 Hz
C 261.63 Hz (Original C)
B 246.94 Hz
A 220.00 Hz
G 196.00 Hz
F 174.61 Hz
E 164.81 Hz
D 146.83 Hz
C 130.81 Hz
B 123.47 Hz

The scale repeats with each C. By examining one sequence, from one C to the next, you can see that it consists of eight pitches. Collectively, these eight pitches are called an *octave*. To distinguish this set of pitches from the next, the first set is said to occur one octave lower than the second.

Just as the different pitches in an octave are labeled, so are the different octaves. However, instead of using a letter of the alphabet, a number is used. The piano key for the original C is found at about the middle of the keyboard. This C is called *middle C*, and begins octave 4. Other octaves are numbered relative to the octave containing middle C. The octave immediately above octave 4 is octave 5. The octaves which are of the most use musically are octaves 1 to 7.

In music notation, the pitch value of a note is represented by its vertical position when drawn on a staff. Thus, C5 (C of the fifth octave) is indicated by placing the note between the second and third lines of the treble staff. The next higher pitch, D5, is indicated by placing the note above the position for C5, except that this time the note is placed on the line. The positions for all notes alternate between being on a staff line or between staff lines, for the entire grand staff. See Figure 7-3 for an illustration.
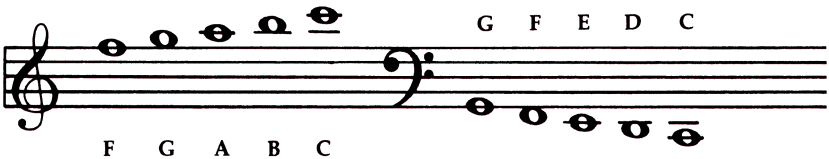
Figure 7-3. Grand Staff



One special case is middle C. The staff line for C4 is placed halfway between the treble and bass staves. The pitches around middle C must take this variation into account. The separation of the two staves creates some space used for messages and special symbols which give additional information to the performer.

Another special situation is when a note is so high or low in pitch that it goes off the grand staff. In such instances, additional staff lines, called leger lines (see Figure 7-4), are added. The pitch of notes drawn on leger lines is still determined in the normal way, by counting staff lines and seeing whether the note is placed on or between lines.

Figure 7-4. Leger Lines



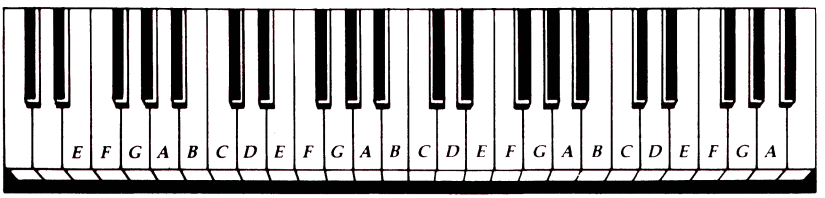
By using the grand staff and leger lines, eight octaves (octaves 0–7) can be displayed.

Sharps and flats. Eight octaves, each containing 7 different pitches, would seem to make a total of 56 pitches. Actually, there are some intermediate pitches between some of these notes. These are called sharps and flats.

C
B
A-sharp B-flat
A
G-sharp A-flat
G
F-sharp G-flat
F
E
D-sharp E-flat
D
C-sharp D-flat
C

A note is sharp if its pitch is a half step above normal. A note is flat if the pitch is a half step below the normal pitch. Notes that are not sharp or flat are said to be *natural*. Figure 7-5 shows some of the natural notes on the piano keyboard.

Figure 7-5. Piano Keyboard
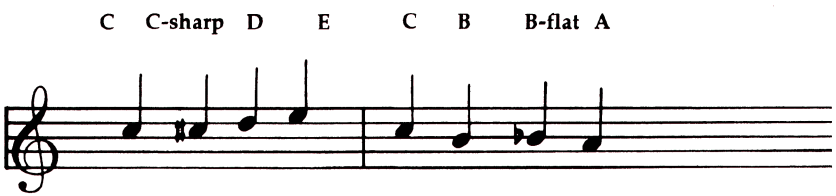


Two important observations should be made. First, every sharp note is equivalent to a flat note. C-sharp and D-flat both denote the same pitch. The difference lies in the viewpoint, whether the intermediate pitch is a half step above C or a half step below D.

Thus far, we've been using the words *sharp* and *flat* for accidental pitches. Another way to indicate that a note is sharp or flat is to use a special symbol. The symbol for a sharp note looks like a slanted pound sign (♯), while the symbol for a flat note looks something like a lowercase letter *B* (♭). The natural symbol is normally not used in front of natural notes.

To show that a note on the grand staff is a sharp or flat note, the appropriate accidental symbol is placed just before the note. The C-sharp and B-flat notes in Figure 7-6 are signified with these symbols. Sharps and flats indicated in this way last only one measure.

Figure 7-6. Accidentals



Including the sharps and flats, one octave consists of 12 different pitches. With eight octaves, the total is now 96 different pitches. Most songs use only notes that come from this palette of 96 pitches.

Key signatures. Just because there are 96 pitches available, does that mean that each one will be used in a song? No, a song may not play in every octave, nor every note within a particular octave. Perhaps it uses just a subset of the 12 pitches within one octave. The selection of notes is determined by the *key* in which the music is written.

The topic of pitch was introduced by starting with a C scale. This is a sequence of notes, starting on C, that continues for one octave. Let's examine the relationship of these notes to the 12 in the entire octave. If the distance between each of the 12 pitches is called a *half step*, the sequence of notes forming the C scale is determined by the following steps: whole, whole, half, whole, whole, whole, half—where a whole step equals two half steps.

C
B
A♯/B♭
A
G♯/A♭
G
F♯/G♭
F
E
D♯/E♭
D
C♯/D♭
C

Now apply that sequence of steps again, but this time start the scale at note A.

A
G♯
F♯
E
D
C♯
B
A (Start here)

This scale contains three sharp notes, as opposed to the earlier scale which contained none. The sharp notes replaced their natural counterparts. This scale is said to be written in the *key of A.* A song written in the key of A will normally use only this set of pitches in each octave. This means that we're back to a situation where we have to deal with only seven pitches per octave.

You can start a scale on any note, and for every starting note, there is a different combination of sharp or flat notes used. Here's another example, this time using flats:

B♭
A
G
F
E♭
D
C
B♭ (Start here)

This is the key of B-flat. The notes were determined by using the sequence of half and whole steps given earlier. The key of B-flat contains two flat notes, B-flat and E-flat. The notes B-natural and E-natural will not normally be used by a song written in the key of B-flat.

Table 7-1 is a complete listing of all the major keys. The keys with less than five sharps or flats are the ones used most often.

Table 7-1. Keys

Key	Notes								Sharps/Flats
C	C	D	E	F	G	A	B	C	0
G	G	A	B	C	D	E	F♯	G	1♯ (F♯)
D	D	E	F♯	G	A	B	C♯	D	2♯ (F♯, C♯)
A	A	B	C♯	D	E	F♯	G♯	A	3♯ (F♯, C♯, G♯)
E	E	F♯	F♯	A	B	C♯	D♯	E	4♯ (F♯, C♯, G♯, D♯)
B	B	C♯	D♯	E	F♯	G♯	A♯	B	5♯ (F♯, C♯, G♯, D♯, A♯)
F♯	F♯	G♯	A♯	B	C♯	D♯	E♯	F♯	6♯ (F♯, C♯, G♯, D♯, A♯, E♯)
C♯	C♯	D♯	E♯	F♯	G♯	A♯	B♯	C♯	7♯ (F♯, C♯, G♯, D♯, A♯, E♯, B♯)
F	F	G	A	B♭	C	D	E	F	1♭ (B♭)
B♭	B♭	C	D	E♭	F	G	A	B♭	2♭ (B♭, E♭)
E♭	E♭	F	G	A♭	B♭	C	D	E♭	3♭ (B♭, E♭, A♭)
A♭	A♭	B♭	C	D	E♭	F	G	A♭	4♭ (B♭, E♭, A♭, D♭)
D♭	D♭	E♭	F	G♭	A♭	B♭	C	D♭	5♭ (B♭, E♭, A♭, D♭, G♭)
G♭	G♭	A♭	B♭	C♭	D♭	E♭	F	G♭	6♭ (B♭, E♭, A♭, D♭, G♭, C♭)
C♭	C♭	D♭	E♭	F♭	G♭	A♭	B♭	C♭	7♭ (B♭, E♭, A♭, D♭, G♭, C♭, F♭)

If you study Table 7-1 carefully, you'll notice some patterns. For example, each key which contains sharp notes contains F♯. The key of G has F♯ as its only sharp note. The key of D keeps the F♯ but adds C♯. Each successive key adds one more sharp note, while retaining all the other sharp notes from before. This pattern works in the same way for keys containing flat notes, starting with the note B♭.

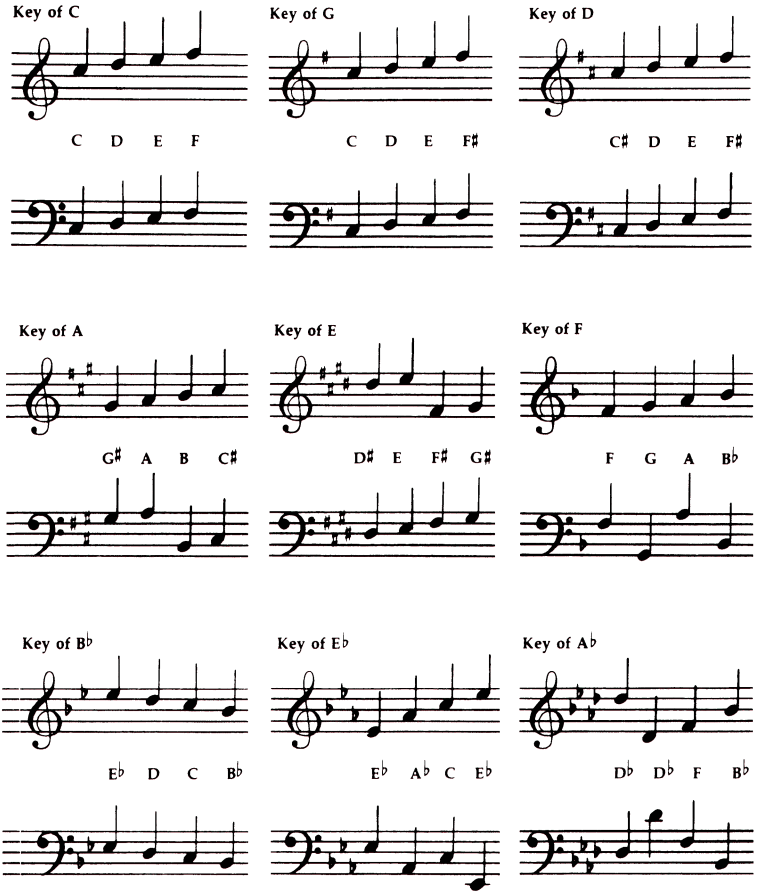
Most of time you can determine the key in which a piece of music is written by counting the number of sharp or flat symbols near the clef symbols on the grand staff. If no sharp or flat symbols appear there, the music is written in the key of C. If one sharp symbol is displayed, the piece is written in the key of G. Two sharp symbols mean that the key of D is to be used, and so on. Likewise, one flat symbol indicates the key of F, two indicate the key of B-flat, on up to seven flat symbols, which indicate the key of C-flat.

Just as the number of sharp or flat symbols is important, so is their position. The sharp symbol for F♯ is always placed on the line that designates note F. Furthermore, when a sharp symbol is put next to the clef symbol, it has the effect of automatically placing a sharp symbol in front of every note on that line. A sharp symbol on line F means that all notes placed on the grand staff in F positions are to be played as F-sharps. Of course, the same is true when flats are used. A flat symbol placed near the clef on the line for B means that all B notes should be played as B-flats.

Sharp and flat symbols placed after a clef symbol is called a *key signature*. The use of a key signature saves a lot of work when writing music, because it's no longer necessary to write a sharp or flat symbol in front of every note that needs one.

Figure 7-7 contains some examples of key signatures. Since all keys that contain sharps contain F♯, all of these keys have a sharp symbol at the F position. Each successive key adds a sharp symbol at a new position while retaining all the old ones. Also notice that a sharp or flat on one line affects not only the notes on that line, but the corresponding notes in the octaves above and below as well.

Figure 7-7. Key Signatures



Duration. The vertical position of a note on the grand staff determines its pitch. The horizontal direction of the staff indicates time. A sequence of notes is played in order from left to right, just as text is read from left to right. By putting the pitches together in a pleasing order, you'll create a melody, the basis for a song.

Pitch, however, is only one major characteristic of a note. Another important quality of a note is its duration. In a song, notes are not always played at the rate of one note every beat. Sometimes a note may be played for two beats. Other times, two notes might be played within the span of one beat, meaning that each note is half a beat long. Thus, every note on the staff is going to have to specify not only its pitch, but also its duration in terms of beats.
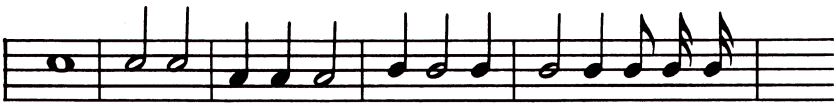
The duration of a note is indicated by its shape. The standard note you've been using thus far is formally called a *quarter note,* and is drawn with a stem and a filled-in oval at the bottom. If we assume a quarter note plays for a duration of one beat, then twice that length, two beats, is indicated by a *half note,* which looks like a quarter note except that the oval is not filled in. Twice the length of a half note is a *whole note,* which plays for four beats and looks like a half note without a stem.

In the other direction, for durations less than one beat, the symbol for a quarter note is used, but flags are added at the top of the stem. An *eighth note* plays for half a beat and has one flag. A *sixteenth note* has two flags. Four sixteenth notes are equal in duration to one quarter note. Thirty-second and sixty-fourth notes do exist, but they're not used very often. Figure 7-8 shows these notes, and how they're written. Take a moment to look over it.

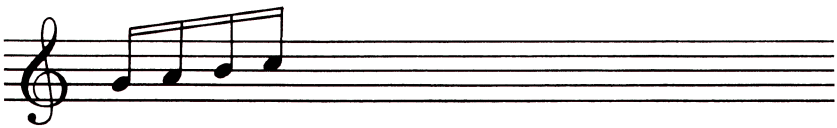
Figure 7-8. Duration
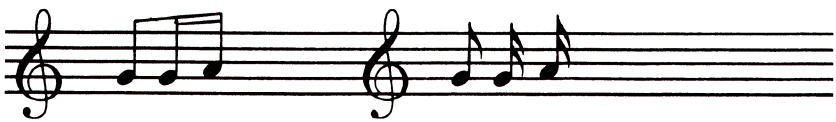


The following combinations are all equivalent in duration to one whole note.



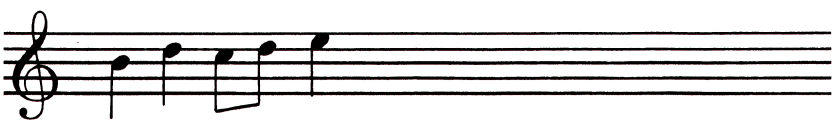
One convention in displaying durations less than one beat is to combine notes of equal duration in sets. Two eighth notes can be drawn by extending the flag from the first one to the top of the stem of the second one. This can also be done with sixteenth notes, except that two lines connect the tops of the stems, because sixteenth notes have two flags.



Notes of different durations can be combined. The notes shown on the left are the same as those shown on the right.
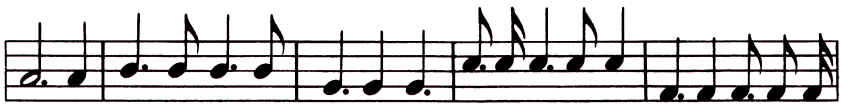


Notes can even be drawn upside down. This is done only when the notes would appear near the top of a staff. The oval portion of the note stays in the same place, so the pitch is not affected. Drawing a note upside down does not affect its duration either.



Dotted notes. With just a few different durations, it's possible to create a variety of different rhythms. But there are still some durations that cannot be expressed using only the notes you've seen so far. For example, how do you show that a note should be played for three beats? Situations like this require the use of dotted notes.

When a dot is placed after a note, it means that the note should be played for one and a half times the normal duration. Given a dotted half note, the half note portion is two beats, and half of that is one beat, for a total of three. A dotted whole note plays for six beats (four beats for the whole note and two beats for the dot). And a dotted quarter note? That plays for one and a half beats.
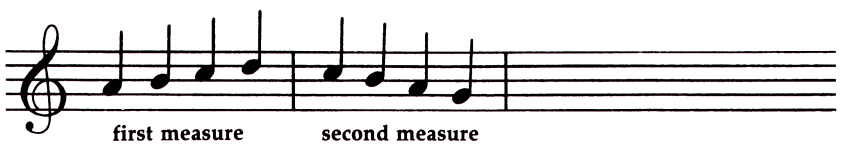
Using the dot, here are some more note combinations which total four beats.



Notice that the dot always appears to the right of the note. If you see a dot placed above or below a note, it has a different meaning and does not affect the note's duration. These dot placements are explained a bit later.

Measures. A song is just a long sequence of notes of different pitches and durations. To make it easier to deal with pitches, they're separated into groups called octaves. Likewise, to make it easier to work with a sequence of notes, the notes are often divided into groups called measures, with each measure consisting of the same number of beats. A common number of beats per measure is four.

In sheet music, a measure is formed by placing a vertical line called a *bar* between each group of notes on the staff.



Measures are used mainly for organization and reference. It's much easier to refer to a note as being the second note in the twenty-third measure than it is to refer to the one hundred forty-seventh note.

Each measure must have the same total duration. Since this total duration is often four beats, or one whole note, you can see why the note for one beat is called a quarter note.

Tempo. You've seen that the length of a note is expressed in beats, and that notes can be organized into groups called measures, which all have the same number of beats. The question is, how long is a beat?

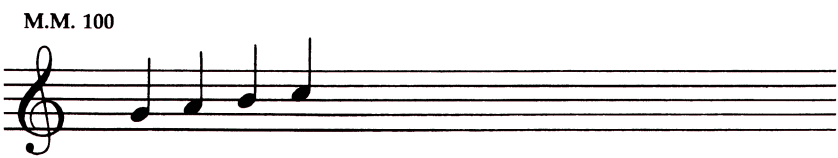
A beat is a unit of time. The shorter the amount of time for each beat, the faster they'll occur. If the time is longer, the beats won't occur as often.

The rate at which the beats occur is called the *tempo*. The faster the tempo, the more quickly the notes are played. At a slower tempo, each beat lasts for a longer amount of time. Another way to look at it is to say that for a fixed amount of time, such as one minute, there will be more beats at a fast tempo than there will be at a slow tempo.

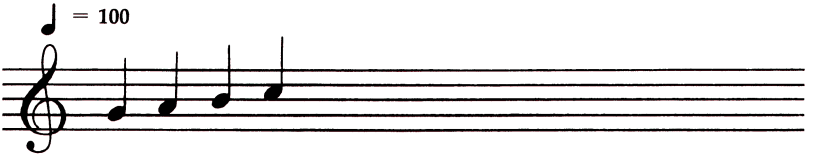
The relationships of quarter notes to half notes and other notes still hold; a half note will always be twice as long as a quarter note. It's just the actual time lengths that change.

The standard method of measuring a tempo is to specify the beats per minute. An average tempo is about 100 beats per minute. A tempo of 60 means that there will be 1 beat every second, which is rather slow. A tempo of 150 is more than 2 beats every second, which is relatively fast.

The tempo is a very important part of a song. A beautiful melody can be ruined if it's played too fast or too slow. Therefore, sheet music usually indicates the proper tempo. At the top of the sheet music you should find the letters *M.M.*, followed by a number. This number indicates the beats per minute, and defines the tempo that should be used in playing the song.



Another way to show the tempo is to replace the M.M. with a quarter note and an equal sign. The quarter note is used because it represents one beat. Either way works just as well.



Using a number for the tempo is convenient because it can be used when setting a metronome. But there's another method of specifying the tempo. Often you'll see words such as *adagio* or *allegro* on sheet music. What do they mean? Just like numbers, they specify tempo. Here's a list of most of the tempo terminology, in order from slowest to fastest:

lento	Very slow
grave	Slow, solemn
largo	Broad
adagio	Leisurely
andante	Walking
andantino	A little faster walk
moderato	Moderate
allegretto	Rather fast
allegro	Fast
vivace	Lively
presto	Very fast
prestissimo	As fast as possible

These values may be modified by one of the following words:

molto	Very
meno	Less
piu	More

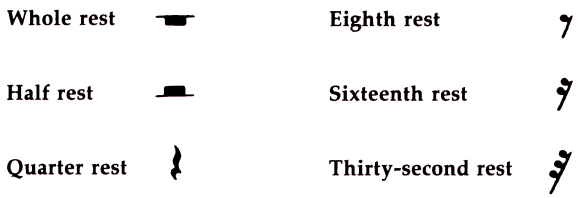
The tempo notation *moderato* indicates medium speed, which roughly corresponds to M.M. 100.

Rests. Pitch and duration are the two most important parts of a note. There's something similar to a note, however, which has only duration and no pitch. It's called a rest. For the specified amount of time, no tone is produced.
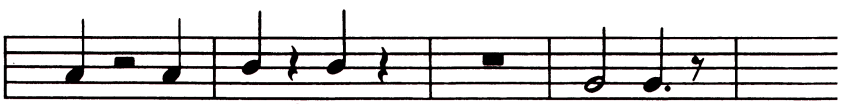
There's a different rest symbol for each duration. Because the idea of pitch does not apply, the vertical position of a rest does not matter, so it's usually placed in the middle of the staff. A whole rest is drawn as a small block placed right below

the second staff line. A half rest looks the same except that the block is placed on top of the third staff line. A quarter rest is a symbol that defies description. Eighth, sixteenth, and thirty-second rests are all drawn as slanted stems with the proper number of flags to the left of the stem. Figure 7-9 shows the rest symbols.
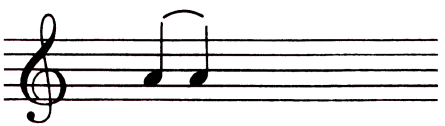
Figure 7-9. Rests



Here are some combinations of notes and rests. They've been divided into measures to show that each group has a total duration of four beats.
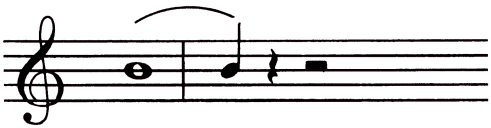


Ties and slurs. Another special symbol is called the *tie* symbol. Two notes are tied together when connected by a symbol that looks like a curved line or arc. The tie means that the two notes are to be played together as one long note, with no break in volume. Thus, two quarter notes tied together will play just like a half note.
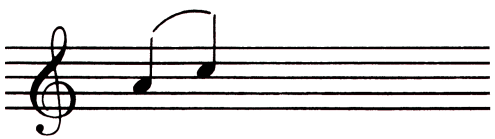


The reason for using ties is that the effect of a tie can extend across one measure or beat into another. In the following two sequences of notes, each sequence sounds the same when played, but the first one cannot be divided into measures.
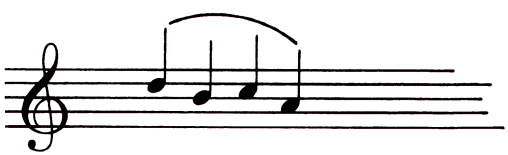


The next example demonstrates the use of a tie to create a note five beats long.



Another application of the tie symbol is to connect notes of different pitches. In this case, the tie is called a *slur* and may be used within a measure as well as between two measures. Playing two quarter notes slurred together is like playing a half note that changes its pitch halfway through playing.



Sometimes a very long tie symbol is used over a long stretch of notes. This produces a smooth, legato effect when the notes are played.



Volume. Yet another major characteristic of a note is its volume. Some parts of a song can be emphasized if they're played loudly. Other parts may be subdued by being played quietly. The level of loudness or softness of a piece of music is referred to as *dynamics.*

Dynamics are indicated on sheet music by letters which appear between the two staves of the grand staff. These letters are listed below, in order from loudest to softest.

fff	(fortissimo)	**mp**	(mezzo piano)
ff	(fortissimo)	**p**	(piano)
f	(forte)	**pp**	(pianissimo)
mf	(mezzo forte)	**ppp**	(pianississimo)

These volume levels range from very very loud (fff) to very very soft (ppp). Extremes such as ffff or pppp are not used very often.

The term *dynamics* should not be confused with the concept of an envelope, which describes the changes in volume as an individual note is played.
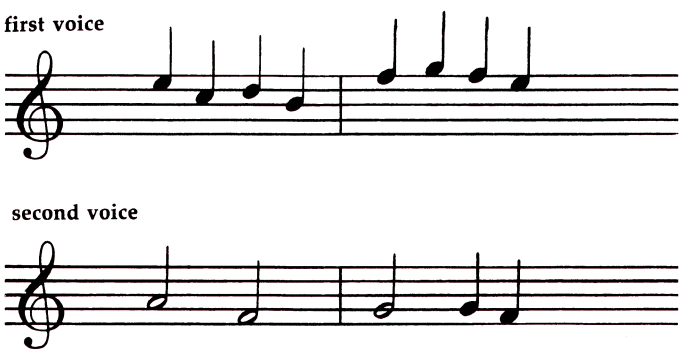
Multiple voices. You've seen the essential characteristics of individual notes and how notes can be combined into groups called measures. The music can then be sung or played on an instrument. The next step is to have several voices or instruments playing at the same time.

A set of notes for one singer or one instrument is generally referred to as one *voice*. With two voices playing simultaneously, one voice can play a melody while the other voice plays a bass part to give a little more body to the song. If a third voice is added, it can be used for harmony or for percussion effects, like drums or cymbals.

Each voice is independent of the others and can play its own notes of different pitches and durations. This brings up only one problem; there must be a way of keeping the voices synchronized. They should start together and end together.
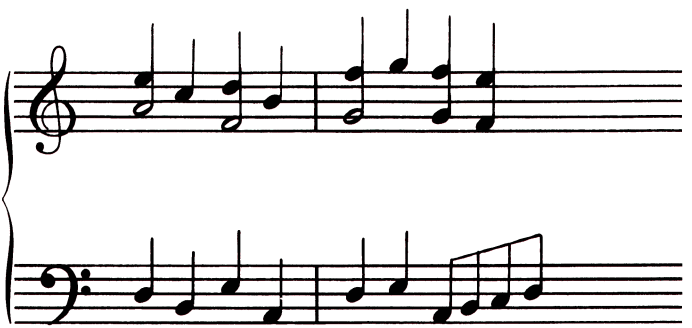
Fortunately, the concept of tempo and the use of measures solve this problem. The voices may be independent, but one thing they must have in common is the tempo. The tempo establishes a beat which all voices can follow. The notes in each voice are divided into measures. Then, even though the durations within a measure may differ for each voice, at least the voices will always be on the same measure at any given instant. Figure 7-10 shows this concept.

Figure 7-10. Multiple Voices, Example 1



The horizontal direction of the grand staff corresponds to time. Because multiple voices are synchronized according to tempo, it's possible to represent more than one voice on just one grand staff. Within each measure, the notes for all of the voices are drawn. It's a rather simple matter to determine which notes go with which voice. Usually, the topmost notes are for the first voice, the notes below those are for the next voice, and so on, with the bottommost notes assigned to the last voice.

Figure 7-11. Multiple Voices, Example 2



Summary

- Notes are shown on a grand staff which consists of the treble and bass staves. Each staff has its own clef symbol.
- The most important characteristics of a note are its pitch and duration.
- Pitches occur in groups called octaves. Each octave contains the pitches labeled C, D, E, F, G, A, and B, plus five intermediate pitches called sharps and flats.
- Only some of the pitches in each octave are used, depending on the key in which the music is written.
- Pitch is indicated by the vertical position of a note on the grand staff. The note may be drawn on a staff line or between staff lines.
- An accidental (a sharp or flat symbol) may be placed immediately before the note to indicate that the note is a sharp or a flat.

- A key signature is indicated by placing sharp or flat symbols near the clef symbol.
- Durations are specified in terms of beats. Common units of duration are whole note, half note, quarter note, and so on, down to sixty-fourth note.
- Durations are indicated by the shapes of the notes.
- Placing a dot after the note means that its duration should be one and a half times normal.
- Notes are organized into groups of equal total duration called measures.
- Measures are indicated by a single vertical line, called a bar, that crosses the staff lines.
- The rate at which the beats occur is called the tempo and is measured in terms of beats per minute.
- The tempo is indicated by an M.M. marking at the beginning of the music.
- A rest is similar to a note in that it plays for a certain duration, but it produces no tone so it has no pitch.
- Rests are indicated by special symbols, one for each duration.
- An arc-type symbol connecting two notes is used to indicate a tie or slur.
- The general volume level of a piece of music is specified by dynamics.
- Dynamics are indicated by letters that appear between the two staves.
- The notes for one singer or one instrument form one voice.
- Several voices can be represented on the same grand staff.
- When played, multiple voices stay synchronized because they share the same tempo.

The Editor

You've been introduced to the Sidplayer, which plays songs. You've even seen some of the elements of music. But in order to play your own songs or tunes, you have to create them.

That's what the "Editor" is for. It's used to enter, edit, and debug up to three voices of a song. The program contains features which make music editing easy. Since there are so many features and the program can appear to be rather intimidating the first time you use it, be assured that you'll see exactly how each feature works.

The first step, however, is to create a working editor program. It's in two parts.

Type in Program 8-1, "EDITOR," using the "Proofreader" from Appendix D. Save it using the filename EDITOR. If you're using the Datassette, change the statement DN=8 in line 1 to DN=1 before you save the program. Be sure to make note of the starting and ending number indicated by the tape counter.

Program 8-1. EDITOR

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
1 DN=8:GOTO 9200:REM EDITOR77 BY HARRY BRATT 10/26
  /84'                                          :rem 250
5 JV=PEEK(56320):JB=1-(JV AND 16)/16:JV=JV AND 15:
  RETURN                                        :rem 180
8 POKE CY,3:PRINT:POKE CX,18-8*AC:PRINTAC$(AC+1):R
  ETURN                                         :rem 143
10 PRINTCHR$(19)CHR$(146)C1$;:POKE CX,25:PRINT"MEM
   :"FMCHR$(157)"{6 SPACES}":RETURN              :rem 5
12 POKE 54283,129:POKE 54283,32:RETURN          :rem 122
19 V0=NT:V1=AC:V2=OC:V3=LN:V4=RS+2*SP:V5=TY:V6=1
                                                 :rem 47
20 H=PEEK(MM+CN):IFHAND3 THEN SP=1:RS=0:TY=0:RETUR
   N                                            :rem 196
22 SP=0:LN=7-(HAND31)/4-.5*((HAND32)=32):TY=-((HAN
   D64)=64)                                     :rem 185
24 H=PEEK(MM+CN+1):IF H=0 THEN RS=1:GOTO 28
                                                :rem 176
26 RS=0:NT=(HAND7)-1:OC=7-(HAND56)/8:AC=(HAND192)/
   64-2                                          :rem 58
28 RETURN                                        :rem 74
```

```
30 POKE CY,21:PRINTC0$:POKE CX,X*5:H=PEEK(MM+CN):I
   F H AND 3 GOTO 36                                :rem 70
31 IF RS THEN PRINT"(R)"Y$;:GOTO 33                 :rem 204
32 PRINTC0$NT$CHR$(ASC(AC$)+162*(ASC(AC$)=194))OC$
   Y$;                                              :rem 201
33 PRINTLN$LM$TY$(TY);                              :rem 120
34 IF V6 THEN NT=V0:AC=V1:OC=V2:LN=V3:RS=V4AND1:SP
   =(V4AND2)/2:TY=V5:V6=0                           :rem 170
35 RETURN                                           :rem 72
36 POKE 251,PEEK(MM+CN+1-(HAND2)/2):POKE 252,-((PE
   EK(MM+CN)AND3)<>1):SYS 1232                      :rem 43
37 PRINTS$(PEEK(252)+1)Y$CHR$(157);:QI=ASC(MID$(S$
   ,PEEK(252)+1))-65                                :rem 255
38 IF ABS(V(QI))<90 THEN QJ=PEEK(251):GOSUB80:GOTO
   34                                               :rem 24
39 QJ=PEEK(MM+CN+1)+256*PEEK(251):GOSUB 80:GOTO 34
                                                    :rem 104
40 POKE 1216+SC,PK:SC=3:POKE 1219,1:PK=14:GOTO 510
                                                    :rem 56
45 GOSUB 5:IF JB=0 THEN RETURN                      :rem 239
47 GOSUB 5:IF JB THEN 47                            :rem 15
48 JB=1:RETURN                                      :rem 128
50 FOR K=0 TO 6:AC%(AK%(ABS(K+6*(KY<0))))=(K<ABS(K
   Y))*SGN(KY):NEXT:RETURN                          :rem 235
60 LN$=CHR$(196+LN):LM$=CHR$(46+14*(LN=INT(LN))):R
   ETURN                                            :rem 218
65 NT$=CHR$(67+NT+7*(NT>4)):AC$=CHR$(194+AC):OC$=R
   IGHT$(STR$(OC),1):RETURN                         :rem 64
70 K=J-INT(J/256)*256:J=(J-K)/256:RETURN            :rem 57
80 IF QI<>2 GOTO 84                                 :rem 226
82 IF QJAND1 THEN PRINT-(11-INT(QJ/16)+12*(QJ/2AND
   7));:RETURN                                      :rem 50
83 QJ=INT(QJ/16)+12*(7-(QJ/2AND7)):GOTO 87          :rem 86
84 IFQI=9 THEN QJ=(QJAND3584)/2+(QJAND255)+((QJAND
   256)=256)*2048:GOTO 87                           :rem 24
85 IF QI>10 THEN ON QI-10 GOTO 88,88,90,90,91,92
                                                    :rem 197
86 IF V(QI)<0 THEN QJ=QJ+256*(QJ>127)               :rem 2
87 PRINT MID$(STR$(QJ),2+(QJ<0));:RETURN            :rem 138
88 IF QJ=0 THEN PRINT"N";:RETURN                    :rem 170
89 PRINTFW$(QJ-1,QI-11);:RETURN                     :rem 179
90 PRINTNU$(QJ,QI-13);                              :rem 59
91 RETURN                                           :rem 74
92 QJ=QJ-256*(QJ=0):QJ=INT(14400/QJ):GOTO 87
                                                    :rem 30
200 REM INPUT SBR                                   :rem 237
210 A$="":LH=0:PRINTCHR$(164)CHR$(157);             :rem 146
215 GET B$:IF B$="" GOTO 215                        :rem 95
220 B=ASC(B$):IF B=13 OR B=141 THEN PRINT" ";:RETU
    RN                                              :rem 88
```

```
230 IF B<>20 OR LH=0 GOTO 240                    :rem 179
232 LH=LH-1:A$=LEFT$(A$,LH-(LH=0))               :rem 122
235 PRINTCHR$(157)"{2 SPACES}"CHR$(157)CHR$(157)CH
    R$(164)CHR$(157);:IF LH=0 THEN A$=""         :rem 46
237 GOTO 215                                     :rem 109
240 IF LH=MX-(ASC(A$+" ")=45) GOTO 215           :rem 140
245 IF B=45 AND (FA AND 1) AND LH=0 GOTO 270
                                                 :rem 103
250 IF B>47 AND B<58 GOTO 270                    :rem 161
260 IF B<32 OR B>90 OR (FA AND 2)=0 GOTO 215
                                                  :rem 80
270 A$=A$+B$:LH=LH+1:PRINTB$;:POKE 212,0:PRINTCHR$
    (164)CHR$(157);:GOTO 215                     :rem 215
500 REM ** MAIN JOYSTICK PART **                 :rem 237
501 IF NN<0 THEN NT=0:OC=4:LN=3:RS=0:SP=0:GOTO 507
                                                 :rem 145
502 IF GK=0 GOTO 507                             :rem 250
503 TN=CN:FOR X=(TN/2-3)*(TN<5) TO 7-(4-(NN-TN)/2)
    *-((NN-TN)<7)                                :rem 173
504 CN=TN-6+2*X:GOSUB19:GOSUB60:GOSUB65:GOSUB30:NE
    XT:CN=TN+2*(TN>NN)                            :rem 72
505 GOSUB20:CN=TN                                 :rem 38
507 GOSUB 555:GOSUB 545                           :rem 19
509 LQ=0                                         :rem 168
510 GOSUB5:GET K$:IF K$="" THEN K$=" ":IF JV=15 AN
    D JB=0 GOTO 510                               :rem 83
511 IF JB OR ASC(K$)=13 GOTO 600                  :rem 90
512 IF JV<>15 THEN POKE 198,0:GOTO 530            :rem 14
513 ON ABS(ASC(K$)-132) GOTO 8000,7000,4100,4000,4
    050,4060                                     :rem 119
514 K=-2*(K$="+")-3*(K$="£")-4*(K$="-"):IF K>0 TH
    EN AC=K-3:GOTO 554                           :rem 101
515 IF K$>"@" AND K$<"H" THEN NT=ASC(K$)-67:NT=NT-
    7*(NT<0):RS=0:SP=0:GOTO 553                  :rem 242
516 IF K$>"/" AND K$<"8" THEN OC=VAL(K$):RS=0:SP=0
    :GOTO 553                                    :rem 140
517 IF K$>"G" AND K$<"X" THEN K=KL%(ASC(K$)-72):IF
     K>-1 THEN LN=K:GOTO 542                     :rem 126
518 IF K$="8" THEN LN=2:GOTO 542                 :rem 155
519 IF K$="." AND LN>0 AND LN<6 THEN LN=INT(LN)+.5
    *(1-(LN*2)AND1):GOTO 542                     :rem 162
520 IF (ASC(K$) AND 127)=20 GOTO 900             :rem 236
521 IF K$="R" THEN RS=1-RS:TY=0:GOSUB555:GOTO 510
                                                  :rem 51
522 IF K$="/" THEN TY=1-TY:RS=0:GOSUB555:GOTO 510
                                                  :rem 25
523 IF ASC(K$)=17 OR ASC(K$)=29 THEN JV=28-ASC(K$)
    :GOSUB650:JV=15                               :rem 81
524 IF K$="M" THEN LQ=1:GOTO 740                 :rem 175
```

```
525 IF ASC(K$)=221 OR ASC(K$)=219 THEN K=220-ASC(K
    $):LQ=1:GOTO 725                              :rem 10
526 IF ASC(K$)=147 GOTO 980                       :rem 118
529 GOTO 510                                      :rem 111
530 IF JV>12 THEN RS=0:GOTO 550                   :rem 182
535 K=.5*((JV>8)-(JV<8)):LN=LN+K:IF LN=.5 OR LN=6.
    5 THEN LN=LN+K                                :rem 71
540 LN=LN+7.5*(LN>7)-7.5*(LN<0)                   :rem 161
542 GOSUB12:IF SP GOTO 554                        :rem 193
543 GOSUB 545:GOTO 510                            :rem 195
545 POKE CY,17:PRINT:POKE CX,21:IF SP THEN PRINTC0
    $"--"SPC(2)"-------":RETURN                   :rem 182
547 POKE CY,17:PRINT:POKE CX,21:GOSUB 60:PRINT C0$
    LN$LM$SPC(2)LN$(LN):RETURN                    :rem 126
550 NT=NT+(JV=13)-(JV=14)                         :rem 152
552 IF NT<0 OR NT>6 THEN OC=OC-(JV=14)+(JV=13):NT=
    3*(1-SGN(NT))                                 :rem 253
553 AC=AC%(NT):IF SP=0 THEN GOSUB 555:GOTO 510
                                                  :rem 42
554 SP=0:GOSUB555:GOSUB545:GOTO 510               :rem 104
555 F1=0:POKE CY,9:PRINTC0$:POKE CX,16:IF RS THEN
    {SPACE}PRINT"(R)":GOTO 567                     :rem 8
556 IF SP THEN PRINT"---":GOTO 567                :rem 110
557 IF OC>7 OR (OC=7 AND NT=6 AND AC=-1) THEN OC=7
    :NT=6:AC=0                                    :rem 184
558 IF OC<0 OR (OC=0 AND NT=0 AND AC=1) THEN OC=0:
    NT=0:AC=0                                     :rem 105
559 POKE 780,(1-RS)*(NT+1+8*(7-OC)+64*(AC+2)):SYS
    {SPACE}1396                                    :rem 59
560 POKE CY,9:POKE 54276,64:GOSUB 65:PRINT:rem 190
565 POKE CX,16:PRINTC0$NT$" "AC$:POKE CY,14:PRINT:
    POKE CX,17                                     :rem 2
567 POKE CY,14:PRINT:POKE CX,17:IF RS OR SP THEN P
    RINT"-":GOTO 576                              :rem 81
568 PRINTOC$                                      :rem 230
576 POKE CY,19:PRINT:POKE CX,9:PRINTC3$CHR$(146-12
    8*RS)"REST"CHR$(146);                         :rem 194
577 PRINT" <--SELECT--> "CHR$(146-128*TY)"TIE":IF
    {SPACE}F1 THEN RETURN                          :rem 99
579 IF AL=AC AND RS=0 AND SP=0 THEN 582           :rem 24
580 PRINTC3$:H=AC:AC=AL:GOSUB 8:AC=H:IF RS OR SP T
    HEN AL=2:RETURN                               :rem 187
582 AL=AC:PRINTC1$:GOSUB 8:RETURN                  :rem 88
600 REM ** BUTTON PRESSED **                       :rem 20
605 FG=0                                          :rem 149
610 GOSUB 5:IF JB=0 THEN 680                      :rem 219
620 IF JV=15 THEN 610                              :rem 48
625 FG=1:IF JV<13 GOTO 610                        :rem 113
632 POKE 1216+SC,PK                                :rem 28
635 K=(JV=13)-(JV=14):SC=SC+K:SC=SC-K*(SC=5):SC=SC
    +6*((SC>6)-(SC<1))                            :rem 138
```

```
640 PK=PEEK(1216+SC):POKE 1216+SC,-(SC=3 OR SC=1)
                                                :rem 188
645 FOR K=1 TO 90:NEXT:GOTO 610                 :rem 206
648 GOSUB650:GOSUB45:IF JB THEN POKE 198,0:GOTO 40
                                                :rem 104
649 GOTO 648                                    :rem 126
650 K=-(JV<8):IF(CN+(K=0))<0 OR(CN>NN AND K=1) OR
    {SPACE}JV>11 OR IN THEN FG=1:RETURN          :rem 70
651 IF K=0 THEN POKE CY,21:PRINT:POKE CX,35:PRINTZ
    $(0);                                       :rem 204
652 POKE CY,21:PRINT:PRINTZ$(1-K);:X=K*7        :rem 179
656 CN=CN-2+4*K:IF CN-2=NN THEN FG=1:RETURN :rem 9
658 TN=CN:IF SQ=0 THEN GOSUB 20:GOSUB 545:GOSUB 55
    5                                           :rem 175
660 CN=CN-6-14*(JV<8):IF CN<0 OR CN>NN GOTO 670
                                                 :rem 39
662 GOSUB19:GOSUB60:GOSUB65:GOSUB30             :rem 234
670 FG=1:CN=TN:GOSUB20:RETURN                   :rem 120
680 IF FG THEN ON SC GOTO 648,800,510,750,510,700
                                                :rem 202
682 IF (FM=0 AND CN>NN) OR (LN=7 AND (TY OR RS)) G
    OTO 510                                     :rem 164
684 IN=IN+(IN>0):POKE MM+CN,4*(7-INT(LN))-32*(LN>I
    NT(LN))+64*TY                               :rem 137
686 POKE MM+CN+1,(1-RS)*(NT+1+8*(7-OC)+64*(AC+2)):
    GOSUB 60:GOSUB 65                           :rem 136
687 IF M1 THEN AC%(NT)=AC                        :rem 62
688 IF NN<CN THEN NN=NN+2:FM=FM-1:GOSUB10 :rem 181
689 X=3:GOSUB30:POKE 780,PEEK(MM+CN+1):SYS 1396:PO
    KE CY,21:PRINT                              :rem 79
690 POKE 54276,64:PRINTZ$(0);:TY=0:F1=1:GOSUB576:C
    N=CN+2                                      :rem 50
691 IF NN-CN<8 OR IN>4 GOTO 510                 :rem 118
692 TN=CN:CN=CN+8:X=7:GOSUB19:GOSUB60:GOSUB65:GOSU
    B30:CN=TN:GOTO 510                           :rem 76
700 REM ** KEY CHANGE **                        :rem 178
710 GOSUB 45:IF JB GOTO 40                      :rem 115
715 IF JV>12 AND JV<15 GOTO 740                  :rem 86
720 K=(JV=11)-(JV=7):IF K=0 THEN 710            :rem 139
725 KY=KY+K:KY=KY+15*(SGN(KY)*(ABS(KY)=8)):GOSUB12
                                                 :rem 25
730 POKE CY,1:PRINT:POKE CX,10:PRINTC1$ABS(KY)CHR$
    (157)CHR$(193-2*(KY<0));                     :rem 71
732 PRINT" ("KY$(KY+7)")":GOTO 745               :rem 52
740 M1=1-M1:GOSUB12:POKE CY,1:PRINT:POKE CX,30:PRI
    NTC1$MID$("ON OFF",4-M1*3,3)                 :rem 60
745 GOSUB 50:ON LQ+1 GOTO 710,509               :rem 240
750 REM ** ACCIDENTAL CHANGE **                 :rem 150
760 RS=0:GOSUB 45:IF JB GOTO 40                 :rem 196
770 K=(JV=7)-(JV=11):IF K=0 THEN 760            :rem 149
```

```
775 AC=AC+K:AC=AC+3*(SGN(K))*(ABS(AC)=2):IF SP=Ø T
    HEN GOSUB 555:GOTO 76Ø                    :rem 6
78Ø SP=Ø:GOSUB 555:GOSUB 545:GOTO 76Ø       :rem 112
8ØØ REM ** REST <--> TIE **                   :rem 24
81Ø GOSUB 45:IF JB GOTO 4Ø                   :rem 116
815 IF JV<>11 GOTO 825                       :rem 129
82Ø RS=1-RS:TY=Ø:GOTO 84Ø                    :rem 226
825 IF JV<>7 GOTO 81Ø                         :rem 81
83Ø TY=1-TY:RS=Ø:GOTO 84Ø                    :rem 235
84Ø IF SP=Ø THEN GOSUB 555:GOTO 4Ø            :rem 96
845 SP=Ø:GOSUB555:GOSUB545:GOTO 4Ø            :rem 57
9ØØ REM ** INS/DEL **                         :rem 19
91Ø K=-(ASC(K$)=2Ø)                          :rem 214
92Ø IF CN>NN OR (FM=Ø AND K=Ø) THEN ON SQ+1 GOTO 9
    7Ø,7375                                  :rem 185
923 GOSUB12:IN=IN-K*2+1:IN=IN-(IN<Ø)         :rem 124
925 IF K=Ø THEN POKE CY,21:PRINT:POKE CX,35:PRINTZ
    $(Ø);                                    :rem 2Ø8
927 POKE CY,21:PRINT:POKE CX,15:PRINTZ$(1-K);
                                             :rem 211
93Ø IF K GOTO 939                             :rem 84
932 FOR I=Ø TO 2 STEP 2:J=MM+CN+I:GOSUB 7Ø:POKE 25
    1+I,K:POKE 252+I,J:NEXT I                :rem 196
936 J=NN-CN+2:GOSUB 7Ø:POKE 781,K:POKE 782,J:SYS 1
    Ø24:NN=NN+2                               :rem 13
938 FM=FM-1:ON SQ+1 GOTO 97Ø,7375             :rem 23
939 IF CN=NN GOTO 95Ø                        :rem 117
94Ø FOR I=Ø TO 2 STEP 2:J=MM+CN+2-I:GOSUB 7Ø:POKE
    {SPACE}251+I,K:POKE 252+I,J:NEXT I        :rem 34
942 J=NN-CN:GOSUB 7Ø:POKE 781,K:POKE 782,J:SYS 1Ø2
    4                                        :rem 161
95Ø NN=NN-2:FM=FM+1:IF NN-CN<8 OR IN>4 THEN ON SQ+
    1 GOTO 97Ø,7375                           :rem 83
96Ø TN=CN:CN=CN+8:X=7:GOSUB19:GOSUB6Ø:GOSUB65:GOSU
    B3Ø:CN=TN:ONSQ+1GOTO 97Ø,7375            :rem 243
97Ø GOSUB1Ø:GOTO 51Ø                         :rem 138
98Ø PRINTCHR$(19)SPC(25)C1$"CLEAR TO END?":rem 132
982 GET K$:IF K$="" GOTO 982                 :rem 135
985 IF K$<>"Y" GOTO 995                      :rem 155
99Ø FOR K=1 TO 5:POKE CY,21:PRINT:POKE CX,15:PRINT
    Z$(Ø);:NEXT                              :rem 133
992 FM=FM+(NN-CN+2)/2:NN=CN-2:IN=Ø           :rem 22Ø
995 GOSUB1Ø:GOTO 51Ø                         :rem 145
4ØØØ REM ** ENTER MEASURE **                 :rem 224
4Ø1Ø IF M1 THEN GOSUB 5Ø                     :rem 23Ø
4Ø15 MZ=MZ+1+1ØØØ*(MZ>999):J=MZ:GOSUB7Ø       :rem 49
4Ø2Ø POKE MM+CN,J*64+3Ø:POKE MM+CN+1,K:B=1:GOTO 73
    ØØ                                       :rem 156
4Ø5Ø K=Ø:GOTO 4Ø65                           :rem 195
4Ø6Ø K=NN+2                                   :rem 75
```

```
4065 IF IN GOTO 510                                    :rem 196
4067 CN=K                                              :rem 234
4070 GK=1:SYS 1802:GOSUB10:GOTO501                     :rem 243
4100 IF NN<2 OR IN GOTO 510                            :rem 101
4105 PRINTCHR$(19)SPC(25)C1$"MEASURE:{10 SPACES}";
     :POKE CX,33                                       :rem 170
4110 FA=0:MX=3:GOSUB 200:IF A$="" THEN GOSUB10:GOT
     O 510                                              :rem 29
4115 J=VAL(A$):GOSUB 70:POKE 1394,J*64+30:POKE 139
     5,K                                               :rem 144
4120 J=MM(VC):GOSUB 70:POKE 251,K:POKE 252,J
                                                       :rem 227
4125 J=MM(VC)+NN:GOSUB 70:POKE 1392,K:POKE 1393,J:
     SYS 1350                                          :rem 31
4130 CN=PEEK(251)+256*PEEK(252)-MM(VC):GOTO 4070
                                                       :rem 148
5000 POKE CY,3:PRINTC0$:POKE CX,10:PRINTCHR$(18)"P
     RESS 'Y' TO CONFIRM"                              :rem 137
5010 GET K$:ON 1-(K$="")-2*(K$="Y") GOTO 8005,5010
     ,5040                                             :rem 155
5040 POKE 781,237:POKE 782,246:SYS 1813:POKE 55,PE
     EK(1020):POKE 56,PEEK(1021)                       :rem 170
5050 POKE 53269,0:POKE 648,4:POKE 56576,PEEK(56576
     )OR3:SYS 65409:CLR:END                             :rem 47
7000 GOSUB 7905:POKE 53280,2:POKE 53265,19:rem 233
7001 POKE 780,24:SYS 1596                              :rem 54
7002 POKE CY,3:PRINTC6$:POKE 53288,3:GOTO 7200
                                                       :rem 101
7004 PRINTC2$CHR$(147);:P$="{19 SPACES}":REM 19 SP
     ACES!                                             :rem 177
7006 PRINTCHR$(18)P$P$" "CHR$(157)CHR$(148)" "CHR$
     (19);                                             :rem 90
7008 U1$=CHR$(18):PRINTC6$:FOR K=1 TO 10:PRINTCHR$
     (185);:U1$=U1$+CHR$(192):NEXT                      :rem 2
7110 H=0:FOR K=1 TO 10:PRINT:PRINTC2$CHR$(18);:J=5
     -LEN(SC$(K))/2                                    :rem 131
7114 PRINTLEFT$(P$,J)SC$(K)LEFT$(P$,J+.5);CHR$(146
     );                                                :rem 181
7115 FOR J=1 TO N(K):H=H+1:PRINT" "C0$S$(H)" ";:NE
     XT:PRINT:PRINTC6$U1$;:NEXT                        :rem 144
7116 PRINTCHR$(145):FOR K=1 TO 10:PRINTCHR$(184);:
     NEXT                                              :rem 144
7120 POKE CY,1:PRINTC6$:FOR K=0 TO 4:POKE CX,20:PR
     INTCHR$(18)P$:NEXT:RETURN                          :rem 33
7200 IF UY>9 THEN POKE 53255,0:POKE 53288,13:GOTO
     {SPACE}7375                                       :rem 135
7201 POKE 53288,3:POKE 53264,-((UX*40)>147)*8:POKE
      53254,(108+UX*40)AND255                          :rem 152
7202 POKE 53255,59+UY*16:POKE CX,21                    :rem 149
7205 PRINT CHR$(18)SC$(UY+1);":";SS$(NN(UY)+UX+1)R
     IGHT$(I$,38-POS(0));                              :rem 219
```

```
7207 GOSUB5:GETK$:K=ASC(K$+" "):IF K<>17 AND K<>29
      GOTO 7209                                  :rem 218
7208 JV=-7*(K=29)-11*(K=17):SQ=1:GOSUB650:SQ=0:POK
     E CY,3:PRINTC6$:GOTO 7207                   :rem 113
7209 IF (KAND127)=20 THEN SQ=1.1:GOTO 900        :rem 181
7211 JV=-JV*(JV<15)-(JV=15)*(15+(K=73)+2*(K=77)+4*
     (K=74)+8*(K=75))                            :rem 175
7212 IF JV<15 THEN GOSUB12:GOTO 7216             :rem 242
7213 IF (JB OR K$=CHR$(13)) AND NOT (FM=0 AND CN>N
     N) GOTO 7250                                :rem 150
7214 IF K$<>CHR$(133) GOTO 7207                   :rem 56
7215 POKE 53255,0:GK=0:GOTO 9000                 :rem 211
7216 UX=UX+(JV=11)-(JV=7):UX=UX+N(UY+1)*(UX=N(UY+1
     ))-6*(UX<0)                                 :rem 94
7217 UY=UY+(JV=14)-(JV=13):UY=UY+11*(UY=11)-11*(UY
     <0)                                         :rem 51
7218 IF UY>9 THEN POKE 53255,0:POKE 53288,13:GOTO
     {SPACE}7350                                 :rem 137
7219 IF N(UY+1)<=UX THEN UX=N(UY+1)-1             :rem 81
7220 GOTO 7200                                   :rem 205
7250 K=ASC(MID$(S$,NN(UY)+UX+1))-65:POKE CY,1:PRIN
     T:POKE CX,21:PRINTCHR$(18);                 :rem 80
7255 IF K>10 THEN PRINTV$(K-11):GOTO 7265 :rem 244
7260 PRINT"(";-(V(K)<0)*(V(K))"TO"ABS(V(K))")"
                                                 :rem 185
7265 POKE CY,5:PRINT:POKE CX,21                   :rem 90
7267 MX=LEN(STR$(ABS(V(K))))-1:FA=-(V(K)<0)
                                                 :rem 167
7270 PRINTCHR$(18)"ENTER:";:GOSUB 200:IF A$="" GOT
     O 7001                                      :rem 75
7280 J=VAL(A$):IF ABS(J)>ABS(V(K)) OR (K=16 AND J<
     56) GOTO 7001                               :rem 40
7282 IF K=16 THEN J=INT(1800/J+.5)*8:J=-J*(J<256)
                                                 :rem 75
7283 IFK=9THENPOKEMM+CN+1,JAND255:POKE MM+CN,(JAND
     1792)/8+10-16*(J<0):GOTO7300                :rem 58
7284 IF K<>2 THEN J=J-256*(J<0):GOTO 7289 :rem 163
7286 H=16*(-11*(J<0)+12*(J/12-SGN(J)*INT(ABS(J)/12
     )))                                        :rem 252
7287 J=INT(H+2*(-7*(J>=0)-SGN(J)*INT(ABS(J)/12))-(
     J<0)+.5)                                    :rem 106
7289 IF K=6 THEN MZ=J                            :rem 132
7290 IF ABS(V(K))>90 GOTO 7295                    :rem 49
7292 POKE MM+CN,1:POKE MM+CN+1,PEEK(1278+NN(UY)+UX
     )+M(K)*J:GOTO 7300                         :rem 168
7295 J1=K:IF J<0 THEN J=-J*(V(C)+1)+1            :rem 163
7297 GOSUB 70:POKE MM+CN,PEEK(1278+NN(UY)+UX)+M(J1
     )*J:POKE MM+CN+1,K                         :rem 219
7300 IF NN<CN THEN NN=NN+2:FM=FM-1:IF B=1 THEN GOS
     UB10                                        :rem 71
```

```
7303 IN=IN+(IN>0)                                    :rem 185
7305 POKE CY,22:PRINT:POKE CX,16:PRINT"{4 SPACES}"
     ;:REM 4 SPACES                                  :rem 223
7310 X=3:GOSUB30:POKE CY,21:PRINT:PRINTZ$(0);:CN=C
     N+2                                             :rem 2
7315 IF NN-CN<8 OR IN>4 GOTO 7320                    :rem 220
7316 TN=CN:CN=CN+8:X=7:GOSUB19:GOSUB60:GOSUB65:GOS
     UB30:CN=TN                                      :rem 115
7320 ON 1-(B=13)-2*(B=1) GOTO 7001,7215,510:rem 67
7350 POKE CX,21:PRINT"{17 SPACES}";:REM 17 SPACES
                                                     :rem 179
7355 GOSUB5:GETK$:K=0:IF K$<>"" THEN K=ASC(K$):K=K
     -57*(K=17)-46*(K=29)                            :rem 94
7356 JV=-JV*(JV<15)-(JV=15)*(15+(K=73)+2*(K=77)+4*
     (K=74)+8*(K=75))                                :rem 185
7357 IF JV=7 OR JV=11 THEN SQ=1:GOSUB650:SQ=0:POKE
      CY,3:PRINTC6$:GOTO 7355                        :rem 73
7358 IF JV=13 OR JV=14 THEN FOR K=1 TO 100*-(K$=""
     ):NEXT:GOSUB12:GOTO 7217                        :rem 255
7360 IF K$<>"" THEN IF (ASC(K$)AND127)=20 THEN SQ=
     1:GOTO 900                                      :rem 139
7365 IF K$=CHR$(133) GOTO 7215                       :rem 1
7370 GOTO 7355                                       :rem 222
7375 K=SQ:SQ=0:POKE CY,3:PRINTC6$:ON 1-(K>1) GOTO
     {SPACE}7355,7207                                :rem 94
7500 NV(VC)=NN:CN(VC)=CN                             :rem 182
7510 IF VC=3OR(VC=2 AND NV(3)=-2)OR(VC=1 AND NV(2)
     =-2 AND NV(3)=-2) GOTO 7530                     :rem 36
7520 J=K5-NV(3)-4+(VC=1)*(NV(2)+4):GOSUB 70:POKE 2
     51,K:POKE 252,J                                 :rem 32
7525 J=MM(1)+NV(1)+4-(VC=2)*(NV(2)+4):GOSUB 70:POK
     E 253,K:POKE 254,J                              :rem 194
7527 J=NV(3)+4-(VC=1)*(NV(2)+4):GOSUB 70:POKE 781,
     K:POKE 782,J:SYS 1024                           :rem 138
7530 MM(2)=MM(1)+NV(1)+4:MM(3)=MM(2)+NV(2)+4
                                                     :rem 88
7535 GOSUB8690                                       :rem 43
7900 POKE 241,PEEK(241)OR128:POKE 53251,0:POKE 532
     81,10                                           :rem 201
7905 POKE 1140,255:POKE 1145,240:POKE 1160,K8:POKE
      1165,K9:SYS 1138                               :rem 84
7910 POKE 53280,10:POKE 53281,1:POKE 53249,0:POKE
     {SPACE}53253,0:RETURN                           :rem 5
8000 IF IN GOTO 510                                  :rem 189
8001 GOSUB 7500                                      :rem 21
8005 POKE 780,9:SYS 1596:POKE 53269,0:GOTO 8225
                                                     :rem 32
8010 P$=CHR$(18)+"{39 SPACES}":REM 39 SPACES
                                                     :rem 237
```

```
8015 P1$=CHR$(162):FOR K=1 TO 4:P1$=P1$+P1$:NEXT:P
     1$=P1$+LEFT$(P1$,6)                    :rem 233
8020 P2$=CHR$(192):P2$=P2$+P2$+P2$:P2$=P2$+P2$+P2$
     +P2$                                   :rem 54
8100 PRINTCHR$(147);                        :rem 127
8110 FOR K=1 TO 6:PRINTCHR$(150)P$:NEXT     :rem 223
8120 PRINTCHR$(18)" "C7$LEFT$(P$,39)        :rem 161
8130 FOR K=1TO13:PRINTCHR$(18)CHR$(150)" "C7$" "CB
     $" "SPC(34)" "C7$" ":NEXT              :rem 45
8140 PRINTCHR$(150)CHR$(18)" "C7$LEFT$(P$,39);
                                            :rem 198
8141 FOR K=0 TO 960 STEP 40:POKE 55335+K,10:POKE 5
     1239+K,160:NEXT                        :rem 223
8142 POKE CY,19:PRINT                       :rem 247
8145 FOR K=1 TO 4:PRINT:PRINTCHR$(150)LEFT$(P$,4)C
     2$LEFT$(P$,35)CHR$(150)"{2 SPACES}";   :rem 219
8146 NEXT                                   :rem 18
8150 PRINTCHR$(19)C6$SPC(13):PRINTCHR$(18)K1$P2$K2
     $                                      :rem 184
8152 PRINTSPC(13):PRINTCHR$(18)CHR$(221)" SID PLAY
     ER "CHR$(221)                          :rem 57
8154 PRINTSPC(13):PRINTCHR$(18)K3$P2$K4$    :rem 103
8162 POKE CY,6:PRINT:POKE CX,3              :rem 40
8165 PRINTCB$LEFT$(P$,7)P1$LEFT$(P$,7)      :rem 157
8170 FOR K=1 TO 11:PRINTSPC(3)LEFT$(P$,7)CHR$(146)
     MID$(P$,2,22)LEFT$(P$,7):NEXT          :rem 171
8175 PRINTSPC(3)LEFT$(P$,7)CHR$(146)P1$LEFT$(P$,7)
                                            :rem 76
8180 POKE CY,7:PRINTC6$                     :rem 99
8190 FOR K=0 TO 5:PRINTSPC(10)K+1;CHR$(157)") "M$(
     K):PRINT:NEXT:RETURN                   :rem 77
8225 GET K$:IF K$<"1" OR K$>"6" GOTO 8225 :rem 219
8230 J=ASC(K$)-48:POKE CY,5+J*2:PRINT:PRINTSPC(14)
     CHR$(18)C2$M$(J-1)                     :rem 163
8260 ON J GOTO 8900,8500,8450,8400,8300,5000
                                            :rem 151
8300 REM ** DIRECTORY **                     :rem 12
8310 IF DN=1 GOTO 8005                       :rem 97
8312 POKE 780,1:POKE 781,8:POKE 782,0:SYS 65466:PO
     KE 581,36                              :rem 172
8314 POKE 780,1:POKE 781,69:POKE 782,2:SYS 65469:S
     YS 65472                               :rem 143
8316 POKE 581,46:POKE 582,77:POKE 583,85:POKE 584,
     83                                      :rem 26
8318 P$=CHR$(18)+"{39 SPACES}":REM 39 SPACES
                                            :rem 248
8320 POKE CY,3:PRINT:POKE CX,15:PRINTCHR$(18)C0$"D
     IRECTORY:"                             :rem 160
8330 POKE CY,6:PRINT:PRINTSPC(3)CB$CHR$(18)LEFT$(P
     1$,16)CHR$(187)CHR$(172);               :rem 72
```

```
8335 PRINTLEFT$(P1$,16)                          :rem 56
8340 FOR K=1 TO 11:PRINTSPC(3)MID$(P$,2,16)CHR$(18
     )CHR$(161)CHR$(146)CHR$(161);               :rem 16
8345 PRINTMID$(P$,2,16):NEXT                     :rem 142
8350 PRINTSPC(3)LEFT$(P1$,16)CHR$(18)CHR$(190)CHR$
     (188)CHR$(146)LEFT$(P1$,16)                 :rem 254
8355 J=0:K=0:POKE CY,7:PRINTC0$                  :rem 68
8360 POKE CX,18*K+3:SYS 1710:H=PEEK(251)+256*PEEK(
     252):IF ST<>0 GOTO 8376                     :rem 254
8365 POKE CX,18*K+15:PRINTLEFT$("{2 SPACES}",-(H<1
     00)-(H<10));H                               :rem 139
8373 J=J+1:IF J>10 THEN J=0:POKE CY,7:PRINT:K=1-K:
     IF K=0 THEN K=-1:GOTO 8380                  :rem 16
8374 GOTO 8360                                   :rem 224
8376 CLOSE 1:SYS 65484                           :rem 195
8380 POKE 198,0:POKE CY,3:PRINT:PRINTSPC(7)CHR$(18
     )C0$"PRESS ANY KEY ";                       :rem 143
8382 IF K<0 THEN PRINT"TO CONTINUE";:GOTO 8385
                                                 :rem 169
8383 PRINT"FOR MAIN MENU";:POKE CY,19:PRINT
                                                 :rem 133
8384 PRINTTAB(12)LEFT$("{BLU}{RVS}{2 SPACES}",2-(H
     <100)-(H<10))H"{LEFT} BLOCKS FREE{2 SPACES}"
                                                 :rem 97
8385 WAIT 198,15                                 :rem 65
8387 POKE CX,7:PRINTCHR$(150)LEFT$(P$,30)        :rem 89
8390 IF K>=0 GOTO 8005                           :rem 95
8395 K=0:GOTO 8330                               :rem 210
8400 REM ** SAVE **                              :rem 135
8402 IF NV(1)<0 AND NV(2)<0 AND NV(3)<0 GOTO 8005
                                                 :rem 194
8404 FOR I=6 TO 2 STEP -2:J=NV(4-I/2)+4:GOSUB 70:P
     OKE MM(1)-I,K                               :rem 150
8406 POKE MM(1)-I+1,J:NEXT                       :rem 222
8408 POKE CY,3:PRINT:PRINTTAB(10)CHR$(18)C0$" CHAN
     GE TEXT LINES? "                            :rem 80
8410 WAIT 198,15:GET A$:ON 1-(A$="N")-2*(A$="Y") G
     OTO 8410,8428,8414                          :rem 141
8414 POKE 780,9:SYS 1596:GOSUB 8490              :rem 161
8418 MX=32:FA=2:FOR K=0 TO 3:POKE CY,20+K:PRINT:PO
     KE CX,4:PRINTCHR$(18)C2$;                   :rem 133
8420 GOSUB200:TX$(K)=A$:NEXT                     :rem 161
8424 POKE 780,3:SYS 1596                         :rem 13
8428 POKE CY,3:PRINT:POKE CX,6:PRINTI$;:POKE CX,17
     :MX=12:FA=2:GOSUB 200                       :rem 47
8429 IF A$="" GOTO 8005                          :rem 82
8430 POKE 780,1:POKE 781,DN:POKE 782,0:SYS 65466:A
     $=A$+".MUS"                                 :rem 62
8432 POKE 780,LH+4:FOR K=0 TO LH+3:POKE TP+K,ASC(M
     ID$(A$,K+1)):NEXT                           :rem 3
```

```
8434 J=TP:GOSUB70:POKE 781,K:POKE 782,J:SYS 65469
                                                :rem 102
8435 J=MM(1)-6:GOSUB70:POKE 251,K:POKE 252,J:POKE
     {SPACE}780,251                             :rem 183
8436 J=MM(1)+NV(1)+NV(2)+NV(3)+12:POKE CY,2:rem 95
8438 FOR K=0 TO 3:IF TX$(K)="" GOTO 8440    :rem 11
8439 FOR I=1 TO LEN(TX$(K)):POKE J,ASC(MID$(TX$(K)
     ,I)):J=J+1:NEXT                            :rem 204
8440 POKE J,13:J=J+1:NEXT:POKE J,0:J=J+1:GOSUB 70:
     POKE 781,K:POKE 782,J                      :rem 80
8446 GOSUB 8495:SYS 65496:POKE 648,200:PRINT:IF PE
     EK(783)AND1 THEN GOSUB8600                 :rem 105
8448 GOTO 8005                                  :rem 222
8450 REM ** LOAD **                             :rem 125
8452 POKE CY,3:PRINT:POKE CX,6:PRINTI$;:POKE CX,17
     :FA=2:MX=12:GOSUB 200                      :rem 44
8456 IF A$="" GOTO 8005                         :rem 82
8460 A$=A$+".MUS":POKE 780,1:POKE 781,DN:POKE 782,
     0:SYS 65466                                :rem 65
8462 POKE 780,LH+4:FOR K=0 TO LH+3:POKE TP+K,ASC(M
     ID$(A$,K+1)):NEXT                          :rem 6
8464 J=TP:GOSUB70:POKE 781,K:POKE 782,J:SYS 65469:
     POKE CY,1                                  :rem 203
8466 POKE 780,0:J=MM(1)-6:GOSUB70:POKE 781,K:POKE
     {SPACE}782,J:GOSUB 8495:SYS 65493          :rem 59
8468 POKE 648,200:PRINT:IF PEEK(783)AND1 THEN GOSU
     B8600:GOTO 8560                            :rem 216
8470 FOR J=6 TO 2 STEP-2:NV(4-J/2)=PEEK(MM(1)-J)+2
     56*PEEK(MM(1)-J+1)-4:NEXT                  :rem 153
8474 MM(2)=MM(1)+NV(1)+4:MM(3)=MM(2)+NV(2)+4:CN(1)
     =0:CN(2)=0:CN(3)=0                         :rem 145
8476 FOR K=0 TO 3:TX$(K)="":NEXT:K=0:J=MM(3)+NV(3)
     +4                                         :rem 113
8478 IF PEEK(J)=0 GOTO 8483                     :rem 167
8480 IF PEEK(J)=13 THEN J=J+1:K=K+1:GOTO 8478
                                                :rem 215
8482 TX$(K)=TX$(K)+CHR$(PEEK(J)):J=J+1:GOTO 8478
                                                :rem 221
8483 POKE 780,9:SYS 1596                        :rem 24
8484 GOSUB8490:POKE CY,20:FOR K=0 TO 3:PRINTC2$:PO
     KE CX,4:PRINTCHR$(18)TX$(K);               :rem 92
8485 NEXT:POKE 780,3:SYS 1596                   :rem 141
8486 FM=INT((K5-MM(3)-NV(3)-2)/2)-1:IF FM>=0 GOTO
     {SPACE}8005                                :rem 130
8487 POKE CY,3:PRINTC0$:POKE CX,10:PRINTCHR$(18)"I
     NSUFFICIENT{2 SPACES}MEMORY"               :rem 26
8488 WAIT 198,255:GET K$:GOTO 8560              :rem 74
8490 POKE CY,20:FOR K=0 TO 3:PRINTC2$:POKE CX,4:PR
     INT CHR$(18)RIGHT$(I$,16);                 :rem 81
8492 PRINT RIGHT$(I$,16);:NEXT:RETURN           :rem 37
```

```
8495 POKE 648,188:PRINT "{3}":RETURN          :rem 51
8500 POKE CY,3:PRINT:POKE CX,10:PRINTC0$CHR$(18)"
     {SPACE}VOICE 1-3 (OR CLR) ";             :rem 193
8505 GET A$:IF A$="" GOTO 8505                :rem 209
8507 VC=VAL(A$):IF ASC(A$)=147 GOTO 8560      :rem 126
8510 IF VC<1ORVC>3 GOTO 8005                   :rem 20
8519 REM ** OH DEAR.{2 SPACES}THIS IS IT. **
                                              :rem 227
8520 IF VC=3OR(VC=2 AND NV(3)=-2)OR(VC=1 AND NV(2)
     =-2 AND NV(3)=-2) GOTO 8540               :rem 40
8530 J=MM(VC+1):GOSUB 70:POKE 251,K:POKE 252,J
                                              :rem 72
8535 J=K5-NV(3)-4+(VC=1)*(NV(2)+4):GOSUB 70:POKE 2
     53,K:POKE 254,J                          :rem 43
8537 J=NV(3)+4-(VC=1)*(NV(2)+4):GOSUB 70:POKE 781,
     K:POKE 782,J:SYS 1024                    :rem 140
8540 MM=MM(VC):NN=NV(VC):CN=CN(VC)            :rem 80
8550 POKE 53269,15:GK=1:GOTO 9000             :rem 18
8560 POKE 780,9:SYS 1596:GOSUB8490           :rem 163
8565 POKE 780,3:SYS 1596:FOR K=1 TO 3:TX$(K)="":NV
     (K)=-2:CN(K)=0:NEXT:TX$(0)=""            :rem 89
8570 MM(2)=MM(1)+2:MM(3)=MM(1)+4:GOSUB8690    :rem 72
8575 FM=INT((K5-MM(1))/2)-67:GOTO 8005        :rem 58
8600 K=PEEK(780):POKE 780,9:SYS 1596          :rem 230
8605 POKE CY,3:PRINTC0$:PRINTCHR$(18);        :rem 24
8610 IF K=4 THEN POKE CX,13:PRINT"FILE NOT FOUND":
     GOTO 8620                                :rem 126
8612 IF K=5 THEN POKE CX,11:PRINT"DEVICE NOT PRESE
     NT":GOTO 8620                            :rem 180
8614 POKE CX,15:PRINT"ERROR -"ST              :rem 152
8620 GET K$:IF K$="" GOTO 8620                :rem 225
8630 RETURN                                   :rem 177
8690 FOR K=1 TO 3:POKE MM(K)+NV(K)+2,1:POKE MM(K)+
     NV(K)+3,79:NEXT:RETURN                   :rem 168
8900 REM PLAY OPTION                          :rem 196
8910 FOR I=6 TO 2 STEP -2:J=NV(4-I/2)+4:GOSUB 70:P
     OKE MM(1)-I,K                            :rem 152
8915 POKE MM(1)-I+1,J:NEXT                    :rem 227
8920 POKE CY,3:PRINT:POKE CX,6:PRINTCHR$(18)C0$"PL
     AY VOICES:";                             :rem 251
8925 B=0:FOR K=1 TO 3:POKE CX,16+K*5:PRINTK;CHR$(1
     57)"?"CHR$(157);                         :rem 140
8930 GET A$:IF A$="" GOTO 8930                :rem 213
8935 ON 1-(A$="Y")-3*(ASC(A$)=13)-2*(A$="N") GOTO
     {SPACE}8930,8937,8938,8939               :rem 160
8937 B=B+2↑(K-1):PRINT" ";:NEXT:GOTO 8940     :rem 88
8938 PRINTCHR$(157)CHR$(157)CHR$(150)"{3 SPACES}"C
     0$;:NEXT:GOTO 8940                        :rem 6
8939 POKE 198,2:POKE 631,89:POKE 632,89:GOTO 8937
                                              :rem 155
```

```
8940 POKE 198,0:SYS 49435:J=MM(1)-6:GOSUB 70:POKE
     {SPACE}781,K:POKE 782,J:SYS 49458     :rem 236
8945 POKE 49152,B:J=0                      :rem 109
8950 IF PEEK(198) THEN GET K$:J=(K$="{F7}"):POKE 4
     9152,0                                 :rem 79
8955 IF NOT PEEK(56320)AND16 THEN POKE 56325,22-PE
     EK(678):GOTO 8960                      :rem 144
8956 POKE 56325,SGN(15-PEEK(56320)AND15)*(132-4*PE
     EK(678))+66-PEEK(678)*2                :rem 169
8960 IF PEEK(49152)AND7 GOTO 8950           :rem 250
8970 SYS 49629:POKE 54276,0:POKE 54283,0:POKE 5429
     0,0                                    :rem 70
8972 POKE 56325,66-PEEK(678)*2:IF PEEK(49152)=0 AN
     D J=0 GOTO 8980                        :rem 151
8974 FOR K=1 TO 3:H=PEEK(49289+K):J=PEEK(49274+K)+
     256*PEEK(49277+K)-MM(K)-2              :rem 113
8975 IF J>=-2 AND J<=NV(K)+2 GOTO 8977      :rem 219
8976 J=PEEK(49292+H)+256*PEEK(49304+H)-MM(K)-2:H=H
     -1:GOTO 8975                           :rem 174
8977 IF (BAND2↑(K-1))=0 THEN J=CN(K)        :rem 174
8978 CN(K)=J:CN(K)=CN(K)*-(CN(K)>=0):NEXT   :rem 94
8980 IF PEEK(49152)=0 GOTO 8005             :rem 86
8981 POKE CX,6:PRINTCHR$(18)CHR$(150)"{28 SPACES}"
     ;:REM 28                               :rem 119
8982 POKE CX,6:PRINTCHR$(18)C2$PE$(PEEK(49152)/8-1
     )" ERROR";                             :rem 117
8984 PRINTCHR$(145);:POKE CX,16:PRINT"VOICE"PEEK(4
     9184)+1CHR$(157)":"                    :rem 242
8986 WAIT 198,255:GET A$:GOTO 8005          :rem 61
9000 POKE 53280,2:POKE 1140,127:POKE 1145,1:POKE 1
     160,147:POKE 1165,4:SYS 1138           :rem 115
9002 POKE 53288,1:GOTO 9130                 :rem 161
9004 PRINTCHR$(147);                        :rem 131
9010 P1$=C6$+CHR$(18)+CHR$(161)+C7$         :rem 13
9012 P$=CHR$(162):P$=P$+P$+P$:P2$=P$+P$+P$+P$+P$:P
     6$=CHR$(146)                           :rem 32
9014 P3$=C6$+P6$+CHR$(161):P4$=CHR$(192):P$=CHR$(2
     9):P7$=CHR$(221)                       :rem 247
9016 P5$=P1$+" "+C0$+P$+" "+P$+" "+P6$+" "+P7$+" "
     +CHR$(18)+" "+P$+" "+P$+" "            :rem 162
9017 P5$=P5$+P$+C7$+" "+P6$+C6$+CHR$(161)  :rem 233
9050 PRINT CHR$(146)C1$"{5 SPACES}VOICE:
     {14 SPACES}MEM:"                       :rem 237
9051 P$=CHR$(183):P$=P$+P$                  :rem 108
9052 PRINT SPC(5)C7$P$P$P$P$SPC(12)P$P$P$P$P$
                                            :rem 21
9054 POKE CX,6:PRINTC1$"KEY:";:POKE CX,22:PRINT"ME
     ASURE:"                                :rem 234
9057 PRINT:PRINTC3$:FOR AC=-1 TO 1 STEP 2:GOSUB 8:
     NEXT:PRINTC1$:AC=0:GOSUB 8             :rem 44
```

```
9060 U1$=P4$+P4$:U1$=U1$+U1$+U1$+U1$       :rem 123
9100 PRINTC0$"{6 SPACES}M{15 SPACES}"C6$CHR$(172)P
     2$CHR$(187)                            :rem 32
9102 PRINTC0$"{6 SPACES}L{8 SPACES}NOTE:{2 SPACES}
     "P1$C7$"{15 SPACES}"P3$                :rem 158
9104 PRINTC0$" NO{3 SPACES}L{8 SPACES}"C6$K1$P4$P4
     $P4$K2$"{2 SPACES}"P5$                 :rem 98
9106 PRINTC0$"{M}PQLLLLLLLLL{3 SPACES}"C6$P7$"
     {3 SPACES}"C6$P7$"{2 SPACES}"P5$       :rem 109
9108 PRINTC0$"{M}RSLLLLLLLLL{3 SPACES}"C6$K3$P4$P4
     $P4$K4$"{2 SPACES}"P5$                 :rem 121
9109 P$=CHR$(163):P8$=P$+P$+P$+P$+P$       :rem 131
9110 PRINTC0$"{M}TU"P$P$P$P$P8$"{10 SPACES}"P5$
                                            :rem 236
9111 P5$=" "+P7$:P5$=P5$+P5$+P5$+P5$+P5$+P5$+" ":P
     9$=C7$+" "+CHR$(146)                   :rem 113
9112 P5$=C6$+CHR$(18)+CHR$(161)+P9$+C0$+P5$+CHR$(1
     8)+P9$+C6$+CHR$(161)                   :rem 84
9113 PRINTC0$"{M}{5 SPACES}"P$"{7 SPACES}OCTAVE: "
     P5$                                    :rem 58
9114 PRINTC0$"{M}VWLLLLLLLLL{4 SPACES}"C6$K1$P4$K2
     $"{3 SPACES}"P5$                       :rem 42
9116 PRINTC0$"{M}XYLLLLLLLLL{4 SPACES}"C6$P7$" "P7
     $"{3 SPACES}"P1$"{15 SPACES}"P3$       :rem 200
9118 PRINTC0$"Z"P8$"L"P8$"{4 SPACES}"C6$K3$P4$K4$"
     {3 SPACES}"CHR$(188)CHR$(18);          :rem 120
9119 PRINTP2$CHR$(146)CHR$(190)             :rem 224
9120 P$=P4$+P4$+P4$:P$=P$+P$+P4$            :rem 156
9121 PRINTC0$"{6 SPACES}L{13 SPACES}"C6$K1$P4$P4$K
     2$K1$P$K2$                             :rem 228
9122 PRINTC0$"{6 SPACES}L{6 SPACES}LENGTH:"C6$P7$"
     {2 SPACES}"P7$P7$"{7 SPACES}"P7$       :rem 207
9124 PRINT"{20 SPACES}"K3$P4$P4$K4$K3$P$K4$
                                            :rem 239
9126 PRINTC3$"{9 SPACES}REST <--SELECT--> TIE"
                                             :rem 75
9128 PRINT:PRINT:PRINT"{40 SPACES}"CHR$(19):RETURN
                                              :rem 9
9130 IF GK THEN POKE 241,PEEK(241)AND127   :rem 137
9132 POKE 780,16+GK:SYS 1596               :rem 251
9135 POKE 53250,140:POKE 53251,224:POKE 53264,0
                                            :rem 183
9140 FOR K=54272 TO 54295:POKE K,0:NEXT:K=54272
                                             :rem 60
9150 POKEK+3,8:POKEK+5,0:POKEK+6,247:POKEK+24,10:P
     OKEK+16,240:POKEK+8,120                :rem 9
9160 PRINTCHR$(19)CHR$(146)C1$;:POKE CX,11:PRINTVC
     ;                                      :rem 6
9170 POKE CX,29:PRINT FM                   :rem 140
9180 POKE CY,1:PRINT:POKE CX,10             :rem 82
```

```
9185 PRINTABS(KY)CHR$(157)CHR$(193-2*(KY<Ø))" ("KY
     $(KY+7)")";                                  :rem 42
9190 POKE CX,30:PRINTMID$("ON OFF",4-M1*3,3)
                                                  :rem 197
9199 GOTO 500                                     :rem 170
9200 PRINT "{CLR}":PRINT " SIDPLAYER EDITOR":PRINT
      " BY HARRY BRATT":PRINT                     :rem 30
9210 POKE 780,1:POKE 781,DN:POKE 782,1:SYS 65466
                                                  :rem 105
9220 A$="EDITOR.OBJ":FOR K=1 TO 10:POKE 584+K,ASC(
     MID$(A$,K)):NEXT                             :rem 217
9230 POKE 780,10:POKE 781,73:POKE 782,2:SYS 65469
                                                  :rem 119
9240 POKE 780,Ø:SYS 65493:IF PEEK(783)AND1 THEN PR
     INT " ERROR":END                             :rem 189
40000 REM *** INITIALIZATION ***                  :rem 252
41000 POKE 251,DN:POKE 53280,7:POKE 53265,8:K=PEEK
      (49)+256*PEEK(50)+4096                      :rem 138
41010 POKE 1020,PEEK(55):POKE 1021,PEEK(56):POKE 5
      5,KAND255:POKE 56,K/256:CLR                 :rem 25
41012 K5=PEEK(1020)+256*PEEK(1021)                :rem 26
41013 K=Ø:J=Ø:I=Ø:H=Ø:JV=Ø:LN=3:CX=211:CY=214:AC=Ø
      :SC=3:PK=14:NN=-2:NT=Ø:OC=4                 :rem 54
41014 DIM KY$(14),S$(36),SS$(36),V(16),M(16),NV(3)
      ,CN(3),MM(3),PE$(5),V$(5)                   :rem 64
41015 DIM FW$(6,1),NU$(1,1),KL%(15)               :rem 235
41017 MM(1)=PEEK(55)+256*PEEK(56)+6:MM(2)=MM(1)+2:
      MM(3)=MM(1)+4:DN=PEEK(251)                  :rem 158
41018 FM=INT((K5-MM(1))/2)-67:NV(1)=NN:NV(2)=NN:NV
      (3)=NN                                      :rem 205
41019 CN(1)=K:CN(2)=K:GN(3)=K:POKE 648,200:POKE 53
      272,36:Z=PEEK(56576)                        :rem 101
41020 FOR TP=1024 TO 1137:READ J:POKE TP,J:NEXT:TP
      =1827                                       :rem 126
41021 POKE 251,98:POKE 252,199:POKE 253,114:POKE 2
      54,4:SYS 1030                                :rem 83
41022 POKE 781,30:POKE 782,7:SYS 1813:POKE 657,128
                                                  :rem 156
41023 K8=PEEK(788):K9=PEEK(789):POKE 1213,K8:POKE
      {SPACE}1214,K9                              :rem 176
41024 FOR K=1 TO 36:READ S$(K),SS$(K):NEXT :rem 27
41025 NN(Ø)=Ø:FOR K=1 TO 10:READ SC$(K),N(K):NN(K)
      =NN(K-1)+N(K):NEXT                          :rem 249
41026 C0$=CHR$(144):C1$=CHR$(5):C3$=CHR$(159):C6$=
      CHR$(31):C7$=CHR$(158)                       :rem 35
41027 CB$=CHR$(154):C2$=CHR$(28)                  :rem 156
41028 K1$=CHR$(176):K2$=CHR$(174):K3$=CHR$(173):K4
      $=CHR$(189)                                 :rem 230
41030 FOR K=Ø TO 18:READ AC:AC$(Ø)=AC$(Ø)+CHR$(AC)
      :NEXT:AC=Ø                                  :rem 235
```

```
41032 FOR K=1 TO 2:AC$(K)=LEFT$(AC$(0),9)+CHR$(193
      +K)+RIGHT$(AC$(0),9):NEXT                  :rem 13
41034 FOR K=0 TO 5:READ M$(K):NEXT               :rem 79
41035 IF DN=1 THEN M$(4)="(NOT AVAILABLE)":rem 245
41060 GOSUB 9004:POKE 780,5:SYS 1596:GOSUB 7004:PO
      KE 780,7:SYS 1596                          :rem 189
41064 GOSUB 8010:POKE 780,3:SYS 1596:POKE 53280,10
                                                 :rem 181
41070 POKE 56578,PEEK(56578)OR3:POKE 56576,(PEEK(5
      6576)AND252)                               :rem 161
41080 POKE 251,0:POKE 252,208:POKE 253,0:POKE 254,
      208:POKE 781,0:POKE 782,8                  :rem 228
41085 POKE 56334,PEEK(56334)AND254:POKE 1,PEEK(1)A
      ND251:SYS 1024                             :rem 35
41090 FOR K=0 TO 223:READ J:POKE 53768+K,J:POKE 54
      792+K,255-J:NEXT                           :rem 123
41095 POKE 1,PEEK(1) OR 4:POKE 56334,PEEK(56334) O
      R 1                                        :rem 242
41122 FOR K=0 TO 7:READ LN$(K):NEXT K            :rem 231
41125 FOR K=53248 TO 53254 STEP 2:POKE K,0:NEXT:PO
      KE 53264,0                                 :rem 216
41130 POKE 52216,48:POKE 52217,49:POKE 52218,50:PO
      KE 52219,49:POKE 53276,11                  :rem 194
41135 POKE 53287,7:POKE 53285,6:POKE 53289,0:POKE
      {SPACE}53290,13                            :rem 55
41138 POKE 53275,10:POKE 53277,10:FOR K=52224 TO 5
      2415:POKE K,0:NEXT                         :rem 150
41140 J=52225:POKE J,85:FOR K=1 TO 6:POKE J+K*3,10
      5:NEXT:POKE J+21,85                        :rem 27
41142 J=52288:FOR I=0 TO 1:FOR K=0 TO 1:POKE J+K*3
      +I*57,85:POKE J+K*3+I*57+1,85              :rem 146
41143 NEXT:NEXT:FOR K=6 TO 54 STEP 3:POKE J+K,106:
      POKE J+K+1,169:NEXT                        :rem 197
41145 J=52398:POKE J,63:POKE J+3,127:POKE J+6,255:
      POKE J+9,254:POKE J+12,124                 :rem 110
41147 FOR I=J-30 TO J-3 STEP 3:POKE I,1:NEXT
                                                 :rem 140
41160 P$=" "+CHR$(20):P$=P$+P$+P$+P$+P$+P$:P$=P$+CHR$
      (17)+P$:Z$(0)=P$                           :rem 120
41162 P$=CHR$(148)+" "+CHR$(157):P$=P$+P$+P$+P$+P$
      :P$=P$+CHR$(17)+P$:Z$(1)=P$                :rem 208
41165 Y$=CHR$(157):Y$=Y$+Y$+Y$+CHR$(17)+" "
                                                 :rem 191
41175 FOR K=0 TO 14:READ KY$(K)                   :rem 99
41177 IF K<6 THEN KY$(K)=KY$(K)+"C"              :rem 182
41178 IF K>12 THEN KY$(K)=KY$(K)+"A"             :rem 228
41179 NEXT K                                     :rem 144
41180 FOR K=0 TO 6:READ AK%(K):NEXT K            :rem 221
41185 TY$(0)="{2 SPACES}":TY$(1)="+{-}"          :rem 155
41196 I$=CHR$(18)+CHR$(150)+"FILENAME: "+CHR$(146)
      +CHR$(161)+C0$                             :rem 86
```

```
41197 I$=I$+"{16 SPACES}":REM 16 SPACES    :rem 208
41220 S$="QFDOFPDDPLEDNEFNADADFMKENNBFHJCGNFPF"
                                            :rem 28
41225 FOR K=0 TO 16:READ V(K),M(K):NEXT    :rem 125
41230 FOR K=0 TO 5:READ V$(K):NEXT          :rem 86
41235 FOR K=0 TO 6:READ FW$(K,0),FW$(K,1):NEXT
                                            :rem 229
41237 FOR K=0 TO 1:READ NU$(K,0),NU$(K,1):NEXT
                                            :rem 238
41240 GOSUB50:AL=2                          :rem 22
41250 FOR K=0 TO 5:READ PE$(K):NEXT        :rem 151
41350 FOR K=0 TO 15:KL%(K)=-1:NEXT          :rem 75
41360 FOR K=0 TO 5:READ I,J:KL%(I-8)=J:NEXT
                                            :rem 127
41370 POKE 53281,1:POKE 53265,27           :rem 145
49999 GOSUB8690:GOTO 8005                  :rem 175
61100 DATA 142,112,4,140,113,4,165,253,56,229,251,
      170,165,254,229,252,168,138          :rem 209
61101 DATA 205,112,4,152,237,113,4,144,35,160,0,17
      4,113,4,240,14,177,251,145           :rem 128
61102 DATA 253,200,208,249,230,252,230,254,202,208
      ,242,174,112,4,240,8,177,251         :rem 242
61103 DATA 145,253,200,202,208,248,96,173,113,4,16
      8,101,252,133,252,152,24,101         :rem 240
61104 DATA 254,133,254,172,112,4,240,9,136,177,251
      ,145,253,192,0,208,247,174           :rem 164
61105 DATA 113,4,240,16,198,252,198,254,136,177,25
      1,145,253,192,0,208,247,202          :rem 221
61106 DATA 208,240,96,177,2                 :rem 56
61150 DATA TEM,SET,UTL,UTILITY SET,VOL,SET,BMP,BUM
      P                                     :rem 34
61151 DATA HED,HEAD,TAL,TAIL,CAL,CALL,DEF,DEFINE,E
      ND,END                               :rem 166
61152 DATA F-M,MODE,AUT,AUTO,RES,RESONANCE,FLT,THR
      OUGH,F-S,SWEEP,F-C,CUTOFF             :rem 91
61153 DATA F-X,EXTERNAL,ATK,ATTACK,DCY,DECAY,SUS,S
      USTAIN,RLS,RELEASE                   :rem 198
61154 DATA PNT,R POINT,WAV,SET,P-W,P WIDTH,P-S,P S
      WEEP,SNC,SYNC,RNG,RING MOD            :rem 7
61155 DATA VDP,VIB DEPTH,VRT,VIB RATE,POR,PORTAMEN
      TO,DTN,DETUNE,TPS,TRANSPOSE           :rem 76
61156 DATA MS#,MEASURE #,3-O,VOICE 3 OFF,FLG,FLAG,
      HLT,HALT,AUX,AUXILIARY               :rem 126
61157 DATA TEMPO,2,VOLUME,2,REPEAT,2,PHRASE,3,FILT
      ER,3,FILTER,4,ENVELOPE,5             :rem 138
61158 DATA WAVEFORM,5,FREQ,5,MISC,5         :rem 91
61190 DATA 162,162,162,18,17,157,157,157,32,193,32
      ,146,17,157,157,157                   :rem 99
61192 DATA 183,183,183                      :rem 73
61194 DATA PLAY MUSIC,EDIT MUSIC,LOAD MUSIC FILE,S
      AVE MUSIC FILE,DISK DIRECTORY        :rem 158
```

```
61195 DATA QUIT                                        :rem 99
61200 REM *** CHR DATA ***                             :rem 208
61201 DATA 108,254,108,254,108,0,0,0,96,124,108,10
      8,124,12,0,0                                     :rem 203
61202 DATA 192,192,240,216,240,0,0,0                   :rem 214
61203 DATA 30,27,30,27,30,123,248,240,30,27,30,27,
      24,120,248,240                                   :rem 52
61204 DATA 30,27,24,24,24,120,248,240                  :rem 16
61205 DATA 24,24,24,24,24,120,248,240,12,12,12,12,
      12,124,204,248                                   :rem 43
61206 DATA 0,0,0,0,120,204,204,120,0,0,99,99,99,99
      ,110,0                                           :rem 169
61207 DATA 0,24,60,102,126,102,102,102                 :rem 47
61208 DATA 255,0,0,0,255,0,0,0,0,0,0,0,255,0,0,0
                                                       :rem 243
61209 DATA 0,0,0,7,12,24,24,24,0,0,0,128,192,192,1
      92,192                                           :rem 167
61210 DATA 255,13,15,6,255,14,27,27,255,128,0,0,25
      5,0,0,0                                          :rem 218
61211 DATA 255,49,55,55,255,54,24,12,255,224,176,1
      52,255,216,216,240                               :rem 34
61212 DATA 255,0,24,25,15,0,0,0,255,96,96,192,0,0,
      0,0                                              :rem 15
61213 DATA 255,48,96,112,255,112,0,0,255,96,51,51,
      255,48,51,99                                     :rem 253
61214 DATA 255,0,1,3,255,28,112,0,255,192,128,0,25
      5,0,0,0,3,0,0,0,0,0,0,0                          :rem 184
61215 DATA 96,96,48,48,28,15,3,0,6,6,12,12,56,240,
      192,0                                            :rem 149
61300 DATA 32ND{3 SPACES},16TH{3 SPACES},EIGHTH ,Q
      UARTER,HALF{3 SPACES},WHOLE{2 SPACES},UTILIT
      Y,ABS SET                                        :rem 175
61306 DATA C,G,D,A,E,B,F ,C ,G ,D ,A ,E ,B ,F,C
                                                       :rem 125
61310 DATA 3,0,4,1,5,2,6                               :rem 130
61450 DATA 15,8,127,256,-95,256,15,16,-127,256,255
      ,256,999,64,16383,4,2047,32                      :rem 253
61452 DATA -2047,16,4095,16,7,32,7,32,1,8,1,8,0,0,
      900,256                                          :rem 234
61454 DATA 1=L 2=B 4=H,0=N 1=T 2=S 4=P,( 0=NO / 1=
      YES ),( 0=UP / 1=DOWN )                          :rem 231
61456 DATA ( 0=YES ),( 56 TO 900 )                     :rem 243
61458 DATA L,T,B,S,LB,TS,H,P,LH,TP,BH,SP,LBH,TSP,N
      O,UP,YES,DWN                                     :rem 94
61460 DATA CLOBBER,ILLEGAL DURATION,DURATION OVERF
      LOW,STACK UNDERFLOW                              :rem 254
61462 DATA STACK OVERFLOW,UNDEFINED PHRASE CALL
                                                       :rem 164
61470 DATA 20,0,19,1,17,3,8,4,23,5,21,6                :rem 103
```

Once the Editor has been saved, load the "MLX" program from Appendix C. MLX is the same program you used to create "BMG.OBJ" and "SID.OBJ." If you already have a copy of MLX from another COMPUTE! publication, you can use it to enter the listings in this book. Tape users using another version of MLX must change line 763 to match the MLX listing in this book. (Tape users who have a copy of MLX which does not have a line 763 *must* use the version of MLX from this book.)

In order to avoid retyping the same data you entered for SID.OBJ, you'll reload the SID.OBJ file through MLX and append the additional data (actually, you'll be overwriting about 72 bytes and adding more than 650 others). Be sure to follow these directions carefully in order to get a working copy of the Editor.

Load and run MLX, and answer the prompts as follows:

Starting Address: 49349
Ending Address: 51736

If you're using a Datassette, insert the tape which contains a copy of SID.OBJ, created in Chapter 7, and fast-foward past the Sid Loader program to the beginning of SID.OBJ (you should have earlier noted the tape counter number for the start of this file).

If you're using disk, just insert the disk containing SID.OBJ in your drive.

The cursor should be waiting for input on line 49349. Press the L key while holding down the SHIFT key. When you see the prompt for a filename, enter SID.OBJ and press RETURN. Next, respond with T or D for tape or disk. SID.OBJ will load into memory.

When MLX has finished loading SID.OBJ and the prompt returns for line 49349, press the N key while holding down the SHIFT key. You'll be asked for a new address: Enter 51041 and press RETURN.

Using the listing of Program 8-2, enter in the data as you would any other MLX listing. When you're finished, save the program using the filename EDITOR.OBJ. Tape users should save the program immediately following program 8-1, EDITOR.

(The Editor uses most of the memory space from 49152 to 53247. For this reason you cannot run other utilities which use this memory, such as the DOS wedge, when using the Editor.)

Program 8-2. EDITOR.OBJ

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C. Be sure to read the directions above before entering this program.

```
51041 :096,120,169,127,141,013,251
51047 :220,169,001,141,026,208,100
51053 :169,049,141,018,208,169,095
51059 :027,141,017,208,169,147,056
51065 :141,020,003,169,004,141,087
51071 :021,003,088,096,173,025,021
51077 :208,141,025,208,041,001,245
51083 :240,020,172,191,004,185,183
51089 :192,004,141,033,208,185,140
51095 :200,004,141,018,208,206,160
51101 :191,004,048,006,104,168,166
51107 :104,170,104,064,160,007,004
51113 :140,191,004,076,000,000,068
51119 :007,002,014,006,001,006,211
51125 :002,002,002,050,246,222,193
51131 :208,106,082,076,064,160,115
51137 :255,200,185,034,005,170,018
51143 :041,001,069,252,208,245,247
51149 :138,009,001,170,037,251,043
51155 :217,254,004,208,234,132,236
51161 :252,138,073,255,037,251,199
51167 :133,251,138,160,255,200,080
51173 :074,176,252,070,251,136,164
51179 :208,251,096,006,022,014,064
51185 :003,054,015,002,006,047,112
51191 :023,150,010,019,102,014,053
51197 :067,004,000,132,008,038,246
51203 :007,002,086,051,035,118,046
51209 :134,003,010,166,030,083,179
51215 :070,079,182,255,255,014,102
51221 :246,255,254,014,014,254,034
51227 :030,255,014,246,255,031,090
51233 :246,134,014,134,014,255,062
51239 :030,015,255,246,246,255,062
51245 :255,003,015,255,063,246,114
51251 :255,254,255,160,000,177,128
51257 :251,205,114,005,208,009,081
51263 :200,177,251,136,205,115,123
51269 :005,240,023,165,251,024,009
51275 :105,002,133,251,144,002,200
51281 :230,252,205,112,005,208,069
51287 :224,165,252,205,113,005,027
51293 :208,217,096,000,000,000,102
51299 :000,170,010,042,042,041,148
51305 :003,168,138,041,007,208,158
51311 :009,169,255,141,001,208,126
51317 :141,005,208,096,133,253,185
51323 :198,253,010,201,008,233,002
51329 :000,024,121,008,006,168,200
51335 :185,012,006,072,185,028,111
51341 :006,133,251,138,041,056,254
51347 :074,074,074,133,254,170,158
51353 :104,202,048,009,192,015,211
51359 :102,251,106,024,202,016,092
51365 :249,141,000,212,165,251,159
51371 :141,001,212,169,065,141,132
51377 :004,212,185,044,006,170,030
51383 :074,169,160,144,002,169,133
```

```
51389 :144,141,001,208,138,041,094
51395 :254,010,141,000,208,169,209
51401 :000,042,141,016,208,169,009
51407 :007,056,229,254,133,254,116
51413 :010,010,010,101,253,056,141
51419 :229,254,010,133,251,169,241
51425 :197,056,229,251,201,141,020
51431 :176,003,056,233,008,208,147
51437 :002,169,137,141,005,208,131
51443 :169,064,141,004,208,096,157
51449 :001,000,255,151,030,024,198
51455 :139,126,250,006,250,006,008
51461 :172,243,230,143,248,046,063
51467 :060,126,134,142,150,159,014
51473 :168,179,168,179,189,200,076
51479 :212,225,238,253,012,152,091
51485 :104,109,112,117,120,128,207
51491 :120,128,133,136,141,144,069
51497 :149,152,104,168,041,001,144
51503 :170,189,169,006,141,112,066
51509 :004,189,171,006,141,113,165
51515 :004,152,074,170,041,003,247
51521 :168,138,074,074,170,189,110
51527 :161,006,133,252,185,161,201
51533 :006,133,254,189,165,006,062
51539 :141,173,006,185,165,006,247
51545 :141,174,006,169,000,133,200
51551 :251,133,253,169,182,141,200
51557 :250,255,169,004,141,251,147
51563 :255,120,165,001,041,253,174
51569 :133,001,032,006,004,173,206
51575 :173,006,133,252,173,174,006
51581 :006,133,254,169,000,133,052
51587 :251,133,253,032,006,004,042
51593 :165,001,009,002,133,001,192
51599 :088,096,200,224,232,240,199
51605 :216,228,236,244,112,232,137
51611 :003,003,000,000,162,001,068
51617 :032,198,255,032,207,255,116
51623 :032,207,255,032,207,255,131
51629 :133,251,032,207,255,133,160
51635 :252,032,207,255,164,144,209
51641 :208,062,201,034,208,245,119
51647 :160,000,032,207,255,201,022
51653 :034,240,006,153,073,002,193
51659 :200,208,243,132,253,032,247
51665 :207,255,168,208,250,164,181
51671 :253,192,005,144,200,162,147
51677 :003,185,072,002,221,069,005
51683 :002,208,190,136,202,016,213
51689 :244,132,253,160,000,185,183
51695 :073,002,032,210,255,200,243
51701 :196,253,208,245,096,169,132
51707 :032,160,079,153,112,203,222
51713 :136,016,250,096,120,142,249
51719 :040,003,140,041,003,088,066
51725 :096,169,255,201,127,096,189
51731 :253,000,255,000,255,000,014
```

Using the Editor

To use the Editor, plug a joystick into port 2. Then load Program 8-1, EDITOR, and type RUN. The EDITOR.OBJ file automatically loads and the screen blanks while the program initializes. The Editor is a graphically complex program, using redefined characters, sprites, and raster scan interrupts, so it takes awhile to initialize. When the program is ready, it displays a main menu.

1) PLAY MUSIC
2) EDIT MUSIC
3) LOAD MUSIC FILE
4) SAVE MUSIC FILE
5) DISK DIRECTORY or
5) (NOT AVAILABLE)
6) QUIT

EDIT Music

Press the 2 key to select EDIT, and then the 1 key to choose voice 1. The display changes to show the editing screen.

The first thing you'll notice is that it's divided into different levels. The top level shows the current voice number and the amount of free memory remaining. The next level displays the current key signature and tells whether the measure feature is on or off. The level below that has one of the three accidental symbols highlighted.

The next level is the main level, where notes are selected. The main level displays the current pitch in three different forms: as a note on the grand staff, as a piano key, and as a letter and octave number. The current duration is indicated by a symbol and a word. The level below this main level is used for entering rests and ties. The bottom level contains a box in which entered notes appear.

In the main level, the method of using the joystick is:

1. Push the stick up or down to change the current pitch.
2. Push the stick left or right to change the current duration.
3. Press and release the joystick button to enter a note.

While you're in the main level, pushing the stick up or down changes the pitch. The quarter note on the grand staff moves, the next key on the piano is indicated, and the written display changes. A full eight octaves is available by pushing the joystick.

Take a moment to enter a few notes. Just select a pitch and press the joystick button. A note appears in the box in the bottom level and then scrolls to the left. The note looks like a quarter note, the current duration, and has the note name and octave number written above it. As you enter more notes, each one displays in the box—all notes scroll to the left to make room for the next.

If you press the button without changing the pitch, the entered note is set at the same pitch as the previous note.

To change the duration, push the stick left or right. Notice that all durations also have a dot option, except the thirty-second note. Now press the button. The note which shows in the box is in the current pitch, but has a new duration.

For the moment, ignore the durations marked UTILITY and ABS SET.

Now that you've entered a few notes, you might like to hear your musical creation. Return to the main menu by pressing the function key f1. Then press the 1 key for PLAY, followed by the RETURN key to play the voice. When the voice is through, the program waits for a new selection from the main menu.

To continue editing, press the 2 key to select EDIT and then the 1 key to choose voice 1 again. The display switches back to the editing screen, and the notes that you've entered appear in the same position as before.

As notes are entered, the previous notes are scrolled to the left. Since the bottom level can display only a few notes, notes have to scroll off the screen. Sometimes you'll want to scroll those notes back onto the screen to review them. Scrolling can be done manually by using the two cursor keys.

Press the cursor key on the left (ignore the up and down markings). All the notes scroll to the right. The note which scrolled off the screen appears at the left edge, and the most recently entered note moves back into the box. Every time you press this cursor key, the notes scroll one position.

As the notes move, the current pitch and duration change to reflect each note which appears. You can scroll as far as you want. Scrolling stops when you come to the beginning of the voice.

Press the cursor key on the right to scroll in the opposite direction. Scrolling stops at the end of the voice.

Replacing notes. It's easy to correct a mistake made while entering notes. Use the cursor keys to scroll the notes until the one which needs to be changed appears in the box. Select the correct pitch and duration and press the button. The old note is replaced with the new one. You can then scroll back to the end of the voice and continue entering notes.

Insert. If you miss a note and need to insert one, scroll the notes until the insertion point is reached. The note in the box will be to the *right* of the inserted note. Next, press SHIFT–INST/DEL. The note in the box and all notes to its right scroll to the right, creating a blank in which you can enter a note.

If necessary, you can insert several blanks. The only thing you cannot do while blanks exist is scroll or return to main menu. You must fill in all of them to make the voice complete before adding new notes at the end.

Delete. To get rid of an extra note or blank, just press the DEL key (the unSHIFTED INST/DEL key). The note or blank in the box is deleted, and the notes to its right scroll to the left to fill in the gap.

When deleting notes, remember that the keyboard is buffered for up to ten keystrokes. If you press the DEL key a second time, before the Editor is finished deleting the first note, a second note will also be deleted.

Clear. The clear feature is used when you want to delete all the notes from the current note to the end of the voice. This allows you to delete several notes at once. Press SHIFT–CLR/HOME. Since clearing can be disastrous when done accidentally, there's a confirmation prompt. Press the Y key to erase the note currently in the box and all the notes to its right, or hit the N key to cancel the clear.

By scrolling to the beginning of the voice and using SHIFT–CLR/HOME, you can erase the entire voice.

Moving to the beginning or end of the voice. To move to the beginning or end of a voice, use the function keys f2 and f4, respectively. Pressing SHIFT-f1 will immediately take you to the beginning of the voice. Likewise, pressing SHIFT-f3 takes you to the end of the voice.

Accidentals. Figure 8-1 shows three symbols, called *accidentals*, that are often found in sheet music.

Figure 8-1. Accidentals
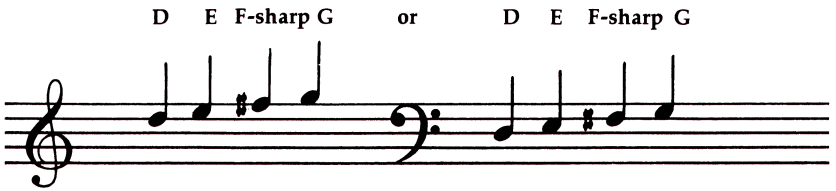
♯ sharp

♭ flat

♮ natural

When you find one of these symbols placed before a note in the sheet music, it means that the pitch of the note should be adjusted up or down slightly. Say, for instance, that you find an F note with a sharp in front of it, something like Figure 8-2.

Figure 8-2. F-sharp
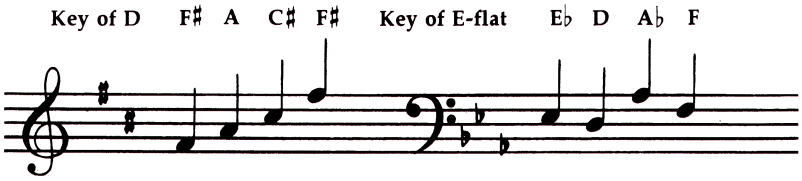


This means that you should enter the note with the pitch F-sharp instead of just a natural F. To do so, use the joystick to move to pitch F, then press the + key (it's on the top row of the keyboard) to select the sharp before you press the button to enter the note.

A sharp increases the pitch of a note. The pitch F-sharp is halfway between pitches F and G. A flat, on the other hand, decreases a note's pitch. To make a pitch flat, press the − key (right beside the + key). If you see a flat symbol before a note, you must move to the pitch, hit the − key, then press the button to enter the note.

To cancel a sharp or flat, press the English pound sign (£) to select the natural pitch.

Key signature. Sometimes you'll find one or more sharp or flat symbols at the left edge of the staff, next to the clef symbol. They might look like this:

Figure 8-3. Key Signatures



These symbols form the *key signature*. The number of sharps or flats in the key signature determines the key in which the music is written. As shown in Figure 8-3, the key of D has two sharps, and the key of E-flat has three flats.

The presence of a sharp or flat in a key signature means that all notes at that position on the staff should be treated as if they had a sharp or flat symbol in front of them. (See Chapter 7 for more details on this and all other aspects of music notation.) For example, a sharp appearing at the position for F in a key signature means that all F notes, in all octaves, should be entered as F-sharps.

Since having to remember which notes should have a sharp or flat can make note entry much more difficult, the Editor has a feature to help. All you have to do is tell the Editor how many sharps or flats are in the key signature, and the Editor automatically selects the appropriate pitches for you.

Look at the level near the top of the screen which displays the number *0* and the letter *C*. The number tells how many sharps or flats are in the current key, and the letter gives the name of the key. The default key is C, which has no sharps or flats.

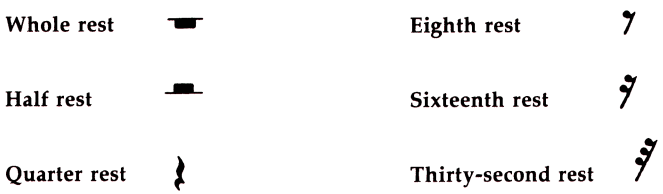
Let's say that the key signature shows two sharp symbols, at the positions for F and C. Press SHIFT-+ (plus key) *twice* to select this key signature. The display changes to show two sharps in the key of D. Now when you push the joystick, the pitches F and C are automatically sharped.

To select a key with more sharps, press SHIFT-+ as necessary. To reduce the number of sharps or to select a key with flats, press SHIFT-− (minus key). Keep pressing until the desired number of sharps or flats is displayed.

If the key has been set and you come across an accidental symbol in front of a note, that accidental overrides what was set by the key signature. You'll have to use the +, −, or £ key to change the pitch for that note.

Rest. A voice does not have to play notes constantly. A rest tells a voice how long it should be silent. Rests have duration but no pitch. Different rest symbols are used for different durations. Figure 8-4 illustrates these symbols.

Figure 8-4. Rests



To select a rest, press the R key. The note name in the main level alters to read (R), and the word *REST* in the level below the main level displays in reverse letters. All notes now entered will be rests.

The Editor cannot display the duration symbols for rests. Rests of different durations can be selected by pushing the stick left or right, but the Editor will still display the duration symbols for normal notes.

The rest mode stays in effect until canceled. To cancel it, either press the R key again, or push the stick up or down to change the pitch.

Tie/slur. Two notes are tied or slurred when they're connected by a symbol which looks like an arc. When the symbol joins two notes of the same pitch, it's called a tie. If the two notes have different pitches, the symbol is called a slur. Notes tied or slurred are played with no break in volume.

Figure 8-5. Ties and Slurs



A tie or slur is selected by pressing the / key (near the right on the bottom row of the keyboard). The word *TIE* in the level below changes to reversed letters. The next note entered is followed by a curved symbol, indicating a tie or slur.

Unlike the rest mode, the tie/slur mode is automatically canceled after a note is entered. To cancel it without entering a note, press the / key a second time.

Special Options

In addition to notes, the Editor lets you enter commands to control things like tempo, volume, waveform, and envelope. Since quite a variety of commands are supported, there's no room to display the choices on the editing screen, so a separate screen is used. Press the function key f3 to switch from the editing screen to the special option screen.

The commands are organized by headings like TEMPO and WAVEFORM. Each command has its own three-letter abbreviation. One of the commands is displayed inside a box, which you can move by pushing the joystick. If the box is moved off the end of one row or column, it wraps around to the other side.

The window in the upper-right corner of the screen displays the full name of the current command. This display changes as the box is moved from one command to another.

To enter a command, first press the joystick button. The window then shows the number range for the selected command. All commands require that a number be entered from the keyboard. Once a number is typed and the RETURN key is pressed, the command is actually entered. The command name and the number show up in the bottom row, the command and the previous notes scroll to the left, and the display returns to the editing screen. If you want to enter another command, you have to press f3 again.

There'll be times when you'll want to enter two or more commands—having to return to the special option screen each time can be inconvenient. Fortunately, there's a way to enter a command without having the display switch back to the editing screen. After you press the button and type the number, press SHIFT-RETURN instead of RETURN. The command is entered, but the screen doesn't change.

If you accidentally press the button on the wrong command, press only RETURN without typing a number. The command is not entered, and you'll be able to choose another.

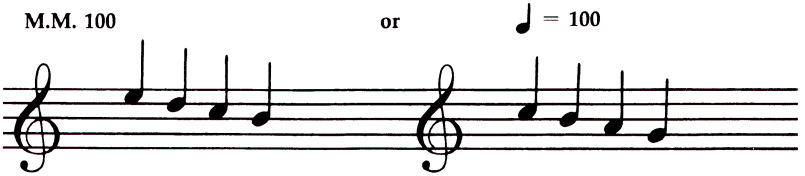
Commands can be scrolled, inserted, and deleted, just like notes. The INST/DEL key can also be used while you're in the special options screen.

To return to the editing screen without entering a command, press the f1 key.

We'll discuss just three commands here. The other commands are for advanced applications and are explained in later chapters.

Tempo. The tempo determines how quickly the notes are played; it is normally specified at the beginning of a song by one of the two methods shown in Figure 8-6.

Figure 8-6. Set Tempo



The TEM command, used to set the tempo, is usually placed at the beginning of a voice. The number part of the command is the desired tempo value. In the preceding example, this number was 100.

There are some restrictions regarding which tempo values can be used. Sidplayer supports only a limited number of tempo values, and of those, some do not permit the use of certain durations. Not all tempo values support sixteenth or thirty-second notes, or dotted whole notes. Table 8-1 shows all the available tempo values and their restrictions.

Those tempos which do not support a sixteenth note do not support a dotted eighth note. Likewise, those tempos which do not support thirty-second notes do not support dotted sixteenth notes.

Tempo values 81 and below do not support dotted whole notes, but this is no major problem. A dotted whole note can be simulated by tying a half note to the whole note.

When you type the number for the TEM command, the Editor uses the closest available tempo value. For example, if you type 160, the command TEM 163 is actually entered because tempo 160 is not available.

In most cases, it doesn't matter if the exact tempo is not available. There's little difference between M.M. 160 and M.M. 163. The only complication would be if the song used sixteenth or thirty-second notes, which are not supported in

Table 8-1. Tempo Values and Restrictions

Tempo	Restrictions
900	No thirty-second notes
600	No sixteenth or thirty-second notes
450	None
360	No sixteenth or thirty-second notes
300	No thirty-second notes
257	No sixteenth or thirty-second notes
225	None
200	No sixteenth or thirty-second notes
180	No thirty-second notes
163	No sixteenth or thirty-second notes
150	None
138	No sixteenth or thirty-second notes
128	No thirty-second notes
120	No sixteenth or thirty-second notes
112	None
105	No sixteenth or thirty-second notes
100	No thirty-second notes
94	No sixteenth or thirty-second notes
90	None
85	No sixteenth or thirty-second notes
81	No thirty-second notes
78	No sixteenth or thirty-second notes
75	None
72	No sixteenth or thirty-second notes
69	No thirty-second notes
66	No sixteenth or thirty-second notes
64	None
62	No sixteenth or thirty-second notes
60	No thirty-second notes
58	No sixteenth or thirty-second notes
56	None

M.M. 163. If that was the case, the tempo value 150 would have to be used.

If you try to use durations like sixteenth notes in a tempo which does not support them, the Editor prints the error message ILLEGAL DURATION when the voice is played. If this should happen, press any key to acknowledge the error, and then change the tempo to one that *does* support the duration.

Sometimes a word like *adagio* or *allegro* is used in place of a number. In these situations, you have to choose the appropriate tempo value based on the information given in Chapter 7.

If the sheet music does not specify any tempo at all, just use whichever tempo sounds best to you. If you don't enter a TEM command, M.M. 100 is used.

Volume. Like tempo, the general volume level (dynamics) of a song is also usually indicated at its beginning. The level is specified with the letters *p* for piano (soft) and *f* for forte (loud).

The master volume of the SID chip can range from 0 to 15, with 15 being the loudest and 0 being off. This volume level is set by the VOL command.
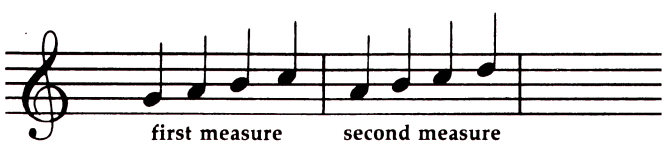
Here are the suggested volume levels for various dynamics markings.

Dynamic	Volume
fff	12
ff	11
f	10
mf	8
mp	7
p	5
pp	4
ppp	3

It's recommended that you not use volume levels above 12, because they can cause notes to distort. If the sheet music does not specify a volume, do not enter a VOL command. The default volume level of 8 (mf) will then be used.

Measures. Notes are organized into groups called measures. Measures help the performer keep in time with other players and are convenient when trying to locate a particular note in a song.

Figure 8-7. Measures



To help keep your place in a voice, measure markers are available with the MS# command. This command is entered with a number ranging from 0 to 999.

Measures in sheet music usually are not numbered, so you'll have to number them yourself. Start at the first measure and write the number *1* above it. Number the next measure *2*, and so on, for the length of the song. Then, before you enter the notes for each measure, enter the appropriate measure marker.

Measure markers have no effect when a voice is played and are used strictly for editing purposes. The use of measure markers is optional but recommended, especially in longer songs.

To enter measure numbers from the editing screen, press the function key f7. The Editor enters the next measure marker, using the number one greater than the previous marker. If the last measure entered was 3, pressing f7 enters MS# 4. This allows you to enter measure markers without going to the special option screen and without typing numbers.

The usual procedure when entering a sequence of measures is to go to the special option screen to set the starting measure number, and then to use f7 to enter successive measure markers.

The real advantage in using measure markers is that the Editor can search for a specific measure marker and move to that point in the voice. This is like moving to the beginning or end, except that now you can quickly move to any location in the voice.

To search for a measure, press the f5 key when in the editing screen. The free memory display changes to a prompt asking for a measure number. Enter the number. The Editor searches the voice from the beginning until it finds the requested measure. If the measure is found, the Editor moves to that marker. If the Editor does not find the measure before it reaches the end of the voice, the Editor simply moves to that position.

If you hit the f5 key accidentally and don't notice the measure number prompt, the joystick and keyboard may seem to be locked up. To recover, just press the RETURN key to cancel the search.

Function Key Summary

At this point it might be good to review the uses of the function keys.
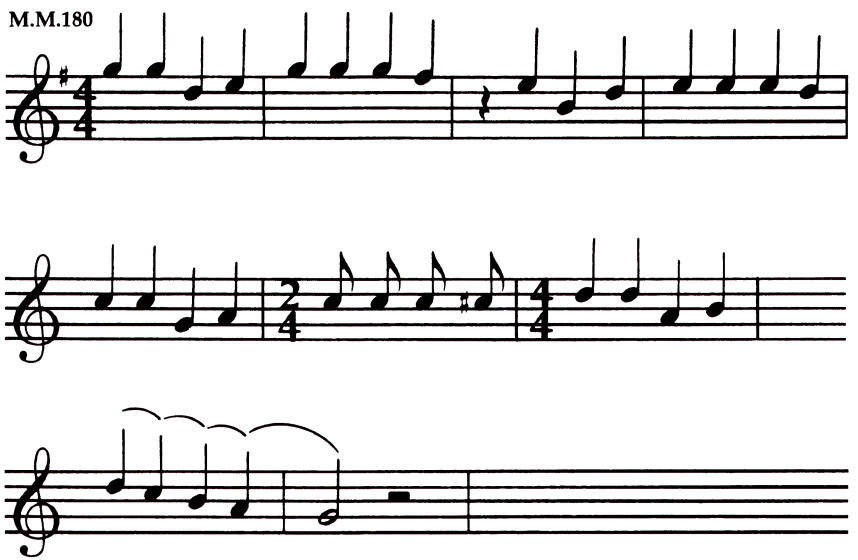
f1 If in editing screen, return to main menu.
If in special options screen, return to editing screen.
f3 Go to special option screen.
f5 Search for measure.
f7 Enter next measure marker.
f2 Move to beginning of voice.
f4 Move to end of voice.

(Note: Figure 8-9 is a cut-out that you can place over the functions keys, and is located at the end of this chapter.)

Entering a Melody

Now that you know something about how the Editor works, how about trying to enter a short song? Just follow this step-by-step example, and you'll have one. Use the sheet music shown as Figure 8-8.

Figure 8-8. Blues



Make sure the Editor is on your screen. Erase the entire voice by moving to the beginning (function key f2) and using SHIFT-CLR/HOME.

The sheet music indicates a tempo of M.M. 180. Go to the special option screen (function key f3) and move to the TEM command. Press the joystick button and type 180, but do not press RETURN. Since you're going to enter a second command, press SHIFT-RETURN instead. This prevents the display from returning to the editing screen when the command is entered.

No volume level is specified, so ignore setting the volume with a VOL command. When the voice is played, the default level of 8 will be used.

You're almost ready to enter the first measure, so move down to the MS# command (the fastest way to get there is to push the stick up and wrap around to the bottom), press the button, and enter the number 1. You can press RETURN this time, because you won't be entering any more commands.

One last thing you need to do before entering notes is set the key. The song is written in the key of G, shown by the key signature displaying one sharp. Select the key of G by pushing SHIFT-+ until the key signature display indicates one sharp.

Select the quarter note duration and enter the first four notes. They are G5, G5, D5, and E5. You've just entered the first measure.

Press the function key f7 to enter the command MS# 2 (beginning the next measure). Type in the next four notes. You'll notice that because of the current key, the F-sharp is automatically selected for the last note instead of F-natural.

Measure 3 begins with a quarter rest. After you press f7 for the next measure marker, press the R key to turn on the rest mode. The rest will be entered when you press the button. The rest mode will still be on, though it's canceled when you move down to enter the note E5.

Measures 4 and 5 are pretty straightforward. Measure 6, however, is different. The notes are eighth notes, not quarter notes, so you'll have to change the duration. The last of these eighth notes has a sharp symbol by it, changing the pitch for that note from C-natural to C-sharp, so you'll have to press the + key to select the sharp.

You may also notice that this measure plays for only two beats, instead of the usual four beats. The fraction 2/4 shows

a change in time signature, an advanced topic which is explained later.

Measure 7 and the following measures are back in 4/4 time. Remember to change the duration back to a quarter note.

The notes in measure 8 are followed by ties, so press the / key to turn on the tie mode before you enter each note.

The last measure, the ninth, should be easy since it contains only one note. Once this note has been entered, the voice is ready to be played.

Error Checking

Sometimes a few mistakes may be made when entering a song. The Editor can find some, like the use of a sixteenth note in a tempo that does not support sixteenth notes, but it cannot find others, such as wrong pitches. Fortunately, the Editor has some features that make it easy to track down mistakes and correct them.

Autostop. If you're playing a voice and the Editor stops with the ILLEGAL DURATION error, the Editor points to the note which caused the error. This means that when you switch to the editing screen, the notes move so that the one with the illegal duration appears in the box.

Manual stop. Playing can also be stopped at any time by pressing the f7 key. When you then go to the editing screen, the note in the box will be the one playing at the moment you hit the key. This feature can be convenient when you're playing a song and you hear a "bad" note.

If you press any key other than f7 while a song is playing, the playing stops, but the Editor leaves the voice alone and doesn't move to the note being played.

Fast forward. Another feature is a fast forward mode. This can be useful when you're entering a long song. Perhaps the beginning of the song plays fine, but some later parts need work. When you choose PLAY from the main menu, playing always starts at the beginning of the song. There's no way to start somewhere in the middle.

With the fast forward mode, you still have to start at the beginning, but you can make it play much faster. If you hold the joystick button down while playing a song, the song plays at three times the normal speed. Playing will return to normal as soon as you release the button.

Slow down. A slow mode is also available. If you let go of the button and push the stick in any direction while playing a song, it slows down to one-third normal speed. Use this feature when you want to listen carefully to a sequence of complicated notes.

Saving the Music

Select this item from the main menu when you want to save a music file to tape or disk. It's a good practice to save your work frequently.

The Editor asks if you want to change the old text lines. The text lines are displayed in the window at the bottom of the main screen. If you've just entered a new song, there will be no text lines, so press the Y key to enter new ones.

The text lines are used to give the title of the song, to identify the composer, and to credit the person who entered the song. Up to four lines of text are allowed. The standard format for the text lines is one or two lines for the title, one line for the composer, and one line for the acknowledgment.

Enter the lines one at a time. Each can hold up to 32 characters. If you don't need all four lines, press RETURN to enter a blank line.

After the text lines have been entered, the Editor requests a filename. Enter one up to 12 characters long, then press RETURN. Do not include the .MUS extension as part of the filename. If you're saving to disk, be sure to choose a filename not already in use. If you're saving to tape, note the exact filename used and the counter number where the SAVE begins. The program will *not* prompt you to press the PLAY and RECORD buttons.

The saving begins once the filename has been entered. When the SAVE is completed, the Editor clears the main menu and waits for your next instruction.

Disk drive errors such as FILE EXISTS and DISK FULL are not reported by the Editor. Always check the red drive light after a SAVE to make sure that it isn't flashing. The only error that the program does report is DEVICE NOT PRESENT, which can happen if the drive is turned off. If this error does occur, press any key to acknowledge it, turn the drive on, and try again. This time, however, you won't have to change the text lines.

To cancel a SAVE, press only RETURN in response to the filename prompt.

Loading a Song

This main menu item is used to load a music file from tape or disk into memory. Type the filename of the desired file (excluding the .MUS extension), and press RETURN. Tape users will *not* be prompted to press PLAY. The Editor loads the song, replacing the one currently in memory.

Note that loading a song erases the one in memory. If you want to preserve the song currently in memory, save it to disk or tape before loading a new one.

If you're using disk and the requested file is not on the disk, the Editor will report the FILE NOT FOUND error.

If you're using tape and enter an incorrect filename, or if you position the tape past the beginning of the data, the program will fail to locate the song. If you believe you entered a filename that is on the tape, but the program fails to find the song, stop the tape and rewind it to a position before the beginning of the data and press PLAY again. The only other way to recover from an incomplete LOAD from tape is to turn the computer off, reload the Editor, and start over.

After the song is loaded, the text lines in the window are updated. The Editor is ready for you to select another item.

If you accidentally choose LOAD, don't type a filename and just press RETURN. The song in memory will not be erased.

What's on the Disk?

This prints a disk directory on the screen. The names of all files with the .MUS extension on the disk are listed in two columns. Printing the directory is useful when you want to check if a filename is already in use or if you want to see how many blocks are free.

The Editor won't let you choose this item if you're using the Datassette (assuming you changed DN=8 to DN=1 as you should have).

Using All Three Voices

You've seen how to enter, edit, and debug one complete voice. But as you know, Sidplayer can play up to three voices at the same time. To access the other two voices, press the 2 or 3 key after choosing EDIT from the main menu.

The voices are completely independent. Editing done on one voice does not affect the others. For instance, you can add

some notes to voice 2 and completely erase voice 3, but when you return to voice 1, you'll find it unchanged.

When it comes to playing, however, the voices are not entirely independent. Multiple voices must share the same tempo and master volume. The TEM and VOL commands can be used on any voice, but they affect all three. If the tempo is set on voice 1, voices 2 and 3 use that same tempo. It's not possible for one voice to play at one tempo while another voice plays at another. The same idea applies to the master volume.

When you choose PLAY from the main menu and press RETURN, the Editor plays all three voices. If one of the voices runs out of notes before the others, that voice stops but the others continue.

It's possible to play individual voices. Instead of pressing RETURN after choosing PLAY, press the Y or N key for each voice. The Editor plays only those voices which were selected by pressing the Y key.

One problem with playing individual voices involves tempo. If the TEM command is used only on voice 1, and voice 1 is not played, the tempo will not be set for the other voices.

When you save or load a file, all three voices are saved or loaded, even if some of the voices are empty.

To clear all three voices at once, press the 2 key in the main menu to select EDIT, then press SHIFT–CLR/HOME.

Program 8-3 is an enhanced version of the blues rhythm given earlier. Voices 2 and 3 are used to add bass and percussion parts. You may wish to load this song into the Editor and examine the individual voices. Voice 1 should be identical to the voice in the earlier demonstration. Voices 2 and 3 use advanced commands explained in later chapters.

Program 8-3. BLUES

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49403
Filename: BLUES.MUS

```
49152 :090,000,054,000,072,000,216
49158 :006,080,030,001,016,149,032
49164 :016,149,016,146,016,147,246
49170 :030,002,016,149,016,149,124
49176 :016,149,016,084,030,003,066
```

```
49182 :016,000,016,147,016,159,128
49188 :016,146,030,004,016,147,139
49194 :016,147,016,147,016,146,018
49200 :030,005,016,145,016,145,149
49206 :016,157,016,158,030,006,181
49212 :020,145,020,145,020,145,043
49218 :020,081,030,007,016,146,110
49224 :016,146,016,158,016,159,071
49230 :030,008,080,146,080,145,055
49236 :080,159,080,158,030,009,088
49242 :012,157,012,000,001,079,095
49248 :006,080,001,071,001,012,011
49254 :001,160,001,132,001,008,149
49260 :030,001,054,016,020,173,146
49266 :001,015,030,003,054,016,233
49272 :020,171,001,015,030,005,106
49278 :054,011,020,169,001,015,140
49284 :020,105,030,007,054,016,108
49290 :020,170,001,015,030,009,127
49296 :012,181,012,000,001,079,173
49302 :006,080,001,007,001,004,249
49308 :001,096,001,132,001,104,235
49314 :038,001,030,001,001,006,239
49320 :016,000,016,141,016,000,101
49326 :016,141,016,000,020,141,252
49332 :020,141,020,000,020,141,010
49338 :020,141,020,000,001,047,159
49344 :030,003,001,002,030,005,007
49350 :016,000,016,141,016,000,131
49356 :016,141,030,006,020,000,161
49362 :020,141,020,141,020,141,181
49368 :030,007,001,002,001,079,080
49374 :066,076,085,069,083,013,102
49380 :067,082,065,073,071,032,106
49386 :067,072,065,077,066,069,138
49392 :082,076,065,073,078,013,115
49398 :013,013,000,013,013,013,055
```

Keyboard Note Entry

The Editor has been designed to use the keyboard, as well as a joystick, for note entry. If you prefer to use the keyboard, or if you just don't have a joystick, follow these instructions.

- Press a letter key from A to G. The current note name will change to that letter.
- To change the octave, press a digit key from 0 to 7.
- The duration can be set by pressing one of the following keys:

W Whole
H Half
Q Quarter
8 Eighth (E is used for a note name)
S Sixteenth
T Thirty-second

- Press the period (.) key if you want a dotted note. Press the key again to cancel the dot.
- Press the RETURN key to enter a note. This is like pressing the joystick button.
- In the special options screen, use the letters *I, J, K,* and *M* to move the box up, left, right, or down. Press RETURN to select a command.

Complete Joystick Editing

If you prefer to use only the joystick to edit, you can. Many of the things done with the keyboard can also be done with the joystick.

Accidentals, rests, and ties can be selected; the joystick can also be used to change the current key and to scroll the notes. But before any of these things can be done, you must first change the levels.

To move from the main level to a different level, press the joystick button and hold it down while you push the stick up or down. The different levels of the screen are highlighted, in order. Release the button to stop on a particular level.

Once you're in a level and have released the button, push the stick left or right to change whatever that particular level controls. For example, if you end up on the level which displays the current key, all the keys will be shown when you push the stick left or right. When you come to the key that you want, press the button to return to the main level.

On the accidental level, the flat, natural, and sharp symbols can be selected one at a time. Wraparound is supported, so if the current accidental is flat, pushing the stick left will switch to sharp. Again, press the button to return to the main level.

The level for selecting a rest or tie works a little differently. As soon as you push the stick left (for rest) or right (for tie), that feature is selected and the Editor automatically returns to the main level. To turn off the rest or tie mode, move to this level and select rest or tie again.

Move to the bottom level and push the stick left or right to scroll the notes. Be aware that keyboard items, including the INST/DEL key, do not work when you're in this level. Press the button to return to the main level.

Leaving the Editor: QUIT

When you're done with the Editor, choose QUIT from the main menu. The prompt CONFIRM? appears. Press the Y key to return to BASIC, or press N to continue editing.

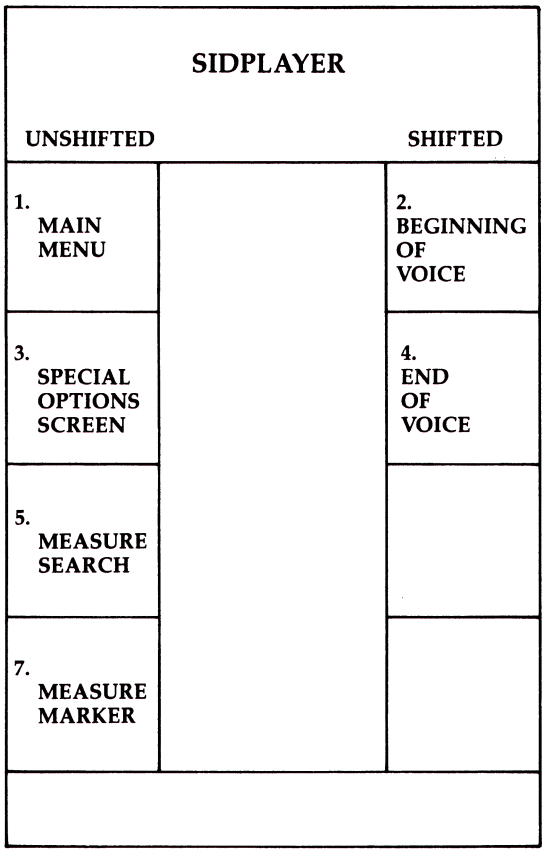
The RUN/STOP key has been disabled for your protection, so you cannot press this key to stop the program. Pressing RUN/STOP–RESTORE will not work either.

One final word of caution: Remember to save your work before you choose QUIT.

This concludes the description of the Sidplayer Editor. You should now be able to enter, edit, and debug a simple piece of sheet music. Music stores and libraries offer a wide variety of classical and contemporary sheet music that will give you plenty of practice.

Once you feel comfortable with the Editor, you may wish to move on to the advanced features explained in the following chapters.

Figure 8-9. Function Key Cutout



Making Music

Waveforms

The *timbre* of a sound is what distinguishes a middle C played on a saxophone from a middle C played on a cello. The main thing that controls a sound's timbre is the type of vibration which produces the sound. There are a few basic types of vibrations, or *waveforms*. These types are named according to their shape when viewed with an oscilloscope.

Triangle waves (Figure 9-1) produce soft, mellow tones. The flute is an example of an instrument which produces triangle waves.

Figure 9-1. Triangle

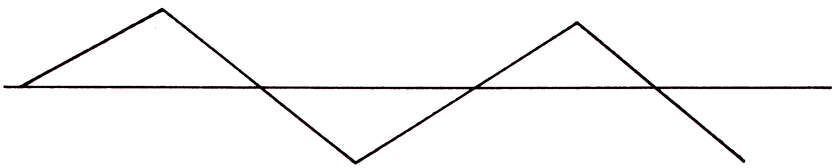


Figure 9-2 shows a waveform for a bright, buzzy tone. Brass instruments produce waves that are basically *sawtooth* waves. (This is sometimes called a *ramp* waveform.)

Figure 9-2. Sawtooth

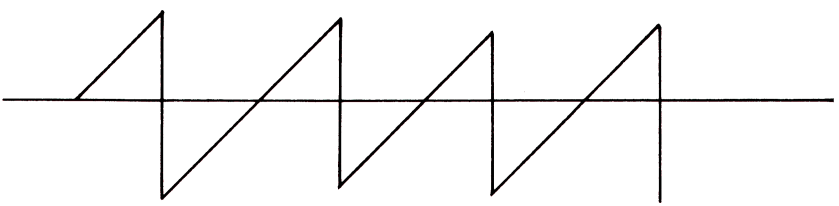
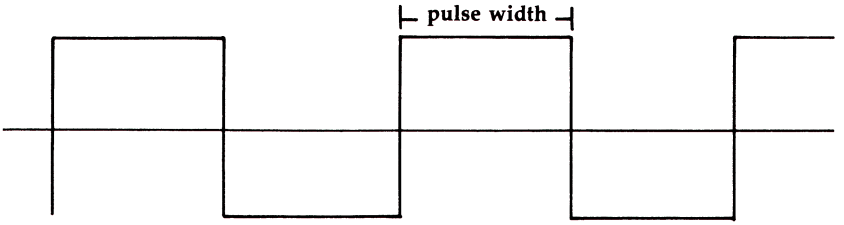


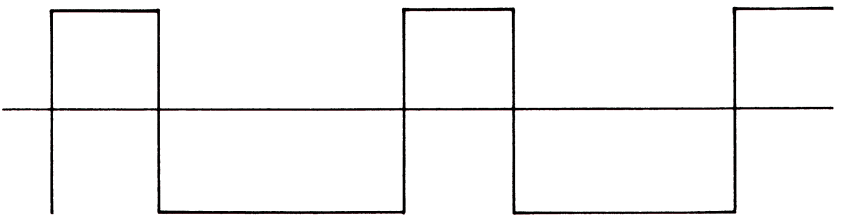
Figure 9-3 illustrates a *square* wave. This waveform sounds rich and hollow, and can be heard from the clarinet.

Figure 9-3. Square



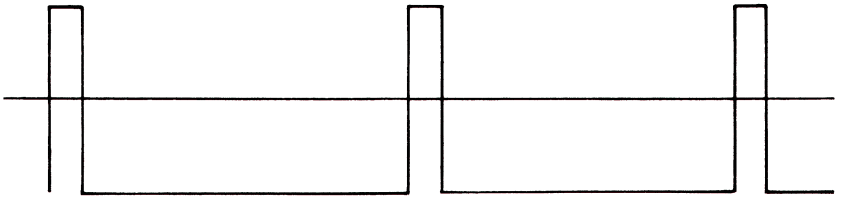
The *pulse* wave alternates between high and low states. The amount of time during one cycle that the wave is high is called the *width*, or *duty cycle*, of the wave. Square waves have a pulse width of 50 percent. When the width is reduced, the wave becomes more rectangular. The waveform shown in Figure 9-4 might be produced by an oboe or bassoon.

Figure 9-4. Rectangular



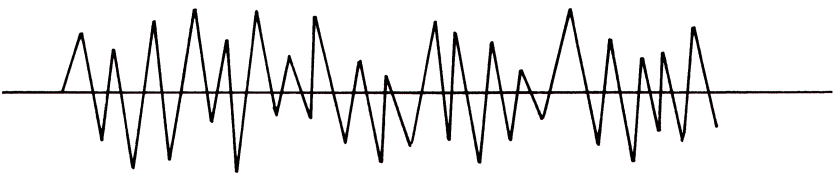
Pulse waves with a very narrow pulse width sound thin and reedy. Pulse waves with widths greater than 50 percent sound just like pulse waves with widths less than 50 percent. For example, a pulse wave with a 40 percent width sounds the same as a pulse wave with a 60 percent pulse width.

Figure 9-5. More Rectangular



Another type of waveform has no definite shape because it's completely random. It's called *noise* (Figure 9-6) because it is the waveform associated with white noise, the sound you hear when a television station goes off the air. Noise is useful for producing percussion effects, such as snare drums.

Figure 9-6. Noise



Another useful waveform, the sine wave, unfortunately is not supported by the SID chip.

The waveforms introduced here are just the basic types. Actual instruments produce more complicated waveforms that may not conform exactly to one of these types.

Setting the waveform. The special option screen in the Editor has a row of commands which pertain to waveforms. Three of these commands are introduced in this chapter. The first command, labeled WAV, is used to select the waveform for a voice.

When you press the joystick button to choose the WAV command, the window in the upper-right corner displays some extra information. The desired waveform has to be specified in the form of a number, so the window shows you the number for each waveform.

0 N Noise
1 T Triangle
2 S Sawtooth
4 P Pulse

After you type the number and press RETURN, the command which appears in the box displays the letter instead of the number.

Each of the three voices can have its own waveform.

Try changing waveforms. Load Program 8-3, "BLUES," from the Editor and play voice 1 with various waveforms.

Pulse width. If you choose the pulse waveform, you may also want to use the P-W command to set the pulse width.

The numeric range for this command is from 0 to 4095, with 2048 being a perfect square wave. Values smaller or larger than 2048 produce rectangular waves.

As you approach the limits of 0 or 4095, the waves become so narrow that the volume begins to decrease. The pulse wave is inaudible when the pulse width is set to 0 or 4095. In most cases, only width values from 100 to 4000 are used.

Try playing a voice with different numbers for the P-W command to hear the effects of the various widths. Just as each voice has its own waveform, each voice also has its own pulse width. Changing the pulse width for one voice will not affect the others.

Pulse width sweeping. An advanced feature of Sidplayer is that it can change the pulse width during a note. When this feature is turned on, the pulse width starts at the specified value, but increases or decreases for the duration of the note. The width is then reset back to the specified value at the beginning of the next note.

The effect of pulse width sweeping is to add a sense of motion to the sound. A common way to use sweeping is to set the pulse width at 2048 and have it increase slowly so that the sweeping is barely noticeable. Another technique is to set the pulse width at 1000 and have it increase rather quickly. You'll hear the sound go from a rectangular wave to a square wave as the width reaches 2048, and then back to a rectangular wave as the width continues to increase.

The P-S command controls pulse width sweeping. Values from 1 to 127 turn on the sweeping. The larger the value, the faster the width increases. Values from −1 to −127 do the same thing, except that they cause the width to decrease. Try 10 for starters. Using the number *0* with the P-S command turns off the sweeping.

If the pulse width is allowed to sweep past 4095, it wraps around to 0. The same thing happens if the width is decreasing and sweeps past 0. When the width wraps around, the effect on the tone is quite noticeable and usually isn't desirable. In most cases the wraparound can be avoided by changing the values for the pulse width, the sweep rate, or the direction.

One interesting use of pulse width wraparound is to set the sweep rate at 127, the maximum value. This produces a raspy tone.

If one note is tied to another note, the pulse width is not

reset when the second note starts playing. The sweeping continues with no break.

When you turn sweeping off with the command P-S 0, the pulse width won't be reset to the specified value at the beginning of the next note. If you want the pulse width to be reset, use the P-W command to set the width again.

Here is a list of the default waveform settings.

```
WAV  P
P-W  2048
P-S  0
```

This gives you a square wave with no sweeping. That's the type of waveform used if none of the waveform commands are placed at the beginning of a voice.

Waveform demonstrations. To close this section, here are two demonstration programs. The first one uses triangle, sawtooth, and pulse wave settings. The second one uses only the pulse wave, but with different pulse widths, plus the use of sweeping.

Program 9-1. FSONATINA.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 50675
Filename: FSONATINA.MUS

```
49152 :238,002,234,001,200,000,163
49158 :006,096,001,174,012,000,039
49164 :001,044,001,096,001,228,127
49170 :001,120,001,039,054,002,235
49176 :038,003,016,148,020,148,141
49182 :020,148,088,148,088,147,157
49188 :088,146,088,145,088,223,046
49194 :088,158,088,157,088,156,009
49200 :038,005,020,155,020,156,186
49206 :020,157,020,158,080,145,122
49212 :020,223,038,003,020,000,108
49218 :030,001,001,228,016,149,235
49224 :020,149,020,149,088,149,135
49230 :088,148,088,147,088,146,015
49236 :088,145,088,223,088,158,106
49242 :088,157,038,005,020,156,042
49248 :020,155,020,156,020,157,112
49254 :038,003,016,158,030,001,092
49260 :001,118,088,145,024,158,130
49266 :088,145,088,158,020,156,001
```

```
49272 :022,009,004,000,092,158,149
49278 :020,157,004,156,092,158,201
49284 :020,157,020,156,001,212,186
49290 :088,148,024,145,088,148,011
49296 :088,145,020,158,004,000,047
49302 :092,145,020,223,004,158,024
49308 :092,145,020,223,020,158,046
49314 :030,003,001,228,088,150,150
49320 :024,148,088,150,088,148,046
49326 :020,146,020,146,088,081,163
49332 :088,146,088,147,088,148,117
49338 :020,149,020,149,001,047,060
49344 :030,002,088,150,088,149,187
49350 :088,148,088,147,088,081,070
49356 :088,146,088,081,088,146,073
49362 :088,081,088,146,088,147,080
49368 :088,148,088,150,088,149,159
49374 :088,150,088,149,088,150,167
49380 :088,149,088,148,088,147,168
49386 :084,146,088,147,088,148,167
49392 :020,145,020,159,020,145,237
49398 :024,159,024,145,024,081,191
49404 :024,146,024,082,024,147,187
49410 :001,142,001,071,001,015,233
49416 :030,004,001,135,130,000,052
49422 :016,148,020,148,020,148,002
49428 :088,148,088,147,088,146,213
49434 :088,145,088,223,088,158,048
49440 :088,157,088,156,020,219,248
49446 :088,154,088,156,020,219,251
49452 :088,154,088,156,016,219,253
49458 :016,000,030,005,066,000,167
49464 :016,211,020,211,020,211,233
49470 :088,211,088,146,088,145,060
49476 :088,223,088,158,088,157,102
49482 :088,156,088,219,020,154,031
49488 :088,153,088,219,020,154,034
49494 :088,153,088,219,016,154,036
49500 :016,000,030,006,130,000,018
49506 :086,008,020,146,088,211,145
49512 :088,089,020,154,088,219,250
49518 :088,089,020,154,084,158,191
49524 :084,092,084,153,020,231,012
49530 :088,166,088,153,020,231,100
49536 :088,166,088,153,016,231,102
49542 :016,000,030,007,020,145,096
49548 :088,159,088,146,020,145,018
49554 :088,159,088,146,020,145,024
49560 :084,149,084,147,084,223,155
49566 :020,158,088,157,088,223,124
49572 :020,158,088,157,088,223,130
49578 :012,158,030,008,086,000,208
49584 :084,223,084,148,084,146,177
49590 :084,158,012,157,084,158,067
49596 :084,147,084,081,084,157,057
49602 :012,156,020,000,084,156,110
49608 :084,146,084,159,020,148,073
49614 :084,146,084,151,084,148,135
49620 :020,138,084,151,084,140,061
49626 :084,138,084,137,084,151,128
49632 :084,150,084,151,020,137,082
49638 :020,000,030,009,088,149,014
```

```
49644 :088,147,088,149,088,147,175
49650 :020,145,020,000,088,137,140
49656 :088,149,088,137,088,149,179
49662 :020,147,020,000,088,139,156
49668 :088,137,088,139,088,137,169
49674 :020,151,084,138,084,140,115
49680 :084,151,020,137,020,000,172
49686 :001,174,001,039,001,252,234
49692 :088,138,088,215,088,138,015
49698 :088,215,020,149,020,000,014
49704 :088,215,088,149,088,215,115
49710 :088,149,020,147,020,000,214
49716 :088,149,088,147,088,149,249
49722 :088,147,020,145,030,010,242
49728 :020,000,001,142,001,071,043
49734 :001,228,001,114,088,138,128
49740 :024,215,088,138,088,215,076
49746 :088,150,088,149,088,150,027
49752 :088,149,088,084,088,149,222
49758 :088,150,088,215,088,138,093
49764 :088,137,088,138,088,137,008
49770 :088,215,088,150,088,149,116
49776 :024,148,084,149,088,150,243
49782 :088,215,020,148,020,147,244
49788 :020,148,030,003,001,135,205
49794 :050,122,020,150,020,148,128
49800 :020,146,076,145,020,145,176
49806 :020,150,020,148,020,146,134
49812 :076,145,020,145,020,150,192
49818 :038,001,024,149,024,148,026
49824 :024,147,024,146,024,145,158
```



```
49830 :024,146,024,145,024,146,163
49836 :024,145,024,146,024,145,168
49842 :024,146,024,145,024,146,175
49848 :024,145,024,146,024,145,180
49854 :024,146,024,145,024,146,187
49860 :024,145,024,146,024,145,192
49866 :024,146,024,145,024,150,203
49872 :024,149,024,148,024,147,212
49878 :024,148,024,147,024,148,217
49884 :024,147,024,215,024,150,036
49890 :024,149,038,003,016,148,092
49896 :016,000,016,215,016,000,239
49902 :044,150,016,000,001,079,016
49908 :006,096,012,000,001,044,147
49914 :001,096,001,228,001,120,185
49920 :038,003,001,039,054,002,137
49926 :024,164,024,166,024,153,049
49932 :024,166,024,164,024,166,068
49938 :024,153,024,166,016,164,053
49944 :016,000,024,231,024,153,216
49950 :024,166,024,153,024,165,074
49956 :024,153,024,164,024,153,066
49962 :016,163,016,000,030,001,012
49968 :024,163,024,231,024,153,155
49974 :024,231,024,163,024,231,239
49980 :024,153,024,231,016,165,161
49986 :016,000,024,166,024,153,193
49992 :024,165,024,153,024,166,116
49998 :024,153,024,163,024,153,107
50004 :016,164,016,000,030,002,056
50010 :001,166,024,166,024,153,112
```

```
50016 :024,166,024,153,024,231,206
50022 :024,153,024,166,024,153,134
50028 :024,231,024,153,024,166,218
50034 :024,153,016,000,024,164,239
50040 :024,153,024,164,024,153,150
50046 :024,165,024,153,024,164,168
50052 :024,153,024,165,024,153,163
50058 :024,164,024,153,030,003,024
50064 :016,000,001,047,076,167,195
50070 :012,153,076,167,012,153,211
50076 :020,000,038,005,084,164,211
50082 :020,165,020,173,016,161,205
50088 :038,003,016,000,001,071,041
50094 :001,015,030,004,001,135,104
50100 :130,000,001,214,054,002,069
50106 :024,164,024,166,024,153,229
50112 :024,166,001,015,016,164,066
50118 :016,000,001,047,024,161,191
50124 :024,164,024,175,024,164,011
50130 :024,161,024,164,024,175,014
50136 :024,164,016,161,016,000,085
50142 :030,005,066,000,001,210,022
50148 :024,239,024,164,024,174,109
50154 :024,164,024,239,024,164,105
50160 :024,174,024,164,016,239,113
50166 :016,000,030,006,130,000,172
50172 :086,008,076,100,012,164,186
50178 :084,165,084,100,084,165,172
50184 :084,100,020,165,084,154,103
50190 :084,231,084,164,030,007,102
50196 :008,163,084,164,084,163,174
50202 :084,164,084,163,020,164,193
50208 :030,008,001,071,084,153,123
50214 :084,166,084,163,012,162,197
50220 :084,163,084,231,084,165,087
50226 :084,162,012,097,084,162,139
50232 :084,166,084,164,084,161,031
50238 :016,175,016,000,008,000,021
50244 :016,156,016,000,030,009,039
50250 :086,000,020,155,020,157,000
50256 :020,155,020,157,020,155,095
50262 :020,157,020,155,020,157,103
50268 :020,155,020,157,020,155,107
50274 :020,157,020,156,020,157,116
50280 :020,156,020,157,020,155,120
50286 :048,000,001,039,020,155,117
50292 :048,000,020,157,048,000,133
50298 :020,223,048,000,001,071,229
50304 :001,162,076,167,016,153,191
50310 :016,000,076,155,016,156,041
50316 :016,000,020,000,084,231,235
50322 :020,153,020,161,016,164,168
50328 :001,135,050,032,016,000,130
50334 :020,000,086,016,084,155,007
50340 :084,157,084,223,016,158,118
50346 :016,000,020,000,084,155,189
50352 :084,157,084,223,016,158,130
50358 :016,000,020,000,084,155,201
50364 :084,157,084,223,084,157,209
50370 :084,155,084,157,084,223,213
50376 :016,158,016,000,016,157,051
50382 :016,000,016,156,016,000,154
```

```
50388 :016,153,016,000,044,164,093
50394 :016,000,001,079,006,096,160
50400 :012,000,001,044,001,096,122
50406 :001,228,001,120,038,003,109
50412 :001,039,054,002,008,000,084
50418 :008,000,008,000,008,000,010
50424 :008,000,008,000,076,164,248
50430 :012,163,076,164,012,163,076
50436 :008,000,001,071,001,015,100
50442 :008,000,008,000,008,000,034
50448 :008,000,008,000,008,000,040
50454 :008,000,012,000,012,000,054
50460 :008,000,008,000,008,000,052
50466 :001,071,012,000,016,165,043
50472 :016,000,001,135,130,000,066
50478 :020,153,020,000,020,153,156
50484 :020,000,020,153,020,000,009
50490 :020,153,020,000,020,153,168
50496 :020,000,020,153,020,000,021
50502 :020,154,020,000,020,154,182
50508 :020,000,020,153,048,000,061
50514 :001,039,020,153,048,000,087
50520 :020,157,048,000,008,000,065
50526 :008,000,012,000,001,135,250
50532 :050,032,076,164,016,163,089
50538 :016,000,076,231,016,166,099
50544 :016,000,008,000,001,135,016
50550 :130,000,086,016,012,000,106
50556 :016,156,016,000,016,000,072
50562 :016,000,016,156,016,000,078
50568 :016,000,016,000,016,000,184
```



```
50574 :016,000,016,156,016,000,090
50580 :016,153,016,000,016,166,003
50586 :016,000,016,149,016,000,095
50592 :044,172,016,000,001,079,216
50598 :083,079,078,065,084,073,116
50604 :078,065,032,073,078,032,018
50610 :070,013,076,046,032,086,245
50616 :065,078,032,066,069,069,051
50622 :084,072,079,086,069,078,146
50628 :013,068,069,077,079,078,068
50634 :083,084,082,065,084,069,157
50640 :083,032,087,065,086,069,118
50646 :070,079,082,077,083,013,106
50652 :067,079,085,082,084,069,174
50658 :083,089,032,074,069,082,143
50664 :082,089,032,066,082,065,136
50670 :068,089,013,000,005,074,231
```

Program 9-2. GSONATINA.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49841
Filename: GSONATINA.MUS

```
49152 :018,001,232,000,096,000,091
49158 :006,096,001,064,086,003,006
49164 :001,214,001,022,076,165,235
49170 :084,166,084,165,084,166,255
49176 :084,167,016,165,084,157,185
```

```
49182 :020,154,084,154,020,167,117
49188 :084,167,020,165,076,153,189
49194 :084,167,084,166,084,167,026
49200 :084,153,016,166,020,154,129
49206 :020,155,020,154,020,167,078
49212 :020,153,020,166,012,165,084
49218 :020,166,020,165,020,166,111
49224 :020,167,012,153,020,155,087
49230 :020,154,020,155,020,092,027
49236 :084,157,020,154,084,155,226
49242 :020,153,084,167,020,154,176
49248 :084,153,020,166,076,166,249
49254 :016,165,016,000,001,047,091
49260 :086,006,076,166,084,153,167
49266 :084,167,084,166,084,167,098
49272 :084,154,084,153,084,158,069
49278 :084,157,084,092,084,155,014
49284 :084,154,084,153,012,167,018
49290 :084,154,084,153,084,167,096
49296 :084,153,084,155,084,154,090
49302 :084,159,084,158,084,157,108
49308 :084,092,084,155,084,154,041
49314 :016,153,084,158,084,157,046
49320 :084,092,084,155,084,154,053
49326 :084,153,016,167,084,157,067
49332 :084,092,084,155,084,154,065
49338 :084,153,084,167,016,166,088
49344 :084,162,084,163,084,100,101
49350 :084,165,084,166,084,167,180
49356 :080,153,084,167,084,153,157
49362 :084,154,084,153,084,166,167
49368 :084,167,001,018,012,154,140
49374 :016,154,016,154,012,154,216
49380 :016,157,016,159,012,146,222
49386 :016,146,016,146,012,159,217
49392 :012,000,012,154,016,154,076
49398 :016,154,012,154,080,157,051
49404 :080,159,016,146,016,000,157
49410 :016,092,016,000,016,157,043
49416 :016,000,016,165,016,165,130
49422 :012,165,012,165,008,165,029
49428 :001,047,001,079,006,096,250
49434 :001,064,002,200,086,020,143
49440 :001,230,001,038,076,175,041
49446 :076,161,016,175,044,000,254
49452 :076,174,076,173,016,108,155
49458 :044,000,020,173,020,162,213
49464 :020,175,020,162,020,173,114
49470 :020,164,020,162,020,164,100
49476 :020,173,020,163,020,161,113
49482 :020,163,020,173,020,161,119
49488 :020,174,020,161,080,175,198
49494 :080,161,080,162,080,170,051
49500 :080,165,080,162,016,173,000
49506 :016,000,001,047,020,108,034
49512 :020,170,020,108,020,170,100
49518 :020,173,020,170,020,173,174
49524 :020,170,016,174,044,000,028
49530 :020,173,020,170,020,173,186
49536 :020,170,020,174,020,170,190
49542 :020,174,020,170,016,175,197
49548 :044,000,016,174,012,000,130
```

```
49554 :016,108,016,173,016,000,219
49560 :016,000,016,171,016,108,223
49566 :044,000,008,000,001,034,245
49572 :001,134,054,002,020,170,033
49578 :020,174,020,161,020,174,227
49584 :001,015,020,170,020,173,063
49590 :020,175,020,173,020,170,248
49596 :020,173,020,175,020,173,001
49602 :001,047,054,002,020,170,232
49608 :020,108,020,174,020,108,138
49614 :001,015,054,002,020,170,212
49620 :020,173,020,175,020,173,025
49626 :001,015,001,130,020,170,043
49632 :020,108,020,174,020,108,162
49638 :020,170,020,174,020,161,027
49644 :020,174,016,175,016,000,125
49650 :016,175,016,175,012,175,043
49656 :012,175,008,175,001,047,154
49662 :001,079,006,096,001,064,245
49668 :002,200,086,020,001,246,047
49674 :001,054,076,173,076,108,242
49680 :016,173,044,000,076,170,239
49686 :076,170,016,170,044,000,242
49692 :008,000,008,000,008,000,052
49698 :008,000,001,047,008,000,098
49704 :008,000,008,000,008,000,064
49710 :016,170,016,000,016,000,008
49716 :016,170,016,170,012,000,180
49722 :016,170,016,170,044,000,218
49728 :008,000,001,050,054,007,184
49734 :008,000,001,015,016,173,027
```

```
49740 :016,000,016,181,016,181,230
49746 :080,181,080,173,080,170,078
49752 :080,183,008,181,001,047,076
49758 :001,079,083,079,078,065,223
49764 :084,073,078,065,032,073,249
49770 :078,032,071,013,076,046,166
49776 :032,086,065,078,032,066,215
49782 :069,069,084,072,079,086,065
49788 :069,078,013,085,083,069,009
49794 :083,032,080,085,076,083,057
49800 :069,032,087,073,068,084,037
49806 :072,032,083,087,069,069,042
49812 :080,073,078,071,013,067,018
49818 :079,085,082,084,069,083,124
49824 :089,032,074,069,082,082,076
49830 :089,032,066,082,065,068,056
49836 :089,013,000,146,024,145,077
```

Envelopes

Dynamics describe the general volume of a song, as set by the master volume, but do not describe the changes in volume which occur while an individual note is playing. These changes in volume over the course of a note are referred to as the *envelope* of the note.

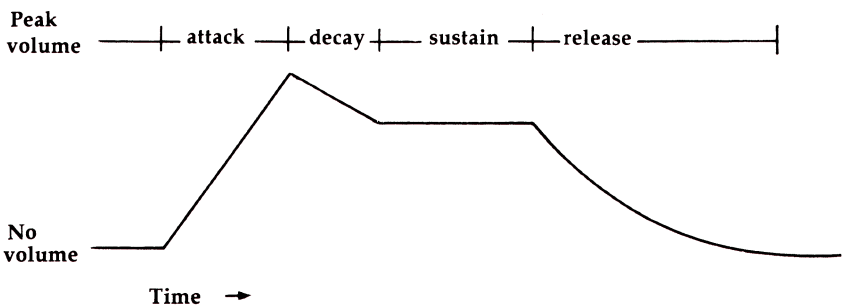
When a note first starts playing, the volume must increase from no volume to the peak volume established by the dynamics. The speed at which the volume rises is called the *attack* rate.

The volume then diminishes slightly until it reaches an intermediate volume level called the *sustain* level. The rate at which the volume falls from the peak level to the sustain level is called the *decay* rate.

Toward the end of a note, the note will be released and the volume will begin to fade away, at a speed called the *release* rate.

A good way to understand the four stages of the envelope is to graph them as shown in Figure 9-7.

Figure 9-7. ADSR Envelope



This standard envelope is sometimes also called an *ADSR envelope*, because of the attack, decay, sustain, and release sequence.

Not all instruments have the same ADSR values. Each instrument has its own characteristic envelope. In electronic music, the ability to control the envelope of a voice lets you more closely approximate a particular instrument.

There are two basic types of envelopes. One type is used for sustaining instruments. These include string instruments which are played with a bow, and wind instruments.

The other type is for nonsustaining instruments. String instruments which are plucked and percussion instruments such as drums are examples of nonsustaining instruments.

Sustaining envelopes. For an example of a sustaining envelope, let's consider a person playing a violin. At the beginning of a note, the player has to dig in with the bow to start the string vibrating. This is the attack.

Once the sound has started, the player does not have to apply quite so much pressure to the string, and the volume is reduced a little. This is the decay part of the envelope.

As the player continues to bow, a constant volume level is maintained. This is the sustain level.

At the end of the note, the player stops the bow, but the string continues to vibrate for a moment until the sound fades away completely. The instant when the player stops the bow is called the release, and the rate at which the volume fades away is the release rate.

The whole process works similarly for a wind instrument, such as a flute. The player has to blow with a little extra force to start the air vibrating, and then eases off slightly. The air continues to vibrate for a moment after the release, when the player stops blowing.

Nonsustaining envelopes. Nonsustaining instruments have completely different envelopes. The most important characteristic of these instruments is that they are struck. The instrument is hit or plucked once for each note. No continual force is applied, so the volume is never sustained.

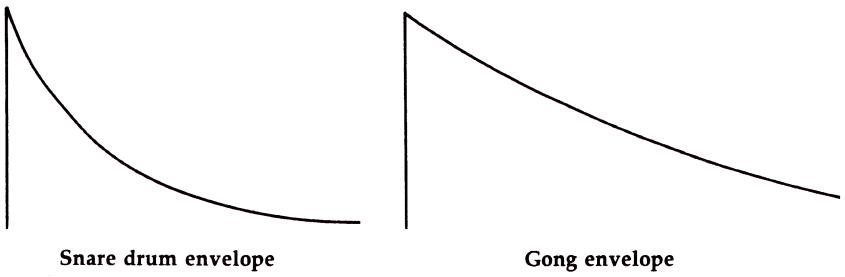
Most percussion instruments have nonsustaining envelopes. These instruments include drums, bells, and others like wood blocks. They usually have very fast attack rates. The decay rate varies from one instrument to another. The sound of a snare drum decays rather quickly, but the sound of a gong takes a long time to decay. Since there is no sustain, there is no sustain part to the envelope, and the envelope can be called an *ADR envelope.*

Figure 9-8 shows two graphs that illustrate nonsustaining envelopes.

Setting the envelope. The special option screen has five commands that are used to configure the envelope.

The **ATK** command lets you choose one of 16 attack rates. The rates are numbered 0–15. At rate 0 the attack takes

Figure 9-8. ADR Envelope



just a fraction of a second (two milliseconds to be precise), while at rate 15 the attack takes eight seconds.

The **DCY** command is used to set the decay rate. Again, the rates are numbered 0–15, with 0 being the fastest, but the range is a little different. The decay takes 6 milliseconds at rate 0, and 24 seconds at rate 15.

Use the **SUS** command to set the sustain level. If the sustain level is set to 0, the volume decays to no volume. If the level is set at 15, the volume doesn't decay at all and stays at the peak volume set by the master volume control. Values between 0 and 15 correspond to evenly spaced volume levels between zero and peak volume. When you're using a nonsustaining envelope, the sustain level must be set to 0.

The **RLS** command sets the release rate. The rates, numbered 0–15, are the same as those for the DCY command. If you're using a nonsustaining envelope, the release rate should be set to the same value as the decay rate.

The release point. The one remaining command is **PNT,** which is used to set the release point. In order to understand how to use this command, it's first necessary to know how Sidplayer handles durations.

Sidplayer deals with note durations in terms of time units called *jiffies*. One jiffy lasts about 1/60 second. Table 9-1 shows the jiffy length of each note duration in each tempo.

Table 9-1. Tempo

M.M.	W	H	Q	E	S	32	64
900	16	8	4	2	1	–	–
600	24	12	6	3	–	–	–
450	32	16	8	4	2	1	–
360	40	20	10	5	–	–	–
300	48	24	12	6	3	–	–
257	56	28	14	7	–	–	–
225	64	32	16	8	4	2	1
200	72	36	18	9	–	–	–
180	80	40	20	10	5	–	–
163	88	44	22	11	–	–	–
150	96	48	24	12	6	3	–
138	104	52	26	13	–	–	–
128	112	56	28	14	7	–	–
120	120	60	30	15	–	–	–
112	128	64	32	16	8	4	2
105	136	68	34	17	–	–	–
100	144	72	36	18	9	–	–
94	152	76	38	19	–	–	–
90	160	80	40	20	10	5	–
85	168	84	42	21	–	–	–
81	176	88	44	22	11	–	–
78	184	92	46	23	–	–	–
75	192	96	48	24	12	6	3
72	200	100	50	25	–	–	–
69	208	104	52	26	13	–	–
66	216	108	54	27	–	–	–
64	224	112	56	28	14	7	–
62	232	116	58	29	–	–	–
60	240	120	60	30	15	–	–
58	248	124	62	31	–	–	–
56	256	128	64	32	16	8	4

The number of jiffies for a whole note is different for each tempo. This number is repeatedly cut in half to give durations like half note, quarter note, and so on. After a certain point, some numbers cannot be evenly divided by two, which is why some tempos do not support sixteenth or thirty-second notes.

The PNT command determines how many jiffies from the end of a note the volume should be released. For an example, let's say that the release point is set at 6 and the current tempo is M.M. 100. A quarter note in this tempo is 36 jiffies long. For the first 30 jiffies, the volume goes through the attack, decay, and sustain phases of the envelope. Then the

voice will be released, and for the last 6 jiffies the volume falls from the sustain level to no volume.

The range of the PNT command is 0–255, but usually only very small values are used. Values like 3, 4, and 5 work best. It's preferable not to set the release point too high, or notes of short duration won't be heard. For instance, in M.M. 100, an eighth note is 18 jiffies long. If the release point was set at 20 jiffies, an eighth note would be released as soon as it started playing. The volume would never have a chance to rise, and the note would not be heard.

For best results, always make sure that the release point is less than the duration of the shortest note in the song.

If you're using a nonsustaining envelope, the PNT command must be used to set the release point at 1.

Here are the default values for the envelope. This configuration produces an organ effect.

ATK 2
DCY 0
SUS 15
RLS 5
PNT 4

If some default settings are satisfactory but others are not, you have to change only the ones that need new values.

Also, remember that you can change the envelope at any point in a song. For example, a voice may briefly switch to a nonsustaining envelope to play a few notes and then switch back.

Each voice can have its own envelope, but it's recommended that the attack rate be nearly the same for all three voices.

Experiment with different envelopes. Load BLUES and try different envelope settings on the first voice.

Sometimes in sheet music you may see what appears to be a very long tie symbol spread over several notes. This symbol indicates that the notes are to be played smoothly, in one breath or one bow movement. The term *legato* is used to describe this type of effect.

A sequence of notes can be played in a legato style by using the command PNT 0. When the release point is set to 0, notes are never released. This saves you from having to put a tie on each note.

Another style of playing is called *staccato,* and is the opposite of legato. A dot placed above or below a note in sheet music means that the note should be played in a quick, light, choppy manner.

One way to produce this effect on Sidplayer is to set the release point as large as possible. A more reliable method is to switch to a nonsustaining envelope.

A nonsustaining envelope is also a good choice when you want to play a series of short, fast notes. The release point usually has to be set very small to play these notes, and if a sustaining envelope is used, the notes may sound too legato. Using a nonsustaining envelope insures that the notes will be played distinctly and will not be run together.

Envelope Example

The following song, "Theme and Variation" by Beethoven, uses a wide variety of envelopes.

Program 9-3. Theme and Variation

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49877
Filename: T&V.MUS

```
49152 :054,001,176,000,148,000,123
49158 :006,192,001,052,054,002,057
49164 :001,022,020,159,020,159,137
49170 :116,158,092,159,092,145,012
49176 :020,155,020,092,020,157,232
49182 :024,154,024,000,020,158,154
49188 :024,154,024,000,020,159,161
49194 :024,154,024,000,001,047,036
49200 :020,145,020,159,016,158,054
49206 :001,018,020,092,020,155,104
49212 :016,154,001,008,001,015,255
49218 :001,020,001,104,054,002,248
49224 :020,146,020,146,084,146,122
49230 :024,145,024,000,120,159,038
49236 :092,158,120,159,092,145,082
49242 :084,159,020,158,020,147,166
49248 :020,147,084,147,024,146,152
49254 :024,000,120,145,092,159,130
49260 :120,145,092,146,084,145,072
49266 :024,159,024,000,001,018,084
49272 :020,145,020,158,016,157,124
```

```
49278 :001,152,001,015,012,000,051
49284 :054,002,001,086,001,004,024
49290 :001,144,001,132,001,152,057
49296 :038,001,024,000,024,146,121
49302 :024,149,024,151,024,147,157
49308 :024,151,024,150,024,137,154
49314 :024,000,001,000,001,252,184
49320 :001,008,038,004,024,153,140
49326 :024,092,024,158,024,167,151
49332 :024,158,024,157,024,159,214
49338 :024,000,001,020,001,088,064
49344 :024,154,088,159,024,158,031
49350 :024,000,024,154,088,145,121
49356 :024,159,024,000,001,047,203
49362 :024,145,024,000,024,159,074
49368 :024,000,024,158,088,089,087
49374 :024,154,001,082,024,092,087
49380 :024,000,024,155,016,154,089
49386 :001,015,054,002,024,000,074
49392 :024,081,088,147,024,146,238
49398 :024,000,024,159,088,146,175
49404 :024,145,024,000,024,223,180
49410 :088,145,024,159,024,159,089
49416 :024,158,024,093,024,158,233
49422 :024,000,024,082,088,084,060
49428 :024,147,024,000,024,081,064
49434 :088,147,024,146,024,000,199
49440 :024,159,088,146,024,145,106
49446 :024,145,024,159,024,223,125
49452 :024,159,001,082,024,145,223
49458 :024,000,024,158,016,157,173
49464 :001,015,001,079,006,192,094
49470 :001,052,054,002,001,038,210
49476 :084,154,020,090,016,155,075
49482 :020,000,020,153,020,167,198
49488 :020,000,020,092,020,000,232
49494 :020,157,020,000,001,047,075
49500 :020,158,020,157,016,092,043
49506 :001,034,020,154,020,089,160
49512 :016,100,001,008,001,015,245
49518 :001,088,054,002,016,154,169
49524 :016,154,016,154,016,154,114
49530 :016,154,016,154,016,154,120
49536 :016,154,001,034,020,155,252
49542 :020,092,016,167,001,152,070
49548 :001,015,012,000,001,020,189
49554 :001,236,001,072,054,002,000
49560 :001,102,016,000,012,000,027
49566 :016,000,020,000,088,157,183
49572 :024,092,020,000,088,158,034
49578 :024,157,024,000,001,047,167
49584 :024,157,024,000,024,157,050
49590 :024,000,024,092,020,000,086
49596 :001,098,024,154,024,000,233
49602 :024,089,016,100,001,015,183
49608 :054,002,016,154,016,154,084
49614 :020,154,020,154,016,154,212
49620 :016,154,016,154,020,154,214
49626 :020,154,016,154,001,098,149
49632 :024,158,024,000,024,092,034
49638 :016,167,001,015,001,079,253
49644 :006,192,001,052,054,002,031
```

```
49650 :001,054,016,165,048,161,175
49656 :020,162,016,173,016,162,029
49662 :016,165,001,047,020,100,091
49668 :020,165,016,154,001,050,154
49674 :020,166,020,165,016,166,051
49680 :001,008,001,015,001,088,130
49686 :054,002,016,167,016,166,187
49692 :016,165,016,162,016,153,044
49698 :016,167,016,100,016,165,002
49704 :001,050,020,161,020,162,198
49710 :016,173,001,152,001,015,148
49716 :012,000,001,020,001,236,066
49722 :001,072,054,002,001,118,050
49728 :016,157,016,153,016,162,072
49734 :016,173,016,162,016,165,106
49740 :001,047,020,163,020,097,168
49746 :016,162,001,114,020,166,049
49752 :020,165,016,162,001,015,211
49758 :054,002,016,167,016,166,003
49764 :020,165,020,165,016,162,136
49770 :016,153,016,167,020,100,066
49776 :020,100,016,165,001,114,016
49782 :020,161,020,162,016,173,158
49788 :001,015,001,079,084,072,120
49794 :069,077,069,032,065,078,008
49800 :068,032,086,065,082,073,030
49806 :065,084,073,079,078,013,022
49812 :076,046,032,086,065,078,019
49818 :032,066,069,069,084,072,034
49824 :079,086,069,078,013,068,041
49830 :069,077,079,078,083,084,124
49836 :082,065,084,069,083,032,075
49842 :069,078,086,069,076,079,123
49848 :080,069,083,013,067,079,063
49854 :085,082,084,069,083,089,170
49860 :032,074,069,082,082,089,112
49866 :032,066,082,065,068,089,092
49872 :013,000,024,148,024,147,052
```

The Filters

Sometimes it's not enough to set the waveform and envelope of a voice in order to imitate the sound of a particular instrument. It may also be necessary to control the harmonic content of the voice by using the filter.

When an oscillator generates a tone, it produces not only the requested pitch, but some harmonics as well. *Harmonics* are frequencies related to the main pitch. The first harmonic is the frequency of the main pitch and is also called the *fundamental frequency*. The second harmonic has a frequency twice the fundamental frequency. The frequency of the third harmonic is three times that of the fundamental frequency, and so on.

Because the volume of the fundamental frequency is always greater than the volume of the other harmonics, the main pitch detected by your ear is that of the fundamental frequency. The harmonics, however, give the tone its *timbre.*

Since each instrument has its own characteristic timbre, the ability to control the harmonic content of a voice can be helpful in emulating a particular instrument. The filter is used to remove selected frequencies from a tone. This enables you to emulate a whole new variety of instruments.

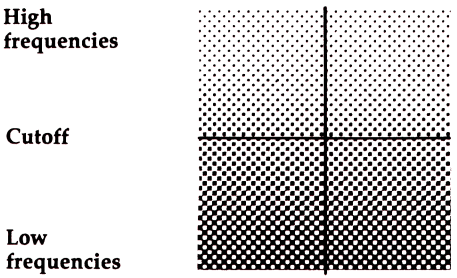
The SID chip's filter has three main control parameters that must be set before the filter can be used. These parameters are the *mode,* the *cutoff frequency,* and the *resonance.*

Filter mode. The filter mode determines which types of frequencies are removed from a tone. The most commonly used mode is the low-pass mode. This mode allows only the frequencies below a certain frequency, called the cutoff frequency, to pass through the filter. Any frequencies above the cutoff are *attenuated* (greatly reduced in volume) so that they are seemingly removed from the tone.

The low-pass filter mode (illustrated by Figure 9-9) produces full-bodied tones. The opposite of the low-pass mode is the high-pass mode, in which frequencies below the cutoff are suppressed while frequencies above the cutoff are passed through unaltered.

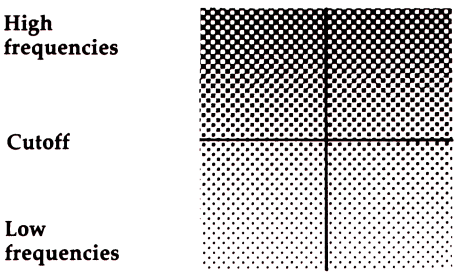
The high-pass mode causes tones to sound tinny or buzzy.

Figure 9-9. Low-Pass Filters



Low-pass filter

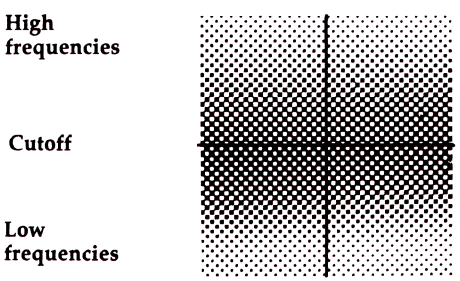
Figure 9-10. High-Pass Filters



High-pass filter

One other filter mode is the band-pass mode. In this mode, only the frequencies near the cutoff are passed through the filter. All other frequencies are attenuated. The band-pass mode produces thin, open tones.

Figure 9-11. Band-Pass Filters



Band-pass filter

Use the F-M command to set the filter mode. This command works similarly to the WAV command. When you press the joystick button for F-M, the window displays a number for each filter mode.

1 L Low pass
2 B Band pass
4 H High pass

Type the number of the desired mode and press RETURN to enter the command. The letter indicating the filter mode appears as part of the command.

You can even select combinations of filter modes. Just add the appropriate numbers together. For example, choosing 5 sets the filter for both low- and high-pass modes. This combination is also called the *band reject* or *notch* mode.

Filter cutoff. The cutoff frequency of the filter acts as a dividing line. In low-pass mode, for example, frequencies above the cutoff are attenuated while frequencies below the cutoff are passed through unaltered. But consider what happens when the pitches of notes in a song fall right around the cutoff frequency. Notes having pitches below the cutoff are played correctly, but notes with pitches above the cutoff are not heard.

What you need is a way to set the cutoff higher or lower. The ideal setting is to place the cutoff right above the highest note to be played. That way, all of the fundamental frequencies will be below the cutoff, so all of the notes will be heard. Most of the harmonics will still be above the cutoff, so they'll be removed from the tone.

The F-C command is used to set the filter cutoff. The range of this command is 0–255, with 0 the lowest and 255 the highest. The standard practice is to start with a value between 0 and 255, play the voice, and then move the cutoff up or down as necessary.

Resonance. The effect of resonance is to produce a sharper tone by emphasizing, or *peaking*, the frequencies in the tone that are close to the cutoff. Actually, the resonance control of the SID chip is not very effective and acts mainly as a way to control the volume of the voice being passed through the filter.

Use the RES command to set the resonance level. The number for this command ranges from 0 (no resonance) to 15 (maximum resonance). In most cases the value 15 is best, but

sometimes you may want to use a smaller value to produce a muted effect.

Passing a voice through the filter. Once you've configured the filter by using the F-M, F-C, and RES commands, use the FLT command to indicate that the voice should be passed through the filter. This command works a little differently from the others. Unlike the previous commands which support a number range, the FLT command gives you a simple choice of either yes (1) or no (0). Enter 1 to indicate that you want the voice to be passed through the filter.

Here's an example showing how you might use the filter. Start with the following commands.

```
F-M L
F-C 255
RES 15
FLT Y
```

Next, enter a few notes. A simple scale is sufficient. When you play the voice, it should play normally. In low-pass mode with the filter cutoff set at the maximum value, all frequencies are passed through the filter.

Now, lower the cutoff by reducing the number for the F-C command, and play the voice again. Do this a few times, each time reducing the F-C command number by about 20 or so. The effects of filtering should gradually become more noticeable.

To stop the filtering, enter the FLT command with the number 0 to indicate that you don't want the voice to be passed through the filter. You don't have to change the mode, cutoff, or resonance settings.

```
FLT N
```

If you want the voice to be passed through the filter later in the song, enter the FLT command with the number 1. The earlier filter settings will still be in effect, but can be changed if necessary.

Autofilter. Trying to find a proper setting for the filter cutoff can often be inconvenient and time-consuming. As an alternative to the F-C command, Sidplayer offers a special feature called the *autofilter mode.* When the autofilter mode is turned on, the filter cutoff is automatically set according to the pitch of each note. Since the cutoff is calculated for each note, all notes, high and low, produce the same filtering effect.

To turn on the autofilter mode, select the command AUT

and enter the number 1. Do this instead of entering the F-C command.

```
F-M  B
AUT  1
RES  15
FLT  Y
```

You may also specify that an offset should be added to the cutoff. When you enter the number for the AUT command, choose any number from 1 to 127 or from −1 to −127. Different offset values give different filtering effects. The autofilter mode is turned off by the command AUT 0.

When using the autofilter feature, it's important that you turn it off whenever you stop passing the voice through the filter.

```
FLT  N
AUT  0
```

Filter sweep. Just as pulse width sweeping changes the pulse width during a note, filter sweeping lets you increase or decrease the cutoff frequency while a note is playing.

Use the F-S command to turn on filter sweeping. The number for this command controls the sweep rate and direction, and ranges from −127 to +127. Values of 1–127 make the cutoff sweep upward. The larger the number, the faster the sweep. Values of −1 to −127 sweep downward. Zero turns off the sweeping.

In most cases, the best results are obtained by using small numbers for the sweep rate, such as values 1–10. If the cutoff is swept too far, it will wrap around, but this is not as useful as it is with pulse width sweeping.

Be sure to turn the filter sweeping off when you stop passing the voice through the filter.

```
FLT  N
F-S  0
```

Or if using the autofilter mode:

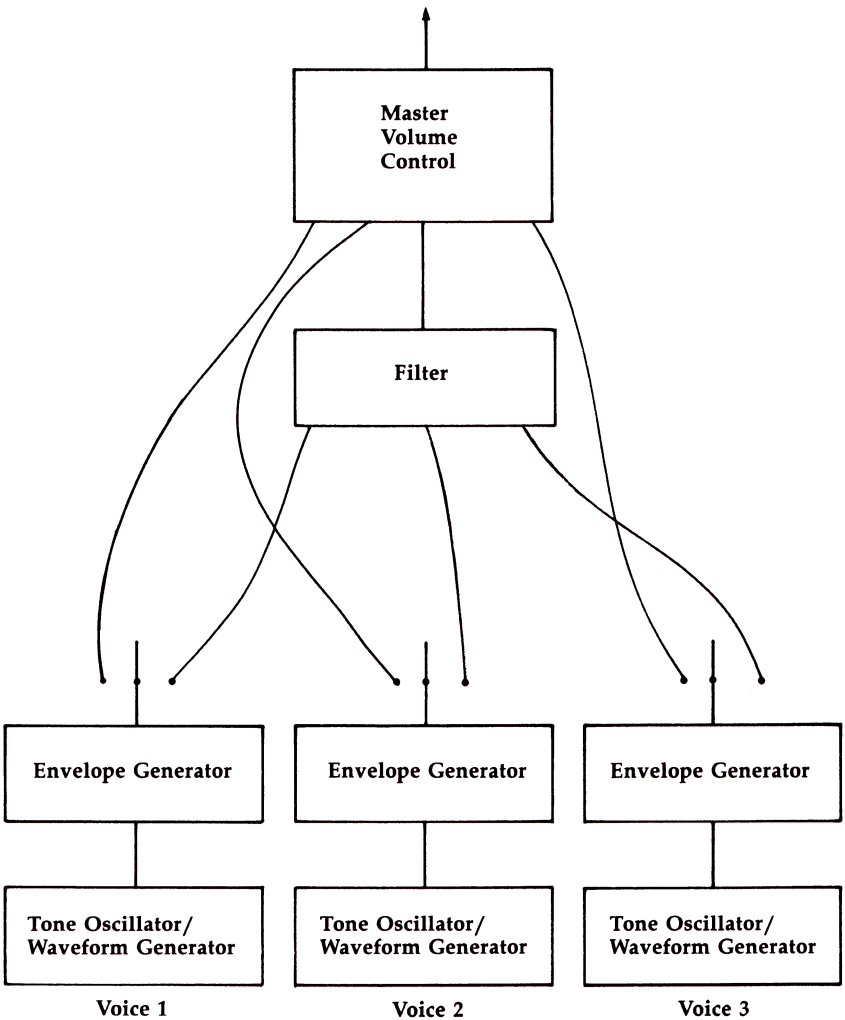
```
FLT  N
AUT  0
F-S  0
```

Restrictions. While the SID chip has three oscillators, and can therefore play three voices, it has only one filter. There's not a separate filter for each voice. Figure 9-12 should help

you understand the arrangement. The illustration is the same as Figure 7-1 except that it has been modified to include the filter.

As shown by Figure 9-12, any voice can be passed through the filter, but only one voice should be routed through the filter at the same time. Passing two or three voices through the filter at the same time can overpower it, causing it to produce an unpleasant buzzy noise.

Figure 9-12. Filtering Sound



The major drawback to using the filter is that there is extreme variance in filtering effects from one computer to another. The same filter settings can produce completely different results on different Commodore 64s. On some computers, especially the older ones, a voice can be almost inaudible when it is passed through the filter, no matter what settings are in effect. The variance lies not only in the SID chip, but in the support circuitry as well, so there's no easy way to fix the problem. If you have one of the earliest versions of the Commodore 64, you may not be able to use the filter.

Three Demonstrations

Three demonstration songs have been provided. In the first one, a theme is played by a trumpet, a trombone, then a tuba. The other two demonstrations emulate a koto and a sitar.

Program 9-4. BRASS.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49307
Filename: BRASS.MUS

```
49152 :072,000,002,000,002,000,076
49158 :006,128,001,022,001,071,235
49164 :001,055,150,020,001,250,233
49170 :001,027,102,003,001,044,196
49176 :001,000,001,252,001,072,095
49182 :038,001,001,047,001,038,156
49188 :020,153,012,156,012,164,041
49194 :020,166,048,153,020,156,093
49200 :048,158,016,156,020,156,090
49206 :008,145,008,153,001,047,160
49212 :150,010,102,000,001,076,143
49218 :166,179,001,034,150,010,094
49224 :166,181,001,034,001,079,022
49230 :001,079,001,079,083,087,152
49236 :079,082,068,032,077,079,245
49242 :084,073,070,013,066,089,229
49248 :032,087,065,071,078,069,242
49254 :082,013,084,082,085,077,013
49260 :080,069,084,044,032,084,245
49266 :082,079,077,066,079,078,063
49272 :069,044,032,084,085,066,244
49278 :065,013,067,079,085,082,005
49284 :084,069,083,089,032,082,059
```

```
49290 :079,066,069,082,084,032,038
49296 :072,073,071,071,073,078,070
49302 :083,013,000,151,024,147,056
```

Program 9-5. KOTO.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49301
Filename: KOTO.MUS

```
49152 :084,000,002,000,002,000,088
49158 :006,144,001,055,150,246,096
49164 :001,250,001,027,102,080,217
49170 :001,004,001,128,001,132,029
49176 :001,008,038,001,001,135,208
49182 :018,044,086,010,016,153,101
49188 :016,156,016,157,016,222,107
49194 :016,157,020,156,024,157,060
49200 :024,222,020,157,020,156,135
49206 :016,218,030,002,020,000,084
49212 :020,156,020,218,020,153,135
49218 :020,218,020,156,020,157,145
49224 :020,222,024,157,024,156,163
49230 :024,157,024,222,020,156,169
49236 :020,218,020,153,001,079,063
49242 :001,079,001,079,074,065,133
49248 :080,065,078,069,083,069,028
49254 :032,075,079,084,079,032,227
49260 :068,069,077,079,078,083,050
49266 :084,082,065,084,073,079,069
49272 :078,013,067,079,085,082,012
49278 :084,069,083,089,032,082,053
49284 :079,066,069,082,084,032,032
49290 :072,073,071,071,073,078,064
49296 :083,013,013,013,000,078,088
```

Program 9-6. SITAR.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49313
Filename: SITAR.MUS

```
49152 :104,000,002,000,002,000,108
49158 :001,151,150,236,001,250,027
49164 :001,027,102,020,001,020,183
49170 :001,160,001,228,001,200,097
```

```
49176 :038,007,001,135,050,232,231
49182 :086,010,006,096,030,001,003
49188 :054,002,016,163,016,165,196
49194 :012,153,044,161,001,015,172
49200 :030,002,054,002,028,145,053
49206 :028,145,028,159,028,159,089
49212 :028,158,028,158,028,157,105
49218 :028,157,028,156,028,156,107
49224 :028,155,028,155,028,154,108
49230 :028,154,028,167,012,167,122
49236 :003,020,012,153,001,015,032
49242 :020,153,020,153,024,153,101
49248 :024,153,028,153,028,153,123
49254 :028,153,028,153,016,153,121
49260 :001,079,001,079,001,079,092
49266 :083,073,084,065,082,032,021
49272 :068,069,077,079,078,083,062
49278 :084,082,065,084,073,079,081
49284 :078,013,067,079,085,082,024
49290 :084,069,083,089,032,082,065
49296 :079,066,069,082,084,032,044
49302 :072,073,071,071,073,078,076
49308 :083,013,013,013,000,137,159
```
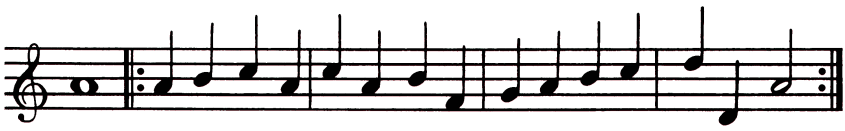
Repetition

As you listen to a song, you can sometimes hear a group of measures played more than once. Repetition occurs often in music, especially in contemporary songs.

The simplest form of repetition is when a group of measures is repeated immediately after being played. So that the repeated measures do not have to be written twice, special symbols are used in sheet music to indicate when a sequence of measures should be repeated. The symbol that marks the beginning of a repeat consists of two vertical lines followed by two dots (shown in Figure 9-13). The end of a repeat is marked by a symbol that looks almost the same, except that the dots come before the double lines, instead of after them.

When the music is played and the first repeat symbol is reached, the playing continues as usual. Upon reaching the end of the repeat, however, playing does not continue to the next measure. Instead, playing jumps back to the measure which had the beginning repeat symbol and continues from that point. When playing comes to the end repeat mark the second time around, it's ignored and playing continues with the next measure.

Figure 9-13. Repeats



Normally, a sequence of notes is repeated only once. There are, however, cases where a voice may repeat several times, such as in a bass line.

Sidplayer has two commands to support repeats. The beginning of a repeat should be marked by the command HED, which stands for *repeat head.* The number for this command determines how many times the sequence of notes is to be played. In a standard repeat, the notes are repeated once, meaning that the sequence is played a total of two times, so you would normally type the number *2.* Then you would press RETURN to enter the command and continue entering notes.

At the end of the repeat, enter the TAL command (repeat tail). There's no data value for this command, so just enter the number *0.*

When the voice is played, the sequence of notes between the HED and TAL commands plays the designated number of times.

The number entered for the HED command must fall in the range 0–255. The number *1* means that the sequence should be played only once, which would seem to make the repeat structure unnecessary. A value of 1 is used only when you're developing a piece of music. If you keep replaying a song to listen for bad notes at the end, you don't want to wait for repeats earlier in the song, so use 1 at first (just remember to change it to the correct value before you save the final version of the song).

Using 0 with HED makes the notes repeat forever. This feature may be useful in some game applications, but should not be used in normal music, because the song will never end.

Repeats cannot be nested. Whenever you have a HED command, it has to be later followed by a TAL command before another HED can be used. It's okay for a voice to contain several repeats; you just cannot have a repeat inside a

repeat. Each voice can have its own repeat, however, so each voice can repeat independently of the others.

When repeats are used properly, there should be one repeat end for every repeat beginning. If a voice is playing and a TAL command is encountered with no previous HED command, playing will repeat forever back to the most recent HED command. If no HED command has been used at all on the current voice, playing will repeat forever back to the beginning of the voice.

Phrases. Occasionally, you'll find that a repeat has a first and a second ending. This means that one set of notes should be played at the end of the sequence the first time through, and a different set should be played the second time. The simple repeat structure of the HED and TAL commands cannot handle this—in this case, you have to use phrases.

If you consider that a repeat is a loop, then a phrase is like a subroutine. A phrase allows the same sequence of notes to be played at different places in the music. The first time the phrase is played, the beginning and end are remembered by Sidplayer. This is called *defining the phrase.* Later in the music when the notes have to be played again, playing can be made to jump back to the beginning of the phrase by the use of a single command. This is known as *calling the phrase.* When the end of the phrase is reached, playing continues with the rest of the song.

It's important to understand that there are some differences between Sidplayer phrases and BASIC subroutines. In BASIC, a subroutine is usually put at the end of a program. Every time the subroutine has to be executed, it's called by the GOSUB statement. With Sidplayer, the notes and commands that form the phrase are placed in the song at the first instance where the phrase must be played. After the phrase has played once, and is thereby defined, it can be played again by use of the phrase call.

To define a phrase, first enter the DEF command. This command needs a number in the range 0–15. For now, just enter the command with 0, then enter the notes which form the phrase.
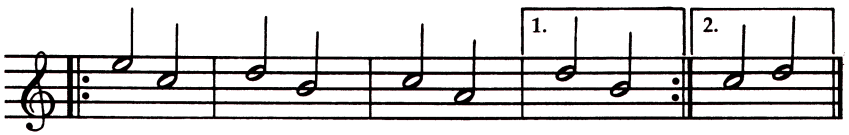
After the last note in the phrase, enter the command END. Like the TAL command, the END command has no data value, so it should be entered by typing the 0 and pressing RETURN.

When playing reaches the END command, the notes have played once, the definition is complete, and the phrase is ready for calling.

To call the phrase, all you have to do is enter the CAL command with the number *0*. This one command takes the place of all the notes in the phrase.

Figure 9-14 demonstrates how phrase calling works.

Figure 9-14. Phrases



Begin by entering the command DEF 0, followed by the notes up to but not including the first ending. Now enter the END command. The notes for the first ending come next. At this point, playing is supposed to repeat back to the beginning. Instead of entering all those notes again, just enter the command CAL 0. Finally, enter the notes for the second ending.

When the voice plays, the phrase plays the first time. The DEF and END commands have no effect. The notes for the first ending then play. The command CAL 0 sends playing back to the first note after the command DEF 0 to play the phrase again. When the END command is reached, playing continues with the first note after the CAL 0 command.

A phrase can be called more than once. A phrase is still defined after it has been called, so it can be called as many times as necessary.

Phrases have many uses besides handling repeats with different endings. In fact, phrases are so useful that quite often 1 is not enough. Don't worry—Sidplayer can remember up to 16 independent phrases. When you enter the DEF command, the number from 0 to 15 identifies which phrase is being defined. That particular phrase can later be called by using the same phrase number with the CAL command.

If, while playing a song, a CAL command tries to call a phrase that has not been defined earlier in the music, the Editor stops with the UNDEFINED PHRASE CALL error.

The 16 phrases are shared among the three voices. For example, even though phrase 7 may be defined on voice 1, it can also be called on voice 2 or 3. This means that playing can temporarily jump to another voice.

This can cause a problem when you are playing individual voices. For example, if you play only voices 2 and 3, and voice 2 contains a call defined in voice 1, playing will stop with the UNDEFINED PHRASE CALL error.

It's even possible for one phrase to call another. Phrase calls can be nested to a limit of four levels on each voice. If you try to exceed the limit, the Editor stops and reports the STACK OVERFLOW error. You can also define one phrase inside another. Be aware that a phrase definition counts as one nesting level. The only thing a phrase definition cannot do is call itself. If the definition of phrase 3 directly or indirectly contains a call to phrase 3, an infinite loop results. Playing will eventually stop with a STACK OVERFLOW error.

Phrases can be redefined. If a phrase is no longer needed, the phrase number can be used in the DEF command of a new phrase. This lets you use more than 16 phrases during the course of a song.

The STACK UNDERFLOW error occurs if DEF and END commands are not properly matched and playing comes across an END command with no previous DEF command.

Da capo, dal segno, and coda. Repeats are not the only kind of repetition. Other forms include *da capo* and *dal segno.* Da capo is indicated in sheet music by the letters *D.C.*, and means that the playing should jump back to the beginning of the voice and continue from there, this time ignoring all repeats. The playing may be stopped before the end of the song by the use of the word *fine.*

Dal segno, identified by the letters *D.S.*, means that playing should jump to the measure marked by a special sign. This sign looks like a slash with dots to either side, passing through a fancy letter *S.* Playing continues from this point, and stops either at the end of the song or at a *fine,* whichever comes first.
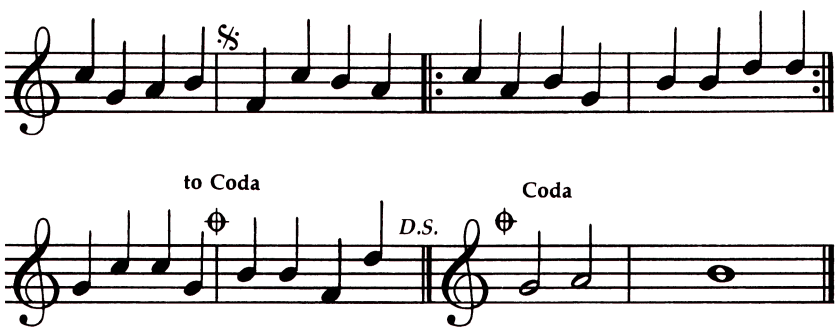
There's one other symbol often encountered when da capo or dal segno has been used. After playing has jumped back to the beginning of the voice or to a particular measure, you may encounter the message *To Coda,* followed by a coda symbol. The coda symbol looks like a letter *O* with a cross

passing through it. This means that playing is going to jump to another place again, but this time, instead of jumping back, the playing skips ahead. At the end of the sheet music you should find some measures labeled as *Coda*, with the coda symbol shown again. Playing jumps to the first of these measures and continues to the end of the song.

Through the clever use of phrase calling, Sidplayer can handle these advanced forms of repetition. The example shown in Figure 9-15, though condensed and not necessarily typical, uses repeats, dal segno, and a coda. Below is the order in which the measures would be played.

1. The first two measures are played (the sign is ignored).
2. The next two measures are played and then repeated.
3. The following two measures are played (the coda symbol is ignored).
4. At the D.S., playing goes back to the measure which had the sign (the second measure) and continues from there.
5. When playing reaches the coda symbol, playing jumps to the coda (the last two measures) and then ends.

Figure 9-15. Repeats, D.S., and Coda



Phrases are also handy if you need to frequently alternate between two or more voice settings. A phrase can consist of only commands and no notes. Just be careful not to make a voice too complex. Too many commands, whether in a phrase or not, can cause the CLOBBER error in the fast-forward play

mode. The CLOBBER error means that processing on one voice took too much time.

One practice to avoid is defining several phrases at the beginning of a song, before any notes are played. This increases the risk of a CLOBBER error. Remember, you should not define a phrase until the first time it's needed, and then call it for all later times.

Chapter 10

Sophisticated Sounds

Advanced Music Theory

Music can be defined in many ways, but one way is to say that it consists of change—changing pitches, changing durations, and changing volumes. Chapter 7 introduced these characteristics of music on a simple level.

This chapter reexamines these same characteristics, but from a broader viewpoint. For example, the volume level may not stay the same throughout a song. It may change at different places in the music. Or the tempo might change. Even the key may change during a song. Changes like these may not happen in a simple piece of music, but they certainly do occur in longer, more sophisticated works.

Now that you have some experience using the Sidplayer, it's time to look at these more advanced aspects of music. The purpose of this chapter is to cover all the remaining elements of notation commonly found in sheet music. When you finish with these descriptions and explanations, you'll be able to enter almost any piece of music with the Sidplayer and Editor.

Tempo changes. There are two kinds of tempo changes. In the first kind, the tempo changes abruptly, perhaps from a slow to a fast speed. This is most often found at the beginning of a movement or major part of a piece of music.

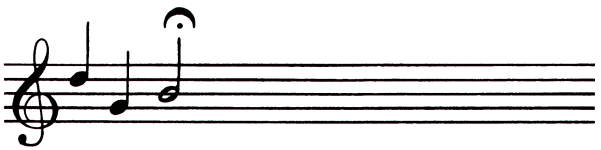
In the second kind of tempo change, the tempo increases or decreases gradually. This type of change is marked by the words *accelerando* and *ritardando*, which respectively mean that the tempo should start getting faster or slower. For instance, if the tempo is currently M.M. 100 and you see *ritard*, you might change the tempo to M.M. 94, then change it to M.M. 90 a few notes later.

When making such tempo changes, be sure that the voices stay synchronized. Make tempo changes only at a point in the music where all three voices are starting a new note. Usually, the beginning of a measure is a suitable place to change the tempo.

(Incidentally, if the tempo is ever indicated by a half note followed by an equal sign, instead of a quarter note and equal sign, double the tempo value when entering the TEM command.)

The symbol used to briefly change the tempo is called a *fermata,* and looks like a narrow semicircle with a dot below it. A fermata can be placed over a particular note or rest to extend its duration. Glance at Figure 10-1 for a sample. This is sometimes referred to as a hold.

Figure 10-1. Fermata

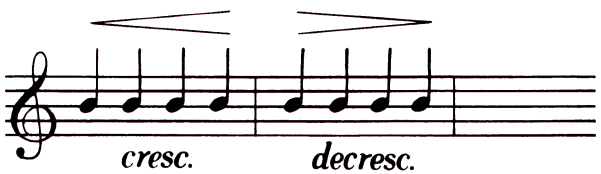


The easiest way to handle a fermata is to enter the corresponding note with a longer duration than written.

Dynamics. Volume changes are the most common type of change in music. As with tempo, the dynamics can change immediately or gradually. Immediate changes are indicated in the normal way, using letter combinations such as *p* and *fff.* Gradual changes are marked by the words *crescendo* and *decrescendo,* which respectively mean that the volume should gradually increase or decrease.

Gradual volume changes can also be denoted by symbols placed above notes, as shown in Figure 10-2.

Figure 10-2. Crescendo and Decrescendo



Use the VOL command for abrupt volume changes, when the master volume is to be changed by more than one level. For gradual changes, however, you may prefer to use the BMP command. This command is used to bump the master volume up or down one level.

When you press the joystick button while on the BMP command in the special option screen, the window indicates that 0 should be entered to bump the volume up one level, or that 1 should be entered to bump the volume down one level.

After you press the RETURN key, the command is displayed with the name *BMP* and the letters *UP* or *DWN*.
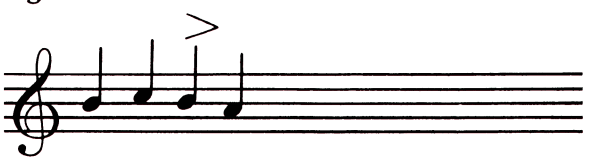
To produce the effect of a crescendo or decrescendo, bump the volume up or down every few measures. The BMP command is also useful in repeat loops at the end of a song, where the volume has to fade out.

A BMP UP command does not change the master volume if the volume is at 15, the highest level. Likewise, a BMP DWN command has no effect if the volume is already 0.

Remember that the VOL command should be used on only one voice. This is also true of the BMP command.

An *accent mark* is the symbol used to change the volume of one note. It looks like a *greater than* sign (Figure 10-3). When placed above a note, it means that the note should be played just a little louder than the others.

Figure 10-3. Accent Mark



Since there is no volume control for individual voices, accents are hard to simulate. You might set the sustain level a little higher, or sustain the note a little longer by using a smaller release point. Insert these changes before the note to be accented. After the note, reset the sustain level or release point to the previous values.

Key changes. Usually, the clef symbols are drawn at the beginning of each staff. They're followed by sharp or flat signs that indicate the key in which the music is to be played. These symbols specify the key signature.

A piece of music does not have to use just one key for an entire song. If a key signature appears somewhere in the middle of the sheet music, it's indicating a change in the key. Any previous sharps or flats are canceled, and only the sharps or flats specified by the new key signature are to be used.

Sometimes you'll see natural signs appearing in a key signature when there is a key change. These natural signs are often used when changing keys to show that the sharps and flats of the previous key no longer apply.

Since key changes are not as common as tempo or volume changes, they can be easy to miss when reading sheet music. A good suggestion is to look over the music before entering notes so that you'll be expecting a key change if there is one.

Time signatures. At the beginning of sheet music, right after the key signature, you'll often find a fraction such as 4/4. The top number tells how many beats there are per measure, and the bottom number tells how many beats there are per whole note. Together, these two numbers define the *time signature.*

Up until now you've used only what's called *4/4 time,* in which each measure contains four beats, and a whole note is four beats long.
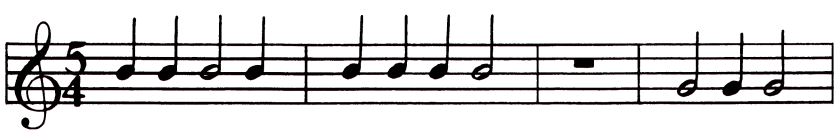
When using different time signatures, there may be more or less than four beats in every measure, and a whole note may not always be four beats long.

One of the more common time signatures is 3/4 time. This is just like 4/4 time, except that there are only three beats in every measure. Waltzes are always written in 3/4 time.

Changing the number of beats in a measure really does not affect Sidplayer. As long as you follow the sheet music, there should be no problem. The important thing is that each measure have the same number of beats.
use the whole rest symbol as a *measure rest* symbol, to indicate a full measure of rest. For example, if a song is written in 5/4 time and you see a whole rest, the sheet music is *not* wrong. It means that the rest should last for five beats.

The five-beat rest shown in Figure 10-4 would be entered as a whole rest and a quarter rest. In other time signatures, a measure rest may last for a different number of beats.

Figure 10-4. Measure Rest



When the bottom number is 4, a whole note plays for four beats, so one-fourth of that duration, a quarter note, plays for one beat. If the bottom number is 2, however, a whole

note is only two beats long, and a half note is one beat long. The quarter note plays for only half a beat. When the number on the bottom is 8, a quarter note plays for two beats, and the note for one beat is now an eighth note. The tempo stays the same; the number of beats per minute remains unchanged. It's the number of beats per note that's different.

Sidplayer is designed to expect a whole note to always be four beats so that a quarter note is always one beat. There's no way to change the number of beats for these standard durations. Time signatures which have a number other than 4 on the bottom can be used indirectly, however, by fooling Sidplayer into thinking that a whole note is longer or shorter than it actually is.

Consider the time signature 2/2, in which a whole note is two beats, compared to four beats in 2/4. A whole note is seemingly reduced to half its normal duration. This can be achieved on Sidplayer by doubling the tempo. At faster tempo selections, more beats per minute means that each beat takes less time, so whole notes are shorter. Therefore, when the tempo is M.M. 100, 2/2 time can be simulated by actually using M.M. 200. To use a time signature where the bottom number is 8, as in 3/8, the tempo should be cut in half, making whole notes play twice as long as normal.

There's another way to show a time signature without using numbers. The letter *C* placed where the time signature belongs indicates 4/4 time. The C stands for *common time*. If the C has a vertical line passing through it, it indicates 2/2 time, also known as *cut time*.

One last word about time signatures—they can change while a song is playing. Such changes are indicated by double bars followed by a new fraction or symbol.

Accidentals. In any octave, there are 12 different pitches, including naturals, sharps, and flats. Earlier you saw that a song will use only 7 of these pitches, according to the current key. Let's retract that now. Once in a while a song may have to play a note using a pitch not in the key. The "BLUES" piece is written in the key of G, meaning that only one note is sharp (F-sharp), but the melody line had to play a C-sharp at one instance.

Special exceptions like this are handled by placing an accidental sign immediately before the note that is to be sharp or flat. This accidental overrides the current key signature for that particular pitch.

Furthermore, the effect of the accidental sign holds true for all following notes of the same pitch. A sharp sign placed in front of a C affects not only that C note but any other C notes in the same octave which may come later. The changed accidental is not permanent, though, and is canceled at the next measure.

An accidental sign used in this way affects only the designated pitch. All other pitches remain the same.

You already know how to change a natural pitch into a sharp or flat. What about the other direction—removing a sharp or flat from a pitch to make it natural? This can be done by using a natural sign. Placed in front of a note, it cancels the sharp or flat for all following notes of the same pitch, but only within the current measure.

A common mistake is to forget that an accidental on one note also affects later notes of that pitch in the same measure. To help with this problem, the Editor supports a measure feature.

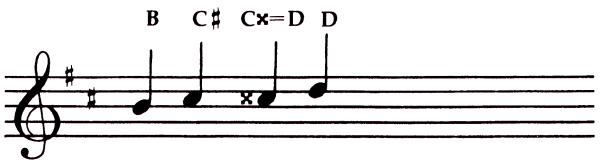
The key signature level shows whether the measure feature is on or off. Press the M key to turn this feature on. Now, whenever you enter a note with an accidental, the Editor remembers that the accidental is in effect. If you enter more notes and then move back to that pitch, the accidental is automatically selected.

Since accidentals affect only those pitches in one measure, accidentals remembered by the Editor are cleared whenever a measure marker is entered.

To turn off the measure feature, press the M key again (it acts as a toggle switch). This feature can also be selected with the joystick. Push the stick up or down while in the key signature level.
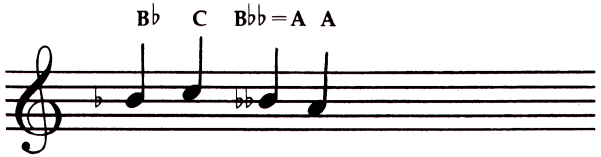
There's one more possibility. Although they're not used often, it's possible to have double sharps or double flats. Changing a natural note into a sharp note is done by increasing the pitch one half step. If this is done on a note already sharp, the pitch is bumped up another half step. The symbol to indicate a double sharp looks something like a letter *X* (shown in Figure 10-5).

Figure 10-5. Double Sharps



Two flat signs are used to indicate a double flat.

Figure 10-6. Double Flats



The double sharp and double flat signs, along with the sharp, natural, and flat signs, give a total of five different accidental signs.

The Editor does not support double sharps and double flats. When you see a note with a double sharp or flat, you must calculate the pitch that should be entered. For example, if you encounter a C-double-sharp, you should enter a D-natural.

Double dots. A dot placed after a note means that the duration should be increased by one-half. Adding a second dot means that the value of the first dot should be increased by one-half. Thus, a double-dotted half note is equal in duration to a half note, plus a quarter note, plus an eighth note. Double dots can be used on other durations as well.
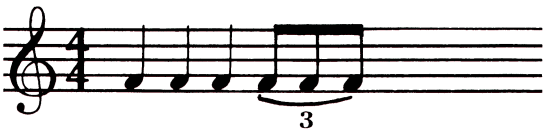
Because double-dotted notes are very rare, they're not directly supported by Sidplayer. You can simulate them, however, by entering the appropriate single-dotted note and tying it to another note of the same pitch. For example, a double-dotted C4 half note could be simulated by entering a dotted C4 half note tied to a C4 eighth note.

Triplets. All the durations, from whole note to thirty-second note, are based on the number *two*. These durations allow notes to be played for one, two, or four beats, half a beat, and so on. This system works very well, except that it's difficult to handle durations based on the number *three*. To

play a note for three beats, a dotted half note can be used. But how do you play a note for one-third of a beat? Using dots will not help there. Instead, you need to use a new kind of duration called a *triplet.*

A triplet consists of three notes played for the amount of time normally allotted to two notes. An eighth triplet is equal in duration to two eighth notes. Because two eighth notes form one beat, each of the three notes in a triplet is one-third of a beat long.

Figure 10-7. Triplets



A sixteenth triplet means that the amount of time used for two sixteenth notes, or one eighth note, is to be divided into three equal parts. Each of the three notes in the triplet plays for that duration. Triplets based on other durations, such as quarter notes, are also possible. Triplets are always written with the number 3 above or below them.

To support the less common durations, Sidplayer has a *utility duration* which can be set to last any amount of time. If you want to play an eighth triplet, for example, all you have to do is set the utility duration to play for the appropriate number of jiffies, then enter the notes of the triplet using the duration marked UTILITY. Push the joystick left or right, or press the U key to select this duration.

The UTL command is used to set the utility duration. The range of this command is from 0 to 255 jiffies, with 0 meaning 256. To calculate the number of jiffies for a particular duration, refer to Table 9-1, "Tempo," in Chapter 9. If you want an eighth triplet, find the jiffy count for a quarter note in the current tempo and divide that number by 3. Enter the result for the UTL command.

Once the utility duration has been set by the UTL command on one voice, it can be used for entering notes and rests on all three voices.

The length of the utility duration stays the same until changed by the UTL command. Since the calculation of the

utility duration depends on the current tempo, the utility duration may have to be changed if the tempo ever changes.

Some tempos do not support the use of eighth or sixteenth triplets. In M.M. 128, for example, a quarter note plays for 28 jiffies. The number *28* is not evenly divisible by 3, so a different tempo has to be used.
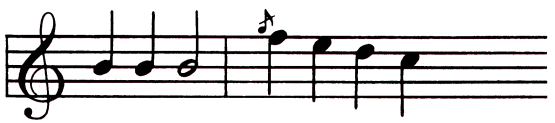
If only a few triplets are used in the music you're translating, try this. To play an eighth triplet in M.M. 128, set the utility duration to 9 jiffies and enter the first two notes of the triplet. Then set the utility duration to 10 jiffies and enter the remaining note. The total duration of the three notes will be 28 jiffies.

This trick of changing the utility duration can also be used to play sixteenth or thirty-second notes in tempos which do not normally support them. An eighth note in M.M. 120 plays for 15 jiffies. To play a pair of sixteenth notes in this tempo, set the utility duration to 7 jiffies for the first note and to 8 jiffies for the second note.

The default value of the utility duration is 12 jiffies, which is the length of an eighth triplet in M.M. 100.

Grace notes. If you see a note written a lot smaller than all the other notes, it's a grace note. A grace note is played very quickly, just long enough to be heard. Refer to Figure 10-8 for an example of a grace note.

Figure 10-8. Grace Note



To enter a grace note, enter the preceding note with a slightly shorter amount of time, then enter the grace note with the remaining duration. Generally, the utility duration must be used to do this.
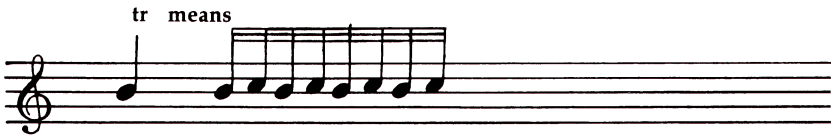
In Figure 10-8, the grace note is preceded by a half note. Let's say that the current tempo is M.M. 100, in which case a half note plays for 72 jiffies. The grace note could be entered by setting the utility duration to 68 jiffies and entering the half note with the utility duration, then setting the utility duration

to 4 jiffies and entering the grace note. The total duration of the two notes (68 + 4 jiffies) will be correct.

There's no definite rule about how many jiffies should be used for a grace note. The grace note in the example could have been played for 6 jiffies, in which case the previous note would have played for 66 jiffies.

Trills. The letters *tr* above a note are used to indicate a trill (Figure 10-9). This means that the note should be played with the pitch rapidly alternating between the designated pitch and the next higher pitch in the current key.

Figure 10-9. Trill



Repeat loops can be useful in entering trills. To enter the trill shown in the example, a pair of thirty-second notes, in the pitches F and G, could be entered between a HED and TAL. The loop would be repeated four times.

Portamento and Vibrato

To make the SID chip produce a pitch, a frequency number has to be POKEd into the chip's frequency registers. The frequency numbers for all eight octaves can be found in Appendix M of the *Commodore 64 User's Guide* (the manual that came with your Commodore 64). Every time a new note is played, Sidplayer examines the pitch of the note and POKEs the corresponding frequency number into the SID chip frequency registers.

The frequency numbers are rather large. For example, the number for middle C is 4291. There's also a large gap between frequency numbers for each half step. The number for C-sharp is 4547, quite a jump from 4291.

All the numbers between 4291 and 4547 make the SID chip produce pitches between C and C-sharp. Normally, these pitches are never played. If Sidplayer plays a C followed by a C-sharp, the frequency number changes immediately from 4291 to 4547. But if the frequency number were to run through all the intermediate values before reaching 4547, the

pitch would make a smooth transition from the C to the C-sharp. The pitch would *glide* from one note to the other.

The effect is called *glissando,* or *portamento* in synthesizer terminology. Gliding can be done between any two pitches, and can go up or down. The trombone is one instrument which does this naturally.

Sidplayer supports a portamento option for each voice. To turn on portamento, enter the POR command with a number greater than zero. Then, as each note is played, the pitch will glide up or down from the previous note until it reaches the new pitch.

The POR command number controls the glide rate. The larger the number, the more quickly the pitch glides from one note to the next.

An important characteristic of the rate number is that larger rate numbers must be used for higher pitches. If you examine the frequency numbers, you'll notice that the difference between the numbers increases as the pitch increases. To be precise, the difference doubles with each octave.

In order to get the same glide effect in each octave, the rate should be doubled or halved as necessary. For instance, if the glide rate is 100 when playing notes in octave 4, use a glide rate of 200 for notes in octave 5, or a rate of 50 for notes in octave 3.

The portamento feature can be turned off by entering the command POR 0.

Usually, gliding is done only a few times in a song. When done continually, the result can be rather comical, as illustrated by Program 10-1.

The effect of *vibrato* is to make the pitch waver slightly. The pitch repeatedly increases and then decreases by a small amount as each note plays. When done properly, the slight but steady fluctuation in pitch is barely noticeable, but it makes a tone sound more natural and alive.

Two commands are needed to control vibrato. The VDP command is used to set the vibrato depth. The number for this command ranges from 0 to 255. The larger the number, the more pronounced the vibrato effect. The most commonly used values are 1–50. Like the glide rate, the depth number should be doubled for each higher octave, and halved for each lower octave.

Program 10-1. ALBUMLEAF.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 50159
Filename: ALBUMLEAF.MUS

```
49152 :080,001,068,001,022,001,173
49158 :001,110,001,012,020,159,053
49164 :015,132,030,001,001,006,197
49170 :052,147,024,084,020,149,238
49176 :020,084,030,002,020,151,075
49182 :020,151,016,147,030,003,141
49188 :020,084,020,084,016,159,163
49194 :030,004,020,147,020,147,154
49200 :020,155,020,159,030,005,181
49206 :052,147,024,084,024,150,023
49212 :024,085,024,149,024,084,194
49218 :030,006,020,151,020,151,188
49224 :016,147,030,007,020,084,120
49230 :020,084,020,159,020,151,020
49236 :001,047,016,147,020,139,198
49242 :020,159,001,002,016,147,179
49248 :020,139,020,000,030,008,057
49254 :054,002,001,142,020,000,065
49260 :020,157,020,000,020,157,226
49266 :030,009,020,000,020,157,094
49272 :020,000,020,157,030,010,101
49278 :020,000,020,092,020,146,168
49284 :020,092,030,011,020,000,049
49290 :020,092,020,146,020,092,016
49296 :030,012,020,000,020,157,127
49302 :020,000,020,157,030,013,134
49308 :020,000,020,157,020,000,117
49314 :020,157,030,014,020,000,147
49320 :020,092,020,000,020,092,156
49326 :030,015,020,000,020,092,095
49332 :020,092,020,092,030,016,194
49338 :001,110,020,000,020,157,238
49344 :020,000,020,157,030,017,180
49350 :020,000,020,157,020,000,159
49356 :020,157,030,018,020,000,193
49362 :020,092,020,146,020,092,088
49368 :030,019,020,000,020,090,139
49374 :020,159,020,090,030,020,049
49380 :020,000,020,155,020,000,187
49386 :020,155,030,021,020,000,224
49392 :020,155,020,000,020,155,098
49398 :030,022,020,000,016,166,244
49404 :020,102,030,023,020,000,191
49410 :016,167,020,159,030,024,162
49416 :052,147,024,084,020,149,228
49422 :020,084,030,025,020,151,088
49428 :020,151,016,147,030,026,154
49434 :020,084,020,084,016,159,153
49440 :030,027,020,147,020,147,167
49446 :020,155,020,159,030,028,194
49452 :052,147,024,084,024,150,013
49458 :024,085,024,149,024,084,184
49464 :030,029,020,151,020,151,201
49470 :016,147,030,030,020,084,133
```

```
49476 :020,084,020,159,020,151,010
49482 :030,031,016,147,020,139,201
49488 :020,000,001,015,001,079,196
49494 :020,000,015,132,000,163,160
49500 :030,001,054,002,020,163,106
49506 :020,157,020,163,020,158,124
49512 :030,002,020,163,020,157,240
49518 :020,163,020,157,030,003,247
49524 :020,163,020,158,020,163,148
49530 :020,158,030,004,020,163,005
49536 :020,157,020,163,020,157,153
49542 :030,005,020,163,020,157,017
49548 :020,163,020,158,030,006,025
49554 :020,163,020,157,020,163,177
49560 :020,157,030,007,020,163,037
49566 :020,158,020,163,020,158,185
49572 :020,171,030,075,020,157,125
49578 :020,171,020,000,001,015,141
49584 :030,008,054,002,052,162,228
49590 :024,097,020,162,020,163,156
49596 :030,009,020,100,020,165,020
49602 :020,166,020,167,030,010,095
49608 :020,167,084,166,016,166,051
49614 :030,011,020,167,084,166,172
49620 :016,166,030,012,052,162,138
49626 :024,097,020,162,020,163,192
49632 :030,013,020,100,020,165,060
49638 :020,166,020,167,030,014,135
49644 :020,167,020,102,020,154,207
49650 :020,153,030,015,016,153,117
49656 :020,167,020,166,030,016,155
```



```
49662 :052,162,024,097,020,162,003
49668 :020,163,030,017,020,100,098
49674 :020,165,020,166,020,167,056
49680 :030,018,020,167,084,166,245
49686 :016,166,030,019,020,165,182
49692 :084,100,016,100,030,020,122
49698 :052,175,024,110,020,175,078
49704 :020,097,030,021,020,098,070
49710 :020,163,020,100,020,165,022
49716 :030,022,020,165,020,100,153
49722 :020,100,020,163,030,023,158
49728 :016,163,020,098,020,000,125
49734 :030,024,020,163,020,157,228
49740 :020,163,020,158,030,025,236
49746 :020,163,020,157,020,163,113
49752 :020,157,030,026,020,163,248
49758 :020,158,020,163,020,158,121
49764 :030,027,020,163,020,157,005
49770 :020,163,020,157,030,028,012
49776 :020,163,020,157,020,163,143
49782 :020,158,030,029,020,163,026
49788 :020,157,020,163,020,157,149
49794 :030,030,020,163,020,158,039
49800 :020,163,020,158,030,031,046
49806 :020,163,020,157,020,171,181
49812 :020,000,001,015,001,079,008
49818 :086,127,020,000,030,001,162
49824 :054,002,020,000,020,167,167
49830 :020,000,020,089,030,002,071
49836 :020,000,020,167,020,000,143
49842 :020,167,030,003,020,000,162
```

```
49848 :020,167,020,000,020,167,066
49854 :030,004,020,000,020,167,175
49860 :020,000,020,167,030,005,182
49866 :020,000,020,167,020,000,173
49872 :020,089,030,006,020,000,117
49878 :020,167,020,000,020,167,096
49884 :030,007,020,000,020,167,208
49890 :020,000,020,167,030,075,026
49896 :020,000,020,167,016,000,199
49902 :001,015,030,008,054,002,092
49908 :076,173,030,009,016,173,209
49914 :016,000,030,010,020,000,070
49920 :020,153,020,000,020,153,110
49926 :030,011,020,000,020,153,240
49932 :020,000,020,153,030,012,247
49938 :076,173,030,013,016,173,243
49944 :016,000,030,014,020,000,104
49950 :020,155,020,000,020,155,144
49956 :030,015,016,000,020,000,117
49962 :020,162,030,016,076,173,007
49968 :030,017,016,173,016,000,044
49974 :030,018,020,000,020,153,039
49980 :020,000,020,153,030,019,046
49986 :020,000,020,153,020,000,023
49992 :020,153,030,020,076,171,030
49998 :030,021,016,171,016,000,076
50004 :030,022,012,161,030,023,106
50010 :012,175,030,024,020,000,095
50016 :020,167,020,000,020,089,156
50022 :030,025,020,000,020,167,108
50028 :020,000,020,167,030,026,115
```



```
50034 :020,000,020,167,020,000,085
50040 :020,167,030,027,020,000,128
50046 :020,167,020,000,020,167,008
50052 :030,028,020,000,020,167,141
50058 :020,000,020,089,030,029,070
50064 :020,000,020,167,020,000,115
50070 :020,167,030,030,020,000,161
50076 :020,167,020,000,020,167,038
50082 :030,031,020,000,020,167,174
50088 :020,163,020,000,001,015,131
50094 :001,079,065,076,066,085,034
50100 :077,076,069,065,070,013,038
50106 :069,068,086,065,082,068,112
50112 :032,071,082,073,069,071,078
50118 :013,085,083,069,083,032,051
50124 :080,079,082,084,065,077,159
50130 :069,078,084,079,013,067,088
50136 :079,085,082,084,069,083,186
50142 :089,032,066,079,066,032,074
50148 :082,069,084,069,076,076,172
50154 :069,013,000,239,024,164,231
```

Program 10-2. K.C.O.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49469
Filename: K.C.O.MUS

```
49152 :220,000,040,000,002,000,006
49158 :001,142,118,012,134,003,160
49164 :001,044,001,072,038,001,169
49170 :001,071,001,006,048,000,145
49176 :088,153,024,219,048,157,201
49182 :088,156,024,219,088,156,249
49188 :088,153,084,153,038,255,039
49194 :001,216,044,153,038,001,239
49200 :001,072,048,000,088,153,154
49206 :024,219,048,157,088,156,234
49212 :024,219,048,156,088,157,240
49218 :024,156,048,231,020,154,187
49224 :001,047,080,153,108,153,102
49230 :038,255,001,216,008,153,237
49236 :038,001,001,072,166,097,203
49242 :001,002,166,014,080,165,006
49248 :080,153,038,255,001,200,055
49254 :012,153,038,001,001,072,123
49260 :001,039,022,013,004,147,078
49266 :022,026,068,145,038,255,156
49272 :038,072,001,200,044,145,108
49278 :038,001,001,072,030,002,014
49284 :048,000,088,215,024,150,145
49290 :048,149,020,215,020,150,228
49296 :084,148,038,255,001,200,102
49302 :044,148,001,072,038,001,198
49308 :048,000,088,149,024,148,101
49314 :048,211,020,149,020,148,246
49320 :084,146,001,200,038,255,124
49326 :044,146,038,001,001,072,220
49332 :048,000,088,211,024,146,185
49338 :048,145,020,211,016,158,016
49344 :084,148,001,200,038,255,150
49350 :076,148,020,148,001,072,151
49356 :038,001,022,013,004,142,168
49362 :001,094,022,026,068,140,049
49368 :038,255,001,200,008,140,090
49374 :016,000,001,079,001,039,102
49380 :001,068,054,002,008,169,018
49386 :001,020,008,172,008,169,100
49392 :008,247,072,169,008,169,145
49398 :166,126,001,015,030,002,074
49404 :008,173,008,172,008,235,088
49410 :008,170,008,169,008,172,025
49416 :001,079,001,079,075,046,033
49422 :067,046,079,046,013,086,095
49428 :073,066,082,065,084,079,213
49434 :032,068,069,077,079,078,173
49440 :083,084,082,065,084,073,247
49446 :079,078,013,066,089,032,139
49452 :072,065,082,082,089,032,210
49458 :066,082,065,084,084,013,188
49464 :013,000,020,151,020,151,155
```

The VRT command specifies the vibrato rate, or how quickly the tone alternates between increasing and decreasing pitches. The number ranges from 0 to 255, with 0 meaning 256. Smaller rate numbers produce faster vibrato rates. The values 1–4 are used most often.

There are no default vibrato settings, so the first time you want to turn vibrato on, both commands should be entered.

To turn vibrato off, use the command VDP 0. You don't have to do anything to the rate when you turn vibrato off. To turn vibrato back on, just enter the VDP command again. The previous rate remains in effect until a new rate is specified by another VRT command.

Program 10-2 demonstrates a brief, but pleasant example of how vibrato can add a nice touch to a song.

Detuning

The DTN command is used to detune a voice. Detuning is accomplished by adding a constant number, specified by the DTN command, to the frequency number of each note played.

Detuning a single voice is not very useful. All it does is make the voice play slightly out of tune. When used with two voices, however, with both voices playing the same notes but one slightly detuned, the result is a "chorus" effect. This effect was used in the "COMMODORE" demonstration song and has some interesting applications.

What's the easiest way to make two voices play the exact same notes? The answer is to use phrases. A phrase defined on voice 1 can be called simultaneously on voice 2 or 3. You don't have to wait for playing to reach the END command before calling the phrase; the phrase is available as soon as Sidplayer has processed the DEF command.

This means that you can define the phrase on voice 1 and call it on voice 2 after having set the detune value.

Voice 1: DEF 0 Play Notes END
Voice 2: DTN 50 CAL 0

Either voice can be detuned. The DTN command can be put either before the DEF 0 or before the CAL 0. Just make sure that the DTN command is not included in the phrase definition. You don't want both voices detuned.

When entering the DTN command, use a number from 1 to 2047 to tune the voice a little sharp. The larger the value,

the more the voice will be detuned. Values around 50 work best. Enter a number from −2047 to −1 to tune the voice flat. For most applications, the direction makes no difference.

As with the glide rate and vibrato depth, larger values have to be used for higher pitches.

Enter the DTN 0 to turn the detuning off.

The only drawback to detuning is that it takes two voices, so only one voice is left free. The effect can sometimes be worth it, though. Program 10-3 is a demonstration of just how realistic a sound can be created with your Commodore 64.

Program 10-3. PIPERS.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49481
Filename: PIPERS.MUS

```
49152 :176,000,020,000,020,000,216
49158 :010,070,001,006,001,135,229
49164 :002,200,038,001,006,192,195
49170 :030,001,092,165,003,100,153
49176 :112,157,003,000,092,092,224
49182 :112,157,092,092,112,157,240
49188 :092,092,112,157,001,047,025
49194 :001,022,030,002,020,157,018
49200 :024,155,020,158,028,159,080
49206 :024,145,020,157,024,155,067
49212 :020,155,028,154,024,153,082
49218 :001,047,001,038,030,003,186
49224 :020,157,024,155,020,158,094
49230 :028,159,024,145,020,157,099
49236 :028,155,024,153,020,154,106
49242 :024,153,001,047,030,004,093
49248 :001,018,001,054,030,005,205
49254 :020,154,024,155,020,156,119
49260 :028,157,024,158,020,157,140
49266 :028,155,024,153,020,154,136
49272 :024,153,001,047,030,006,125
49278 :001,070,020,145,024,157,031
49284 :020,145,024,158,020,145,132
49290 :024,157,024,155,024,154,164
49296 :024,153,001,047,030,007,150
49302 :001,086,020,145,024,157,071
49308 :020,145,024,158,024,157,172
49314 :024,155,024,153,020,154,180
49320 :024,153,001,047,030,008,175
```

```
49326 :001,066,030,009,001,050,075
49332 :001,079,166,065,001,002,238
49338 :008,157,008,157,008,157,169
49344 :008,157,008,157,008,157,175
49350 :008,157,001,079,001,002,190
49356 :001,018,001,034,001,018,021
49362 :001,050,001,066,001,082,155
49368 :001,066,001,050,001,079,158
49374 :084,072,069,032,082,069,118
49380 :068,067,079,065,084,083,162
49386 :032,065,082,069,032,067,069
49392 :079,077,073,078,071,033,139
49398 :033,033,013,066,065,071,015
49404 :080,073,080,069,032,068,142
49410 :069,077,079,078,083,084,216
49416 :082,065,084,073,079,078,213
49422 :013,085,083,069,083,032,123
49428 :068,069,084,085,078,073,221
49434 :078,071,032,065,078,068,162
49440 :032,084,082,065,078,083,200
49446 :080,079,083,073,084,073,254
49452 :079,078,013,067,079,085,189
49458 :082,084,069,083,089,032,233
49464 :082,079,066,069,082,084,006
49470 :032,072,073,071,071,073,198
49476 :078,083,013,000,020,151,157
```

Transposing

Detuning works by adjusting the pitch of each note so that it's slightly sharp or flat. Transposing also changes the pitch of each note, but in quite a different way.

When a voice is transposed, the pitch of each note is shifted up or down a designated number of half steps. For example, if a voice is being transposed up one half step, all notes entered as C play as C-sharps, all C-sharps play as D, every D plays as D-sharp, and so on. If the voice is being transposed down one half step, D notes play as C-sharps, C sharps as C-naturals, C-naturals become B-naturals, and so on. You can check these alterations by tracing the transposing using Table 10-1.

Table 10-1. Transposing

	C
	B
A♯	/B♭
	A
G♯	/A♭
	G
F♯	/G♭
	F
	E
D♯	/E♭
	D
C♯	/D♭
	C
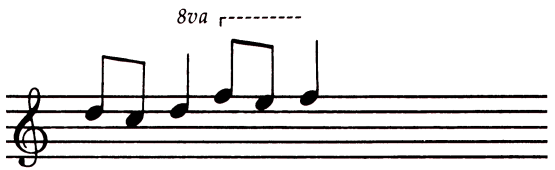
Transposing can be done by more than 1 half step. In a voice being transposed up by 7 half steps, for example, a note entered as C will play using the pitch for G. Since there are 12 half steps in an octave, transposing up or down by 12 half steps makes a voice play an octave higher or lower.

The TPS command is used to specify the number of half steps a voice should be transposed. Enter this command with a number in the range 1–95 to transpose a voice up by 1 to 95 half steps. Enter −1 to −95 to transpose the voice down. Enter 0 to turn transposing off.

Transposing applies to all notes equally, so unlike some of the previous commands, the value does not need to be changed for higher or lower pitches.

There are many useful applications for the TPS command. For instance, you may sometimes see a dashed line appearing above or below a sequence of notes.

Figure 10-10. Octave Offset



The dashed line, along with the message *8va* or *8vaba,* means that the notes should be played one octave higher or lower than written. This is done because writing the notes

with the correct pitches would require too many leger lines, and the notes would be difficult to read.

The easiest way to handle an octave offset is to enter the notes as written, and then insert a TPS 12 or TPS −12 command at the beginning of the offset.

An interesting effect is created by having two voices play the same note, while one of the voices is transposed up one or two octaves. This produces a rich, warm tone. The effect was used in the COMMODORE demonstration song. Another possibility is to use transposing with detuning. While one voice plays a series of notes, the other plays the same notes, but is detuned and transposed up one or two octaves. Programs 10-4 and 10-5 use the technique to imitate a calliope and an accordion.

Synchronization

In the previous section, it was mentioned that a rich, warm tone can be created when two voices play the same notes and the second voice is transposed up by 12 or 24 half steps. But if you try this technique with the second voice transposed up by other numbers of half steps, like 6, 11, or 13, the resulting tone sounds less than desirable.

The reason that the pleasant tone is produced only when the second voice is transposed up an octave or two is because the voices become synchronized. Remember that the frequency of a note is doubled when it's played an octave higher. When two voices play the same notes, one an octave higher, the direct relationship between the frequencies causes the tones to be synchronized.

The tone produced by two synchronized voices sounds rich and warm because it contains more harmonics than usual. The second voice augments the harmonics of the first one. This method of adding harmonics to a tone is called *additive synthesis.* (Filtering is called *subtractive synthesis* because the filter removes harmonics from a tone.)

Synchronization occurs naturally at intervals of 12 half steps, but it can be made to occur at any half-step interval if the synchronization mode of the SID chip is turned on. The advantage to using the synchronization mode is that different half-step intervals produce different harmonic patterns. Synchronization, therefore, lets you produce many new types of tones.

The SNC command is used to control the synchronization

mode. Enter 1 (yes) to turn the mode on.

A specific procedure must be followed in order to synchronize two voices. Let's say that you want to synchronize voices 1 and 2. First, define voice 1 to be a phrase. On voice 2 you should enter the SNC YES command and the TPS command. Choose any number of half steps, such as 8. Then call the phrase that was defined on voice 1.

DEF on voice 1
SNC on voice 2
TPS on voice 2
CAL on voice 2

Play the two voices several times, each time transposing voice 2 by a different number of half steps, to hear the various types of tones that can be created. Each waveform produces different effects when synchronized, so you might also try changing the waveform.

Both voices should have the same envelope, but they do not have to have the same waveform. For example, you might use a square wave on voice 1 and a triangle wave on voice 2.

To synchronize voices 1 and 3, use this procedure:

SNC on voice 1
TPS on voice 1
DEF on voice 1
CAL on voice 3

Use this procedure to synchronize voices 2 and 3:

DEF on voice 2
SNC on voice 3
TPS on voice 3
CAL on voice 3

It's very important that you use only these procedures. Synchronization will not work correctly if the SNC or TPS commands are used on the wrong voice. Also, only positive transpose values should be used. Negative values do not seem to be very useful.

When you want to stop using the synchronization mode and return to normal playing, enter the SNC command with the value 0 (no) to turn the mode off. Do this on the voice which earlier turned the mode on. You may also want to enter a TPS 0 command to cancel the transposing.

Program 10-6 uses different half-step intervals and different waveforms to demonstrate some of the effects possible with synchronization.

Program 10-4. CALLIOPE.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49481
Filename: CALLIOPE.MUS

```
49152 :146,000,106,000,004,000,000
49158 :166,010,010,070,001,246,253
49164 :001,135,130,000,001,004,027
49170 :001,192,001,236,030,001,223
49176 :048,155,020,090,020,155,000
49182 :020,157,012,145,020,159,031
49188 :020,145,030,002,020,146,143
49194 :020,145,020,159,020,145,039
49200 :020,155,020,157,044,158,090
49206 :030,003,048,156,020,155,210
49212 :020,156,020,158,012,159,073
49218 :020,094,020,159,030,004,137
49224 :020,145,020,159,020,094,018
49230 :020,159,020,156,020,159,100
49236 :044,155,030,005,048,155,009
49242 :020,090,020,155,020,157,040
49248 :012,145,020,159,020,145,085
49254 :030,009,020,146,020,145,216
49260 :020,159,020,145,020,155,115
49266 :020,157,044,158,030,010,021
49272 :048,158,020,158,020,159,171
49278 :020,145,012,157,020,148,116
49284 :020,147,030,011,020,146,250
49290 :020,145,020,159,020,158,148
49296 :020,159,020,146,044,145,166
49302 :001,079,001,039,030,001,045
49308 :001,006,020,165,020,153,009
49314 :020,153,020,173,020,153,189
49320 :020,153,001,047,001,002,136
49326 :030,002,001,002,001,022,232
49332 :020,164,020,166,020,166,224
49338 :020,172,020,166,020,166,238
49344 :001,047,030,003,001,018,036
49350 :001,018,030,004,001,018,014
49356 :020,165,020,153,020,153,223
49362 :020,165,020,166,020,167,000
49368 :030,005,001,002,001,002,001
49374 :030,006,001,002,001,018,024
49380 :030,007,001,018,001,002,031
49386 :030,008,020,162,020,165,127
49392 :020,165,020,162,020,165,024
49398 :020,165,020,161,020,163,027
49404 :020,165,048,153,001,079,206
49410 :001,242,001,079,067,065,201
49416 :076,076,073,079,080,069,205
49422 :032,068,069,077,079,078,161
49428 :083,084,082,065,084,073,235
49434 :079,078,013,066,089,032,127
49440 :082,079,066,069,082,084,238
49446 :032,072,073,071,071,073,174
49452 :078,083,013,013,067,065,107
49458 :084,067,072,032,084,072,205
49464 :065,084,032,066,082,065,194
49470 :083,083,032,082,073,078,237
49476 :071,046,046,046,013,000,034
```

Program 10-5. BISTRO.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49523
Filename: BISTRO.MUS

```
49152 :020,001,010,000,008,000,039
49158 :001,071,010,070,001,006,165
49164 :006,128,022,002,030,001,201
49170 :020,153,020,156,020,157,032
49176 :016,222,020,223,020,145,158
49182 :020,223,020,222,020,157,180
49188 :020,156,020,219,048,089,076
49194 :068,154,068,155,068,156,199
49200 :068,157,068,158,068,159,214
49206 :084,145,016,145,108,223,007
49212 :020,223,030,002,020,153,252
49218 :020,155,020,156,016,157,078
49224 :020,222,020,223,020,210,019
49230 :022,005,068,145,068,210,084
49236 :022,006,004,145,020,223,248
49242 :020,222,020,157,108,156,005
49248 :020,156,052,000,001,047,116
49254 :001,002,001,022,030,003,161
49260 :020,156,020,157,020,158,127
49266 :016,145,020,158,016,145,102
49272 :020,158,048,145,030,004,013
49278 :020,158,020,145,020,158,135
49284 :020,146,016,145,020,146,113
49290 :020,145,020,158,044,223,236
49296 :030,005,020,147,020,148,002
49302 :020,149,020,148,016,147,138
49308 :020,146,020,147,020,148,145
49314 :048,147,030,006,020,094,251
49320 :020,145,020,146,016,145,148
49326 :020,155,112,156,084,156,089
49332 :022,002,068,157,068,158,143
49338 :068,223,068,145,068,146,136
49344 :068,147,068,148,068,148,071
49350 :052,148,030,007,020,148,091
49356 :020,082,016,146,020,223,199
49362 :080,146,048,146,020,147,029
49368 :020,149,020,148,020,146,207
49374 :016,145,020,156,080,158,029
49380 :048,158,030,008,020,159,139
49386 :020,146,020,145,020,158,231
49392 :016,145,020,157,080,145,035
49398 :048,145,030,009,020,157,143
49404 :020,158,016,223,020,155,076
49410 :080,156,092,157,092,158,225
49416 :092,223,092,145,092,146,030
49422 :092,147,092,148,092,148,221
49428 :016,148,001,047,001,079,056
49434 :166,179,001,002,001,002,121
49440 :001,018,001,079,001,002,134
49446 :001,002,001,018,001,079,140
49452 :080,073,069,067,069,083,229
49458 :032,070,082,065,078,067,188
49464 :065,073,083,069,083,013,186
49470 :065,067,067,079,082,068,234
```

```
49476 :073,065,078,032,068,069,197
49482 :077,079,078,083,084,082,045
49488 :065,084,073,079,078,013,216
49494 :067,079,085,082,084,069,040
49500 :083,089,032,082,079,066,011
49506 :069,082,084,032,072,073,254
49512 :071,071,073,078,083,013,237
49518 :013,000,020,157,030,003,077
```

Program 10-6. TPI #14.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 50825
Filename: TPI #14.MUS

```
49152 :040,003,060,000,226,002,075
49158 :001,006,006,112,001,004,136
49164 :001,208,001,228,001,168,107
49170 :030,001,020,000,088,223,124
49176 :088,145,088,146,088,145,212
49182 :084,223,084,148,084,146,031
49188 :084,215,084,148,030,002,087
49194 :020,146,088,148,088,211,231
49200 :088,146,088,211,084,148,045
49206 :084,223,084,146,084,156,063
49212 :084,222,030,003,020,157,064
49218 :088,219,088,156,088,157,094
49224 :088,156,084,219,084,223,158
49230 :084,157,084,211,084,223,153
49236 :030,004,020,157,088,223,094
49242 :088,222,088,157,088,222,187
49248 :084,223,084,219,084,157,179
49254 :084,153,084,219,030,005,165
49260 :020,166,088,153,088,154,009
49266 :088,219,088,154,084,153,132
49272 :084,158,084,156,084,145,063
49278 :084,158,030,006,020,211,123
49284 :088,156,088,157,088,158,099
49290 :088,157,084,156,084,145,084
49296 :084,158,084,148,084,145,079
49302 :030,007,016,146,016,000,109
49308 :020,000,001,047,001,006,231
49314 :088,149,088,148,088,211,166
49320 :088,148,020,149,030,008,099
49326 :016,145,016,000,020,000,115
49332 :001,047,001,006,088,148,215
49338 :088,211,088,146,088,211,250
49344 :020,148,030,009,016,223,126
49350 :016,000,020,000,001,047,026
49356 :001,006,088,211,088,146,232
49362 :088,145,088,146,020,211,140
49368 :030,010,020,158,088,145,155
49374 :088,223,088,158,088,223,066
49380 :020,145,016,156,016,000,069
49386 :001,047,030,011,001,006,074
49392 :001,071,001,192,001,132,126
49398 :001,200,038,001,016,148,138
49404 :016,156,016,158,016,145,247
49410 :030,012,012,148,012,000,216
```

```
49416 :030,013,016,000,001,047,115
49422 :001,006,016,156,016,223,176
49428 :016,146,030,014,012,148,130
49434 :012,000,030,015,016,000,099
49440 :001,047,001,006,016,157,004
49446 :016,223,016,145,030,016,228
49452 :012,147,012,000,001,047,007
49458 :030,017,001,006,020,000,124
49464 :001,039,001,252,001,152,246
49470 :038,004,088,156,088,157,081
49476 :088,158,088,157,084,156,031
49482 :084,145,084,158,084,148,009
49488 :084,145,030,018,020,150,015
49494 :088,211,088,146,088,145,084
49500 :088,146,084,211,084,158,095
49506 :084,145,084,221,084,146,094
49512 :030,019,016,223,080,146,106
49518 :080,223,080,157,030,020,188
49524 :016,222,080,148,080,222,116
49530 :080,156,030,021,020,157,074
49536 :001,047,001,006,088,153,168
49542 :088,154,088,219,088,154,157
49548 :084,153,084,157,084,219,153
49554 :084,145,084,157,030,022,156
49560 :020,146,001,047,001,006,117
49566 :088,222,088,157,088,156,189
49572 :088,157,084,222,084,154,185
49578 :084,156,084,167,084,157,134
49584 :001,047,030,023,001,006,028
49590 :016,219,016,000,020,000,197
49596 :001,047,001,135,050,232,142
```

```
49602 :001,006,088,153,088,154,172
49608 :088,219,088,154,020,153,154
49614 :030,024,084,145,080,145,202
49620 :084,223,020,158,088,156,173
49626 :088,157,088,158,088,157,186
49632 :020,156,030,025,084,148,175
49638 :080,148,084,211,020,146,151
49644 :088,223,088,145,088,146,246
49650 :088,145,020,223,030,026,006
49656 :084,215,080,215,084,214,116
49662 :020,149,088,215,088,214,004
49668 :088,149,088,214,020,215,010
49674 :030,027,020,211,088,149,023
49680 :088,148,088,211,088,148,019
49686 :020,149,020,145,088,211,143
49692 :088,146,088,145,088,146,217
49698 :020,211,030,028,020,158,245
49704 :088,145,088,146,088,211,038
49710 :088,146,020,145,020,148,101
49716 :088,222,088,157,088,156,083
49722 :088,157,020,222,030,029,092
49728 :020,157,088,223,088,145,017
49734 :088,146,088,145,020,223,012
49740 :020,211,088,157,088,156,028
49746 :088,219,088,156,020,157,042
49752 :030,030,020,156,088,158,058
49758 :088,223,088,145,088,223,181
49764 :020,158,020,146,088,156,176
49770 :088,219,088,154,088,219,194
49776 :020,156,030,031,020,219,076
49782 :088,157,088,158,088,223,152
```

```
49788 :088,158,020,157,020,145,200
49794 :088,219,088,154,088,153,152
49800 :088,154,020,219,030,032,167
49806 :016,154,016,000,020,000,092
49812 :001,047,001,006,130,000,077
49818 :001,112,001,212,088,223,023
49824 :088,145,088,146,088,145,092
49830 :084,223,030,033,084,148,000
49836 :084,146,084,215,084,148,165
49842 :020,146,088,148,088,211,111
49848 :088,146,088,211,084,148,181
49854 :030,034,084,223,084,211,088
49860 :084,223,084,211,020,157,207
49866 :088,219,088,156,088,157,230
49872 :088,156,084,219,030,035,052
49878 :084,223,084,157,084,211,033
49884 :084,223,020,157,088,223,247
49890 :088,222,088,157,088,222,067
49896 :084,223,030,036,016,219,072
49902 :080,211,020,211,088,211,035
49908 :088,146,088,145,088,146,177
49914 :020,211,030,037,016,156,208
49920 :080,211,020,211,088,145,243
49926 :088,146,088,211,088,146,005
49932 :020,145,030,038,020,148,157
49938 :088,146,088,145,088,223,028
49944 :088,145,020,146,020,156,087
49950 :020,223,020,145,020,158,104
49956 :030,039,008,223,008,000,088
49962 :001,047,001,079,001,059,230
49968 :166,174,001,002,166,010,055
```



```
49974 :001,002,166,012,001,002,238
49980 :166,124,001,002,166,058,065
49986 :001,002,166,090,001,002,072
49992 :166,106,001,002,166,190,191
49998 :001,002,166,060,001,002,054
50004 :166,108,001,002,166,156,171
50010 :001,002,001,071,166,108,183
50016 :001,002,001,039,166,110,159
50022 :001,002,001,079,006,112,047
50028 :034,188,001,004,001,192,016
50034 :001,132,001,200,038,001,231
50040 :030,001,016,231,016,239,141
50046 :016,162,016,164,030,002,004
50052 :012,231,012,000,030,003,164
50058 :016,000,016,239,016,227,140
50064 :016,165,030,004,012,231,090
50070 :012,000,030,005,016,000,213
50076 :016,239,016,161,016,227,063
50082 :030,006,012,166,012,000,132
50088 :030,007,020,000,114,108,191
50094 :001,228,001,152,038,004,086
50100 :088,231,088,153,088,154,214
50106 :088,153,020,231,016,219,145
50112 :016,000,030,008,020,000,010
50118 :088,166,088,231,088,153,244
50124 :088,231,020,166,016,154,111
50130 :016,000,030,009,020,000,029
50136 :088,165,088,166,088,231,018
50142 :088,166,020,165,020,153,066
50148 :088,161,088,162,088,227,018
50154 :088,162,020,161,030,010,193
```

```
50160 :080,164,016,172,020,000,180
50166 :084,153,084,166,020,164,149
50172 :030,011,020,000,088,164,053
50178 :088,165,088,166,088,165,250
50184 :084,164,084,153,084,166,231
50190 :084,156,084,153,030,012,021
50196 :020,166,088,153,088,231,254
50202 :088,166,088,231,084,153,068
50208 :084,164,084,166,084,161,007
50214 :084,227,030,013,020,162,062
50220 :088,239,088,161,088,162,102
50226 :088,161,084,239,084,164,102
50232 :084,162,084,231,084,164,097
50238 :030,014,020,162,088,164,028
50244 :088,227,088,162,088,227,180
50250 :084,164,084,239,084,162,123
50256 :084,173,084,239,030,015,193
50262 :020,171,088,173,088,174,032
50268 :088,239,088,174,084,173,170
50274 :084,163,084,161,084,165,071
50280 :084,163,030,016,020,231,136
50286 :088,161,088,162,088,163,092
50292 :088,162,084,161,084,165,092
50298 :084,163,084,153,084,165,087
50304 :030,017,016,166,001,212,058
50310 :080,153,080,166,080,164,089
50316 :030,018,016,229,080,166,167
50322 :080,229,080,162,030,019,234
50328 :020,165,088,173,088,174,092
50334 :088,239,088,174,084,173,236
50340 :084,162,084,239,084,165,214
50346 :084,162,030,020,020,231,205
50352 :088,230,088,165,088,164,231
50358 :088,165,084,230,084,162,227
50364 :084,164,084,239,084,162,237
50370 :030,021,016,227,080,165,221
50376 :080,227,080,161,030,022,032
50382 :016,175,080,162,080,175,126
50388 :080,173,030,023,020,161,187
50394 :050,132,001,244,088,169,134
50400 :088,170,088,235,088,170,039
50406 :020,169,084,161,080,161,137
50412 :084,239,030,024,020,174,039
50418 :088,172,088,173,088,174,001
50424 :088,173,020,172,084,164,181
50430 :080,164,084,227,030,025,096
50436 :020,162,088,239,088,161,250
50442 :088,162,088,161,020,239,000
50448 :084,231,080,231,084,230,188
50454 :030,026,020,165,088,227,066
50460 :088,164,088,165,088,164,017
50466 :020,227,084,219,080,219,115
50472 :084,154,030,027,020,153,252
50478 :088,219,088,154,088,153,068
50484 :088,154,020,219,020,166,207
50490 :088,153,088,231,088,166,104
50496 :088,231,020,153,030,028,102
50502 :020,164,088,166,088,231,059
50508 :088,153,088,231,020,166,054
50514 :020,154,088,164,088,227,055
50520 :088,162,088,227,020,164,069
50526 :030,029,020,227,088,165,141
```

```
50532 :088,166,088,231,088,166,159
50538 :020,165,020,153,088,227,011
50544 :088,162,088,161,088,162,093
50550 :020,227,030,030,020,162,095
50556 :088,164,088,165,088,166,115
50562 :088,165,020,164,020,231,050
50568 :088,162,088,161,088,239,194
50574 :088,161,020,162,030,031,122
50580 :020,161,088,227,088,164,128
50586 :088,165,088,164,020,227,138
50592 :020,166,088,161,088,239,154
50598 :088,174,088,239,020,161,168
50604 :030,032,020,239,001,071,053
50610 :088,231,088,153,088,154,212
50616 :088,153,084,231,084,156,212
50622 :084,154,084,223,084,156,207
50628 :030,033,020,154,088,156,165
50634 :088,219,088,154,088,219,034
50640 :084,156,084,231,084,154,233
50646 :084,164,084,230,030,034,072
50652 :020,165,088,227,088,164,204
50658 :088,165,088,164,084,227,018
50664 :084,231,084,165,084,219,075
50670 :084,231,030,035,020,165,035
50676 :088,231,088,230,088,165,110
50682 :088,230,084,231,084,227,170
50688 :084,165,084,239,084,162,050
50694 :030,036,020,161,088,172,001
50700 :088,173,088,174,088,173,028
50706 :084,172,084,161,084,174,009
50712 :084,227,084,161,030,037,135
50718 :020,174,088,161,088,239,032
50724 :088,174,088,239,084,161,102
50730 :084,172,084,174,084,169,041
50736 :084,235,030,038,020,170,113
50742 :088,239,088,161,088,162,112
50748 :088,161,020,239,016,164,236
50754 :016,172,030,039,008,239,058
50760 :008,000,001,079,084,087,075
50766 :079,032,080,065,082,084,244
50772 :032,073,078,086,069,078,244
50778 :084,073,079,078,032,035,215
50784 :049,052,013,074,046,083,157
50790 :046,032,066,065,067,072,194
50796 :013,067,079,085,082,084,006
50802 :069,083,089,032,082,079,036
50808 :066,069,082,084,032,072,013
50814 :073,071,071,073,078,083,063
50820 :013,013,000,077,029,192,200
```

Ring Modulation

A ring modulator takes two frequencies and produces two new frequencies in their place. The new frequencies are the sum and difference of the originals. For example, if a ring modulator is given frequencies of 200 and 300 Hz, it will produce the frequencies of 100 and 500 Hz, but not 200 or 300 Hz.

Sometimes it can be hard to discern the pitch of a ring-modulated tone. The two frequencies may be interpreted by your ear as a single pitch, but the frequency corresponding to that pitch is not one of the two produced by the ring modulation, so it does not really exist.

Ring-modulated tones are useful for creating percussion effects. Bells, chimes, steel drums, and various instruments made of metal, wood, or glass can be approximated.

When used simultaneously with synchronization, a whole new set of tone colors (timbres) is available.

To use ring modulation, a definite procedure must be followed. Two voices must play the same sequence of notes, just like when using the synchronization mode. With ring modulation, however, only voices 1 and 3 can be used.

Voice 1 is used to supply the waveform and envelope of the ring-modulated tone, plus one of the two frequencies needed for the process. The first thing voice 1 should do is select the triangle waveform. This waveform must be used in order for ring modulation to work.

Since ring-modulated tones are used mainly for percussion effects, a nonsustaining envelope should be selected.

Finally, enter the command RNG with the value for Yes to turn ring modulation on. Voice 1 is now ready to start playing notes. These notes should be put in a phrase definition so that they can be called by voice 3.

Voice 3 is used only to supply the second frequency needed for ring modulation, so the waveform and envelope of voice 3 do not have to be set. In fact, it's best to turn off the output from this voice completely, by using the 3-O (3 OFF) command. Enter this command with the value for Yes at the beginning of voice 3. The frequencies of notes played by voice 3 are still used in the ring-modulation process, but the output from voice 3 itself is not heard. Voice 3 can now call the phrase defined in voice 1.

Here's an example sequence of commands that could be entered to use ring modulation.

Voice 1
WAV T
ATK 0
DCY 9
SUS 0
RLS 9
PNT 1
RNG Y
DEF

Voice 3
3-O Y
CAL

As with synchronization, different tone types can be produced by using the TPS command. With ring modulation, however, transposing one voice can cause the resulting pitch to be out of tune, so it's often necessary to transpose the other voice as well to bring the pitch back in tune.

Table 10-2 is a chart ordered by increasing half-step intervals. For each interval, the appropriate transpose values for voices 1 and 3 are given.

Table 10-2. Transpose Values for Use with Ring Modulation

Halfstep Offset	V1 TPS	V3 TPS	Highest Note
−20	7	−13	E7
−19	7	−12	E7
−18	7	−11	E7
−17	–	–	
−16	6	−10	F7
−15	6	−9	F7
−14	6	−8	F7
−13	–	–	
−12	0	−12	B7
−11	–	–	
−10	2	−8	A7
−9	4	−5	G7
−8	2	−6	A7
−7	7	0	E7
−6	10	4	C♯7
−5	0	−5	B7
−4	2	−2	A7
−3	–	–	
−2	–	–	
−1	–	–	

0	0	0	B7
1	−1	0	B7
2	–	–	
3	–	–	
4	0	4	G7
5	−5	0	B7
6	4	10	C♯7
7	0	7	E7
8	–	–	
9	−5	4	G7
10	4	14	A6
11	–	–	
12	0	12	B6
13	–	–	
14	−4	10	C♯7
15	−5	10	C♯7
16	0	16	G6
17	−5	12	B6
18	1	19	E6
19	0	19	E6
20	–	–	
21	–	–	
22	−2	20	E♭6
23	4	27	G♯5
24	0	24	B5
25	–	–	
26	0	26	A5
27	1	28	G5
28	0	28	G5
29	–	–	
30	−5	25	B♭5
31	0	31	E5
32	−5	27	G♯5
33	1	34	C♯5
34	0	34	C♯5
35	0	35	C5
36	0	36	B4
37	0	37	B♭4
38	0	38	A4
39	0	39	G♯4
40	0	40	G4

The transpose values for some of these intervals are close approximations and don't produce a distinct pitch, which is why they're best used for percussion effects. Those intervals

for which no satisfactory transpose values could be found have been marked with a dash (–).

When the transpose value for a voice is 0, no TPS command has to be entered for that voice.

The rightmost column tells you the highest note that can be played using the given transpose values. For example, when the 9 half-step interval is used, only notes up to G7 should be used. Notes above G7 won't play properly.

You can transpose the tone up or down one or more octaves by adding a multiple of 12 half steps to each transpose value. Let's say that you are using the 26 half-step interval, in which the transpose values are 0 and 26. Adding −12 half steps gives the transpose values −12 and 14.

Be aware that the effect of ring modulation may vary from one octave to another. Notes played in one octave may sound quite different when played with the same half-step interval in a different octave.

By now you might be wondering just what ring-modulated tones sound like. Program 10-7 is a demonstration that uses the 26 half-step interval.

Only a few of the half-step intervals listed in Table 10-2 produce very pleasant or useful sounds. Too often, the sound produced by one interval includes an undesirable high-pitched tone. But if you pass the ring-modulated tone through the filter, with the filter mode set to low pass, the squeal can be eliminated. Most of the intervals listed in the table then become useful.

Use the filter on voice 1 only. Here are some example settings.

```
F-M  L
AUT  1
RES  15
FLT  Y
```

Try using different filter modes, including combinations. Also try different cutoff settings on the AUT command. The resonance value set by the RES command can be adjusted to take some of the "bite" out of a tone.

Since there are so many variable factors, some experimentation will be needed. This can be worthwhile, however, because you'll discover some timbres that you've never heard

before from the SID chip. Some intervals to try are the intervals with −7, 28, and 38 half steps.

One remaining possibility is to use synchronization and ring modulation at the same time. This technique yields yet another set of timbres.

To use both modes simultaneously, follow this procedure.

Voice 1
WAV T
SNC Y
RNG Y
TPS
DEF

Voice 3
3-O Y
CAL

The triangle waveform is still required, but now any envelope can be used. Also, every half-step interval produces a usable tone, so you can disregard Table 10-2. The transposing always stays in tune, so it has to be done only on voice 1. As with synchronization, however, negative transpose values are not very useful.

To turn ring modulation off, enter the command RNG N on voice 1. Enter the command 3-O N on voice 3 to reenable the output of that voice. The command TPS 0 should be entered on those voices that were being transposed. If you were also using the filter, enter the commands FLT N and AUT 0 on voice 1. Or, if you were using synchronization, enter the command SNC N on voice 1.

Whether used alone, with the filter, or with synchronization, the ring-modulation feature of the SID chip can produce some very interesting sounds. Program 10-8 is just one more example of the techniques.

Advanced Techniques

This section explores some new uses for the commands introduced in the previous chapters. The techniques suggested here should give you an idea of just what's possible with Sidplayer.

Waveform. Probably the most overlooked waveform is the noise waveform. Although this waveform does not have a definite pitch quality, its character does change as the pitch is

changed. High pitches produce a hissing sound, whereas low pitches create more of a rumble.

An interesting use of the noise waveform is to play white-noise notes with different pitches to create a repeating percussion line. The sequence of notes should use short durations and should repeat every one or two measures. With just a little effort, you can create a fancy rhythm that adds a nice touch to a song.

The character of the noise waveform can be altered significantly by passing it through the filter. Try this with the filter set for the high-pass mode.

As for the other three waveforms, one possibility is to use two waveforms at the same time on the same voice, with the objective of creating new types of sounds. Unfortunately, only the combination of triangle and pulse waves is audible.

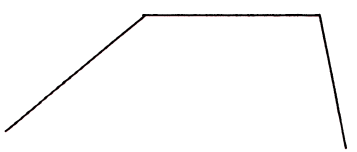
To select the triangle and pulse wave combination, enter the WAV command with the number 5 (1 for triangle plus 4 for pulse). This waveform sounds a lot like the pulse wave with the pulse width set to about 200, but with more of a buzz.

When a voice is playing with the triangle/pulse combination, the pulse width acts as a volume control for the voice. Setting the pulse width above 2048 makes the voice inaudible; smaller values make the voice sound louder.

Envelope. Thus far, you've concentrated on emulating conventional instruments. A synthesizer, however, is capable of producing any kind of sound, natural or unnatural.

One way to make a voice sound unnatural is to use a backward envelope, as shown in Figure 10-11. Such an envelope has a slow attack rate and a very fast release rate.

Figure 10-11. Backward Envelope



Program 10-7. HOLST.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49415
Filename: HOLST.MUS

```
49152 :152,000,002,000,016,000,170
49158 :001,043,001,039,166,044,044
49164 :001,004,001,144,001,132,039
49170 :001,152,038,001,020,000,230
49176 :001,022,020,157,020,223,211
49182 :016,145,020,145,020,211,075
49188 :052,146,024,223,020,211,200
49194 :020,148,016,211,016,146,087
49200 :020,145,020,146,016,145,028
49206 :016,223,012,157,020,157,127
49212 :020,223,016,145,020,145,117
49218 :020,211,052,146,024,223,230
49224 :020,211,020,148,016,149,124
49230 :016,149,020,149,020,148,068
49236 :016,211,016,148,012,211,186
49242 :001,047,001,038,020,215,156
49248 :020,149,016,148,016,148,081
49254 :020,211,020,149,016,148,154
49260 :016,223,020,215,020,149,239
49266 :016,148,016,148,020,149,099
49272 :020,215,012,137,020,137,149
49278 :020,138,016,203,016,138,145
49284 :016,137,016,215,016,203,223
49290 :016,149,020,148,020,211,190
49296 :016,148,016,149,012,215,188
49302 :001,047,166,042,001,018,169
49308 :001,079,001,079,001,091,152
49314 :166,179,020,000,001,018,034
49320 :001,034,166,014,001,018,146
49326 :001,079,084,072,069,077,044
49332 :069,032,070,082,079,077,077
49338 :032,084,072,069,032,074,037
49344 :085,080,073,084,069,082,153
49350 :032,077,079,086,069,077,106
49356 :069,078,084,013,079,070,085
49362 :032,034,084,072,069,032,021
49368 :080,076,065,078,069,084,156
49374 :083,034,013,066,089,032,027
49380 :071,085,083,084,065,086,190
49386 :069,032,072,079,076,083,133
49392 :084,013,067,079,085,082,138
49398 :084,069,083,089,032,072,163
49404 :065,082,082,089,032,066,156
49410 :082,065,084,084,013,000,074
```

Program 10-8. PROMENADE.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49625
Filename: PROMENADE.MUS

```
49152 :090,001,002,000,018,000,111
49158 :166,078,001,059,001,043,098
49164 :001,055,150,251,001,250,208
49170 :001,027,001,006,001,039,093
49176 :001,028,001,080,001,212,091
49182 :001,152,038,006,001,047,019
49188 :030,001,001,022,016,157,007
49194 :016,156,016,223,020,145,106
49200 :038,005,020,148,038,001,042
49206 :016,146,030,002,020,145,157
49212 :038,005,020,148,038,001,054
49218 :016,146,016,223,016,145,116
49224 :016,157,016,156,001,047,209
49230 :030,105,001,134,001,004,097
49236 :001,176,001,132,038,001,177
49242 :001,047,001,051,166,179,023
49248 :001,018,001,150,001,002,013
49254 :001,047,030,005,001,059,245
49260 :166,092,001,246,001,038,140
49266 :016,156,016,157,016,154,117
49272 :020,156,020,157,001,047,009
49278 :001,118,016,153,030,006,194
49284 :020,157,020,158,016,156,147
49290 :001,047,001,130,016,148,225
49296 :016,146,020,145,020,223,202
49302 :016,156,030,007,001,051,155
49308 :166,179,001,034,016,219,003
49314 :030,008,020,223,020,145,096
49320 :016,222,016,214,016,148,032
49326 :020,211,020,210,016,222,105
49332 :030,009,001,246,016,222,192
49338 :016,223,016,222,020,223,138
49344 :020,145,020,211,020,223,063
49350 :016,222,030,010,020,210,194
49356 :020,211,020,148,020,214,069
49362 :020,213,020,148,020,211,074
49368 :020,213,020,148,020,210,079
49374 :016,211,001,047,001,047,033
49380 :030,011,001,059,166,097,080
49386 :001,246,016,223,016,145,113
49392 :016,223,020,211,020,148,110
49398 :020,149,020,146,016,145,230
49404 :020,148,020,149,020,150,247
49410 :020,137,020,215,020,150,052
49416 :020,149,020,215,020,150,070
49422 :020,148,016,149,020,150,005
49428 :020,147,016,148,016,150,005
49434 :016,146,030,012,001,047,022
49440 :001,051,001,246,016,150,241
49446 :016,146,020,148,020,145,021
49452 :016,146,016,145,016,158,029
49458 :016,145,016,148,020,211,094
49464 :020,146,020,145,020,223,118
49470 :020,145,020,158,030,013,192
```

```
49476 :001,047,001,246,016,145,012
49482 :016,223,016,211,020,148,196
49488 :020,215,016,149,016,211,195
49494 :016,214,088,148,016,148,204
49500 :012,211,001,079,001,079,219
49506 :001,091,001,018,001,018,228
49512 :001,242,001,242,166,026,014
49518 :001,242,001,242,001,079,164
49524 :084,072,069,077,069,083,058
49530 :032,070,082,079,077,032,238
49536 :034,080,082,079,077,069,037
49542 :078,065,068,069,034,013,205
49548 :079,070,032,080,073,067,029
49554 :084,085,082,069,083,032,069
49560 :065,084,032,065,078,032,252
49566 :069,088,072,073,066,073,087
49572 :084,073,079,078,013,066,045
49578 :089,032,077,079,068,069,072
49584 :083,084,069,032,077,085,094
49590 :083,083,079,082,071,083,151
49596 ;075,089,013,067,079,085,084
49602 :082,084,069,083,089,032,121
49608 :072,065,082,082,089,032,110
49614 :066,082,065,084,084,013,088
49620 :000,223,020,158,088,156,089
```

If you've ever heard a recording of a song played backward, you've heard the backward envelope. This type of envelope is not characteristic of any normal instrument.

A major drawback to the SID chip is that there's no master volume control for each voice. To some extent, the sustain level can be used as a voice volume control. Select sustain levels less than 15 to make a voice a little quieter. You may also want to adjust the decay rate so that the volume does not drop from the peak to the sustain level too rapidly.

One other technique is to experiment with the attack rate and release point so that each note is released before the volume hits the peak level (before the attack phase is complete). In order for this to work, the release rate must be set so that each note fades to complete silence before the next note begins. As you might guess, this isn't an easy technique to use.

Filter. Pin 5 of the monitor jack can be used to send an audio signal to the SID chip. This input signal is mixed with the output from the three voices before being sent to the master volume control.

If you want to pass the external audio signal through the filter, use the F-X command. This command works just like the FLT command, and accepts a yes/no data value.

Since using an external audio input is an advanced application of the SID chip, you'll rarely use the F-X command.

Portamento. When the portamento feature is turned on, the pitch glides from one note to the next. The starting pitch of the glide is the pitch of the previous note, and the ending pitch is the pitch of the new note. For example, if a G note is played after a C, the pitch will start at C and end at G.

The *absolute set pitch* command can be used to change the starting pitch of a glide. In the earlier example, a command to absolutely set the pitch at E could have been placed between the notes C and G. When the G was then played, the pitch would have glided from E to G, instead of from C to G.

Unlike the other commands, the absolute set pitch command is entered from the editing screen. To absolutely set the pitch, select the name, accidental, and octave of the pitch, change the duration so that it reads ABS SET, and press the joystick button.

Although this command is entered like a note, it is still a command because it has no duration. The command is entered

like a note because the editing screen offers a convenient method of specifying a pitch.

One application of this command is to create pitch-bending effects. To bend the pitch of a note, turn portamento on, set the pitch one half step below the pitch of the note, play the note, and turn portamento off. Another application is discussed later, in the section on synchronization.

The absolute set pitch command has no effect on playing when the portamento feature is turned off.

Vibrato. An advanced technique to use with vibrato is to change the depth of the vibrato during a note.

Let's say that a voice is playing a whole note. Break the whole note into four quarter notes. Enter the first quarter note with a tie, and follow it with a VDP command to set the vibrato depth. Do this again for the next two quarter notes, each time using a larger depth value. Enter the last quarter note without a tie.

When played, the tied quarter notes sound like a whole note, and the different VDP commands make the vibrato effect deepen gradually. The result is a more natural vibrato effect.

Remember to reset the vibrato depth for the next note.

Detuning. Sidplayer can play a note using any frequency, including frequencies between the normal half steps. Just choose a note with a pitch near the desired frequency, and use the DTN command to set a frequency offset value. Refer to Appendix M of the *Commodore 64 User's Guide* to find the frequency number for each pitch.

As an example, let's say that you want to play a note with a frequency of 500 Hz. Use the following formula to calculate the corresponding frequency number.

Frequency number = Frequency/0.06097

The frequency number for 500 Hz is 500/0.06097, or 8201. The pitch with the closest frequency is B4, which has the frequency number 8101. Therefore, to play a note at 500 Hz, all you have to do is enter the command DTN 100 followed by the note B4.

Transposing. By using the TPS command, it's possible to play notes in a ninth octave. If you play notes in octave 0 after entering the command TPS −12, the notes will actually play in octave −1.

It's very hard to hear the pitch of notes in octave 0, and

even harder to hear notes in octave −1, so this technique is probably useful only for creating special effects.

Unfortunately, this technique won't work with notes above octave 7. If a note is transposed into octave 8, it's still played in octave 7.

One other possibility is to play a phrase in different keys by entering the TPS command with the correct number of half steps before each phrase call.

Synchronization. When two voices are synchronized by the sync mode, a rather unusual effect can be created by using portamento on the voice which adds the harmonics.

One technique is to enter an absolute set pitch command before each note. The command should set the pitch one octave below the pitch of the following note so that the pitch glides up one full octave when the note is played. This technique was used in the song "ETAL" in Chapter 7.

Future expansion. The AUX command is reserved for possible future expansion. At present, this command is ignored by Sidplayer.

Uncommon instruments. Sidplayer can emulate instruments which are no longer commonly used. Program 10-9, for example, imitates instruments that were popular in the late Renaissance period.

Sound effects. Sidplayer can be used to produce sound effects as well as music. Program 10-10 demonstrates white noise to produce an explosion.

Phonetics. Certain combinations of frequencies are associated with different vowel sounds, as demonstrated in Program 10-11.

More than three voices. The most serious limitation of the SID chip is that it supports only three voices. Many classical music pieces can be played with three voices, but most of today's songs need at least four.

Sometimes when a song has to play four notes, one of the notes can be eliminated without significantly affecting the music. When a note in the bass clef is also played a couple of octaves higher in the treble clef, the treble clef note can be dropped. If the treble clef contains notes which do not seem to be part of the melody, try deleting them. When it's not obvious which note should be dropped, experiment.

Program 10-9. COURANTE.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49679
Filename: COURANTE.MUS

```
49152 :190,000,090,000,132,000,156
49158 :006,128,001,039,001,028,209
49164 :001,000,001,244,001,072,075
49170 :038,001,166,010,030,001,008
49176 :024,155,024,156,020,157,048
49182 :020,157,052,157,024,156,084
49188 :020,155,016,153,030,002,156
49194 :024,156,024,157,020,158,069
49200 :020,156,020,156,024,155,067
49206 :024,156,020,157,020,155,074
49212 :020,155,030,003,020,155,187
49218 :020,154,020,154,020,154,076
49224 :024,153,024,154,020,155,090
49230 :020,153,020,153,030,004,202
49236 :024,155,024,156,024,157,112
49242 :024,158,024,157,024,158,123
49248 :024,157,024,158,024,157,128
49254 :024,156,020,155,020,153,118
49260 :020,153,030,005,024,156,240
49266 :024,157,020,158,020,156,137
49272 :020,156,024,155,024,156,143
49278 :020,157,020,155,020,155,141
49284 :030,006,020,155,020,154,005
49290 :020,154,020,154,024,153,151
49296 :024,154,020,155,020,153,158
49302 :020,153,030,007,024,167,039
49308 :024,153,020,154,020,154,169
49314 :052,154,024,153,020,167,220
49320 :016,165,030,008,020,154,049
49326 :020,167,016,165,020,157,207
49332 :020,155,024,154,024,153,198
49338 :020,165,030,009,020,167,085
49344 :020,153,001,079,006,128,067
49350 :001,135,002,100,001,004,185
49356 :001,000,001,188,001,072,211
49362 :038,001,166,179,030,001,113
49368 :020,000,012,165,112,161,174
49374 :030,002,020,161,012,172,107
49380 :112,161,030,003,020,161,203
49386 :012,165,112,161,030,004,206
49392 :020,161,012,165,112,161,103
49398 :030,005,020,161,012,172,134
49404 :112,161,030,006,020,161,230
49410 :012,165,112,161,030,007,233
49416 :020,161,012,173,112,173,147
49422 :030,008,076,173,020,173,238
49428 :016,161,020,173,030,009,173
49434 :012,161,001,079,006,128,157
49440 :001,039,001,004,001,080,158
49446 :001,132,001,008,038,001,219
49452 :166,181,030,001,001,070,237
49458 :054,002,020,000,020,137,027
49464 :020,000,020,137,024,137,138
49470 :024,137,020,137,020,137,025
```

```
49476 :020,137,001,015,030,003,018
49482 :020,000,020,137,024,137,156
49488 :024,137,020,137,024,137,047
49494 :024,137,024,137,024,137,057
49500 :024,137,024,137,020,137,059
49506 :001,047,030,004,001,066,247
49512 :030,007,020,000,020,137,062
49518 :024,137,024,137,020,137,077
49524 :024,137,024,137,020,137,083
49530 :020,137,020,137,030,008,218
49536 :020,137,020,137,016,137,083
49542 :020,137,020,137,024,137,097
49548 :024,137,024,137,024,137,111
49554 :030,009,020,137,020,137,243
49560 :028,137,028,137,028,137,135
49566 :024,137,001,079,067,079,033
49572 :085,082,065,078,084,069,115
49578 :032,070,082,079,077,032,030
49584 :034,084,069,082,080,083,096
49590 :073,067,079,082,069,034,074
49596 :013,077,073,067,072,065,043
49602 :069,076,032,080,082,065,086
49608 :069,084,079,082,073,085,160
49614 :083,032,040,049,053,055,006
49620 :049,045,049,054,050,049,252
49626 :041,013,080,073,080,069,062
49632 :044,032,082,069,071,065,075
49638 :076,044,032,065,078,068,081
49644 :032,084,065,066,079,082,132
49650 :013,065,082,082,065,078,115
49656 :071,069,068,032,066,089,131
```



```
49662 :032,082,079,066,069,082,152
49668 :084,032,072,073,071,071,151
49674 :073,078,083,013,000,149,150
```

Program 10-10. JOKE.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49295
Filename: JOKE.MUS

```
49152 :074,000,016,000,016,000,106
49158 :001,078,038,005,050,032,210
49164 :054,002,001,022,024,155,014
49170 :024,154,052,153,024,154,067
49176 :020,153,020,153,020,155,033
49182 :020,157,020,156,020,158,049
49188 :001,047,020,081,020,081,030
49194 :020,000,020,159,020,159,164
49200 :012,000,001,015,001,018,095
49206 :020,145,020,145,001,254,127
49212 :001,022,007,044,001,007,142
49218 :001,004,001,160,001,132,109
49224 :038,001,000,169,044,161,229
49230 :001,079,054,005,008,000,225
49236 :001,015,044,000,020,000,164
49242 :166,179,001,018,001,079,022
49248 :054,005,008,000,001,015,179
49254 :044,000,020,000,166,181,001
49260 :001,018,001,079,065,032,048
```

```
49266 :076,073,084,084,076,069,064
49272 :032,074,079,075,069,013,206
49278 :070,082,079,077,032,065,019
49284 :032,067,065,082,084,079,029
49290 :079,078,013,013,013,000,078
```

Program 10-11. YOY.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 49295
Filename: YOY.MUS

```
49152 :054,000,002,000,032,000,088
49158 :006,128,001,039,001,059,240
49164 :010,070,054,002,001,006,155
49170 :031,008,000,093,052,066,012
49176 :001,047,001,022,000,074,169
49182 :052,085,001,047,001,002,218
49188 :052,000,001,015,001,002,107
49194 :166,062,001,002,166,113,040
49200 :001,002,166,145,001,018,125
49206 :166,014,001,002,001,079,061
49212 :001,079,001,039,054,003,237
49218 :052,153,052,153,052,153,169
49224 :052,000,052,153,052,153,022
49230 :052,153,052,000,052,153,028
49236 :052,219,052,230,052,231,152
49242 :052,153,001,079,087,072,022
49248 :089,032,079,072,032,087,231
49254 :072,089,063,013,087,072,242
49260 :089,032,079,072,032,087,243
49266 :072,089,063,013,087,072,254
49272 :089,032,087,072,089,032,009
49278 :087,072,089,032,079,072,045
49284 :032,087,072,089,063,013,232
49290 :013,000,013,013,013,000,190
```

Contemporary sheet music is often written with two treble staves and one bass staff. The top treble staff usually has just the notes for the melody, and the other one has the melody notes plus extra notes for chords. If you enter the top treble staff and the bass staff, you'll still have one voice free for percussion effects.

A clever technique is to have one voice play both a bass part and percussion part. Bass notes can be played on each beat, and percussion effects, such as snare drum strikes, can be played on the off beat. To make this as easy as possible, define a phrase which selects a nonsustaining envelope with the noise waveform, plays a note, and then resets the envelope and waveform. To enter the voice, just enter bass notes separated by phrase calls.

Instrument parameters. Table 10-3 lists suggested values for emulating different instruments. Filter values may have to be adjusted for your particular computer. This list is by no means complete; many other instrument settings are possible and can be found by experimentation.

Table 10-3. Sidplayer Instrument Parameters

Instrument	ATK	DCY	SUS	RLS	PNT	WAV	P-W	P-S	F-M	AUT	RES	FLT	F-S	F-C	SNC	RNG	VDP	VRT	TPS	DTN
DEFAULT	2	0	15	5	4	P	2048	0		0	0	N	0		N	N	0	0	0	
VIOLIN	0	0			4	P	800										30	2	12	
CELLO	5	0	12	4	4	P	2048										20	2	−12	
BASS (pluck)	1	10	0	10	1	P	1800		L	−10	9	Y							−12	
HARP	0	9	0	0	1	T			LB	−1	15	Y								
BANJO	1	8	0	0	1	P	100		B	3	12	Y	5							
MANDOLIN	1	8	0	0	1	P	500		H	−20	12	Y								
BAZOOKI	1	10	0	0	1	P	1000		H	−20	12	Y	5							
LUTE	0	8	0	0	1	T			L	−40	15	Y								
KOTO	0	8	0	0	1	P	300		L	−10	15	Y	20							
SITAR	2	10	12	12	7	P	100	10	H	−20	15	Y	5							
FLUTE	3	0	14	4	1	T														
PICCOLO	3	0	14	4	1	T													12	
RECORDER—ALTO	3	0	14	4	1	T														
CLARINET	3	0	14			P	2048													
OBOE	7	0	15	2	1	P	700		B	−20	15	Y								
BASSOON	7	8	14	4	1	P	700		B	−20	15	Y							−12	
BAGPIPE	2	0	15	5	1	P	200													70
HARMONICA	5	0	8	4	4	P	2048										20	2		
ACCORDION	2	0	15	4	4	S														70
SHAWM	2	2	14	2	1	P	100		B	5	15	Y								
TRUMPET	5	0	15	4	1	S			L	20	15	Y	1							
TROMBONE	9	0	15	4	1	S			L	1	15	Y								
TUBA	9	0	15	4	1	S			L	10	15	Y								
HARPSICHORD 4′	0	12	8	9	8	P	100													
HARPSICHORD 8′	0	12	8	9	8	P	300													
ORGAN: FLUTE	3	0	14	4	1	T														
ORGAN: PRINCIPAL	3	2	10	4	1	P	2048													
ORGAN: REED	3	0	14	4	1	P	200													
ORGAN: TRUMPET	3	0	15	4	1	S														
CALLIOPE	0	12	13	5	4	P	2048												12	70
FINGER DRUM	0	9	0	0	1	T														
BELLS	0	10	0	10	1	T										Y			7,16	
CHIMES	0	10	0	10	16	P	2048													70
WOOD BLOCK	0	2	0			T														

Music and Your BASIC Program

Merging Sidplayer with BASIC Programs

Run Sidplayer, and while it's playing a song, press the RUN/STOP key. The music still plays, even though the program has stopped. Now type LIST. The music continues as the program is listed on the screen. Enter a few more statements, such as POKEs to change the screen colors, and then enter the CONT command to make Sidplayer resume. Perhaps not the most exciting demonstration you've seen, but it does illustrate something important.

Sidplayer has been designed so that it can play music while BASIC executes commands, statements, or even a whole program. Every 1/60 second, BASIC processing is temporarily set aside and Sidplayer is allowed to process the music. When the music processing is done, BASIC resumes. Since this happens 60 times a second, the continual interruption of BASIC is too fast to be noticeable. BASIC and the Sidplayer appear to be executing simultaneously.

Although this technique causes BASIC processing to run a little slower than normal, it does make it easy to add music to games, adventures, or educational programs. And that can enhance your own BASIC programs.

Load and play procedure. To demonstrate how to merge Sidplayer with a BASIC program, let's begin with a simplified Player program. The Player listed in Program 11-1 contains only the statements necessary to load and play a song, and can be readily merged with another program.

Program 11-1. SID.BAS

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SIDPLAYER":PRINT " BY
    {SPACE}CRAIG CHAMBERLAIN":PRINT          :rem 173
120 DN=8:SA=780:SX=781:SY=782:SP=783         :rem 134
130 GOSUB 57000:REM LOAD SIDPLAYER           :rem 251
200 F$="COMMODORE":LA=PEEK(49)+256*PEEK(50)+1000:G
    OSUB 57500:REM LOAD SONG                 :rem 242
210 SYS HK:REM HOOK (INSTALL)                :rem 220
220 POKE SX,LO:POKE SY,HI:SYS PL:REM SET FOR PLAYI
    NG                                       :rem 225
```

```
230 K=PEEK(SX)+256*PEEK(SY):REM GET ADDRESS OF TEX
    T LINES                                    :rem 171
240 IF PEEK(K) THEN PRINT CHR$(PEEK(K));:K=K+1:GOT
    O 240:REM PRINT UNTIL CHR$(0)              :rem 27
250 POKE SS,7:REM START MUSIC                  :rem 252
260 IF PEEK(SS)AND7 GOTO 260:REM STILL PLAYING
                                               :rem 216
270 SYS DP:REM DROP (REMOVE)                   :rem 158
280 END                                        :rem 113
57000 POKE SA,1:POKE SX,DN:POKE SY,1:SYS 65466:F$=
      "SID.OBJ":GOSUB 59000                    :rem 106
57010 POKE SA,0:SYS 65493:IF PEEK(SP)AND1 GOTO 591
      00                                       :rem 126
57020 SS=49152:HK=49435:PL=49458:DP=49629:RETURN
                                               :rem 77
57100 GOSUB 57000:POKE SX,LA-256*INT(LA/256):POKE
      {SPACE}SY,INT(LA/256):SYS 51042          :rem 218
57110 LA=LA+1398:RETURN                        :rem 111
57500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".MUS":GOSUB 59000                    :rem 61
57510 HI=INT(LA/256):LO=LA-256*HI              :rem 137
57520 POKE SA,0:POKE SX,LO:POKE SY,HI:SYS 65493:IF
       PEEK(SP)AND1 GOTO 59100                 :rem 49
57530 LA=PEEK(SX)+256*PEEK(SY):RETURN          :rem 29
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                                   :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                                 :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END                  :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                               :rem 36
59120 PRINT ST:END                             :rem 70
```

When adding this to your own program, all of the lines before line 57000 can be renumbered to start at any place in the program. The lines starting at 57000, however, must not be renumbered.

The program begins with the standard assignments for using tape or disk I/O and assigns values to variables to be used with SYS (line 120).

DN=8:SA=780:SX=781:SY=782:SP=783

Change the statement DN=8 to DN=1 if you are using the Datassette.

The next step is to load the Sidplayer machine language into memory (line 130 of Program 11-1).

GOSUB 57000

This subroutine loads the "SID.OBJ" file, and also assigns the variables SS, HK, PL, and DP. These four variables are used later in the program. If you're using tape, be sure to position the tape to the spot where SID.OBJ begins. Disk users must have the proper disk in the drive.

Before you can load a song into memory, the music filename and the LOAD address must be set. Assign the name of the music file to the variable F$ (line 200). Do not include the .MUS extension as part of the filename.

F$="COMMODORE"

Now calculate the LOAD address (line 200).

LA=PEEK(49)+256*PEEK(50)+1000

The LOAD address is the address of free memory, plus 1000 bytes for a safety margin. This is the address in memory where the music file will be stored.

Call the subroutine at line 57500 to actually load the song (line 200). Again, be sure the proper disk is in the drive or that the tape is positioned properly. If you assigned F$ as above for a demonstration, make sure that the "COMMODORE.MUS" file is on that disk or tape.

GOSUB 57500

Besides loading the requested music file, the subroutine at 57500 also assigns the variables LO and HI to be the low and high bytes of the LOAD address. These variables are used later in the program.

At this point, everything has been loaded into memory. There's still a little bit of preparation which must be done before the playing can begin.

SYS HK

This statement (line 210) calls the Sidplayer HOOK routine, which installs Sidplayer as part of the normal interrupt processing that's done every 1/60 second.

POKE SX,LO:POKE SY,HI:SYS PL

The PLAY (SYS PL) routine tells Sidplayer where the song begins in memory, and sets all the default values, such as tempo and volume. It also returns the address of the text lines in locations SX and SY. It's not necessary to print the text lines in order to play the song, so the next two program lines (lines 230 and 240) are optional. If you *do* want to print the text lines, the program should start displaying characters at the returned address and stop when it reaches a CHR$(0).

The song is now ready to start playing. The following statement makes the playing begin (line 250).

POKE SS,7

Location SS is the Sidplayer status value. The last three bits of this location control voice processing.

Voice	SS Bit	POKE Value
1	0	1
2	1	2
3	2	4

When a bit is set, the corresponding voice plays. Playing for that voice stops when the bit is cleared, or set to 0. An individual voice can be played by POKEing location SS with the value 1, 2, or 4. To play all three voices, POKE SS with 1+2+4, or 7.

Playing continues as long as the three bits are set. Playing can be stopped at any time by POKEing SS with a 0. The bits are automatically cleared by Sidplayer when it reaches the end of the song or if an error occurs.

Program 11-1 simply loops until the three bits in SS are cleared when the end of the song is reached, but anything can be done during this time. The program could draw pictures on the screen, change the colors, or move sprites.

After a song is done, it can be replayed if you like. Just call the PLAY routine and set the three status bits again. Use this line:

POKE SX,LO:POKE SY,HI:SYS PL:POKE SS,7:REM REPLAY

When playing is finished, call the DROP routine to remove Sidplayer from the interrupt processing.

SYS DP

The DROP routine undoes everything done by the HOOK routine and restores the interrupt processing to normal.

You must be careful in using the HOOK and DROP routines. Do not call the HOOK routine if Sidplayer is already installed. Also, do not call the DROP routine if Sidplayer has already been removed or has not yet been installed. The correct order is HOOK, PLAY, then DROP. Calling these routines in the wrong order can cause the computer to crash.

While a song is playing, BASIC is free to execute any statements and do any processing you wish. The only restrictions concern using string variables and tape or disk I/O. If

string variables are assigned frequently, the program should periodically call the free memory function, as in K=FRE(0), to reorganize free memory. Otherwise, the string data may interfere with the free memory used for storing the song.

The processing that's done every 1/60 second is handled differently when the computer is communicating with the Datassette or disk drive. Therefore, tape or disk files should not be accessed while a song is playing.

The flag command. Sometimes it would be convenient if a BASIC program could determine which part of a song Sidplayer is currently playing. Such information would be helpful in synchronizing screen displays to the music. What's needed is a method of communication between Sidplayer and the BASIC program.

The BASIC program can control Sidplayer by setting or clearing the three status bits in location SS. For communication from Sidplayer to the BASIC program, the FLG command is available. This command is entered in the Editor with a number from 0 to 255. When playing reaches the command, Sidplayer POKEs the number into location SS+1, the *flag* location. The BASIC program can monitor this location to watch for specific values and change the screen accordingly.

Several FLG commands may be used in a song. The general procedure is to wait for the value in the flag location to change, update the screen, wait for the next value, and so on. To detect a change in the flag location, a few different methods can be used.

The first is to wait for a specific value to be POKEd into the flag location, as in:

```
400 IF PEEK(SS+1)<>2 GOTO 400
```

This method requires a direct correspondence between the flag values in the music and the values being checked in the program. Usually, the flag commands will use incrementing numbers, so the BASIC program would first watch for the value 1, then for the value 2, and so on.

Another method is to disregard the value in the flag location and just wait for it to change. This can be done in two ways.

```
400 P=PEEK(SS+1)
410 IF PEEK(SS+1)=P GOTO 410
```

or

400 POKE SS+1,0
410 IF PEEK(SS+1)=0 GOTO 410

The advantage of this method is that you don't have to remember the exact flag values used in the music.

One other method is to use the WAIT statement:

400 WAIT SS+1,1:REM WAIT FOR ODD NUMBER

or

400 WAIT SS+1,1,1:REM WAIT FOR EVEN NUMBER

The advantage of using the WAIT statement is that you don't have to use a whole program line. Other statements can be placed after the WAIT. (Refer to Chapter 4 for more details on WAIT.) The only drawback is that the flag values must alternate between even and odd numbers.

To show how to merge music with a BASIC program and how to use the FLG command, we've provided a demonstration program. Actually two programs—the first, Program 11-3, is a BASIC program which uses colorful screen displays and sprites while the music plays. Enter Program 11-3 first, especially if you're using tape.

The other part of the demonstration is a music file, listed as Program 11-2. It must be entered with MLX and saved as SCIPIO.MUS. This song contains FLG commands on voice 1.

Since Program 11-3, "SIDDEMO," must first load SID.OBJ and then SCIPIO.MUS, tape users must position the tape to the start of SID.OBJ before running the program. The subroutine beginning at line 900 was included so the program will prompt tape users to advance the tape to SCIPIO.MUS once SID.OBJ is loaded. If you use a disk drive, just make sure SIDDEMO, SCIPIO.MUS, and SID.OBJ are all on the same disk.

Remember to change line 120 to DN=1 if you have a Datassette.

Program 11-2. SCIPIO.MUS

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

Starting Address: 49152
Ending Address: 50051
Filename: SCIPIO.MUS

```
49152 :054,001,004,001,008,001,069
49158 :006,112,001,136,038,008,051
```

```
49164 :001,006,030,001,116,154,064
49170 :024,155,030,002,070,001,044
49176 :001,252,012,092,070,002,197
49182 :012,158,030,003,070,003,050
49188 :044,146,001,228,070,004,017
49194 :080,155,030,004,116,092,007
49200 :088,157,116,158,088,159,046
49206 :080,158,116,157,088,092,233
49212 :030,005,044,155,001,047,086
49218 :001,022,080,158,030,006,107
49224 :070,005,116,159,088,081,079
49230 :080,146,080,081,080,159,192
49236 :030,007,076,158,012,157,012
49242 :001,047,030,008,116,092,128
49248 :088,155,116,154,088,157,086
49254 :080,092,116,155,088,154,019
49260 :030,009,044,154,001,094,184
49266 :080,084,030,010,070,000,132
49272 :080,147,080,146,084,081,226
49278 :084,146,080,147,030,011,112
49284 :076,158,012,157,030,012,065
49290 :080,092,084,157,084,092,215
49296 :080,155,080,154,030,013,144
49302 :044,158,080,147,030,014,111
49308 :116,159,088,158,116,159,184
49314 :088,081,080,159,080,158,040
49320 :030,015,044,159,080,081,065
49326 :030,016,080,146,084,081,099
49332 :084,159,080,081,116,159,091
49338 :088,158,030,017,044,158,169
49344 :016,157,030,018,001,003,161
49350 :070,001,116,092,088,155,208
49356 :116,092,024,157,016,092,189
49362 :016,155,030,019,001,003,178
49368 :012,157,012,092,030,020,027
49374 :001,003,016,159,016,158,063
49380 :016,157,016,092,030,021,048
49386 :001,003,044,155,001,174,100
49392 :016,158,030,022,070,000,024
49398 :116,159,088,081,016,146,084
49404 :016,081,016,159,030,023,065
49410 :012,158,012,157,030,024,139
49416 :116,092,088,155,116,154,217
49422 :088,157,080,092,116,155,190
49428 :088,154,030,025,044,154,003
49434 :001,142,001,002,001,003,176
49440 :001,018,001,174,116,092,178
49446 :088,155,006,128,116,154,173
49452 :088,157,080,092,006,160,115
49458 :116,155,088,154,044,154,249
```

```
49464 :016,000,001,079,006,112,014
49470 :001,136,038,008,001,038,028
49476 :030,001,016,000,030,002,147
49482 :001,252,012,154,012,155,148
49488 :030,003,044,092,001,228,222
49494 :080,089,030,004,080,154,011
49500 :080,089,080,154,116,155,254
49506 :088,154,030,005,044,089,252
49512 :080,089,030,006,116,154,067
49518 :088,155,116,092,088,157,038
49524 :080,158,080,089,030,007,048
49530 :016,154,012,154,016,089,051
49536 :001,047,030,008,016,166,140
49542 :016,167,016,154,016,165,156
49548 :030,009,044,100,080,158,049
49554 :030,010,076,158,012,158,078
49560 :030,011,016,154,012,154,017
49566 :016,089,030,012,016,154,219
49572 :016,000,016,000,016,000,212
49578 :030,013,080,089,080,154,104
49584 :016,155,080,081,030,014,040
49590 :080,092,080,155,080,092,249
49596 :080,092,030,015,080,093,066
49602 :080,090,016,155,080,155,002
49608 :030,016,012,158,016,158,078
49614 :052,093,024,000,030,017,166
49620 :080,089,080,154,016,155,018
49626 :016,089,030,018,012,089,216
49632 :012,089,030,019,012,154,028
49638 :016,154,016,089,030,020,043
49644 :012,154,016,089,016,154,165
49650 :030,021,044,089,016,089,019
49656 :030,022,116,154,088,155,045
49662 :116,092,024,157,016,158,049
49668 :016,089,030,023,016,154,076
49674 :012,154,016,089,030,024,079
49680 :052,154,024,000,052,167,209
49686 :024,000,080,154,052,089,165
49692 :024,000,030,025,044,100,251
49698 :001,034,030,022,052,154,071
49704 :024,000,052,167,024,000,051
49710 :080,154,052,089,024,000,189
49716 :030,023,016,100,016,000,237
49722 :016,000,016,000,001,079,170
49728 :006,112,001,136,038,008,109
49734 :001,054,030,001,016,000,172
49740 :030,002,012,162,012,097,135
49746 :030,003,044,175,016,174,012
49752 :030,004,016,162,016,163,223
49758 :016,100,016,165,030,005,170
```

```
49764 :016,166,016,163,016,174,139
49770 :016,100,030,006,016,175,193
49776 :016,167,016,166,016,165,146
49782 :030,007,012,100,016,163,190
49788 :016,166,001,047,030,008,136
49794 :016,162,016,165,016,166,159
49800 :016,174,030,009,016,162,031
49806 :016,174,016,170,080,154,240
49812 :030,010,080,089,080,167,092
49818 :080,166,080,165,030,011,174
49824 :076,100,016,163,016,166,185
49830 :030,012,080,162,080,162,180
49836 :080,097,080,175,030,013,135
49842 :080,174,080,175,016,097,032
49848 :080,174,030,014,080,162,212
49854 :080,097,080,162,080,175,096
49860 :030,015,080,163,080,100,152
49866 :016,101,080,166,030,016,099
49872 :080,100,080,162,080,163,105
49878 :080,171,030,017,080,174,254
49884 :080,175,016,097,016,174,010
49890 :030,018,116,162,088,097,225
49896 :116,162,024,163,016,162,107
49902 :016,097,030,019,012,175,075
49908 :012,174,030,020,016,165,149
49914 :016,100,016,163,016,162,211
49920 :030,021,016,166,016,163,156
49926 :016,174,016,100,030,022,108
49932 :016,175,016,167,016,166,056
49938 :016,165,030,023,012,100,108
49944 :016,163,016,166,030,024,183
49950 :016,162,016,173,016,174,075
49956 :016,174,030,025,016,162,203
49962 :016,174,016,170,001,050,213
49968 :016,162,016,173,016,174,093
49974 :016,174,070,001,016,162,237
49980 :070,002,016,174,070,003,139
49986 :016,170,016,000,001,079,092
49992 :077,065,082,067,072,032,211
49998 :070,082,079,077,032,034,196
50004 :083,067,073,080,073,079,027
50010 :034,013,071,046,070,046,114
50016 :032,072,065,078,068,069,224
50022 :076,013,067,079,085,082,248
50028 :084,069,083,089,032,072,025
50034 :065,082,082,089,032,066,018
50040 :082,065,084,084,013,013,205
50046 :000,162,016,164,030,002,244
```

Program 11-3. SIDDEMO

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SIDPLAYER DEMONSTRATIO
    N"                                          :rem 171
110 PRINT " BY CRAIG CHAMBERLAIN":PRINT         :rem 65
120 DN=8:SA=780:SX=781:SY=782:SP=783            :rem 134
130 CY=214:DIM C$(15):FOR K=0 TO 15:READ P:C$(K)=C
    HR$(P):NEXT                                 :rem 83
140 GOSUB 57000:F$="SCIPIO":GOSUB 900:LA=PEEK(49)+
    256*PEEK(50)+1000:GOSUB 57500               :rem 171
150 SYS HK:POKE SX,LO:POKE SY,HI:SYS PL         :rem 170
200 PRINT CHR$(147):POKE 53280,0:POKE 53281,0
                                                :rem 156
210 FOR K=832 TO 894:READ P:POKE K,P:NEXT       :rem 54
220 FOR K=0 TO 7:POKE 2040+K,13:READ C:POKE 53287+
    K,C                                         :rem 50
230 POKE 53248+K*2,(34+K*40)AND255:POKE 53249+K*2,
    70+RND(0)*20:NEXT                           :rem 93
240 POKE 53264,192:POKE 53271,0:POKE 53277,0
                                                :rem 37
250 FOR K=1 TO 3:POKE CY,3:PRINT:PRINT TAB(5) C$(K
    ) "PRESENTING..."                           :rem 44
260 FOR J=1 TO 20:NEXT J,K:FOR K=1 TO 750:NEXT
                                                 :rem 8
270 FOR K=0 TO 3:POKE CY,3:PRINT:PRINT TAB(5) C$(3
    -K) "PRESENTING..."                        :rem 141
280 FOR J=1 TO 20:NEXT J,K:FOR K=1 TO 300:NEXT:POK
    E SS,7:FL=SS+1                             :rem 126
300 FOR K=1 TO 3:POKE CY,6:PRINT                :rem 55
310 IF PEEK(FL)<>K GOTO 310                    :rem 189
320 PRINT TAB(10) C$(K) "SIDPLAYER":NEXT:WAIT FL,1
    ,1:PRINT:PRINT                              :rem 97
330 WAIT FL,1:PRINT TAB(10) "A COMPLETE":PRINT TAB
    (10) "MUSIC SYSTEM"                        :rem 226
340 PRINT TAB(10) "FOR THE COMMODORE 64"       :rem 200
400 WAIT FL,1,1:PRINT CHR$(147):POKE 53269,255:POK
    E CY,8:PRINT                                :rem 52
410 PRINT "FEATURES:":PRINT:PRINT "JOYSTICK OR KEY
    BOARD NOTE ENTRY":PRINT                    :rem 194
415 PRINT "SUPPORTS ANY TIME SIGNATURE":PRINT
                                               :rem 139
420 PRINT "MEASURE EDITING":PRINT:PRINT "TRANSPOSI
    NG BY HALF STEPS":PRINT                    :rem 179
430 PRINT "EASY INSERT/DELETE/REPLACE EDITING":PRI
    NT                                          :rem 71
440 PRINT "SIMPLE ENOUGH FOR A NOVICE TO USE":PRIN
    T                                           :rem 60
450 POKE 53249+INT(RND(0)*8)*2,70+RND(0)*20:IF PEE
    K(FL)=0 GOTO 450                           :rem 24
```

```
500 POKE 53269,0:PRINT CHR$(147):PRINT:PRINT:PRINT
     "PLUS..."                                  :rem 69
510 FOR K=5 TO 13 STEP 2:READ S$:POKE CY,K:PRINT:P
    RINT TAB(10) S$                             :rem 134
520 POKE CY,K:PRINT:PRINT TAB(10);:FOR J=1 TO 10:P
    RINT CHR$(20);                              :rem 190
530 FOR I=1 TO 20:NEXT I,J,K:PRINT:PRINT:PRINT
                                                :rem 63
540 WAIT FL,1,1:PRINT "AND BEST OF ALL,":PRINT "MU
    SIC CREATED BY SIDPLAYER"                   :rem 140
550 PRINT "CAN BE MERGED WITH":PRINT "YOUR BASIC P
    ROGRAMS":WAIT FL,1                          :rem 57
600 PRINT CHR$(147):POKE CY,10:PRINT:PRINT " SIDPL
    AYER";:WAIT FL,1,1                          :rem 24
610 PRINT TAB(15) "SIDPLAYER";:WAIT FL,1:PRINT TAB
    (30) "SIDPLAYER":WAIT FL,1,1                :rem 58
630 PRINT CHR$(147):FOR K=1 TO 23:PRINT C$(RND(0)*
    12+4) " SIDPLAYER"                          :rem 183
640 FOR J=1 TO 30:NEXT J,K                      :rem 121
650 FOR K=22 TO 0 STEP -1:POKE CY,K:PRINT       :rem 30
660 PRINT TAB(15) C$(RND(0)*12+4) "SIDPLAYER":FOR
    {SPACE}J=1 TO 30:NEXT J,K                   :rem 202
670 POKE CY,0:PRINT:FOR K=1 TO 23:PRINT C$(RND(0)*
    12+4) TAB(30) "SIDPLAYER"                   :rem 185
680 FOR J=1 TO 30:NEXT J,K                      :rem 125
690 IF PEEK(FL)<>1 GOTO 690                     :rem 185
700 PRINT CHR$(147):POKE CY,16:PRINT:PRINT TAB(25)
     C$(3) "THAT'S":WAIT FL,1,1                 :rem 92
710 PRINT TAB(27) "ALL,":WAIT FL,1:PRINT TAB(25) "
    FOLKS!"                                     :rem 151
720 IF PEEK(SS)AND7 GOTO 720                    :rem 32
730 SYS DP:PRINT CHR$(147) CHR$(154);:POKE 53280,1
    4:POKE 53281,6:END                          :rem 228
800 DATA 144,151,152,155,5,28,30,31,129,149,150,15
    3,154,156,158,159                           :rem 235
810 DATA 0,3,192,0,3,224,0,3,240,0,3,120,0,3,56,0,
    3,28,0,3,12,0,3,0,0,3,0                     :rem 155
820 DATA 0,3,0,7,227,0,31,251,0,63,255,0,127,255,0
    ,127,255,0,127,254,0                        :rem 75
830 DATA 63,252,0,31,248,0,7,224,0,0,0,0,0,0,0,0
                                                :rem 160
840 DATA 10,7,4,13,5,3,8,15                     :rem 32
850 DATA "ENVELOPES AND WAVEFORMS","REPEATS AND PH
    RASES"                                      :rem 166
860 DATA "AUTOMATIC FILTER MODE","GLISSANDO AND VI
    BRATO"                                      :rem 140
870 DATA "MANY OTHER FEATURES"                  :rem 19
900 IF DN=8 THEN RETURN                         :rem 62
910 PRINT "POSITION TAPE TO THE BEGINNING OF":PRIN
    T F$; " THEN PRESS RETURN"                  :rem 0
```

```
920 WAIT 198,15:GET K$:RETURN                    :rem 167
57000 POKE SA,1:POKE SX,DN:POKE SY,1:SYS65466:F$="
      SID.OBJ":GOSUB 59000                       :rem 106
57010 POKE SA,0:SYS 65493:IF PEEK(SP)AND1 GOTO 591
      00                                         :rem 126
57020 SS=49152:HK=49435:PL=49458:DP=49629:RETURN
                                                 :rem 77
57100 GOSUB 57000:POKE SX,LA-256*INT(LA/256):POKE
      {SPACE}SY,INT(LA/256):SYS 51042            :rem 218
57110 LA=LA+1398:RETURN                          :rem 111
57500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".MUS":GOSUB 59000                      :rem 61
57510 HI=INT(LA/256):LO=LA-256*HI                :rem 137
57520 POKE SA,0:POKE SX,LO:POKE SY,HI:SYS 65493:IF
       PEEK(SP)AND1 GOTO 59100                   :rem 49
57530 LA=PEEK(SX)+256*PEEK(SY):RETURN            :rem 29
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                                     :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                                   :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END                    :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                 :rem 36
59120 PRINT ST:END                               :rem 70
```

The halt command. With the FLG command, the BASIC program waits until Sidplayer reaches a certain point in the music. An alternative is to have the playing stop and wait until the program is ready for the music to continue. This is possible with the HLT command.

HLT is used to stop the playing on a particular voice. When processed, this command clears the corresponding status bit and sets the frequency to 0 to reduce the background noise of the SID chip. To make the playing continue, the BASIC program can simply set the status bit again.

Multiple songs. Once a song is through playing, it's a simple matter to load and play another song. Just assign the name of the new song to F$, recalculate the LOAD address and assign it to LA, and call the subroutine at 57500. To play the song, follow the standard HOOK, PLAY, and DROP procedure.

Another possibility is to hold more than one song in memory at the same time. The only limit to the number of songs that can be stored in memory simultaneously is the amount of free memory available. When the subroutine at

57500 loads a song, it changes the value of LA to the address of the first free byte after the song, which is the address where the next song should be loaded. This makes it easy to load one song after another.

Load the first song in the usual way.

```
F$="SONG1":LA=PEEK(49)+256*PEEK(50)+1000:GOSUB 57500
```

Remember the values of LO and HI for later use.

```
L1=LO:H1=HI
```

To load the second song, assign the new filename to F$, but don't change the value of LA. Call the subroutine at 57500, and remember the new values of LO and HI.

```
F$="SONG2":GOSUB 57500:L2=LO:H2=HI
```

This procedure can be repeated as many times as necessary. To play one of the songs, just POKE the appropriate low and high byte values into locations SX and SY before calling the PLAY routine.

```
400 SYS HK:REM INSTALL SIDPLAYER
410 POKE SX,L1:POKE SY,H1:SYS PL:POKE SS,7:REM START
    SONG1
420 IF PEEK(SS)AND7 GOTO 420:REM WAIT UNTIL SONG1
    ENDS
430 POKE SX,L2:POKE SY,H2:SYS PL:POKE SS,7:REM PLAY
    SONG2
440 IF PEEK(SS)AND7 GOTO 440:REM WAIT UNTIL SONG2
    ENDS
450 SYS DP:REM REMOVE SIDPLAYER
```

Compatibility. Sidplayer uses memory from location 49152 to location 51199, and zero page addresses 251 through 255. Sidplayer should be compatible with other utilities, such as machine language routines for bitmapped graphics drawing or sprite animation, as long as they don't use these memory locations.

Since the bitmapped graphics routines published in this book use some of the same memory as Sidplayer, there would seem to be a problem. Fortunately, it's possible to relocate a portion of Sidplayer to free memory so that there's no conflict.

To load and relocate the Sidplayer machine language, calculate the LOAD address, assign it to the variable LA, and call the subroutine at 57100. Do this instead of calling the subroutine at 57000.

```
200 LA=PEEK(49)+256*PEEK(50)+1000:GOSUB 57100
```

The subroutine loads SID.OBJ into memory, then moves a portion of it to the specified address in free memory. After the relocation, only memory from 49152 to 49663 is used by Sidplayer.

The subroutine at 57100 also sets LA to the address of the first free byte after the Sidplayer code. Therefore, the LOAD address should not be changed when the song is loaded.

```
210 F$="SONG":GOSUB 57500
```

The song is now ready to be played.

If you're using bitmapped graphics shapes, the procedure is just a little more complicated. The normal procedure to load a shape file is to assign the name of the shape file to F$, calculate the LOAD address and assign it to LA, and call the routine at 56500. When using shapes with Sidplayer, the same procedure should be used, except that the value of LA should *not* be changed.

```
220 F$="SHAPES":GOSUB 56500
```

Utility Programs

Here are four utility programs to help with song debugging and music file management. Although they may look similar to those utilities offered in the bitmapped graphics and sprite sections of this book, they are for use only with Sidplayer music files.

Each program, as listed, is set to work with the disk drive. To make a program work with tape, find the statement DN=8 near the beginning of the program and change it to DN=1.

Lister. The bottom level of the editing screen can display only a few notes at a time. Sometimes it would be helpful to see more notes at one time. This would make it easier to search for a particular note or command, such as a phrase definition, and would also make it easier to find mistakes, such as a measure with the wrong number of beats.

"Lister," Program 11-4, is used to list the notes and commands of the three voices in a music file. Although the notes are not displayed in the grand staff format, the program does let you see one or two full measures at a time when the listing is shown on the screen. The listing can also be sent to a printer for permanent reference.

Tape users should remember to change DN=8 to DN=1 in line 820 when entering the program.

Program 11-4. Lister

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SIDPLAYER MUSIC FILE L
    ISTER"                                    :rem 56
110 PRINT " BY CRAIG CHAMBERLAIN":PRINT:GOTO 800
                                              :rem 76
300 K=LA+6+FNDP(LA)+FNDP(LA+2)+FNDP(LA+4):S$=" "
                                              :rem 138
310 J=PEEK(K):K=K+1:IF J AND J<>13 THEN S$=S$+CHR$
    (J):GOTO 310                              :rem 123
320 PRINT#1,S$:IF J THEN S$=" ":GOTO 310      :rem 90
330 B=LA+6:IF SV>1 THEN B=B+FNDP(LA):IF SV=3 THEN
    {SPACE}B=B+FNDP(LA+2)                     :rem 101
332 FOR V=SV-1 TO 2:PRINT#1," VOICE";V+1:MD=0
                                              :rem 20
335 T(1)=12:N=144:FOR J=2 TO 7:T(J)=N:N=N/2:NEXT
                                              :rem 196
340 FOR K=B TO B+FNDP(LA+2*V)-2 STEP 2:N=FRE(0)
                                              :rem 43
350 N=PEEK(K):N2=PEEK(K+1):J=12:ON N AND 3 GOTO 50
    0,530,690                                 :rem 164
400 T=N2 AND 7:S$="{5 SPACES}"+MID$("RCDEFGAB",T+1
    ,1)+" "                                   :rem 117
410 IF T=0 THEN S$=S$+"{4 SPACES}":GOTO 430
                                              :rem 184
420 S$=S$+MID$("S F",N2/64,1)+STR$(NOT N2/8 AND 7)
    +" "                                      :rem 164
430 T=N/4 AND 7:IF T=0 THEN PRINT#1,S$;"ABS":GOTO
    {SPACE}480                                :rem 132
440 S$=S$+MID$("UWHQEST",T,1):IF N AND 32 THEN S$=
    S$+" D":GOTO 460                          :rem 248
450 S$=S$+"{2 SPACES}"                        :rem 51
460 IF N AND 64 THEN S$=S$+" T"               :rem 209
470 PRINT#1,S$:MD=MD+T(T):IF N AND 32 THEN MD=MD+T
    (T+1)                                     :rem 57
480 NEXT K:B=B+FNDP(LA+2*V):PRINT#1,"{5 SPACES}MEA
    SURE DURATION:";MD:PRINT#1:NEXT V         :rem 34
490 CLOSE 1:END                               :rem 85
500 N=N2:IF (N AND 15)=7 THEN T=INT(N/32)*3+1:ON (
    (N AND 16)=0)+2 GOTO 570,575              :rem 185
510 J=1:IF (N AND 7)=3 THEN T=INT(N/16):PRINT#1,"
    {5 SPACES}";CN$(28+T);:GOTO 580           :rem 16
530 IF (N AND AM(J))<>CM(J) THEN J=J+1:GOTO 530
                                              :rem 206
540 PRINT#1,"{5 SPACES}";CN$(J);:IF J<7 THEN PRINT
    #1,(N AND NOT AM(J))/16;                  :rem 207
```

```
550 IF J<10 THEN PRINT#1:GOTO 480          :rem 47
560 ON J-9 GOTO 600,600,610,620,630,640,650,650,65
    0,660,660                              :rem 112
565 ON J-20 GOTO 650,650,660,680,650,675,670
                                           :rem 161
570 PRINT#1,"{5 SPACES}WAVEFORM ";MID$("N
    {2 SPACES}T{2 SPACES}S{2 SPACES}TS P{2 SPACES}
    TP SP TSP",T,3):GOTO 480               :rem 58
575 PRINT#1,"{5 SPACES}FILTER MODE ";MID$("OFFL
    {2 SPACES}B{2 SPACES}LB H{2 SPACES}LH BH LBH",
    T,3):GOTO 480                          :rem 204
580 IF T THEN PRINT#1,MID$(" NO YES",1+(N/8 AND 1)
    *3,4):GOTO 480                         :rem 59
585 PRINT#1,MID$(" UP{2 SPACES}DOWN",((N AND 8)=0)
    *4+5,5):GOTO 480                       :rem 225
600 PRINT#1,N/8 AND 15:GOTO 480            :rem 160
610 PRINT#1,N2+16*(N AND 240):GOTO 480     :rem 247
620 T=N2+8*(N AND 224):IF N AND 16 THEN T=T-2048
                                           :rem 160
625 PRINT#1,T:GOTO 480                      :rem 13
630 IF N2=0 THEN N2=256                    :rem 158
635 PRINT#1,INT(14400/N2):FOR J=2 TO 7:T(J)=N2:N2=
    N2/2:NEXT:GOTO 480                     :rem 137
640 T(1)=N2                                 :rem 45
650 PRINT#1,N2:GOTO 480                     :rem 55
660 PRINT#1,N2+256*(N2>127):GOTO 480       :rem 211
670 PRINT#1,MD:PRINT#1,N2+4*(N AND 192):MD=0:GOTO
    {SPACE}480                             :rem 224
675 PRINT#1,8*N2:GOTO 480                  :rem 160
680 IF N2 AND 1 THEN PRINT#1,-(11-INT(N2/16)+12*(N
    2/2 AND 7)):GOTO 480                    :rem 79
685 PRINT#1,INT(N2/16)+12*(7-(N2/2 AND 7)):GOTO 48
    0                                      :rem 186
690 PRINT#1,"{5 SPACES}PORTAMENTO";N2+64*(N AND 25
    2):GOTO 480                            :rem 141
800 DV=3:PRINT "{2 SPACES}OUTPUT DEVICE ("DV")";:I
    NPUT DV:IF DV<1 GOTO 800               :rem 189
805 OPEN 1,DV                              :rem 198
810 INPUT "{7 SPACES}MUSIC FILENAME";S$:IF S$="" O
    R LEN(S$)>12 GOTO 810                   :rem 93
815 SV=1:PRINT " STARTING VOICE ("SV")";:INPUT SV
                                           :rem 142
816 IF SV<1 OR SV>3 OR SV<>INT(SV) GOTO 815
                                           :rem 127
820 K=0:J=0:T=0:B=0:MD=0:SA=780:SX=781:SY=782:SP=7
    83:DN=8                                :rem 140
825 DEF FN DP(K)=PEEK(K)+256*PEEK(K+1)      :rem 63
830 DIM CN$(33),AM(27),CM(27):FOR K=1 TO 33:READ C
    N$(K):NEXT                             :rem 206
```

```
840 FOR K=1 TO 27:READ AM(K),CM(K):NEXT:LA=PEEK(49
    )+256*PEEK(50)+1000                          :rem 57
850 POKE SA,2:POKE SX,DN:POKE SY,0:SYS 65466
                                                 :rem 69
860 S$=S$+".MUS":FOR K=1 TO LEN(S$):POKE 584+K,ASC
    (MID$(S$,K)):NEXT                            :rem 9
865 POKE SA,LEN(S$):POKE SX,73:POKE SY,2:SYS 65469
                                                 :rem 157
870 POKE SA,0:POKE SX,LA-256*INT(LA/256):POKE SY,L
    A/256:SYS 65493                              :rem 242
875 PRINT:IF (PEEK(SP)AND1)=0 GOTO 300           :rem 161
880 PRINT " ERROR:";:T=PEEK(SA):IF T=4 THEN PRINT
    {SPACE}"FILE NOT FOUND":GOTO 490             :rem 112
882 IF T=5 THEN PRINT "DEVICE NOT PRESENT":GOTO 49
    0                                            :rem 201
885 PRINT ST:GOTO 490                            :rem 233
900 DATA DECAY RATE,RELEASE RATE,CALL PHRASE,DEFIN
    E PHRASE,RESONANCE                           :rem 13
902 DATA MASTER VOLUME,REPEAT TAIL,PHRASE END,HALT
    ,ATTACK RATE,SUSTAIN LEVEL                   :rem 102
910 DATA PULSE WIDTH,DETUNE ,TEMPO,UTILITY DURATIO
    N,RELEASE POINT                              :rem 220
912 DATA REPEAT HEAD,FLAG,PULSE WIDTH SWEEP ,"FILT
    ER CUTOFF SWEEP "                            :rem 9
914 DATA VIBRATO DEPTH,VIBRATO RATE,AUTOFILTER ,TR
    ANSPOSE ,AUXILIARY                           :rem 173
916 DATA FILTER CUTOFF,"MEASURE DURATION:":rem 105
920 DATA BUMP VOLUME,FILTER,RING MODULATION,SYNC M
    ODE,EXTERNAL FILTER,3OFF                     :rem 40
930 DATA 15,0,15,8,15,2,15,6,15,10,15,14  :rem 148
932 DATA 255,15,255,47,255,79                :rem 169
934 DATA 135,4,135,132                       :rem 58
940 DATA 15,2,15,10,255,6,255,22,255,38,255,54,255
    ,70,255,86,255,102                           :rem 28
942 DATA 255,118,255,134,255,150,255,166,255,182,3
    1,14,63,30                                  :rem 150
```

Once run, the program first asks for an output device number. Enter 3 for the screen, or 4 for the printer. For now, just press the RETURN key to choose 3, the default value displayed in parentheses.

The next prompt asks for the name of the file to be listed. As always, do not include the .MUS extension as part of the filename. A good song to examine for purposes of explanation is COMMODORE.MUS, Program 7-4.

Lister then requests a starting voice number. Usually a listing begins with voice 1, but sometimes you may want to

skip voice 1 or voice 2. Enter the number of the first voice you want to list, or just press the RETURN key to start at voice 1. The program loads the requested file and begins the listing.

Each note is listed by its name, accidental, octave number, and duration.

C S 4 Q (C sharp, octave 4, quarter note)

The first letter specifies the note name, and can be a letter *A* through *G* or the letter *R* for rest. If the note is sharp or flat, S or F is displayed after the note name. The octave number, 0–7, comes next, followed by a letter for the duration. The duration can be W, H, Q, E, S, T, or U. Respectively, these letters stand for whole, half, quarter, eighth, sixteenth, and thirty-second notes. U represents the utility duration.

If the note is dotted, a D shows after the duration. If the note is tied, the note listing ends with a T.

C S 4 Q D T (C sharp, octave 4, quarter note, dotted, and tied)

Commands are listed with their full names. If a command has a data value, such as a number or yes/no indication, the value is placed after the command name.

TEMPO 90
WAVEFORM T

Measure numbers are displayed at the left edge of the listing, separate from the notes and commands. This makes it easy to find a particular measure. At the end of each measure, the total jiffy count for that measure is printed. This number is affected by the tempo and the time signature. For example, COMMODORE.MUS is written in 4/4 time and is played in M.M. 90. If you refer to Table 9-1, four beats at tempo 90 give a total of 160 jiffies. That's why the measure duration for each measure in COMMODORE.MUS is 160 jiffies.

If you're listing a song and one of the measures has a different jiffy count from the others, it could be an indication that the measure doesn't have the correct total number of beats. Either a duration is wrong, a note is missing, or there's an extra note.

The measure duration feature won't work properly when repeats or phrases are used. Also, the total number of jiffies for a measure may change from one voice to another if the TEM command is not used on each voice.

If the listing goes by too fast, you can make the screen scroll slower by pressing the CTRL key. If that's still too fast, or if you want to stop the listing, just hit the RUN/STOP key. Enter CONT to make the listing continue.

To send the listing to the printer, enter 4 in response to the OUTPUT DEVICE prompt. Make sure that the printer is connected and turned on.

If you have an RS-232 printer instead of a parallel printer, you may have to change the OPEN statement in line 805 of the Lister.

Merge. The Editor has a limited amount of memory available for editing a song. Some songs may be just too long for it to handle. The solution is to edit the song in sections, and then merge the sections together.

"Merge," Program 11-5, is used to combine two or more music files into one larger file. This file can then be played by the Player. Another use of Merge is to copy a music file from one tape to another tape or from one disk to another disk. The program can also be used to transfer a song from tape to disk.

Tape users must change line 120 so that DN=1.

Program 11-5. Merge

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SIDPLAYER MUSIC FILE M
    ERGE UTILITY"                                  :rem 9
110 PRINT " BY CRAIG CHAMBERLAIN":PRINT     :rem 65
120 DN=8:SA=780:SX=781:SY=782:SP=783:TA=0:TB=0:DIM
     LA(3),LB(3)                                   :rem 98
130 DEF FNH(N)=INT(N/256):DEF FNL(N)=N-256*INT(N/2
    56)                                           :rem 206
140 DEF FNDP(N)=PEEK(N)+256*PEEK(N+1)       :rem 62
150 FOR K=49152 TO 49313:READ N:POKE K,N:NEXT:MB=4
    9152:BG=49263                                 :rem 15
160 S=0:D=0:L=0:BA=FNDP(49)+100:MT=FNDP(51)-100
                                                  :rem 134
300 F$="":INPUT " LOAD FILENAME";F$:IF LEN(F$)>12
    {SPACE}GOTO 300                               :rem 98
310 GOSUB 600:POKE SA,0:POKE SX,FNL(BA):POKE SY,FN
    H(BA)                                         :rem 232
320 SYS 65493:IF PEEK(SP)AND1 GOTO 700      :rem 85
330 FOR V=1 TO 3:LA(V)=FNDP(BA+2*V-2):TA=TA+LA(V):
    NEXT:A=FNDP(SX)                               :rem 79
400 F$="":INPUT " APPEND FILENAME";F$:IF LEN(F$)>1
    2 GOTO 400                                    :rem 252
410 PRINT:IF F$="" GOTO 500                       :rem 164
```

```
420 F$=F$+".MUS":OPEN 1,DN,0,F$:GET#1,L$,H$:rem 40
430 FOR V=1 TO 3:GET#1,L$,H$:LB(V)=ASC(L$+CHR$(0))
    +256*ASC(H$+CHR$(0))                    :rem 222
435 TA=TA+LB(V)-2:NEXT                      :rem 59
440 IF TA>MT-BA THEN PRINT " ERROR: NOT ENOUGH MEM
    ORY":CLOSE 1:END                        :rem 145
450 A=BA+6:S=A+LA(1):L=LA(2)+LA(3)-2:D=MT-L:GOSUB
    {SPACE}630:A=A+LA(1)-2:S=A:L=LB(1)      :rem 70
460 GOSUB 660:A=A+L:L=LA(2)-2:S=MT-LA(3)-L:D=A:GOS
    UB 630:A=A+L:S=A:L=LB(2)                :rem 24
470 GOSUB 660:A=A+L:L=LA(3)-2:S=MT-L:D=A:GOSUB 630
    :A=A+L:S=A:L=LB(3):GOSUB 660            :rem 51
480 A=A+L:IF ST GOTO 520                    :rem 18
490 K=FRE(0):GET#1,F$:IF F$<>"" THEN POKE A,ASC(F$
    ):A=A+1:GOTO 490                        :rem 58
495 CLOSE 1:POKE A,0:A=A+1:FOR V=1 TO 3:LA(V)=LA(V
    )+LB(V)-2:NEXT:GOTO 400                 :rem 125
500 FOR V=1 TO 3:POKE BA+2*V-2,FNL(LA(V)):POKE BA+
    2*V-1,FNH(LA(V)):NEXT                   :rem 252
510 F$="":INPUT " SAVE FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 510                        :rem 3
520 GOSUB 600:POKE SA,251:POKE 251,FNL(BA):POKE 25
    2,FNH(BA)                               :rem 45
530 POKE SX,FNL(A):POKE SY,FNH(A):SYS 65496:IF PEE
    K(SP)AND1 GOTO 700                      :rem 188
540 PRINT " SAVED" A-BA "BYTES":END         :rem 170
600 PRINT:POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466
    :F$=F$+".MUS"                           :rem 225
610 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                    :rem 241
620 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                 :rem 159
630 POKE 251,FNL(S):POKE 252,FNH(S):POKE 253,FNL(D
    ):POKE 254,FNH(D)                        :rem 3
640 POKE SX,FNL(L):POKE SY,FNH(L):SYS MB:RETURN
                                            :rem 185
660 POKE 251,FNL(S):POKE 252,FNH(S):POKE 253,FNL(L
    ):POKE 254,FNH(L)                       :rem 22
670 SYS BG:IF PEEK(SP)AND1 THEN CLOSE 1:GOTO 700
                                            :rem 235
680 RETURN                                  :rem 126
700 PRINT " ERROR":END                       :rem 3
800 DATA 142,109,192,140,110,192,165,253,56,229,25
    1,170,165,254,229,252,236,109           :rem 62
801 DATA 192,237,110,192,144,35,160,0,174,110,192,
    240,14,177,251,145,253,200              :rem 143
802 DATA 208,249,230,252,230,254,202,208,242,174,1
    09,192,240,8,177,251,145,253             :rem 8
803 DATA 200,202,208,248,96,173,110,192,168,101,25
    2,133,252,152,24,101,254,133            :rem 243
```

```
804 DATA 254,172,109,192,240,9,136,177,251,145,253
    ,192,0,208,247,174,110,192              :rem 174
805 DATA 240,16,198,252,198,254,136,177,251,145,25
    3,192,0,208,247,202,208,240             :rem 228
806 DATA 96,0,0,162,1,32,198,255,176,43,165,251,16
    8,101,253,133,253,165,252,101            :rem 52
807 DATA 254,133,254,169,0,133,251,32,207,255,176,
    21,145,251,200,208,2,230,252            :rem 246
808 DATA 196,253,208,240,165,252,197,254,208,234,3
    2,204,255,24,96                         :rem 156
```

The program begins by asking for a file to load. After you enter the filename and the song is loaded, the program requests the name of the file to be appended to the first file. When you enter this second filename, the program reads each voice in the file and adds it to the end of each voice in the first song.

The program then asks for the name of another file to be appended. You can append as many files as you like. When you don't want to append any more files, just press RETURN. The program finally asks for a filename to use in saving the composite file.

To copy a music file from one tape to another tape or from one disk to another disk, use Merge to load the file you want to move, do not append any other files, and then save the file to the new tape or disk. This is more convenient than using the Editor to copy files.

With a slight modification, the Merge program can be used to transfer a music file from tape to disk. Change the DN=8 in line 120 to DN=1, and insert the statement DN=8 at the beginning of line 520, right before the GOSUB 600. Now when you load a music file, it will come from the Datassette, but when you save the file again, it will go to the disk drive.

Extract. "Extract," Program 11-6, loads a music file, asks for starting and ending measure numbers, extracts the measures in the specified range, and saves them as a new file. The original file retains all its measures. Extract does not remove measures from the original—it only pulls out specified measures and places them in a new file. (Change DN to equal 1 in line 120 if you are using tape.)

Program 11-6. Extract

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SIDPLAYER":PRINT " MUS
    IC FILE EXTRACTION UTILITY"              :rem 165
110 PRINT " BY CRAIG CHAMBERLAIN":PRINT      :rem 65
120 DN=8:SA=780:SX=781:SY=782:SP=783:DIM LN(3)
                                             :rem 184
130 DEF FNH(N)=INT(N/256):DEF FNL(N)=N-256*INT(N/2
    56)                                      :rem 206
140 DEF FNDP(N)=PEEK(N)+256*PEEK(N+1):BA=FNDP(49)+
    500:PT=BA+6:DP=PT                        :rem 140
300 F$="":INPUT " LOAD FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 300                        :rem 238
310 GOSUB 600:POKE SA,0:POKE SX,FNL(BA):POKE SY,FN
    H(BA)                                    :rem 232
320 SYS 65493:IF PEEK(SP)AND1 GOTO 700       :rem 85
330 FOR V=1 TO 3:LN(V)=FNDP(BA+2*V-2):NEXT:rem 128
350 SM=-1:INPUT " STARTING MEASURE";SM:IF SM<-1 OR
     SM>1023 GOTO 350                        :rem 83
360 EM=0:INPUT "{3 SPACES}ENDING MEASURE";EM:IF EM
    <0 OR EM>1023 GOTO 360                   :rem 10
370 PRINT:SL=SM AND255:SH=(SM/4 AND192)+30:EL=EM A
    ND255:EH=(EM/4 AND192)+30                :rem 255
400 FOR V=1 TO 3:PRINT " PROCESSING VOICE";V:DB=DP
    :MP=PT+LN(V):IF SM<0 GOTO 450            :rem 128
410 IF PEEK(PT)=SH THEN IF PEEK(PT+1)=SL GOTO 450
                                             :rem 248
420 PT=PT+2:IF PT<MP GOTO 410                :rem 140
430 PRINT " ERROR:STARTING MEASURE NOT FOUND":END
                                             :rem 40
450 P=PEEK(PT):Q=PEEK(PT+1):IF P=EH THEN IF Q=EL G
    OTO 480                                  :rem 19
460 POKE DP,P:POKE DP+1,Q:DP=DP+2:PT=PT+2:IF PT<MP
     GOTO 450                                :rem 223
470 PRINT " NOTE:RAN TO END OF VOICE":GOTO 490
                                             :rem 82
480 POKE DP,1:POKE DP+1,79:DP=DP+2           :rem 173
490 POKE BA+2*V-2,FNL(DP-DB):POKE BA+2*V-1,FNH(DP-
    DB):PT=MP:NEXT:PRINT                     :rem 238
500 P=PEEK(PT):PT=PT+1:POKE DP,P:DP=DP+1:IF P GOTO
     500                                     :rem 178
510 F$="":INPUT " SAVE FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 510                         :rem 3
520 GOSUB 600:POKE SA,251:POKE 251,FNL(BA):POKE 25
    2,FNH(BA)                                :rem 45
530 POKE SX,FNL(DP):POKE SY,FNH(DP):SYS 65496:IF P
    EEK(SP)AND1 GOTO 700                     :rem 98
540 CLR:END                                  :rem 139
```

```
600 PRINT:POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466
    :F$=F$+".MUS"                              :rem 225
610 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                       :rem 241
620 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                    :rem 159
700 PRINT " ERROR":END                           :rem 3
```

When you run the program, the first prompt asks for the name of the file to load. After you enter the filename, the program loads the song.

The program requests a starting measure number. Type the number of the first measure that should be extracted. If you want the extraction to start at the beginning of the file instead of at a later measure, just press RETURN.

When prompted for the ending measure number, type the number of the measure at which the extraction should stop. For example, if you want to extract measures 26 through 50 inclusive, the ending measure should be 51. Press RETURN by itself if you want the extraction to go to the end of the voice.

The program displays the message PROCESSING VOICE 1 while it searches for the starting measure marker. If the measure marker is not found in the voice, the program stops with the message ERROR:STARTING MEASURE NOT FOUND.

Having found the starting measure marker, the program continues to search until it comes to the ending measure marker or the end of the voice. If the searching reaches the end of the voice, the program prints the warning NOTE:RAN TO END OF VOICE.

The program repeats this procedure for voices 2 and 3.

Finally, the program asks for the filename to use in saving the new music file.

One use of the Extract program is to break a large music file into parts. Another application is to copy a sequence of notes in a song to the end of the song. For example, a song may repeat a chorus, but the chorus might be played a little differently the second time. Repeats or phrases cannot be used if there are minor changes when the notes are repeated. Instead of entering all the notes in the chorus again, just extract the notes which form the chorus, use Merge to merge them to the end of the song, and make the necessary changes.

Cross File Merge. The last utility, Program 11-7, is a cross file merger. This program lets you construct a new music file from the voices of different music files. Voice 1 of the new file can come from one file, voice 2 can come from another file, and voice 3 can come from yet another. (Tape users need to change DN=8 in line 120 to DN=1.)

Program 11-7. Cross File Merge

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SIDPLAYER CROSS FILE M
    ERGE UTILITY"                                  :rem 18
110 PRINT " BY CRAIG CHAMBERLAIN":PRINT            :rem 65
120 DN=8:SA=780:SX=781:SY=782:SP=783:TA=0:TB=0:DIM
     LA(3),LB(3):Z$=CHR$(0)                        :rem 217
130 DEF FNH(N)=INT(N/256):DEF FNL(N)=N-256*INT(N/2
    56)                                            :rem 206
140 DEF FNDP(N)=PEEK(N)+256*PEEK(N+1):BA=FNDP(49)+
    100:A=BA+6                                     :rem 118
300 FOR V=1 TO 3                                    :rem 20
310 F$="":PRINT " FILE FOR VOICE";V;:INPUT F$:IF F
    $="" OR LEN(F$)>12 GOTO 310                    :rem 100
320 VN=0:INPUT "{2 SPACES}USE WHICH VOICE ";VN:IF
    {SPACE}VN<>INT(VN) OR VN<1 OR VN>3 GOTO 320
                                                   :rem 102
330 F$=F$+".MUS":OPEN 1,DN,0,F$:GET#1,L$,H$:PRINT
                                                   :rem 239
340 FOR K=1 TO 3:GET#1,L$,H$:LB(K)=ASC(L$+Z$)+256*
    ASC(H$+Z$):NEXT                                :rem 57
350 FOR K=1 TO 3:FOR J=1 TO LB(K)/2:N=FRE(0):GET#1
    ,L$,H$                                         :rem 218
360 IF K=VN THEN POKE A,ASC(L$+Z$):POKE A+1,ASC(H$
    +Z$):A=A+2                                     :rem 37
370 NEXT:NEXT:LA(V)=LB(VN):CLOSE 1:NEXT:POKE A,0:A
    =A+1                                           :rem 53
500 FOR V=1 TO 3:POKE BA+2*V-2,FNL(LA(V)):POKE BA+
    2*V-1,FNH(LA(V)):NEXT                          :rem 252
510 F$="":INPUT " SAVE FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 510                               :rem 3
520 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=F$
    +".MUS"                                        :rem 27
530 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                           :rem 242
540 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
                                                   :rem 134
550 POKE SA,251:POKE 251,FNL(BA):POKE 252,FNH(BA)
                                                   :rem 224
```

```
560 POKE SX,FNL(A):POKE SY,FNH(A):SYS 65496:IF PEE
   K(SP)AND1 GOTO 700                          :rem 191
570 CLR:END                                    :rem 142
700 PRINT " ERROR":END                           :rem 3
```

The program begins by constructing voice 1. It asks for the name of the file which contains the voice to be used for voice 1. The program then asks which of the three voices in that file is to be used. For instance, if the first voice of the file being created is to come from voice 3 of the song COMMODORE.MUS, here's how you would respond:

FILE FOR VOICE 1 ? COMMODORE
USE WHICH VOICE ? 3

The program loads the requested voice from the specified file. This may take a while. The procedure is then repeated for voices 2 and 3.

After all the voices have been loaded, the program requires one more filename to use when saving the new file.

The "Cross File Merge" program can be used to change the order of the voices in a file. This is sometimes necessary if you start a song and later decide to use the synchronization mode or ring modulation. As an example, let's say that you want to switch voices 2 and 3 in the song "MINUET." Here's how you would reply to the program prompts:

FILE FOR VOICE 1 ? MINUET
USE WHICH VOICE ? 1

FILE FOR VOICE 2 ? MINUET
USE WHICH VOICE ? 3

FILE FOR VOICE 3 ? MINUET
USE WHICH VOICE ? 2

If you're using disk, remember to choose a filename other than MINUET when saving the new file to avoid the FILE EXISTS error.

Sidplayer Editor Command Index

SYS Calls

DP (DROP routine)	354
HK (HOOK routine)	353
PL (PLAY routine)	353–54

Sidplayer Editor commands

ABS SET (absolute set pitch) 342–43
ATK (attack rate) 283–84
AUT (autofilter mode) 293–94
BMP (bump the master volume) 306–307
CAL (call a phrase) 301–302
DCY (decay rate) 284
DEF (define a phrase) 300–301
DTN (detune a voice) 320
END (end definition of a phrase) 300–301
F-C (filter cutoff) 292
FLG (flag location) 355–56
FLT (voice pass through filter) 293
F-M (filter mode) 290, 292
F-S (filter sweep) 294
HED (repeat head) 299–300
HLT (halt a voice) 362
MS# (measures) 258–59
N (key signature level) 310
PNT (release point) 284–87
POR (controls glide rate) 315
P-S (pulse width sweeping) 274–75
P-W (pulse width) 273–74
RES (resonance level) 292–93
RLS (release rate) 284
RNG (ring modulation) 333–37
SNC (synchronization mode) 324–25
SUS (sustain level) 284
TAL (repeat tail) 299–300
TEM (tempo) 256–58
TPS (transpose a voice) 323
UTL (utility duration) 312–13
VDP (vibrato depth) 315
VOL (volume) 258
VRT (vibrato rate) 320
WAV (waveform) 273

Part 4
Sprite Control System

Chapter 12

Introduction to the Sprite Control System

So far, you've seen utilities that let you access the Commodore 64's excellent bitmapping and sound capabilities. There's one more area to cover. One of the most widely used features of your computer is sprites, the movable object blocks of the 64. Games especially make wide use of sprites.

This last part of this book deals with sprites and sprite animation. In the pages which follow, you'll see how easy it can be to define a sprite and make it move across the screen.

Just as you were provided with utilities to help you use the bitmap screen and create sophisticated sound, so you'll have a Sprite Editor at your disposal. The Editor lets you use the joystick to create sprite definitions. You'll also find several machine language routines you can use to make it easier to move the sprites you create. You'll be able to get up to eight sprites moving at the same time, each with its own shape, color, size, direction, and speed. Best of all, you'll be able to do all of this from a BASIC program, without using a single POKE!

Before you see how to define and animate sprites, however, it's important to understand the basic characteristics of animation, and the advantages of sprite graphics over character and bitmapped graphics.

Principles of Animation

There are two ways in which an object can be animated. The object can change its position or it can change its appearance.

Motion, which occurs when the position of an object is changing, is probably the more common type of animation. Two important characteristics of a moving object are its direction and its speed.

But an object can be animated even when it's not moving, simply by changing its appearance in some way. This can be done by altering the shape of the object, but it can also be done by changing its color or size.

To create effective animation on the Commodore 64, you

need a graphics system in which an object's position *or* appearance can be changed quickly and easily.

Character Graphics

There are many reasons why it's preferable not to use character graphics for animation. New character shapes can be created by redefining the character set, but each character is always limited to an 8 × 8 grid. This size is often too small to be useful.

Another problem is in motion. A character can be moved to another place on the screen by just a couple of POKE statements, but the positioning is limited to 40 columns × 25 rows, so motion appears jerky.

Bigger shapes can be created by using several characters, but motion is then more difficult because *each* character comprising the large figure has to be moved.

Thus, for many applications, character graphics are simply unsuitable for animation.

Bitmapped Graphics

With bitmapped graphics, there are no size or positioning restrictions. A shape of any size can be drawn, starting at any of 64,000 (320 × 200) positions. Sometimes these shapes can be redrawn fast enough so that they appear to change shape. (Refer back to the shape demonstrations in Part 2 for a few examples.)

The only drawback to using bitmapped graphics is that motion is impossible in BASIC. Moving an object means erasing each point in the shape and replotting it at an adjacent position. Bitmapped graphics shapes usually contain too many points to be erased and redrawn quickly. If a shape is moved like this, it moves very slowly, with considerable flicker.

These problems are magnified when you want the shape to move without disturbing other things already on the screen. For instance, you may want to make a figure walk in front of a building. As the points in the figure are plotted in new positions, they may be placed in the same positions as points used to display the building. Those points must be restored after the figure has walked past. Motion is not as simple as just plotting and erasing points. For each point, the original contents must be remembered before the position is plotted so that the contents can later be restored.

Although bitmapped graphics has many applications (just some of which you saw in Part 2), animation is not one of them.

Sprite Graphics

This graphics system combines the best features of character and bitmapped graphics. A sprite is a super character, because it can be defined on a grid of 24 × 21 points, about the size of eight characters. This size is large enough to create detailed shapes. Positioning is allowed on any of the 320 × 200 points, so a sprite can be placed anywhere on the screen. It also means sprite motion is smooth.

The best feature, however, is that the sprite definition is stored in memory separate from screen memory. This means you can use sprites at the same time as character graphics or bitmapped graphics. Also, the screen memory is not changed when the sprite is moved, so the background does not have to be restored.

The fact that a sprite is completely independent of the screen greatly simplifies its display and movement. Only three POKE statements are needed to move a whole sprite. Just one changes its shape, size, or color.

If this were all the Commodore 64 did, it would be an excellent system. What makes it even better is that the 64 supports up to eight sprites. Each sprite can have its own shape, color, size, and position. Other features, such as multicolor mode and priority, will be discussed later.

The Sprite Control System

Sprites are well suited for animation, but there are a couple of problems. One is that it's difficult to create a sprite definition. The other is that the three POKE statements required to position a sprite are rather complicated and take time to execute. BASIC programs run too slowly to make a sprite move very quickly. And making eight sprites move at once is simply not feasible in BASIC.

That's where the Sprite Control System comes in. The Control System consists of a group of programs and machine language routines which make sprite animation easier than you ever thought possible.

Sprite Editor. The first part of the Control System is a Sprite Editor. Using the joystick, you plot points on a 24 × 21

grid to define a shape. A variety of editing features are available to scroll the shape, flip it to create a mirror image, turn it upside down, and so on. If necessary, over 100 shapes can be defined. The definitions can be saved as a disk or tape file, or can be converted into DATA statements and merged with a BASIC program. You'll never have to do any calculations or work with bits and bytes.

Machine language routines. The second part of the Control System is a set of machine language routines. These routines can be called from BASIC to make a sprite change its shape, color, size, or position. The routines can also make a sprite move in a given direction at a specified speed. There are even advanced features to make a sprite move under joystick control or change shape automatically whenever it changes direction. All of this is done without a single POKE statement.

Before You See It Work

There are three impressive demonstration programs included here, but before you can enter them, you have to prepare two other files. Type in the following lines and save them with the filename SCS.BAS. It's important that you use this filename. We'll be referring to this filename throughout the next few chapters.

Program 12-1. SCS.BAS

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
58000 POKE SA,1:POKE SX,DN:POKE SY,1:SYS 65466:F$=
      "SCS.OBJ":GOSUB 59000                  :rem 116
58010 POKE SA,0:SYS 65493:IF PEEK(SP)AND1 GOTO 591
      00                                     :rem 127
58020 IN=49664:RE=49667:RL=49670:RS=49673:DF=49676
      :BL=49679:CO=49682:WI=49685            :rem 172
58030 HE=49688:PO=49691:DI=49694:WR=49697:BO=49700
      :MO=49703:LB=49706:RB=49709            :rem 148
58040 TB=49712:BB=49715:FR=49718:UN=49721:PR=49724
      :MU=49727:EN=49730:JS=49733            :rem 149
58050 AU=49736:CH=49739:RETURN                :rem 14
58100 GOSUB 58000:SYS 51123,LA:LA=LA+1191:RETURN
                                             :rem 11
58400 POKE 49749,LA-256*INT(LA/256):POKE 49750,INT
      (LA/256)                               :rem 182
58410 READ J:POKE 49751,J:FOR K=0 TO J:READ J:POKE
       LA,J:LA=LA+1                          :rem 16
58420 IF J THEN FOR LA=LA TO LA+J-1:READ I:POKE LA
      ,I:NEXT LA                             :rem 92
```

```
58430 NEXT K:RETURN                                  :rem 168
58500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".DEF":GOSUB 59000                          :rem 24
58510 POKE SA,0:POKE SX,LA-256*INT(LA/256):POKE SY
      ,INT(LA/256)                                   :rem 78
58520 SYS 65493:IF PEEK(SP)AND1 GOTO 59100 :rem 44
58530 LA=LA+1:POKE 49749,LA-256*INT(LA/256):POKE 4
      9750,INT(LA/256)                              :rem 167
58540 POKE 49751,PEEK(LA-1):LA=PEEK(SX)+256*PEEK(S
      Y):RETURN                                      :rem 31
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                                         :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                                       :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END                       :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                     :rem 36
59120 PRINT ST:END                                   :rem 70
```

These lines *do not* form a complete program. Rather, they're subroutines which must be included in every program that uses the Sprite Control System.

The next file contains the machine language routines used to display sprites and make them move. As with all the complete machine language programs in this book, you have to use "The Machine Language Editor: MLX," found in Appendix C, to type it in. Be sure to read the accompanying article if you haven't already. If you have a copy of MLX from another COMPUTE! publication, you *can* use it to enter the listings in this book. However, if you use tape and have another version of MLX, you must change line 763 to match what's listed in this book. (Tape users who have a copy of MLX which does not have a line 763 *must* use this book's version of MLX.)

Once you've loaded and run MLX, you'll be asked to make two responses. Here are the numbers you should enter:

Starting Address: 49664
Ending Address: 51199
Filename: SCS.OBJ

Begin typing in Program 12-2. Follow the instructions in Appendix C if you enter the program in more than one sitting. When you're finished, save this file using the filename SCS.OBJ. If you're saving to tape, put it on a cassette all by itself so that you can easily find it when you need it. A bit later, you'll have to load it into memory to see the demonstration.

Program 12-2. SCS.OBJ

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

```
49664 :076,012,195,076,177,195,219
49670 :076,100,199,076,157,199,045
49676 :076,210,195,076,052,196,049
49682 :076,073,196,076,097,196,220
49688 :076,100,196,076,134,196,034
49694 :076,007,197,076,061,197,132
49700 :076,064,197,076,025,197,159
49706 :076,085,197,076,088,197,249
49712 :076,119,197,076,122,197,067
49718 :076,188,196,076,200,196,218
49724 :076,103,196,076,106,196,045
49730 :076,109,196,076,210,196,161
49736 :076,146,197,076,253,196,248
49742 :076,173,197,000,000,007,019
49748 :000,000,000,000,000,000,084
49754 :000,000,000,000,000,000,090
49760 :000,001,002,004,008,016,127
49766 :032,064,128,254,253,251,060
49772 :247,239,223,191,127,000,111
49778 :009,008,010,002,006,004,153
49784 :005,001,012,000,003,015,156
49790 :004,008,001,002,005,009,155
49796 :006,010,002,004,002,008,164
49802 :000,002,006,007,008,187,092
49808 :194,186,194,032,241,183,150
49814 :224,008,176,003,134,002,185
49820 :096,076,072,178,032,147,245
49826 :194,032,241,183,138,166,092
49832 :002,096,032,253,174,032,245
49838 :138,173,032,155,188,165,001
49844 :101,164,100,166,002,096,041
49850 :064,169,015,141,025,208,040
49856 :173,089,194,048,007,169,104
49862 :000,141,088,194,240,061,154
49868 :088,216,173,001,220,073,207
49874 :031,141,090,194,173,000,071
49880 :220,073,031,141,091,194,198
49886 :174,092,194,240,014,173,085
49892 :030,208,141,093,194,173,043
49898 :031,208,141,094,194,162,040
49904 :000,173,088,194,061,097,085
49910 :194,240,006,142,089,194,087
49916 :032,078,194,232,224,008,252
49922 :208,237,169,255,141,089,077
49928 :194,108,081,194,169,194,180
49934 :205,021,003,208,001,096,036
49940 :169,000,141,021,208,141,188
49946 :023,208,141,027,208,141,006
49952 :028,208,141,029,208,141,019
49958 :016,208,160,016,169,050,145
49964 :136,153,000,208,169,024,222
49970 :136,153,000,208,208,242,229
49976 :162,007,189,224,236,157,007
49982 :039,208,169,000,157,060,183
49988 :003,157,068,003,157,116,060
49994 :003,169,017,157,076,003,243
50000 :169,012,157,084,003,169,162
50006 :159,157,092,003,169,050,204
```

```
50012 :157,100,003,169,228,157,138
50018 :108,003,202,016,211,142,012
50024 :088,194,142,089,194,169,212
50030 :004,141,037,208,169,000,157
50036 :141,038,208,173,030,208,146
50042 :173,031,208,169,127,141,203
50048 :013,220,162,001,189,020,221
50054 :003,157,081,194,189,143,133
50060 :194,157,020,003,189,145,080
50066 :194,157,250,255,202,016,196
50072 :235,169,249,141,018,208,148
50078 :173,017,208,041,127,141,097
50084 :017,208,169,015,141,025,227
50090 :208,169,001,141,026,208,155
50096 :096,169,194,205,021,003,096
50102 :208,025,169,000,141,026,239
50108 :208,141,021,208,173,081,252
50114 :194,141,020,003,173,082,039
50120 :194,141,021,003,169,129,089
50126 :141,013,220,096,173,086,167
50132 :194,240,014,032,163,194,025
50138 :072,032,241,183,236,087,045
50144 :194,144,005,240,003,076,118
50150 :072,178,104,160,000,132,108
50156 :099,074,102,099,074,102,018
50162 :099,013,084,194,133,100,097
50168 :173,085,194,133,097,173,079
50174 :086,194,133,098,232,202,175
50180 :240,013,177,097,056,101,176
50186 :097,133,097,144,244,230,187
50192 :098,208,240,165,001,072,032
```

```
50198 :041,252,120,133,001,138,195
50204 :160,063,136,145,099,208,071
50210 :251,177,097,240,008,168,207
50216 :177,097,136,145,099,208,134
50222 :249,104,133,001,088,096,205
50228 :032,160,194,160,248,132,210
50234 :099,172,083,194,132,100,070
50240 :164,002,145,099,160,000,122
50246 :056,176,046,032,160,194,222
50252 :157,039,208,032,121,000,121
50258 :240,012,032,241,183,142,164
50264 :037,208,032,241,183,142,163
50270 :038,208,096,160,008,044,136
50276 :160,002,044,160,006,044,004
50282 :160,007,044,160,000,132,097
50288 :020,032,160,194,164,020,190
50294 :074,185,021,208,061,105,004
50300 :194,144,003,029,097,194,017
50306 :153,021,208,096,032,147,019
50312 :194,032,170,194,133,020,111
50318 :132,021,032,170,194,072,251
50324 :138,010,168,104,024,105,185
50330 :050,120,153,001,208,165,083
50336 :020,024,105,024,153,000,230
50342 :208,165,021,105,000,074,227
50348 :173,016,208,061,105,194,161
50354 :144,003,029,097,194,141,018
50360 :016,208,088,096,032,163,019
50366 :194,073,255,045,088,194,015
50372 :141,088,194,096,032,163,142
50378 :194,013,088,194,141,088,152
```

```
50384 :194,096,032,160,194,168,028
50390 :240,022,201,003,176,058,146
50396 :009,001,133,020,032,121,024
50402 :000,240,011,032,170,194,105
50408 :041,003,010,010,005,020,065
50414 :133,020,166,002,189,060,040
50420 :003,041,224,005,020,157,182
50426 :060,003,096,032,160,194,027
50432 :041,007,056,042,042,208,140
50438 :231,032,160,194,201,009,065
50444 :176,008,168,185,113,194,088
50450 :157,068,003,096,076,072,234
50456 :178,032,160,194,041,007,124
50462 :010,010,010,133,020,032,245
50468 :163,194,041,007,005,020,210
50474 :157,076,003,032,121,000,175
50480 :240,010,032,163,194,030,205
50486 :076,003,074,126,076,003,156
50492 :096,169,064,044,169,128,218
50498 :133,020,032,160,194,074,167
50504 :189,060,003,041,063,144,060
50510 :002,005,020,157,060,003,069
50516 :096,169,000,044,169,008,058
50522 :133,020,032,147,194,032,136
50528 :170,194,024,105,024,072,173
50534 :152,105,000,074,104,106,131
50540 :072,138,024,101,020,168,119
50546 :104,153,084,003,096,169,211
50552 :000,044,169,008,133,020,238
50558 :032,147,194,032,170,194,127
50564 :024,105,050,072,138,024,033
```



```
50570 :101,020,168,104,153,100,016
50576 :003,096,032,160,194,168,029
50582 :240,017,136,136,152,041,104
50588 :007,009,128,157,116,003,064
50594 :032,163,194,157,124,003,067
50600 :096,157,116,003,096,189,057
50606 :076,003,016,005,165,162,089
50612 :074,176,245,189,060,003,159
50618 :133,251,041,003,240,117,203
50624 :074,144,033,168,185,090,118
50630 :194,041,015,168,240,015,103
50636 :165,251,041,004,240,006,143
50642 :152,074,144,253,208,093,110
50648 :152,208,087,165,251,041,096
50654 :008,208,084,152,240,078,224
50660 :165,251,041,028,074,168,187
50666 :185,000,208,133,252,185,173
50672 :001,208,133,253,152,074,037
50678 :168,173,016,208,057,097,197
50684 :194,240,001,056,102,252,073
50690 :138,010,168,169,000,133,108
50696 :254,173,016,208,061,097,049
50702 :194,240,001,056,185,000,178
50708 :208,106,197,252,240,007,006
50714 :169,004,176,001,010,133,007
50720 :254,185,001,208,197,253,106
50726 :240,008,169,001,176,001,121
50732 :010,005,254,044,165,254,008
50738 :157,068,003,138,010,168,082
50744 :189,068,003,240,071,041,156
50750 :003,240,062,074,189,076,194
```

```
50756 :003,041,007,133,252,185,177
50762 :001,208,144,017,229,252,157
50768 :221,100,003,176,039,165,016
50774 :251,010,016,019,189,108,167
50780 :003,144,029,101,252,221,074
50786 :108,003,144,022,240,020,123
50792 :165,251,010,048,012,144,222
50798 :016,189,068,003,073,003,206
50804 :157,068,003,208,006,189,235
50810 :100,003,153,001,208,189,008
50816 :068,003,041,012,240,066,046
50822 :189,076,003,074,074,074,112
50828 :041,007,133,252,185,000,246
50834 :208,133,253,173,016,208,113
50840 :061,097,194,240,002,169,147
50846 :001,133,254,189,068,003,038
50852 :074,074,074,165,253,144,180
50858 :031,229,252,133,253,165,209
50864 :254,233,000,133,254,074,100
50870 :165,253,106,221,084,003,246
50876 :176,061,165,251,010,016,099
50882 :033,189,092,003,144,043,186
50888 :240,070,101,252,133,253,225
50894 :165,254,105,000,133,254,093
50900 :074,165,253,106,221,092,099
50906 :003,144,030,240,028,165,060
50912 :251,010,048,012,144,042,219
50918 :189,068,003,073,012,157,220
50924 :068,003,208,032,189,084,052
50930 :003,010,133,253,169,000,042
50936 :042,133,254,165,253,153,224
```

```
50942 :000,208,070,254,173,016,207
50948 :208,061,105,194,144,003,207
50954 :029,097,194,141,016,208,183
50960 :189,116,003,240,078,041,171
50966 :007,074,133,251,168,189,076
50972 :068,003,208,007,144,065,011
50978 :185,134,194,208,041,185,213
50984 :122,194,208,010,189,068,063
50990 :003,074,144,253,208,047,007
50996 :169,015,061,068,003,240,096
51002 :040,160,007,217,126,194,034
51008 :240,005,136,016,248,048,245
51014 :028,165,251,201,002,208,157
51020 :002,136,136,152,024,125,139
51026 :124,003,072,169,248,133,063
51032 :251,173,083,194,133,252,150
51038 :138,168,104,145,251,096,228
51044 :173,002,221,009,003,141,137
51050 :002,221,032,241,183,138,155
51056 :141,136,002,024,105,003,011
51062 :141,083,194,041,192,141,142
51068 :084,194,010,042,042,073,057
51074 :003,077,000,221,041,003,219
51080 :077,000,221,141,000,221,028
51086 :138,010,010,077,024,208,097
51092 :041,240,077,024,208,141,111
51098 :024,208,096,169,004,141,028
51104 :136,002,162,002,189,000,139
51110 :221,009,003,157,000,221,009
51116 :202,202,016,244,076,129,017
51122 :255,032,170,194,133,251,189
```

```
51128 :132,252,056,233,012,133,234
51134 :253,152,233,195,133,254,130
51140 :160,081,185,254,193,024,069
51146 :101,253,153,254,193,185,061
51152 :255,193,101,254,153,255,139
51158 :193,136,136,136,208,234,233
51164 :169,012,133,253,169,195,127
51170 :133,254,162,004,177,253,185
51176 :145,251,200,208,249,230,235
51182 :252,230,254,202,208,242,090
51188 :177,253,145,251,200,192,182
51194 :167,208,247,096,013,013,226
```

Once you have this program on disk or tape, you can easily make copies of it to place on other disks or tapes. This will be necessary when you begin to write your own programs which make use of "SCS.OBJ" (the Sprite Control System, explained in more detail in Chapters 14 and 15).

Here's how you can create copies. First of all, load and run MLX. When you see the beginning and ending address prompts, enter 49664 and 51199, respectively. Press the SHIFT and L keys at the same time (this is MLX's Load feature). You'll be asked for a filename. Enter SCS.OBJ and press RETURN. Another prompt appears, asking you whether you're using disk or tape. Make sure the disk or tape containing SCS.OBJ is in the drive or Datassette, then press D or T. The program loads into memory. You'll know it's completed when you see the DONE message. Press the SHIFT and S (Save) keys at the same time, then enter the filename SCS.OBJ. Check that the disk or tape you want SCS.OBJ to be *copied to* is in the drive or recorder before hitting D or T. When the DONE message appears again, you know that the machine language file has been successfully copied to the new disk or tape.

You can use this same procedure to make copies of "SID.OBJ," "BMG.OBJ," "EDITOR.OBJ," or any machine language program which requires MLX for its entry. Of course, the beginning and ending addresses, as well as the filenames, will change.

On with the Show

Each of these three examples is composed of several parts. You need all of them in order to make the Sprite Control System operate as it should. SCS.OBJ, the program you just typed in, displays the sprites and makes them move. You also need a BASIC program (which contains all the lines from Program 12-1, "SCS.BAS") and another machine language file that creates the sprite shapes or definitions.

If you're using disk, just make sure all three components are on the same disk. If you're using tape, however, you have to save the programs in a specific order.

Save the BASIC program first, then the machine language definition file. Since SCS.OBJ, Program 12-2, is on a tape by itself already, all you'll have to do is follow the prompts on the screen, changing tapes after loading the BASIC program

and before loading the definition file.

You're now ready to enter the first example program. Notice that much of this program is a repetition of what you entered for Program 12-1, SCS.BAS. When you type in Program 12-3, you can save time by first loading Program 12-1 into memory, then adding the additional lines. Once SCS.BAS is in memory, just type in all the lines you see up to line 58000. Save this program using the filename DEMO#1. (If you're using the Datassette, change the DN=8 in line 120 to DN=1.)

Program 12-3. DEMO#1

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 POKE 53280,12:POKE 53281,0:PRINT "{CLR}{DOWN}
    {5} SPRITE CONTROL SYSTEM DEMO #1"     :rem 182
110 PRINT " BY CRAIG CHAMBERLAIN":PRINT     :rem 65
120 DN=8:SA=780:SX=781:SY=782:SP=783:REM DN=1 FOR
    {SPACE}TAPE                            :rem 181
125 F$="SCS.OBJ":GOSUB 900                 :rem 200
130 GOSUB 58000:REM LOAD CONTROL SYSTEM     :rem 85
140 F$="BUNNY":LA=16384:GOSUB 900:GOSUB 58500:REM
    {SPACE}LOAD DEFINITIONS                :rem 159
300 SYS IN:REM INSTALL                      :rem 94
310 SYS DF,255,0:REM DEFINE BLOCK 255 WITH DEFINIT
    ION 0                                  :rem 102
320 FOR SN=0 TO 7:REM LOOP WITH SPRITE NUMBER
                                           :rem 152
330 SYS PO,SN,150,90:REM SET POSITION (SAME FOR AL
    L SPRITES)                              :rem 56
340 SYS DI,SN,INT(SN/2)*2+1:REM SET DIRECTION (DIF
    FERENT FOR EACH SPRITE)                :rem 116
350 SYS MO,SN,SN/2+1,(SN+1)/2+1:REM SET SPEED (DIF
    FERENT FOR EACH SPRITE)                :rem 243
360 SYS BO,SN,1:REM TURN ON BOUNCE          :rem 19
370 SYS BL,SN,255:REM ASSIGN BLOCK 255 TO SPRITE (
    AND ENABLE)                            :rem 235
380 NEXT SN                                :rem 123
390 GET K$:IF K$="" GOTO 390:REM WAIT FOR KEYPRESS
                                            :rem 41
400 SYS RE:REM REMOVE                       :rem 22
410 END                                    :rem 108
800 REM DEFINITION DATA (USED IN LATER DEMONSTRATI
    ON)                                    :rem 247
801 DATA 0,63,12,0,48,15,129,240,12,195,48,12,102,
    48,7,102,224,3,231,192,0,195           :rem 229
802 DATA 0,1,255,128,3,24,192,7,255,224,12,60,48,1
    5,0,240,7,255,224,0,255,0,28           :rem 230
```

```
803 DATA 60,56,7,60,224,1,255,128,48,126,12,124,60
    ,62,127,60,254,255,255,255                    :rem 163
900 IF DN=8 THEN RETURN                           :rem 62
910 PRINT "POSITION TAPE TO THE BEGINNING OF":PRIN
    T F$;" AND PRESS RETURN"                      :rem 164
920 WAIT 198,15:GET K$:RETURN                     :rem 167
58000 POKE SA,1:POKE SX,DN:POKE SY,1:SYS 65466:F$=
      "SCS.OBJ":GOSUB 59000                       :rem 116
58010 POKE SA,0:SYS 65493:IF PEEK(SP)AND1 GOTO 591
      00                                          :rem 127
58020 IN=49664:RE=49667:RL=49670:RS=49673:DF=49676
      :BL=49679:CO=49682:WI=49685                 :rem 172
58030 HE=49688:PO=49691:DI=49694:WR=49697:BO=49700
      :MO=49703:LB=49706:RB=49709                 :rem 148
58040 TB=49712:BB=49715:FR=49718:UN=49721:PR=49724
      :MU=49727:EN=49730:JS=49733                 :rem 149
58050 AU=49736:CH=49739:RETURN                    :rem 14
58100 GOSUB 58000:SYS 51123,LA:LA=LA+1191:RETURN
                                                  :rem 11
58400 POKE 49749,LA-256*INT(LA/256):POKE 49750,INT
      (LA/256)                                    :rem 182
58410 READ J:POKE 49751,J:FOR K=0 TO J:READ J:POKE
       LA,J:LA=LA+1                               :rem 16
58420 IF J THEN FOR LA=LA TO LA+J-1:READ I:POKE LA
      ,I:NEXT LA                                  :rem 92
58430 NEXT K:RETURN                               :rem 168
58500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".DEF":GOSUB 59000                       :rem 24
58510 POKE SA,0:POKE SX,LA-256*INT(LA/256):POKE SY
      ,INT(LA/256)                                :rem 78
58520 SYS 65493:IF PEEK(SP)AND1 GOTO 59100 :rem 44
58530 LA=LA+1:POKE 49749,LA-256*INT(LA/256):POKE 4
      9750,INT(LA/256)                            :rem 167
58540 POKE 49751,PEEK(LA-1):LA=PEEK(SX)+256*PEEK(S
      Y):RETURN                                   :rem 31
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                                      :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                                    :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END                     :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                  :rem 36
59120 PRINT ST:END                                :rem 70
```

Now type in and save the sprite definition file. It's in machine language, so you'll need MLX to enter it. These are the responses you need to give:

Starting Address: 49152
Ending Address: 49217
Filename: BUNNY.DEF

Make sure the filename is BUNNY.DEF, or the "DEMO#1" program will not be able to load it properly. If you're using disk, place this on the same disk as the previous two programs. If you're using tape, make sure BUNNY.DEF follows DEMO#1.

Program 12-4. BUNNY.DEF

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

```
49152 :000,063,012,000,048,015,138
49158 :129,240,012,195,048,012,130
49164 :102,048,007,102,224,003,242
49170 :231,192,000,195,000,001,125
49176 :255,128,003,024,192,007,121
49182 :255,224,012,060,048,015,132
49188 :000,240,007,255,224,000,250
49194 :255,000,028,060,056,007,192
49200 :060,224,001,255,128,048,252
49206 :126,012,124,060,062,127,053
49212 :060,254,255,255,255,013,128
```

Load and run DEMO#1. If you're using disk, the SCS.OBJ and BUNNY.DEF files are automatically loaded. All eight sprites then appear on the screen.

However, if you're using the Datassette, you'll have to do some tape switching. Once DEMO#1 is run, you'll see the prompt POSITION TAPE TO THE BEGINNING OF SCS.OBJ AND PRESS RETURN. Take the tape containing DEMO#1 out of the Datassette and insert the tape you made earlier which has SCS.OBJ on it. (Make sure this second tape is rewound to the beginning of the SCS.OBJ file.) Press RETURN, then the PLAY button on the recorder. The machine language file will load. Once it's finished, you'll see another prompt, this time POSITION TAPE TO THE BEGINNING OF BUNNY AND PRESS RETURN. Insert the first tape (don't rewind it—leave it where it ended after loading DEMO#1). Press RETURN, and then PLAY. The machine language sprite definition file now loads. When that's completed, you'll see the eight sprites

bouncing around the screen.

The sprites all have the same shape, but each sprite has a different color, direction, and speed. Also notice that whenever a sprite hits the screen edge, it rebounds.

While the program is running, press the RUN/STOP key. The sprites continue to move! Type LIST, and watch the bunnies go bouncing over the program listing. This illustrates that the Sprite Control System can animate sprites while BASIC is doing something else. A program can handle things like printing a score and generating sound effects while the Control System is moving the sprites.

Enter the command CONT to make the program resume execution, then press any key to end the program and turn off the sprites.

Getting Fancy

The second example demonstrates some of the more advanced features of the Sprite Control System. Enter the following program and save it with the filename DEMO#2. Again, if you're using tape, change line 120 so that DN=1.

Program 12-5. DEMO#2

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 POKE 53280,0:POKE 53281,12:PRINT "{CLR}{DOWN}
    {BLK} SPRITE CONTROL SYSTEM DEMO #2"  :rem 175
110 PRINT " BY CRAIG CHAMBERLAIN":PRINT    :rem 65
120 DN=8:SA=780:SX=781:SY=782:SP=783:REM DN=1 FOR
    {SPACE}TAPE                           :rem 181
125 F$="SCS.OBJ":GOSUB 900                :rem 200
130 GOSUB 58000:REM LOAD CONTROL SYSTEM    :rem 85
140 F$="GOBBLER":LA=16384:GOSUB 900:GOSUB 58500:RE
    M LOAD DEFINITIONS                     :rem 16
300 SYS IN:REM INSTALL                     :rem 94
310 FOR K=0 TO 11:SYS DF,244+K,K:NEXT:REM DEFINE B
    LOCKS 244 TO 255                      :rem 135
320 SYS CO,0,7:REM SET COLOR TO YELLOW     :rem 237
330 SYS JS,0,2:REM SELECT JOYSTICK 2       :rem 108
340 SYS MO,0,3,2:REM SET MOTION (SPEED)    :rem 237
350 SYS WR,0,1:REM TURN ON WRAPAROUND        :rem 0
360 SYS PO,0,80,80:REM POSITION SPRITE     :rem 37
370 SYS BL,0,252:REM ASSIGN BLOCK 252 TO SPRITE 0
    {SPACE}(AND ENABLE)                   :rem 164
380 SYS AU,0,9,244:REM AUTOSHAPE 8 DIRECTIONS PLUS
     CENTERING, BASE BLOCK = 244           :rem 23
390 SYS CO,1,2,1,0:REM COLORS RED, BLACK, AND WHIT
    E                                     :rem 117
```

```
400 SYS MU,1,1:REM TURN ON MULTICOLOR      :rem 253
410 SYS PO,1,240,120:REM POSITION SPRITE   :rem 123
420 SYS MO,1,1,1:REM SET MOTION (SPEED)    :rem 234
430 SYS CH,1,0:REM HAVE SPRITE 1 CHASE SPRITE 0
                                           :rem 143
440 SYS BL,1,254:K=20:REM ASSIGN BLOCK 254 TO SPRI
    TE 1 (AND ENABLE)                      :rem 204
450 IF PEEK(53278) GOTO 510:REM COLLIDED   :rem 212
460 K=K-1:IF K GOTO 450:REM LOOP 20 TIMES OR UNTIL
     COLLISION                             :rem 198
470 SYS BL,1,255:K=20:REM ASSIGN BLOCK 255 TO SPRI
    TE 1                                    :rem 6
480 IF PEEK(53278) GOTO 510:REM COLLIDED   :rem 215
490 K=K-1:IF K GOTO 480:REM LOOP 20 TIMES OR UNTIL
     COLLISION                             :rem 204
500 GOTO 440:REM CONTINUE ALTERNATING BETWEEN TWO
    {SPACE}SHAPES                          :rem 234
510 SYS EN,1,0:REM DISABLE SPRITE 1        :rem 251
520 SYS JS,0,0:SYS DI,0,0:REM STOP MOVING SPRITE 0
                                           :rem 164
530 SYS AU,0,0:SYS BL,0,253:REM TURN AUTOSHAPE OFF
    , DISPLAY FROWN FACE                   :rem 151
540 FOR K=1 TO 999:NEXT:REM DELAY LOOP     :rem 199
550 SYS RE:REM REMOVE                       :rem 28
560 END                                    :rem 114
900 IF DN=8 THEN RETURN                     :rem 62
910 PRINT "POSITION TAPE TO THE BEGINNING OF":PRIN
    T F$;" AND PRESS RETURN"               :rem 164
920 WAIT 198,15:GET K$:RETURN              :rem 167
58000 POKE SA,1:POKE SX,DN:POKE SY,1:SYS 65466:F$=
      "SCS.OBJ":GOSUB 59000                :rem 116
58010 POKE SA,0:SYS 65493:IF PEEK(SP)AND1 GOTO 591
      00                                   :rem 127
58020 IN=49664:RE=49667:RL=49670:RS=49673:DF=49676
      :BL=49679:CO=49682:WI=49685          :rem 172
58030 HE=49688:PO=49691:DI=49694:WR=49697:BO=49700
      :MO=49703:LB=49706:RB=49709          :rem 148
58040 TB=49712:BB=49715:FR=49718:UN=49721:PR=49724
      :MU=49727:EN=49730:JS=49733          :rem 149
58050 AU=49736:CH=49739:RETURN              :rem 14
58100 GOSUB 58000:SYS 51123,LA:LA=LA+1191:RETURN
                                           :rem 11
58400 POKE 49749,LA-256*INT(LA/256):POKE 49750,INT
      (LA/256)                             :rem 182
58410 READ J:POKE 49751,J:FOR K=0 TO J:READ J:POKE
       LA,J:LA=LA+1                        :rem 16
58420 IF J THEN FOR LA=LA TO LA+J-1:READ I:POKE LA
      ,I:NEXT LA                           :rem 92
58430 NEXT K:RETURN                        :rem 168
58500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".DEF":GOSUB 59000                :rem 24
```

```
58510 POKE SA,0:POKE SX,LA-256*INT(LA/256):POKE SY
      ,INT(LA/256)                              :rem 78
58520 SYS 65493:IF PEEK(SP)AND1 GOTO 59100 :rem 44
58530 LA=LA+1:POKE 49749,LA-256*INT(LA/256):POKE 4
      9750,INT(LA/256)                         :rem 167
58540 POKE 49751,PEEK(LA-1):LA=PEEK(SX)+256*PEEK(S
      Y):RETURN                                 :rem 31
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                                    :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                                  :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END                   :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                :rem 36
59120 PRINT ST:END                              :rem 70
```

You'll also need to type in and save the file containing the definitions used by the "DEMO#2" program. That's this next program, which is in machine language. The starting and ending addresses you'll need for MLX are:

Starting Address: 49152
Ending Address: 49829
Filename: GOBBLER.DEF

Save it with the filename GOBBLER.DEF immediately after the DEMO#2 program if you're using tape.

If you've got a Datassette, the procedure for loading the three sections is the same as that for the DEMO#1 program. Refer to the explanation if you've forgotten the process.

Before you run DEMO#2, plug a joystick into port 2. Be ready to move the joystick as soon as you see the sprites.

The motion of the first sprite, the one with the smiling face, is controlled by the joystick. If you move this sprite past the edge of the screen, it wraps around to the other side. Also notice that this sprite changes shape when you change directions. There's a different shape for each of the eight directions, plus a shape for when the sprite is stationary.

The second sprite alternates between two shapes and chases the first sprite. The program ends when the second sprite catches the first.

While all the action is taking place, the BASIC program is doing only two things. It's alternating the second sprite between the two shapes, and it's watching for when the two sprites collide.

Program 12-6. GOBBLER.DEF

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

```
49152 :011,054,000,000,000,000,065
49158 :000,000,000,000,000,001,007
49164 :255,128,007,255,224,015,128
49170 :255,240,031,159,248,063,246
49176 :015,252,063,159,252,127,124
49182 :255,254,031,255,254,007,062
49188 :255,254,001,255,252,000,029
49194 :127,252,063,255,248,031,250
49200 :255,240,007,255,224,001,006
49206 :255,128,054,000,000,000,235
49212 :000,000,000,000,000,000,060
49218 :001,255,128,007,255,224,168
49224 :015,255,240,031,249,248,086
49230 :063,240,252,063,249,252,173
49236 :127,255,254,127,255,248,070
49242 :127,255,224,063,255,128,118
49248 :063,254,000,031,255,252,183
49254 :015,255,248,007,255,224,082
49260 :001,255,128,054,000,000,034
49266 :000,000,000,000,000,000,114
49272 :000,003,007,128,007,015,024
49278 :224,015,015,240,031,031,170
49284 :248,063,031,252,063,063,084
49290 :252,127,063,190,127,127,000
49296 :030,127,255,030,063,255,136
49302 :188,063,255,252,031,255,170
49308 :248,015,255,240,007,255,152
49314 :224,001,255,128,054,000,056
49320 :000,000,000,000,000,000,168
49326 :000,000,001,255,128,007,053
49332 :255,224,015,255,240,031,176
49338 :255,248,063,255,252,061,040
49344 :255,252,120,255,254,120,168
49350 :254,254,125,252,254,063,120
49356 :252,252,063,248,252,031,022
49362 :248,248,015,240,240,007,184
49368 :240,224,001,224,192,054,127
49374 :000,000,000,000,000,000,222
49380 :000,000,000,001,255,128,100
49386 :007,255,224,007,255,240,198
49392 :007,255,248,003,255,252,236
49398 :003,252,252,113,248,126,216
49404 :120,252,254,124,255,254,231
49410 :062,255,252,063,255,252,117
49416 :031,255,248,015,255,240,028
49422 :007,255,224,001,255,128,116
49428 :054,000,000,000,000,000,074
49434 :000,000,000,000,001,255,026
49440 :128,007,255,224,015,255,148
49446 :224,031,255,224,063,255,066
49452 :192,063,063,192,126,031,199
49458 :142,127,063,030,127,255,026
49464 :062,063,255,124,063,255,110
49470 :252,031,255,248,015,255,094
49476 :240,007,255,224,001,255,026
49482 :128,054,000,000,000,000,000
49488 :000,000,000,000,000,001,081
49494 :255,128,007,255,224,015,202
49500 :255,240,031,255,248,063,160
```

```
49506 :159,252,063,015,252,127,198
49512 :159,254,127,255,254,127,000
49518 :255,254,063,031,252,060,001
49524 :063,252,000,127,248,000,038
49530 :255,240,001,255,224,001,074
49536 :255,128,054,000,000,000,053
49542 :000,000,000,000,000,000,134
49548 :001,255,128,007,255,224,242
49554 :015,255,240,031,255,248,166
49560 :063,249,252,063,240,252,247
49566 :127,249,254,127,255,254,144
49572 :127,255,254,063,248,252,083
49578 :063,252,060,031,254,000,062
49584 :015,255,000,003,255,128,064
49590 :001,255,128,054,000,000,108
49596 :000,000,000,000,000,000,188
49602 :000,001,255,128,007,255,072
49608 :224,015,255,240,030,126,066
49614 :120,060,060,060,062,126,182
49620 :124,127,231,254,127,195,246
49626 :254,115,231,206,057,255,056
49632 :156,060,000,060,030,000,018
49638 :120,015,129,240,007,255,228
49644 :224,001,255,128,054,000,130
49650 :000,000,000,000,000,000,242
49656 :000,000,001,255,128,007,127
49662 :255,224,015,255,240,030,249
49668 :126,120,062,126,124,062,112
49674 :126,124,127,231,254,127,231
49680 :195,254,127,219,254,063,104
49686 :255,252,063,129,252,031,236
```

```
49692 :000,248,015,060,240,007,086
49698 :255,224,001,255,128,063,192
49704 :002,170,128,010,170,160,168
49710 :042,170,168,169,105,106,038
49716 :169,105,106,171,107,106,048
49722 :171,107,106,171,107,106,058
49728 :170,170,170,170,170,170,060
49734 :170,170,170,170,105,170,001
49740 :170,150,170,170,170,170,052
49746 :042,170,168,042,170,168,074
49752 :010,000,160,010,000,160,172
49758 :010,000,160,042,128,160,082
49764 :000,002,168,062,002,170,248
49770 :128,010,170,160,042,170,018
49776 :168,169,105,106,169,105,166
49782 :106,169,233,234,169,233,238
49788 :234,169,233,234,170,170,054
49794 :170,170,170,170,170,170,126
49800 :170,170,105,170,170,150,047
49806 :170,170,170,170,042,170,010
49812 :168,042,170,168,010,000,194
49818 :160,010,000,160,010,000,238
49824 :160,010,002,168,042,128,158
```

Hi, Guy

This final program demonstrates a few other features of the Sprite Control System. Enter Program 12-7 and save it with the filename DEMO#3. Remember to change the value of DN in line 120 to 1 if you're using the Datassette.

Program 12-7. DEMO#3

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 POKE 53280,12:POKE 53281,0:PRINT "{CLR}{DOWN}
    {5} SPRITE CONTROL SYSTEM DEMO #3"     :rem 184
110 PRINT " BY MARK THOMAS":PRINT          :rem 188
115 PRINT:FOR K=1 TO 17:PRINT "{6 SPACES}{RVS}
    {4 SPACES}{OFF}{7 SPACES}{RVS}{5 SPACES}{OFF}
    {7 SPACES}{RVS}{4 SPACES}":NEXT        :rem 231
120 DN=8:SA=780:SX=781:SY=782:SP=783:REM DN=1 FOR
    {SPACE}TAPE                            :rem 181
125 F$="SCS.OBJ":GOSUB 900                 :rem 200
130 GOSUB 58000:REM LOAD CONTROL SYSTEM     :rem 85
140 F$="GUY":LA=16384:GOSUB 900:GOSUB 58500:REM LO
    AD DEFINITIONS                          :rem 8
300 SYS IN:REM INSTALL                      :rem 94
310 SYS DF,254,0:SYS DF,255,1:REM DEFINE BLOCKS
                                           :rem 175
320 SYS FR,1+2+4+8:REM FREEZE FIRST FOUR SPRITES
                                           :rem 117
330 REM SET FIRST PERSON                   :rem 197
340 SYS CO,0,3,7,10:SYS CO,1,13:REM SET COLORS
                                           :rem 103
350 FOR SN=0 TO 1                           :rem 97
360 SYS MU,SN,1:REM TURN ON MULTICOLOR MODE
                                           :rem 151
370 SYS WI,SN,1:SYS HE,SN,1:REM SET DOUBLE WIDTH A
    ND HEIGHT                               :rem 36
380 SYS PO,SN,84,40+SN*42:REM SET POSITION  :rem 9
390 SYS JS,SN,2,3:REM SET JOYSTICK 2 CONTROL, 4 DI
    RECTIONS, MOMENTUM                     :rem 129
400 SYS MO,SN,1,1:REM SET SPEED             :rem 49
410 SYS WR,SN,1:REM TURN ON WRAPAROUND     :rem 110
420 SYS RB,SN,319-48:SYS TB,SN,SN*42:SYS BB,SN,199
    -73+42*SN:REM SET BOUNDARIES           :rem 147
430 NEXT SN                                :rem 119
440 REM SET SECOND PERSON                  :rem 251
450 SYS CO,2,4:SYS CO,3,6:REM SET COLORS    :rem 80
460 FOR SN=2 TO 3                          :rem 103
465 SYS PR,SN,1:REM SET PRIORITY BEHIND SCREEN DAT
    A                                      :rem 122
```

```
470 SYS MU,SN,1:REM TURN ON MULTICOLOR MODE
                                              :rem 153
480 SYS WI,SN,1:SYS HE,SN,1:REM SET DOUBLE WIDTH A
    ND HEIGHT                                 :rem 38
490 SYS PO,SN,180,102+(SN-2)*42:REM SET POSITION
                                              :rem 23
500 SYS JS,SN,1,3:REM SET JOYSTICK 1 CONTROL, 4 DI
    RECTIONS, MOMENTUM                        :rem 120
510 SYS MO,SN,1,1:REM SET SPEED               :rem 51
520 SYS BO,SN,1:REM TURN ON BOUNCE            :rem 17
530 SYS RB,SN,319-48:REM SET BOUNDARIES       :rem 80
535 SYS TB,SN,(SN-2)*42:SYS BB,SN,199-73+42*(SN-2)
    :REM SET BOUNDARIES                       :rem 254
540 NEXT SN                                   :rem 121
550 SYS BL,0,254:SYS BL,1,255:SYS BL,2,254:SYS BL,
    3,255:REM MAKE SPRITES APPEAR             :rem 47
560 SYS UN,1+2+4+8:REM UNFREEZE SPRITES       :rem 101
570 IF (PEEK(56320)AND16) AND (PEEK(56321)AND16) G
    OTO 570:REM WAIT FOR TRIGGER              :rem 36
580 SYS RE:REMOVE SPRITE CONTROL SYSTEM       :rem 24
585 POKE 198,0:REM CLEAR KEY BUFFER           :rem 247
590 END                                       :rem 117
900 IF DN=8 THEN RETURN                       :rem 62
910 PRINT "{CLR}{DOWN}POSITION TAPE TO THE BEGINNI
    NG OF":PRINT F$;" AND PRESS RETURN"       :rem 72
915 PRINT:FOR K=1 TO 17:PRINT "{6 SPACES}{RVS}
    {4 SPACES}{OFF}{7 SPACES}{RVS}{5 SPACES}{OFF}
    {7 SPACES}{RVS}{4 SPACES}":NEXT           :rem 239
920 WAIT 198,15:GET K$:RETURN                 :rem 167
58000 POKE SA,1:POKE SX,DN:POKE SY,1:SYS 65466:F$=
      "SCS.OBJ":GOSUB 59000                   :rem 116
58010 POKE SA,0:SYS 65493:IF PEEK(SP)AND1 GOTO 591
      00                                      :rem 127
58020 IN=49664:RE=49667:RL=49670:RS=49673:DF=49676
      :BL=49679:CO=49682:WI=49685             :rem 172
58030 HE=49688:PO=49691:DI=49694:WR=49697:BO=49700
      :MO=49703:LB=49706:RB=49709             :rem 148
58040 TB=49712:BB=49715:FR=49718:UN=49721:PR=49724
      :MU=49727:EN=49730:JS=49733             :rem 149
58050 AU=49736:CH=49739:RETURN                :rem 14
58100 GOSUB 58000:SYS 51123,LA:LA=LA+1191:RETURN
                                              :rem 11
58400 POKE 49749,LA-256*INT(LA/256):POKE 49750,INT
      (LA/256)                                :rem 182
58410 READ J:POKE 49751,J:FOR K=0 TO J:READ J:POKE
       LA,J:LA=LA+1                           :rem 16
58420 IF J THEN FOR LA=LA TO LA+J-1:READ I:POKE LA
      ,I:NEXT LA                              :rem 92
```

```
58430 NEXT K:RETURN                                    :rem 168
58500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".DEF":GOSUB 59000                            :rem 24
58510 POKE SA,0:POKE SX,LA-256*INT(LA/256):POKE SY
      ,INT(LA/256)                                     :rem 78
58520 SYS 65493:IF PEEK(SP)AND1 GOTO 59100 :rem 44
58530 LA=LA+1:POKE 49749,LA-256*INT(LA/256):POKE 4
      9750,INT(LA/256)                                 :rem 167
58540 POKE 49751,PEEK(LA-1):LA=PEEK(SX)+256*PEEK(S
      Y):RETURN                                        :rem 31
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                                           :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                                         :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END                          :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                       :rem 36
59120 PRINT ST:END                                     :rem 70
```

Type in and save the following machine language file to tape or disk. Your responses to MLX's prompts will be:

Starting Address: 49152
Ending Address: 49265
Filename: GUY.DEF

If you have a tape drive, place this file immediately after "DEMO#3."

Program 12-8. GUY.DEF

To enter this program, you must use "The Machine Language Editor (MLX)," a program found in Appendix C.

```
49152 :001,063,000,085,000,000,149
49158 :085,000,001,085,064,001,242
49164 :255,064,001,255,064,003,142
49170 :060,192,003,255,192,000,208
49176 :195,000,000,255,000,000,218
49182 :060,000,002,170,128,010,144
49188 :170,160,042,170,168,042,020
49194 :170,168,040,170,040,040,158
49200 :170,040,040,170,040,040,036
49206 :170,040,040,170,040,040,042
49212 :170,040,060,170,060,048,096
49218 :060,085,060,002,170,128,059
49224 :002,170,128,002,170,128,160
49230 :002,170,128,002,170,128,166
49236 :002,130,128,002,130,128,092
49242 :002,130,128,002,130,128,098
```

```
49248 :002,130,128,002,130,128,104
49254 :002,130,128,003,195,192,240
49260 :015,195,240,063,195,252,044
```

Loading from tape is identical to the previous two examples. Make sure you have the tape with the SCS.OBJ file handy.

A joystick plugged into port 2 controls the figure on the left. The figure actually consists of two sprites, one above the other. The sprites are synchronized so that they move together to create the appearance of one large object.

Also notice that the figure doesn't stop moving when you stop pushing the joystick. Instead, it keeps moving in the current direction. Another feature is that only four directions are allowed. The joystick is ignored when it's pushed in a diagonal direction.

To make the other guy move, plug the joystick into port 1. The Sprite Control System can control sprites through either joystick port, or even both at the same time. This second figure does not wrap around the screen, but simply rebounds off the edges.

One other feature demonstrated by this program is the foreground/background priority feature. The first figure (initially on the left) moves in front of the drawing on the screen, while the second moves behind the drawing.

Press the joystick button to end this program.

These three demonstration programs have illustrated many of the things possible with the Sprite Control System. But this has been only an introduction. The next chapters will show you how to design your own sprites using the Sprite Editor, how to use the Sprite Control System in your own BASIC programs, how to control sprites with joysticks, how to automatically change their shapes, even how to detect sprite-to-sprite collisions. You've just started. But as promised, you'll find the Control System simple to use, easy to learn, and a tremendous aid when it comes to anything dealing with sprites.

Defining Sprite Shapes

You've just seen how the Sprite Control System can quickly and easily move and animate sprites on your Commodore 64's screen. But before you can use the Control System, you have to have some sprites. "The Sprite Editor" is a utility which helps you design and create sprites, without the tedious task of plotting on graph paper, counting up bit values, or even retyping the sprite definition into the computer. The Editor does all this for you. With this program and a joystick, it's easy, even fun, to design sprite shapes.

Here's the listing for the Sprite Editor. Remember to change DN=8 in line 110 to DN=1 if you're using tape.

Program 13-1. The Sprite Editor

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SPRITE DEFINITION EDIT
    OR":REM V1.0                              :rem 161
110 PRINT " BY CRAIG CHAMBERLAIN":DN=8:SA=780:SX=7
    81:SY=782:SP=783:GOTO 800                 :rem 178
200 IF SN AND SN=NS AND SL=0 THEN NS=NS-1:TA=AS-1:
    RETURN                                     :rem 23
205 IF SN>NS AND SL THEN NS=SN:AS=TA+1:SL%(SN)=0:T
    A=AS                                      :rem 191
206 IF SN>NS THEN RETURN                      :rem 182
210 IF SL=SL%(SN) GOTO 230                    :rem 130
220 SYS MV,AS+SL%(SN),TA-AS-SL%(SN),AS+SL:TA=TA-SL
    %(SN)+SL:SL%(SN)=SL                       :rem 224
230 SYS MV,DB,SL,AS:RETURN                    :rem 142
250 PRINT "{UP}" TAB(6) N "{LEFT}{2 SPACES}":FOR K
    =1 TO 30:NEXT:WAIT 56320,28,28            :rem 122
260 P=PEEK(56320):IF PAND4 GOTO 270           :rem 141
262 IF N AND N<=NS THEN N=N-1:SL=SL%(N):A=A-SL-1:S
    YS 51257,A,SL,DB:GOTO 250                 :rem 229
264 IF N=0 AND SL%(NS) AND NS<MX THEN N=NS+1:SL=0:
    SYS 51266:GOTO 250                         :rem 8
266 N=NS:SL=SL%(N):A=TA-SL:SYS 51257,A,SL,DB:GOTO
    {SPACE}250                                :rem 141
270 IF PAND8 GOTO 280                          :rem 86
272 IF N<NS THEN N=N+1:A=A+SL+1:SL=SL%(N):SYS 5125
    7,A,SL,DB:GOTO 250                        :rem 130
274 IF N=NS AND SL AND NS<MX THEN N=NS+1:SL=0:SYS
    {SPACE}51266:GOTO 250                      :rem 99
276 N=0:SL=SL%(0):A=BA+2:SYS 51257,A,SL,DB:GOTO 25
    0                                         :rem 126
```

```
280 IF PAND16 GOTO 250                          :rem 131
290 WAIT 56320,16:POKE 198,0:RETURN             :rem 228
300 GOSUB 620:PRINT "EDITING" SN                 :rem 91
310 K=FRE(0):SYS 50832:SL=PEEK(SX):GET K$       :rem 117
320 IF K$="E" AND (NS OR SN OR SL) GOTO 500
                                                 :rem 110
325 IF K$="C" AND (NS OR SN OR SL) GOTO 510
                                                 :rem 114
330 IF K$="{INST}" THEN A=DB+3*PEEK(50983):SYS 505
    11,A,E-A,A+3,A:GOTO 310                      :rem 163
335 IF K$=CHR$(20) THEN A=DB+3*PEEK(50983):SYS 505
    33,A+3,E-A,A:GOTO 310                         :rem 27
340 IF K$="{LEFT}" THEN SYS 50643:GOTO 310:rem 234
342 IF K$="{RIGHT}" THEN SYS 50671:GOTO 310
                                                 :rem 109
344 IF K$="{UP}" THEN SYS 50549:GOTO 310         :rem 231
346 IF K$="{DOWN}" THEN SYS 50568:GOTO 310:rem 106
350 IF K$="F" THEN SYS 50699:GOTO 310            :rem 159
355 IF K$="I" THEN SYS 50587:GOTO 310            :rem 163
360 IF K$="{F1}" THEN POKE 50371,170:GOTO 310
                                                 :rem 198
362 IF K$="{F3}" THEN POKE 50371,85:GOTO 310
                                                 :rem 158
364 IF K$="{F5}" THEN POKE 50?371,255:GOTO 310
                                                 :rem 208
370 IF K$<>"{F2}" GOTO 373                       :rem 181
371 GOSUB 640:IF N>=0 THEN POKE 53288,N:POKE 50373
    +PEEK(50370),N:SYS 50735                      :rem 27
372 GOTO 300                                     :rem 104
373 IF K$<>"{F4}" GOTO 376                       :rem 188
374 GOSUB 640:IF N>=0 THEN POKE 53285,N:POKE 50374
    -PEEK(50370),N:SYS 50735                      :rem 30
375 GOTO 300                                     :rem 107
376 IF K$<>"{F6}" GOTO 380                       :rem 187
377 GOSUB 640:IF N>=0 THEN POKE 53286,N:POKE 50375
    ,N:SYS 50735                                 :rem 129
378 GOTO 300                                     :rem 110
380 IF K$<>"{F8}" GOTO 390                       :rem 184
381 GOSUB 640:IF N>=0 THEN POKE 53282,N           :rem 84
382 GOTO 300                                     :rem 105
390 IF K$="M" THEN SYS 51179:GOTO 310            :rem 164
400 IF K$="W" THEN POKE 53277,NOTPEEK(53277)AND2:G
    OTO 310                                      :rem 119
405 IF K$="H" THEN POKE 53271,NOTPEEK(53271)AND2:G
    OTO 310                                       :rem 97
410 IF K$<>"{CLR}" GOTO 420                      :rem 179
412 GOSUB 620:PRINT "ERASE?":GOSUB 630:IF K$="Y" T
    HEN SYS 51266                                  :rem 0
414 GOTO 300                                     :rem 101
420 IF K$="L" GOTO 520                            :rem 49
```

```
430 IF K$="S" AND (NS OR SN OR SL) GOTO 530
                                              :rem 129
440 IF K$="D" AND (NS OR SN OR SL) GOTO 540
                                              :rem 116
450 IF K$<>"Q" GOTO 310                       :rem 115
452 GOSUB 620:PRINT "QUIT?":GOSUB 630:IF K$="N" GO
    TO 300                                    :rem 102
454 POKE 648,4:SYS 51386:END                   :rem 31
500 GOSUB 200:IF TA>MT THEN F$="MEMORY FULL":GOTO
    {SPACE}710                                :rem 21
505 GOSUB 620:PRINT "EDIT":N=SN:A=AS:GOSUB 250:SN=
    N:AS=A:GOTO 300                           :rem 158
510 GOSUB 200:GOSUB 620:PRINT "COPY":N=SN:A=AS:GOS
    UB 250:GOTO 300                           :rem 73
520 GOSUB 620:PRINT "LOAD:";:F$="":GOSUB 670:IF F$
    ="" GOTO 300                              :rem 75
521 GOSUB 660:SYS 51266:GOSUB 600             :rem 65
522 POKE SA,0:POKE SX,FNL(BA):POKE SY,FNH(BA):POKE
     214,20:GOSUB 680                         :rem 131
523 SYS 65493:GOSUB 690:IF PEEK(SP)AND1 GOTO 700
                                              :rem 179
524 IF PEEK(SX)+256*PEEK(SY)>MT THEN F$="NOT ENOUG
    H MEMORY":GOTO 710                        :rem 243
525 NS=PEEK(BA):A=BA+1:FOR K=0 TO NS:SL%(K)=PEEK(A
    ):A=A+SL%(K)+1:NEXT                        :rem 14
526 TA=A:SL=SL%(0):SYS 51260,AS,SL,DB:GOTO 300
                                              :rem 133
530 GOSUB 620:PRINT "SAVE:";:F$="":GOSUB 670:IF F$
    ="" GOTO 300                               :rem 91
531 GOSUB 200:POKE BA,NS:A=BA+1:FOR K=0 TO NS:POKE
     A,SL%(K):A=A+SL%(K)+1:NEXT               :rem 167
532 GOSUB 600:POKE SA,251:POKE 251,FNL(BA):POKE 25
    2,FNH(BA)                                  :rem 48
534 POKE SX,FNL(TA):POKE SY,FNH(TA):POKE 214,21:GO
    SUB 680                                    :rem 82
535 SYS 65496:GOSUB 690:ON (PEEK(SP)AND1)+1 GOTO 3
    00,700                                     :rem 51
540 GOSUB 620:PRINT "LINE:";:D$=""            :rem 124
541 WAIT 198,15:GET K$:IF D$="" THEN ON (K$=CHR$(1
    3))+2 GOTO 300,544                        :rem 175
542 IF K$=CHR$(20) THEN PRINT K$;:D$=LEFT$(D$,LEN(
    D$)-1):GOTO 541                            :rem 6
543 IF K$=CHR$(13) THEN N=VAL(D$):GOTO 546:rem 245
544 IF VAL(D$)>6399 OR K$<"0" OR K$>"9" OR (K$="0"
     AND D$="") GOTO 541                       :rem 13
545 PRINT K$;:D$=D$+K$:GOTO 541               :rem 195
546 GOSUB 200:POKE BA,NS:A=BA+1:FOR K=0 TO NS:POKE
     A,SL%(K):A=A+SL%(K)+1:NEXT               :rem 173
547 GOSUB 620:P=BA:A=TA                        :rem 79
```

```
550 PRINT N:PRINT "{UP}{2 RIGHT}";:Q=76-LEN(STR$(N
    )):D$=STR$(PEEK(P))                          :rem 231
551 P=P+1:IF P=TA GOTO 554                       :rem 150
552 K$=STR$(PEEK(P)):J=FRE(0)                    :rem 155
553 IF LEN(D$)+LEN(K$)<Q THEN D$=D$+","+MID$(K$,2)
    :P=P+1:IF P<TA GOTO 552                      :rem 107
554 R=A+6+LEN(D$):POKE A,FNL(R):POKE A+1,FNH(R)
                                                 :rem 156
555 POKE A+2,FNL(N):POKE A+3,FNH(N):POKE A+4,131
                                                 :rem 144
556 FOR K=1 TO LEN(D$):POKE A+4+K,ASC(MID$(D$,K)):
    NEXT:POKE R-1,0:A=R                          :rem 116
557 IF A>MT THEN F$="NOT ENOUGH MEMORY":GOTO 710
                                                   :rem 5
558 IF P<TA THEN N=N+1:GOTO 550                  :rem 195
559 POKE A,0:POKE A+1,0:A=A+2                     :rem 39
560 GOSUB 620:PRINT "FILE:";:F$="":GOSUB 670:IF F$
    =""{2 SPACES}GOTO 300                         :rem 79
561 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=F$
    +".DAT":GOSUB 610                             :rem 85
562 POKE SA,251:POKE 251,FNL(TA):POKE 252,FNH(TA):
    POKE SX,FNL(A):POKE SY,FNH(A)                :rem 104
563 POKE 214,21:GOSUB 680:SYS 65496:GOSUB 690:ON (
    PEEK(SP)AND1)+1 GOTO 300,700                  :rem 27
600 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=F$
    +".DEF"                                      :rem 244
610 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                         :rem 241
615 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                      :rem 163
620 POKE 214,22:PRINT:PRINT "{20 SPACES}":PRINT "
    {UP}{2 SPACES}";:RETURN                      :rem 177
630 GET K$:IF K$<>"Y" AND K$<>"N" GOTO 630 :rem 87
635 RETURN                                       :rem 126
640 GOSUB 620:PRINT "ENTER COLOR:";               :rem 47
641 GET K$:IF K$=CHR$(13) THEN N=-1:RETURN :rem 71
642 IF K$<"0" OR K$>"9" GOTO 641                 :rem 233
643 PRINT K$;:N=ASC(K$)-48                       :rem 201
644 GET K$:IF K$=CHR$(13) THEN RETURN             :rem 39
645 IF K$=CHR$(20) THEN PRINT K$;:GOTO 641         :rem 2
646 IF N<>1 OR K$<"0" OR K$>"5" GOTO 644         :rem 134
647 PRINT K$;:N=ASC(K$)-38                       :rem 204
648 GET K$:IF K$=CHR$(13) THEN RETURN             :rem 43
649 IF K$=CHR$(20) THEN PRINT K$;:N=1:GOTO 644
                                                 :rem 255
650 GOTO 648                                     :rem 118
660 NS=0:SN=0:AS=BA+2:SL=0:SL%(0)=0:TA=AS:RETURN
                                                 :rem 201
670 K=FRE(0):WAIT 198,15:GET K$:IF K$<" " OR K$>"Z
    " OR LEN(F$)=12 GOTO 674                       :rem 8
```

```
672 F$=F$+K$:PRINT K$;:GOTO 670                    :rem 203
674 IF K$=CHR$(20) AND F$>"" THEN PRINT K$;:F$=LEF
    T$(F$,LEN(F$)-1):GOTO 670                      :rem 212
676 IF K$<>CHR$(13) GOTO 670                       :rem 167
678 RETURN                                         :rem 133
680 POKE 648,4:PRINT "{WHT}":POKE 53269,0:RETURN
                                                   :rem 199
690 POKE 53269,3:POKE 648,192:PRINT "{HOME}{43}":RE
    TURN                                           :rem 216
700 P=PEEK(SA):IF P=4 THEN F$="FILE NOT FOUND":GOT
    O 710                                          :rem 106
702 IF P=5 THEN F$="DEVICE NOT PRESENT":GOTO 710
                                                   :rem 209
704 F$="ERROR"+STR$(ST)                            :rem 80
710 GOSUB 620:PRINT F$:WAIT 198,15:GET K$:GOTO 300
                                                   :rem 19
800 FOR K=50370 TO 51421:READ P:POKE K,P:NEXT
                                                   :rem 245
810 DB=50304:MV=50376:E=50367:DEF FNH(N)=INT(N/256
    ):DEF FNL(N)=N-256*FNH(N)                      :rem 209
815 SYS 51149:SYS MV,53248,2048,53248:FOR K=53464
    {SPACE}TO 53503:READ P:POKE K,P:NEXT           :rem 124
820 SYS 51164:POKE 648,192:PRINT "{CLR}{WHT}{H}":S
    YS 51272                                       :rem 82
835 FOR K=1 TO 21:PRINT "{2 SPACES}{RVS}[£[£[£[
    £[£[£[£[£[£[£[£[£[£↑":NEXT                     :rem 206
840 PRINT "{2 SPACES}{RVS}]]]]]]]]]]]]]]]]]]]]]]]
    ←{43}{HOME}"                                   :rem 64
845 FOR K=1 TO 8:PRINT TAB(28) "{10 SHIFT-SPACE}":
    NEXT                                           :rem 114
848 PRINT:FOR K=1 TO 12:READ F$:PRINT TAB(28) F$:N
    EXT                                            :rem 13
850 SYS 51246:SYS MV,DB,63,50176:SYS MV,DB,63,5024
    0                                              :rem 58
860 FOR K=0 TO 18 STEP 3:READ P,Q,R:POKE 50176+K,P
                                                   :rem 183
865 POKE 50240+K,Q:POKE 50241+K,R:NEXT             :rem 146
869 MX=255:DIM SL%(MX)                             :rem 243
870 BA=PEEK(49)+256*PEEK(50)+500:MT=PEEK(51)+256*P
    EEK(52)-500:GOSUB 660                          :rem 170
890 POKE 53269,3:GOTO 300                          :rem 62
900 DATA 0,170,1,14,2,7,32,70,197,132,251,133,252,
    32,70,197,140,68,197,141,69                    :rem 203
901 DATA 197,32,70,197,132,253,133,254,165,253,56,
    229,251,170,165,254,229,252                    :rem 235
902 DATA 236,68,197,237,69,197,144,35,160,0,174,69
    ,197,240,14,177,251,145,253                    :rem 253
903 DATA 200,208,249,230,252,230,254,202,208,242,1
    74,68,197,240,8,177,251,145                    :rem 219
```

```
904 DATA 253,200,202,208,248,96,173,69,197,168,101
    ,252,133,252,152,24,101,254          :rem 218
905 DATA 133,254,172,68,197,240,8,136,177,251,145,
    253,152,208,248,174,69,197           :rem 206
906 DATA 240,15,198,252,198,254,136,177,251,145,25
    3,152,208,248,202,208,241,96          :rem 34
907 DATA 0,0,32,253,174,32,138,173,76,247,183,32,2
    00,196,32,70,197,132,251,133           :rem 4
908 DATA 252,169,0,160,2,145,251,136,16,251,76,47,
    198,169,0,162,2,157,191,196          :rem 233
909 DATA 202,16,250,32,200,196,76,47,198,162,2,189
    ,128,196,157,191,196,202,16          :rem 242
910 DATA 247,162,0,32,176,199,76,47,198,162,2,32,1
    76,199,162,2,189,191,196,157          :rem 52
911 DATA 128,196,202,16,247,76,47,198,160,27,162,3
    3,185,128,196,72,189,128,196          :rem 56
912 DATA 153,128,196,104,157,128,196,185,129,196,7
    2,189,129,196,153,129,196,104        :rem 110
913 DATA 157,129,196,185,130,196,72,189,130,196,15
    3,130,196,104,157,130,196,232         :rem 87
914 DATA 232,232,136,136,136,16,206,48,92,162,60,1
    72,194,196,189,128,196,10,62          :rem 31
915 DATA 130,196,62,129,196,62,128,196,136,16,240,
    202,202,202,16,232,48,64,162          :rem 12
916 DATA 60,172,194,196,189,130,196,74,126,128,196
    ,126,129,196,126,130,196,136          :rem 53
917 DATA 16,240,202,202,202,16,232,48,36,160,60,18
    5,128,196,32,142,199,72,185          :rem 219
918 DATA 130,196,32,142,199,153,128,196,185,129,19
    6,32,142,199,153,129,196,104          :rem 50
919 DATA 153,130,196,136,136,136,16,222,169,42,133
    ,97,169,216,133,98,169,2,133          :rem 36
920 DATA 99,173,194,196,56,42,133,254,169,23,133,1
    00,164,99,185,128,196,133,253         :rem 95
921 DATA 162,8,165,253,37,254,168,185,196,196,164,
    100,145,97,172,194,196,240,9          :rem 52
922 DATA 198,100,70,253,202,164,100,145,97,198,100
    ,70,253,202,208,223,198,99           :rem 174
923 DATA 165,100,16,208,165,97,24,105,40,133,97,14
    4,2,230,98,165,99,24,105,6           :rem 174
924 DATA 133,99,201,65,208,182,96,32,130,199,165,1
    98,240,11,162,63,189,127,196          :rem 44
925 DATA 208,3,202,208,248,96,173,0,220,73,15,41,1
    5,240,6,32,40,199,32,130,199           :rem 0
926 DATA 173,0,220,41,16,208,219,173,38,199,74,74,
    74,133,254,173,39,199,10,109          :rem 33
927 DATA 39,199,101,254,168,173,38,199,41,7,174,19
    4,196,208,10,170,185,128,196          :rem 58
928 DATA 93,26,199,76,251,198,74,170,189,34,199,17
    0,45,195,196,133,254,138,57           :rem 28
```

```
929 DATA 128,196,197,254,208,5,89,128,196,176,8,13
    8,73,255,57,128,196,5,254,153          :rem 124
930 DATA 128,196,32,47,198,32,130,199,173,0,220,41
    ,16,208,130,173,0,220,73,15            :rem 203
931 DATA 41,15,240,240,32,40,199,76,181,198,128,64
    ,32,16,8,4,2,1,192,48,12,3,0           :rem 249
932 DATA 0,172,39,199,74,144,5,136,16,2,160,20,74,
    144,7,200,192,21,208,2,160,0           :rem 241
933 DATA 140,39,199,174,38,199,172,194,196,240,15,
    74,144,6,202,202,16,2,162,22            :rem 23
934 DATA 74,144,20,232,208,11,74,144,5,202,16,2,16
    2,23,74,144,6,232,224,24,208           :rem 240
935 DATA 1,170,142,38,199,173,38,199,10,10,10,105,
    41,141,0,208,173,39,199,10,10           :rem 44
936 DATA 10,105,59,141,1,208,96,162,8,165,162,197,
    162,240,252,202,208,247,96             :rem 178
937 DATA 174,194,196,208,11,162,8,10,102,253,202,2
    08,250,165,253,96,162,4,10,38           :rem 57
938 DATA 254,10,102,253,70,254,102,253,202,208,243
    ,165,253,96,169,128,133,251            :rem 216
939 DATA 133,253,169,196,133,252,133,254,169,131,1
    49,251,169,63,141,68,197,169            :rem 55
940 DATA 0,141,69,197,76,223,196,173,14,220,41,254
    ,141,14,220,165,1,41,251,133           :rem 249
941 DATA 1,96,165,1,9,4,133,1,173,14,220,9,1,141,1
    4,220,96,173,194,196,170,73,1           :rem 42
942 DATA 141,194,196,24,105,16,168,189,42,200,141,
    28,208,32,205,199,189,44,200            :rem 22
943 DATA 162,6,157,225,208,202,16,250,32,220,199,7
    8,38,199,14,38,199,140,248             :rem 194
944 DATA 195,32,107,199,173,197,196,174,198,196,14
    2,197,196,141,198,196,76,47             :rem 40
945 DATA 198,2,0,255,127,169,0,160,62,153,128,196,
    136,16,250,96,32,46,200,32             :rem 170
946 DATA 200,196,76,47,198,32,46,200,76,47,198,120
    ,169,217,141,40,3,169,200,141           :rem 74
947 DATA 41,3,88,173,2,221,9,3,141,2,221,173,0,221
    ,41,252,141,0,221,169,4,141            :rem 176
948 DATA 24,208,169,11,141,32,208,169,1,141,33,208
    ,169,0,141,34,208,141,35,208             :rem 0
949 DATA 141,39,208,141,27,208,141,28,208,169,91,1
    41,17,208,169,16,141,248,195            :rem 35
950 DATA 169,18,141,249,195,169,8,141,2,208,169,2,
    141,23,208,141,29,208,141,16            :rem 18
951 DATA 208,169,66,141,3,208,169,14,141,40,208,16
    9,2,141,37,208,169,7,141,38            :rem 227
952 DATA 208,76,107,199,120,169,237,141,40,3,169,2
    46,141,41,3,88,173,2,221,9,3            :rem 16
953 DATA 141,2,221,173,0,221,9,3,141,0,221,76,129,
    255,169,255,201,127,96                 :rem 212
```

```
960 DATA Ø,127,127,127,127,127,127,127,Ø,127,127,1
    27,127,127,127,127,Ø,255,255              :rem 5
961 DATA 255,255,255,255,255,127,127,127,127,127,1
    27,127,127,127,255,255,255               :rem 196
962 DATA 255,255,255,255                     :rem 175
97Ø DATA EDIT,COPY,LOAD,SAVE,DATA,QUIT,,FLIP,INVER
    T,MULTICOLOR,WIDTH,HEIGHT                :rem 241
98Ø DATA 254,255,254,198,192,6,198,192,6,198,192,6
    ,198,192,6,198,192,6,254,255              :rem 76
981 DATA 254                                  :rem 87
```

Plug a joystick into port 2 and run the program. It takes a moment to initialize. You'll see a 24 × 21 grid on the left side of the screen. In the upper-right corner will be a black square, and below that a list of some of the editing commands. At the bottom of the screen the message EDITING 0 appears, indicating that you're currently editing definition number 0.

Joystick Editing

The cursor is displayed in the grid as a small box with thick edges. Push the joystick in any of the eight directions to move the cursor from one grid square to another. If you move the cursor off one side of the grid, it wraps around to the other side.

Press the joystick button to fill a square. To fill an area or draw a line, hold down the button while you push the joystick. Erasing is just as simple; move the cursor to the desired square and press the button again. If you hold the button down and move the cursor over filled areas, you can erase entire sections.

As you edit on the large grid, you'll notice that the shape is displayed as an actual sprite in the black square at the top-right corner of the screen. This can give you an idea of what the sprite will look like in a BASIC program.

The remainder of this chapter explains each of the Sprite Editor's features and commands. As you read, try them out with a shape you've drawn on the editing grid.

The Sprite Editor Commands and Features

Flip and Invert. To flip the current definition left to right, press the F key. This gives you a mirror image of the definition. Of course, the change will not be noticeable if the definition is symmetrical. Press the I key to invert the definition. This turns the definition upside down.

Scrolling. Use the cursor keys (along with the SHIFT key to move up or left) to scroll the shape in any of four directions.

Insert. Sometimes it's necessary to expand or contract a sprite definition. To expand it, move the cursor to the row where you want the expansion to occur and press the SHIFT and INST/DEL keys at the same time. The row on which the cursor is positioned, and all the rows below it, will scroll down one row to create a gap. The bottom row will scroll off the grid and will not be recoverable.

Delete. This contracts a sprite definition. Press the DEL key (unSHIFTed INST/DEL) to delete the row on which the cursor is currently positioned. All the rows below the cursor will scroll up one row to fill in the gap.

Erase. To clear the editing grid, press SHIFT–CLR/HOME and then hit the Y key in response to the ERASE? prompt. If you accidentally hit SHIFT–CLR/HOME, press the N key to cancel the clear.

Change color. You can change the color of the sprite at any time. This color, however, does not necessarily have to be the same color that will be used to display the sprite in BASIC.

To change the current color, press the f2 key (SHIFT and f1 together). When the prompt ENTER COLOR: appears, type a color number from 0 to 15. The color numbers are listed in the *Commodore 64 User's Guide.* The color changes as soon as you press the RETURN key. Notice that the color changes in both the editing grid and in the shape box in the upper-right corner.

If you decide not to change the color, just press the RETURN key without entering a number. The color will not be changed.

You can also change the background color of the box in the upper right. Press the f8 key (SHIFT and f7 keys) and enter the number of the desired color. This feature shows you how a sprite will look on different background colors.

To set the box background back to black, press the f8 key and enter 0.

Width and Height. Press the W key to make the sprite shown in the upper right alternate between normal and double width. Press the H key to alternate the sprite between normal and double height. You can use these keys in combination to

reduce or enlarge the sprite in either, or both, directions.

Like the color change feature, this is provided for editing purposes only. The size chosen in the Editor has no effect on the size used when the sprite is displayed by a BASIC program.

Multicolor. The Sprite Editor also supports multicolor mode editing. In multicolor mode, a sprite can display up to three different colors, instead of just one. The only trade-off is that the horizontal resolution of the sprite is reduced from 24 to 12 pixels across.

Press the M key to switch to multicolor mode. The grid will change to show a grid 12 × 21. Any previous drawing will appear as garbage, so you may also want to press SHIFT–CLR/HOME.

Push the joystick and press the trigger as normal to fill in the squares with the first color. To change to the second color, press f3. All subsequent drawing is now done in the second color. Press the f5 key to switch to the third color. When you want to return to the first color, press the f1 key.

Key effect. f1: Draw using first color; f3: Draw using second color; f5: Draw using third color.

If you want to change one of these colors, just press SHIFT with the corresponding function key. Recall that you pressed f2 (SHIFT and f1) to change the first color when you were in normal color mode. Press f4 (SHIFT-f3) or f6 (SHIFT-f5) to change the second or third color.

Key effect. f2: Change first color; f4: Change second color; f6: Change third color.

Changing a color does not change which color is currently used for drawing. This means that if you're filling in squares with the first color and then press f6 to change the third color, the third color will change, but drawing will continue with the first color.

Press the M key a second time to turn off the multicolor mode. The drawing in the grid will appear distorted again.

Edit. The Sprite Editor lets you edit multiple definitions. In fact, the program can handle up to 256 definitions, but it's doubtful that you'll ever need that many.

The Edit feature is used to switch from one definition to another. The program won't let you use this feature until at least one of the squares in definition 0 has been filled. Once definition 0 has been edited, definition 1 can be accessed.

To switch from one definition to another, select the Edit feature by pressing the E key. Now as you push the joystick left or right, the grid will switch between definitions 0 and 1. Definition 1 will be blank because it has not been edited yet. To edit definition 1, push the stick until definition 1 appears, press the joystick button to select that grid, and then begin editing. The message at the bottom of the screen should now read EDITING 1.

As soon as you've filled some of the squares in definition 1, you can press the E key and push the stick left or right to select definition 0, 1, or 2. You can go back to definition 0, continue editing definition 1, or create a new definition 2. Just press the joystick button when you see the desired definition.

After definition 2 has been defined, definition 3 can be accessed, and so on. This process of creating new definitions one at a time can be continued for as long as necessary.

When you have a sequence of definitions, press the E key and push the joystick left or right to cycle through them all. This creates an animation effect. For example, you could define a sequence of definitions to show a person walking. The person's legs are in a slightly different position in each definition. Pressing the E key and cycling through the definitions will make the person appear to walk.

The number of definitions is reduced whenever the last one is erased. For instance, if definitions 0 through 6 have been edited, pressing the E key lets you cycle through definitions 0 to 7. Definition 7 will be blank because it has not been defined yet. But if definition 6 is erased, it will be blank, so the Edit feature will only cycle through definitions 0 to 6.

This happens only when the last definition is erased. If definitions 0 to 6 have been edited and you erase definition 3, you'll still be able to access all the definitions. Definition 3 will just be blank.

One final note is that when you press the E key, the other editing features won't work until you select a definition. If it seems that the keyboard has locked up, it's probably because you forgot to press the joystick button to select a definition.

Copy. Use this feature to copy a definition into the current definition space. This can be a very handy feature when two definitions are very similar to each other. For example, definition 0 may show a figure walking to the left. You want definition 1 to show the same figure, only walking to the

right. Instead of drawing definition 1 from scratch, it's a lot easier to copy definition 0 into definition 1 and then use the F key to flip the definition.

To use the Copy feature, make sure you're in the destination definition, the one that is going to be changed. You should see the prompt EDITING *n*, where *n* is the number of the definition. Press the C key and push the stick left or right. When you come to the source definition, the one you want to copy to the destination definition, press the joystick button. You'll now be editing the destination definition. (The source definition has not been altered.)

Let's run through an example. You want to copy definition 2 into definition 4. Definition 4 is the destination, definition 2 is the source. Make sure you're currently editing definition 4. If not, use the Edit feature to select it. Then press the C key, push the joystick until definition 2 appears, and press the button. Definition 4 now matches definition 2. Definition 2, remember, has not been changed.

Save. The Save feature is used to store the current set of definitions as a tape or disk file. This file can later be loaded by a BASIC program, or retrieved for more editing by the Editor.

To save the current definitions, press the S key and enter a filename up to 12 characters long. You may have noticed in Chapter 12 that the definition filenames have the extension .DEF—don't include this in the filename. It's automatically added by the Editor.

If you're saving the file to disk, be sure that the filename is not already in use to avoid the FILE EXISTS error. If an error does occur, the drive light will start to flash. The only error reported by the program is the DEVICE NOT PRESENT error.

The normal PRESS PLAY & RECORD ON TAPE prompt will not appear if you're using the Datassette, so you'll have to remember to press both the buttons yourself. Also make sure that the tape has been advanced to an unused area so that you're not overwriting another program you may want intact.

If you accidentally press the S key and don't want to save a file, just press the RETURN key without entering a filename.

When you are defining a lot of shapes, it's a good idea to periodically save your work. There may be a power failure, or surge, which could erase all your work currently in memory.

Load. Use this feature to retrieve a set of definitions stored on tape or disk by the Save feature.

Since loading a new set of definitions erases those already in memory, you may want to save the current definitions before using Load.

To load a definition file, press the L key and enter the desired filename. The requested file is loaded into memory, replacing the current definitions. If you're using disk, the error FILE NOT FOUND will be reported if the requested file is not on the disk.

If you're using the Datassette, there will be no PRESS PLAY ON TAPE prompt, so you must remember to press the PLAY button on the Datassette. Be careful that you don't also press the RECORD button.

If you accidentally press the L key and don't want to load a new file, just press the RETURN key without entering a filename. The current definitions will not be erased.

(Note: The Load feature can be used to load the definition files used by the demonstration programs you earlier typed in from Chapter 12.)

Errors. If you get an error while using the Sprite Editor, such as DEVICE NOT PRESENT or FILE NOT FOUND, the program will wait for you to press a key to acknowledge the error. You can then continue editing, or try to load or save again.

Data. This feature lets you save the definition information to tape or disk in the form of DATA statements. The DATA statements can then be merged with a BASIC program.

To create the file of DATA statements, press the D key. The Editor will ask for a starting line number, the line number used for the first DATA statement. Line numbers will be incremented by one for the succeeding DATA statements.

After you enter the starting line number and press RETURN, the Editor begins building the DATA statements in memory. The line number is displayed as the line is built. This process takes awhile.

The Sprite Editor then asks for a filename before it saves the file. Type in a filename (up to 12 characters long). The Editor automatically adds the extension .DAT to the filename and saves the file to tape or disk. As with the Save feature, you should choose a filename not already in use on your disk. Since the prompts do not appear, remember to press both the

PLAY and RECORD buttons if you're using the Datassette.

The file created by the Data feature is an actual program which can be loaded in BASIC. A merge utility is provided in Appendix E to help you append this data file with other programs.

If the Editor reports the error NOT ENOUGH MEMORY while it's building DATA lines, it means that the current definition file is too large to be converted. Use Program 16-2, "Extract" to break the file into two parts, then convert each to DATA statements. Be aware, however, that the DATA lines for large definition files eat up memory very quickly and are usually not practical.

The Load feature can load only definition files stored by the Save feature and cannot be used to load definition files stored as DATA statements by the Data feature. Therefore, whenever you store the definitions as a file of DATA statements, it's a good idea also to save the definitions with the Save feature, in case you want to modify them later.

To cancel the Data feature, press the RETURN key instead of entering either the starting line number or the filename.

The advantage of storing sprite definitions as data in a program is that when the program runs, one less file has to be loaded. Each time you ran Program 12-3, "DEMO#1," for instance, it had to load both "SCS.OBJ" *and* a definition file. If you're using the Datassette, it can take awhile for these two files to be found and loaded. Storing the definitions in DATA statements in the program eliminates the need for loading the second file.

If you choose not to use the Sprite Control System machine language routines for sprite motion, you can still use the Sprite Editor to create definitions and convert them to DATA statements. The format used to store definitions as data is very simple.

Here's a sample sprite definition data file. It's what you could use in your own BASIC program to recreate the bunny shape you saw in Chapter 12.

The first data number is the number of definitions minus one. For example, if you have a set of four definitions, numbered 0 to 3, the first data number will be a 3. Notice that in Program 13-2, the first number is 0. It indicates that there's only one definition.

Program 13-2. Bunny Data

```
3000 DATA 0,63,12,0,48,15,129,240,12,195,48,12,102
     ,48,7,102,224,3,231,192,0,195
3001 DATA 0,1,255,128,3,24,192,7,255,224,12,60,48,
     15,0,240,7,255,224,0,255,0,28
3002 DATA 60,56,7,60,224,1,255,128,48,126,12,124,6
     0,62,127,60,254,255,255,255
```

The next number specifies the number of bytes in the first definition. This number can range from 0, if the definition is empty, to 63, if the definition is full. The bunny definition is full, for it has 63 bytes. If the definition is not empty, the following data numbers form the actual definition. If the length is less than the full 63 bytes, the remaining bytes in the definition block should be set to 0.

Assume that the second data number indicates the definition is 43 bytes long. The next 43 numbers form the sprite definition, and the remaining 20 bytes in the definition block would all be set to 0.

This format, a length byte followed by 0 to 63 definition bytes, is repeated for each definition.

Quit. When you're finished editing and have saved your work, press the Q key and type Y in response to the QUIT? confirmation prompt. The program ends and returns you to BASIC. If you accidentally press the Q key and don't want to quit the Editor yet, just press N to cancel.

Note that the RUN/STOP key has been disabled to prevent you from accidentally stopping the program without having a chance to save your created definitions or convert them to DATA statements. The only way to exit the Sprite Editor is to use the Quit command.

Sprites and More Sprites

What sprites you design with the Editor are, of course, up to you. The Editor is only a tool. It can't design the sprites for you, it can only make that design easier. That's the purpose of this program, and it fills that purpose very well. You can create several sprites with the Editor in the time that it used to take you to design just one on paper. Better yet, since the Editor is so easy to use, and because it does so much for you, you

can almost forget about the mechanics and concentrate just on sprite design. In a matter of a few minutes, you can have a handful of intricate sprite shapes ready to be placed in motion or animation.

To do that, however, you need to know more about the Sprite Control System, the machine language routines which make it simple to move and control sprites from BASIC programs. That's what Chapter 14 is all about.

Chapter 14

Using the Sprite Control System in BASIC

Moving and controlling sprites in a BASIC program can be a difficult task. Simply getting a sprite across the invisible "seam" on the screen means POKEs and constant checking. Even more complicated sprite movements are almost impossible in BASIC, especially if you're planning on having more than one or two sprites on the screen at the same time. The more sprites, the more time it takes for your Commodore 64 to execute the commands. In BASIC, this can slow a program down to a crawl.

That's why you see so many programs with sprites written in machine language. Machine language gives you the speed to move all eight sprites on the screen at once, without slowing anything down. Unfortunately, not everyone knows how to write machine language programs. If you're like most people, you learned to program in BASIC. And even if you do know machine language programming, BASIC is often the language of choice, for it's simple to write.

With the Sprite Control System, first demonstrated in Chapter 12, you can move and control sprites with the speed of machine language, but from the convenience of your own BASIC program. Just like the demonstration programs in Chapter 12, your BASIC programs, whether they're spreadsheets or arcade games, can call on the Control System.

Sophisticated

The Sprite Control System, sophisticated as it is, requires you to follow a very specific procedure when you use it with a BASIC program. Because of the Control System's power, this may seem complicated at first. It *can* be a bit involved. But with some practice, you'll soon find it second nature.

This procedure consists of calling two subroutines in BASIC and using several SYS statements. To help you learn this procedure, we'll go through it step by step in the immediate mode (without using line numbers). As you become more familiar with them, the steps will seem both simple and logical.

Every program that uses the Sprite Control System must contain the lines in the "SCS.BAS" file (Program 12-1), so before you begin, load these lines into memory with the command:

```
LOAD "SCS.BAS"
```

Make sure you add 8 if you're using a disk. You'll now execute several statements in the immediate mode, in the order that they would actually be used in a program calling the Sprite Control System.

Variable Assignments

The first step is to perform the standard assignments that have been used throughout this book in preparation for SYS calls and disk and tape file input/output (I/O). Enter the following statement. As always, change the DN=8 to DN=1 if you're using the Datassette.

```
DN=8:SA=780:SX=781:SY=782:SP=783
```

Load Machine Language Routines

Now that these variables have been assigned, you can load the "SCS.OBJ" file (Program 12-2), which contains the machine language routines. To do this, make sure that the program is on the disk currently in the drive or that the Datassette contains the tape with the program. Type in the following line. It calls the subroutine at line 58000.

```
GOSUB 58000
```

Besides loading the SCS.OBJ file, this subroutine assigns several more variables that will be used later in SYS statements.

Load Definitions

You also need to load a definition file. For this demonstration, use the bunny definition from Chapter 12. To load the definition file, you must assign the filename to a string variable, specify a LOAD address, and call a subroutine.

Assign the name of the definition file to the variable F$. Do not include the .DEF extension as part of the filename. Assuming that you called the file BUNNY, you'd type:

```
F$="BUNNY"
```

The LOAD address is set to be in the middle of free memory with the statement:

LA=16384

Finally, call the subroutine at 58500 to load the definitions with:

GOSUB 58500

You've just loaded the "BUNNY.DEF" file.

Install Sprite Control System

Next, use the following SYS statement to install the Control System. This SYS call establishes a raster scan interrupt and initializes sprite graphics.

SYS IN

This SYS call should be done only once, and must be done before you use any of the Control System SYS calls described below.

In case you're wondering, the variable IN, which stands for INstall, was assigned by the subroutine at 58000. You could have used SYS 49664 instead, the value assigned to IN. But it's much easier to remember variable names than the corresponding numbers.

The only problem is that you must type the variable name exactly. Remember that all variables have an initial value of 0. If you make a mistake and type a variable name which has not been assigned, the address for the SYS statement will be interpreted as location 0, and the computer may crash. The keyboard may lock up, and the screen may be altered.

If the computer does crash, no permanent damage has been done. Just turn the power off and back on, and start over from the beginning of the procedure. To avoid this inconvenience, be careful as you're typing.

(Note: Since entering a program line clears all variables, do not enter any program lines while working in the immediate mode.)

Define Block

To make a sprite appear, you have to give it a shape. Unfortunately, it's not possible to directly assign one of the definitions loaded from the definition file to a sprite. Instead, you have to copy a definition into a definition block, and *then* assign the definition block to the sprite.

Sprite memory is divided into 64-byte sections. These sections are called *definition blocks* and are numbered from 0 to 255. Each block can hold one definition.

To define a block with one of the definitions loaded in the file, you'll use the following SYS statement format:

SYS DF,*block number,definition number*

(The SYS statement normally allows only one number after the keyword, but the machine language routines of the Sprite Control System have been designed so that several numbers can be used.)

Let's assume that block 255 is available. The BUNNY.DEF file contains only one definition, numbered 0, so to define block 255 with definition 0, enter the statement:

SYS DF,255,0

Definition 0 has now been copied into block 255. Block 255 is ready to be assigned to a sprite.

For now, you'll be using only block 255. A more complete discussion on which definition blocks are available is provided in the next chapter.

The preceding SYS calls have been general system calls. The SYS IN installs the Sprite Control System so that sprites can be used, and SYS DF defines blocks that can be used by all of the sprites.

Now that the general SYS calls have been issued, you can take a look at SYS statements which pertain to an individual sprite. Each of the following SYS statements contains a sprite number to identify which sprite is to be affected. The letters *SN* represent this Sprite Number. For purposes of demonstration, you'll use only sprite 0.

Assign Block to Sprite

The following statement format assigns a block to a sprite to give it a shape and make it appear.

SYS BL,SN,*block number*

You've already defined block 255, so to make sprite 0 appear, enter:

SYS BL,0,255

The sprite should appear in the upper-left corner of the screen. If it didn't display, you've made an error somewhere in

the procedure. Double-check what you typed in (much of it should still be on the screen) or begin again.

Set Sprite Color

The format to set a sprite's color is:

SYS CO,SN,*color number*

where *color number* ranges from 0 to 15. The default color of sprite 0 is white, or color 1. Type in the following to change sprite 0's color to light red (color 10). You can find the color numbers listed in the *Commodore 64 User's Guide.*

SYS CO,0,10

Set Sprite Size

One of the interesting features of sprites on the Commodore 64 is the ability to enlarge, both horizontally and vertically, a sprite shape. To do this with the Sprite Control System, use the following statements:

SYS WI,SN,0 to select normal width (default)
SYS WI,SN,1 to select double width

SYS HE,SN,0 to select normal height (default)
SYS HE,SN,1 to select double height

For instance, to expand sprite 0 to twice its normal width, just type:

SYS WI,0,1

When you double the width of a sprite, the expansion always occurs to the right. Doubling the height makes the sprite expand downward. Let's expand the bunny's height as well with this statement:

SYS HE,0,1

Set Sprite Position

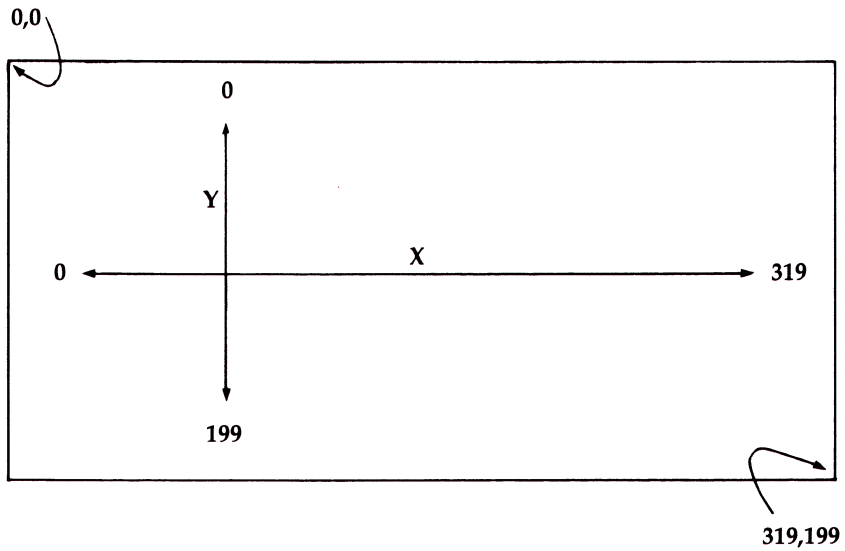
Another vital part of sprite manipulation is its position. You have a huge number of possible positions for any sprite—62,400 to be exact. Placing a sprite at any of these locations insures that at least part of the sprite will appear on the screen.

To position a sprite, you'll use this format:

SYS PO,SN,*X,Y*

where X and Y represent points on a coordinate system. This system is the same as the one used for bitmapped graphics in Part 2. Take a look at Figure 14-1 to see how the coordinates are mapped out.

Figure 14-1. Sprite Position Coordinates



The coordinates you specify refer to the upper-left corner of the sprite. For example, if a sprite is moved to the position 80,120, the upper-left pixel of the sprite appears at this position. The rest of the definition is displayed below and to the right of this location.

The default position of a sprite is 0,0, known as the *home position.* Let's move sprite 0 out of the home position. To move sprite 0 to the center of the screen, enter this statement.

```
SYS PO,0,160,100
```

Remember that the position refers to the upper-left corner of the sprite. That's why the sprite may not seem to be actually centered on the screen.

Type in this statement to move sprite 0 to the right edge of the screen.

```
SYS PO,0,319-24,100
```

The 24 was subtracted from the rightmost position to account for the sprite width. If the X coordinate was 319, only the leftmost edge of the sprite would be visible. But because you earlier expanded the sprite horizontally, you still see only part of the shape. Reduce the sprite to normal size by entering:

SYS WI,0,0

and

SYS HE,0,0

Now the entire sprite should be visible. The rightmost edge of the shape just fits on the screen.

This statement places sprite 0 in the lower-right corner of the screen.

SYS PO,0,319−24,199−21

To make sure all the sprite stayed on the screen, 21 was subtracted from the maximum Y coordinate. The numbers which are subtracted need to be adjusted if the sprite is displayed with double width or height.

Negative values or values greater than 319 or 199 may be used to position a sprite so that it's partially off the screen. When this is done, the part off the screen is covered by the border.

SYS PO,0,−10,−10

You can even make the sprite disappear from the screen display area altogether. This feature can be used to make a sprite appear to vanish without actually disabling or turning off the sprite. Try this:

SYS PO,0,−10,−30

Now the bunny is completely hidden. You can bring it back on the screen with something like:

SYS PO,0,160,100

All of the above SYS calls have dealt with a sprite's appearance—how and where the sprite is displayed. The following SYS calls deal with motion.

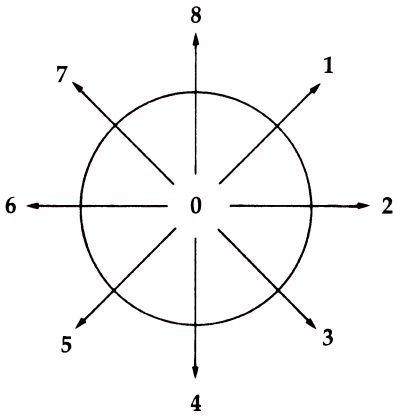
Set Sprite Direction

Moving the sprite is just as simple as placing it. The movement statement format is:

SYS DI,SN,*direction number*

where *direction number* refers to one of the eight directions. Figure 14-2 illustrates the directions and their numbers.

Figure 14-2. Sprite Direction Numbers



Go ahead and enter this statement. Try several different direction numbers. To make sprite 0 move up and to the left, for instance, type:

```
SYS DI,0,7
```

You can change the direction at any time. To stop a sprite's movement, use 0 for the direction number.

Select Wraparound or Bounce

You've already noticed that the screen edges act as boundaries which limit a sprite's motion. When a sprite reaches one of these boundaries, it stops moving.

You may elect instead to have the sprite wrap around to the opposite side of the screen when it reaches a boundary. To select the wraparound option, you'll use the following format:

```
SYS WR,SN,1
```

Thus, to select the wraparound option for sprite 0, you'd use:

```
SYS WR,0,1
```

Sprite 0 will now move continuously, as long as the direction set by SYS DI is a value from 1 to 8.

The wraparound option is turned off by:

SYS WR,SN,0

Another boundary option is bounce. With this option selected, a sprite reverses direction whenever it hits a boundary. This was used in Program 12-3, "DEMO#1."

Turning on the bounce option requires this format:

SYS BO,SN,1

Selecting bounce for sprite 0 just needs:

SYS BO,0,1

Bounce is turned off by:

SYS BO,SN,0

The wraparound and bounce options cannot be used at the same time. Turning on one turns off the other.

For the next few examples, the sprite needs to continue moving, so leave the bounce option turned on.

Set Sprite Speed

When a sprite's direction is other than 0, it appears to move because its position is changed at regular intervals. This interval is called a *frame* and lasts 1/60 second (1/50 second on European Commodore 64s). A value is added to either or both of the sprite's coordinates each frame, depending on the current direction.

If this value is 1, the sprite moves one position every frame. If the value is larger, the sprite moves several positions each frame so that it appears to move faster. To control the speed of a sprite, then, all you have to do is specify how far it should be moved each frame.

To set the speed of a sprite, you'll use this format:

SYS MO,SN,*delta X*,*delta Y*

The *delta* values are the offsets added to the position coordinates each frame, and can range from 0 to 7. The larger the delta value, the further the sprite moves in that direction each frame, so the faster it appears to move.

The default values are 2 for delta X and 1 for delta Y. To see sprite 0 move a little faster, enter:

SYS MO,0,3,3

A sprite can move extremely fast. Enter this statement to see sprite 0 move at top speed.

SYS MO,0,7,7

It's almost out of control!

A delta value of 0 prevents a sprite from moving in the corresponding direction. Let's try it out. The next statement sets the delta Y value at 0, so the sprite never moves vertically, no matter which direction number is used. All the sprite does is bounce back and forth across the screen.

SYS MO,0,3,0

Slow the sprite down by typing:

SYS MO,0,1,1

This is pretty slow, but for some applications this speed may still not be slow enough. To make the speed even slower, add a *,1* to the end of the SYS MO statement. This selects the half-speed option.

SYS MO,0,1,1,1

When the half-speed option is in effect, a sprite moves every other frame. For the best results, use this option only with delta X and delta Y set to 1. Motion appears too jerky when other delta values are used.

To return to normal speed, either remove the *,1* or change it to 0.

SYS MO,0,1,1

or

SYS MO,0,1,1,0

Though it might be easier to remember the variable name SP (for SPeed), that variable is already used for tape and disk I/O SYS calls. The variable name MO, representing MOtion, is used instead.

Set Sprite Boundaries

The movement boundaries are normally set at the screen edges, but they can be changed. To modify a boundary, use one of these SYS statement formats.

SYS Call	**Boundary**
SYS LB,SN,*X*	Left
SYS RB,SN,*X*	Right
SYS TB,SN,*Y*	Top
SYS BB,SN,*Y*	Bottom

X and *Y* refer to the normal coordinate locations. If you wanted to move the top boundary for sprite 0 down to the middle of the screen, for example, you could use:

SYS TB,0,100

Sprite 0 is now confined to the bottom half of the screen.

The default values for the boundaries are:

Boundary	Default Values
Left	0
Right	295 (319–24)
Top	0
Bottom	178 (199–21)

As with the SYS PO statement, values are subtracted to take into consideration a sprite's width or height. These values may have to be adjusted in some cases. Negative boundary values are also allowed.

Remove Sprite Control System

To turn off sprite graphics and remove the Sprite Control System from the raster scan interrupt processing, simply enter:

SYS RE

This call undoes everything done by the SYS IN. Tape and disk I/O will not work properly when the Sprite Control System is installed, so you must remember to remove the system when you've finished using sprites. Once the Control System has been removed, it can be reinstalled by using SYS IN again.

The SYS RE should be done only after the Control System has been installed, and should not be done a second time unless the system is installed again.

Hitting RUN/STOP–RESTORE also removes the Sprite Control System. If you hit this key combination, you must reinstall the Control System in order for motion to work.

Review

Here's a quick review of the whole procedure. You might find this checklist useful as a reminder once you're actually trying to place the SYS calls in your own program.

1. Make sure the lines from the SCS.BAS file (Program 12-1) are included as part of your BASIC program.
2. Assign variables and load the SCS.OBJ file and a definition file.

DN=8:SA=780:SX=781:SY=782:SP=783
GOSUB 58000
F$=*filename*:LA=16384:GOSUB 58500

3. Install the Sprite Control System as part of regular interrupt processing and define a block with a definition from the definition file.

   SYS IN
   SYS DF,*block number,definition number*

4. Use the following SYS calls, in any order, to make a sprite appear and to set its color, size, position, direction, speed, boundaries, and boundary options.

   Appear—**SYS BL,SN,*block number***
   Color—**SYS CO,SN,*color number***
   Width—**SYS WI,SN,*width number***
   Height—**SYS HE,SN,*height number***
   Position—**SYS PO,SN,*X,Y***
   Direction—**SYS DI,SN,*direction number***
   Wraparound—**SYS WR,SN,*wraparound on/off***
   Bounce—**SYS BO,SN,*bounce on/off***
   Speed—**SYS MO,SN,*delta X,delta Y*** (,1 for half speed)
   Boundaries—**SYS LB,SN,*X*; SYS RB,SN,*X*; SYS TB,SN,*Y*; SYS BB,SN,*Y***

   It's suggested that you set values for color, size, and position before using SYS BL to make the sprite appear.

5. Remove the Sprite Control System when you are finished using sprites.

   SYS RE

   The procedure is really very simple. Only the description is long.

Errors

Some errors in SYS calls are detected. If you forget one of the required parameters, BASIC will report a SYNTAX ERROR. This also occurs if you try to use SYS DF before loading a definition file.

The ILLEGAL QUANTITY error may occur when a value is out of range. Using a sprite number out of the range 0–7 is one way of causing this error.

Multiple Sprites

Now that you've seen how to animate one sprite, let's see how easy it is to animate several.

Begin by reinstalling the Sprite Control System with:

SYS IN

The machine language routines and the definition file should still be in memory, which is why you don't need to load them again. In fact, block 255 should still be defined, so you can skip ahead and assign block 255 to sprite 0 to make the sprite appear.

SYS BL,0,255

The SYS IN resets all the default values, which is why the sprite appeared back in the upper-left corner, normal sized, and in white.

Set the sprite so that it moves continuously.

SYS DI,0,3:SYS BO,0,1

Though this is the first time you've put two SYS statements on one line, remember that it's always allowed, whether you're in program or immediate mode. Just separate the statements with the usual colon (:).

Now make sprite 1 appear:

SYS BL,1,255

and make it move continuously.

SYS DI,1,3:SYS BO,1,1

How about the rest?

FOR SN=2 TO 7:SYS BL,SN,255:SYS DI,SN,3:SYS BO,SN,1:NEXT

Wasn't that easy?

Take the time now to experiment with multiple sprites. When you're done, remember to remove the Control System.

SYS RE

What you've just done in the immediate mode can also be done in a program. It would be a good idea to go back and look at the listing of Program 12-3, "DEMO#1." You should now be able to understand everything that happens in the program. You may also want to run the program several times, changing some of the statements and adding new ones each time. One suggestion is to change the size of the sprites. If you do this, you will also need to change the right and bottom boundaries. Experiment with the Sprite Control System. The more you play with it, the more familiar you'll become with its powerful features and abilities.

SCS Advanced Features

There's more to the Sprite Control System than just positioning and moving sprites. What you saw demonstrated in Chapter 14 isn't all the Control System can do. Various advanced features, including vital elements like joystick control and automatic shape changing, are also available. As always, the best way to learn how these features work, and how to work them yourself, is through example. Let's take a look at the advanced capabilities of the Sprite Control System.

Available Definition Blocks

A definition block must be assigned to a sprite so that it can display a shape. As you saw both in Chapter 12 and in Chapter 14, all eight sprites can use the same definition block. (Remember the bunnies?) But if you want each sprite to display a different shape, each requires a different definition block. To do this, more blocks are needed.

There are 256 definition blocks, numbered 0–255. This would seem to be enough. The only problem is that many of these blocks are not available. Some are located in memory that's needed by the Commodore 64 Kernal or by BASIC. Others are located in memory which holds the current BASIC program. The remaining, located in BASIC free memory, are the ones safe to use.

Since the amount of free memory changes according to the size of the current BASIC program, the number of available blocks may vary. Higher numbered blocks are usually available, but lower numbered ones are not.

To find out exactly which are available, execute the following line anytime after most of the variables have been assigned and all arrays have been dimensioned.

```
PRINT INT((PEEK(49)+256*PEEK(50))/64)+16
```

This will tell you the number of the lowest block that can be used. All blocks from this number through block 255 should be available for use.

You'll probably never run out of definition blocks. A typical application may need 20, or at most 30, blocks. In Program 12-3, "DEMO#1," for instance, only one block is used, but the program is so short that blocks 81–255 are available. Program

12-5, "DEMO#2," is a little longer, but with blocks 96–255 available, there are still more than enough. The only time when you should be concerned about available blocks is when your program starts to get larger than 8K.

Just to be safe, it's a good idea to always work from the top down. If you need only 1 block, use block 255. If you need 5 blocks, use blocks 251–255. Or, if you need 16 blocks, use blocks 240–255.

A quick and easy way to define several blocks at once is to use a FOR-NEXT loop. If a definition file contains 16 definitions, a line like this copies them in order to 16 blocks.

```
FOR K=0 TO 15:SYS DF,240+K,K:NEXT
```

Block 240 gets definition 0, block 241 gets definition 1, and so on.

To assign one of these blocks to a sprite, just use the corresponding block number in the SYS BL call (see Chapter 14 for a description of this SYS statement). Here's how you would assign block 247 to sprite 5:

```
SYS BL,247,5
```

The block assigned to a sprite can be changed at any time. That's often how animation is done with sprites. When you have several blocks defined in a sequence, you can simply cycle through them. The second sprite in Program 12-5 alternated between two shapes like this. A bird could be made to fly by cycling through several sprite definition blocks, each showing the bird with its wings in a slightly different position.

Automatic Shape Changing

When a sprite moves in different directions, it makes the movement more realistic if the sprite displays a different shape for each direction. The autoshape feature can do this shape changing for you automatically.

To show how to use this feature, let's start with a sprite that can move in all eight directions. Eight definition blocks are needed, one for each direction. These blocks must be consecutive, and they must be defined in a particular order. This order is:

Block Definition	Direction
Base+0	Left
Base+1	Right
Base+2	Up

Base+3	Down
Base+4	Up and left
Base+5	Up and right
Base+6	Down and left
Base+7	Down and right

The number of the first block being used is called the *Base*, so if blocks 240–247 are being used, block 240 must contain the definition for left, block 241 must contain the definition for right, and so on.

To turn on the autoshape feature, use the following statement format (in program or immediate mode).

SYS AU,SN,*direction code,base block number*

Direction code specifies whether two, four, or eight directions are to be supported. The values 2, 4, or 8 indicate the number. In the example above, eight directions are supported, and the base block number is 240, so the SYS statement would look like this:

SYS AU,SN,8,240

The sprite shape will now automatically change to reflect the direction in which it's moving.

The autoshape feature does not always have to work with all eight directions. Only two directions, left and right, may be supported. The direction code for this is 2, and the order in which the blocks must be defined is:

Block Definition	**Direction**
Base+0	Left
Base+1	Right

To enable the autoshape feature using only these two directions (and assuming the Base is still block definition 240), you'd enter:

SYS AU,SN,2,240

Now, whenever the sprite moves in a leftward direction, whether it's left, up and left, or down and left, the shape for left displays. The shape for right appears when the sprite moves rightward. If the sprite moves straight up or down, the shape does not change from its previous shape.

To support only the four cardinal directions, use 4 for the direction code and define the blocks like this:

Block Definition	Direction
Base+0	Left
Base+1	Right
Base+2	Up
Base+3	Down

One other arrangement that's supported is two directions, up and down. The direction code for this arrangement is 6, and the blocks must be defined in this order:

Block Definition	Direction
Base+0	Up
Base+1	Down

Each of these four arrangements supports a centering option. With centering, one additional block is specified which is displayed by the sprite only when the sprite isn't moving.

To select centering, add 1 to the direction code, and put the centered definition in the last block. For example, to support eight directions with centering, the direction code would be 9 and block BASE+8 would hold the definition used when the sprite stops moving.

For reference, look to the following two tables.

Direction Code	Directions
2	Horizontal directions only
4	Four cardinal directions only
6	Vertical directions only
8	All eight directions

To support no motion (centering), add 1 to the direction code.

Direction Code	Order of Blocks
2	Left, right
3	Left, right, center
4	Left, right, up, down
5	Left, right, up, down, center
6	Up, down
7	Up, down, center
8	Left, right, up, down, up left, up right, down left, down right
9	Left, right, up, down, up left, up right, down left, down right, center

The autoshape feature is turned off with:

SYS AU,SN,0

Multicolor Mode

When a sprite is displayed in multicolor mode, it can consist of three different colors. The trade-off is that the sprite's horizontal resolution is reduced from 24 to 12 pixels. (For help in creating multicolored sprites, take a look at the Sprite Editor in Chapter 13.)

To turn the multicolor mode on or off, use these statements:

SYS MU,SN,0 Turn off multicolor mode (default)
SYS MU,SN,1 Turn on multicolor mode

The sprite's main color, the one displayed when the multicolor mode is turned off, is used as the first of the three colors. To set the additional colors, use the following variation of the SYS CO format:

SYS CO,SN,*first color,second color,third color*

Any combination of sprites may select the multicolor mode. When two or more are displayed in multicolor mode, however, there's one restriction. *All sprites in multicolor mode must share the same second and third colors.* If you change the second and third colors on one sprite, those colors will change on all the other sprites displayed in multicolor mode.

Each sprite has its own default color, the color which it shows if it's not changed.

Sprite	Color
0	1 (White)
1	2 (Red)
2	3 (Cyan)
3	4 (Purple)
4	5 (Green)
5	6 (Blue)
6	7 (Yellow)
7	12 (Medium Gray)

When in multicolor mode, the second and third colors also have default settings. Normally, the second color is 4 (purple) and the third color is 0 (black).

Priority

Sprites are completely independent of the normal screen display, and except for sharing colors in multicolor mode, they're independent of each other. This independence is handy, because it lets you use several sprites on one screen, and it lets

you use sprites along with text or bitmapped graphics.

The only time that this causes a problem is when a sprite is positioned at the same location as another sprite or other screen information. What happens when two sprites are placed so that they partially overlap? The sprites' colors don't blend together—only one sprite is displayed in the overlapped area. But which one?

This is where *priority* is important. It establishes which sprite will appear in front of other sprites, or whether a sprite seems to be in front of, or behind, other screen displays. *Sprite priority* is fixed; lower numbered sprites always have priority over higher numbered sprites whenever they overlap. If, for instance, sprites 2 and 3 overlap, only sprite 2 will be displayed in the overlapped area. In other words, sprite 2 appears to be in front of sprite 3. Sprite 0 always seems to be in front of all the other sprites; sprite 7 always appears behind the others.

Priority over normal screen information, however, can be individually set for each sprite. A sprite can be made to appear in front of or behind text characters and bitmapped graphics. To set this priority, use one of the following statements:

SYS PR,SN,0 Make sprite appear in front of screen data (default)
SYS PR,SN,1 Make sprite appear behind screen data

Sprites always appear in front of screen data displayed by the multicolor mode bit pair 01, regardless of how the priority is set. If multicolor bitmapped graphics is used (modes 1, 3, 5, and 7 in Part 2), sprites will always appear in front of points displayed by the bit pair 01 (those points drawn with pen 1).

Priority can be used to add the illusion of depth to a screen for three-dimensional effects. Program 12-7, "DEMO#3," was a good example.

Joystick Control

Up to now, the direction of a moving sprite has been set in the immediate mode or in a program by the SYS DI call. With the Sprite Control System, a sprite's motion can also be controlled by a joystick. This is the general form of the statement to switch from program control to joystick control.

SYS JS,SN,*joystick port number*

Thus, to make sprite 0 move under control of a joystick plugged into port 2, you'd use:

SYS JS,0,2

Sprite 0 now moves in the same direction that the joystick is pushed. The sprite will not move when the joystick is not being pushed.

Speed. When a sprite is being moved by a joystick, its speed is still controlled by the delta values. The following statement makes sprite 0 move very quickly whenever the joystick is pushed.

SYS MO,0,5,5

For slow movement, use this statement.

SYS MO,0,1,1

In other words, simply use the usual SYS MO call, as you did earlier for setting sprite speed under program control. Note that you don't have to set the speed with a SYS MO, since the default values are 2 for delta X and 1 for delta Y. However, if you want a different speed, do use the SYS MO.

Limiting directions. If you want a sprite to move in only two directions, set one of the delta values to 0. In this line, the delta Y value is 0, so sprite 0 won't move up or down even when the joystick is pushed in those directions.

SYS MO,0,3,0

How about limiting the sprite's movement to up and down? It's just as simple—set the delta X value to 0, as in:

SYS MO,0,0,3

(Reset the delta X value by entering SYS MO,0,2,1 before going on.)

To limit a sprite's movements to just the four cardinal directions, you can use a special option of the SYS JS call. If a *,1* is added to the end of the call, the diagonal directions of the joystick will be ignored. This is what the format should look like:

SYS JS,SN,*joystick number*,1 (Allows only four directions)

An interesting effect is to force the sprite to continue to move, even when the joystick isn't being pushed. With the Sprite Control System, this effect is just a matter of adding *,2* to the end of the SYS JS call. When this is used, the sprite will keep moving in the last direction the joystick was pushed. It's sort of a "momentum" effect.

SYS JS,SN,*joystick number*,2 (Turn on momentum)

Both of these options may be used at the same time. Just add the 1 and 2 together.

SYS JS,SN,*joystick number*,3 (Select both four directions *and* momentum)

To turn off these options, simply omit the additional value after the joystick number, or use 0.

SYS JS,SN,*joystick number*

or

SYS JS,SN,*joystick number*,0

Port 1. You used port 2 in all the previous examples, but port 1 can also be used. In fact, you can have one sprite controlled by a joystick in port 2, while another is controlled by a joystick plugged into port 1. This feature is vital for two-player games.

Keep in mind, however, that if you experiment with joystick control in the immediate mode, use only port 2. Be careful not to push the joystick while you're typing. The keyboard and the joystick ports can interfere with each other, and using both at the same time causes the wrong keys to be read.

This interference between the keyboard and joystick ports is even worse when port 1 is used. Pushing a joystick plugged into this port makes the computer think that keys are being pressed. It's okay to use port 1 in a program as long as that program doesn't require input from the keyboard. At the end of the program, use the following POKE statement to clear the keyboard buffer:

POKE 198,0

Back to program control. To take a sprite off joystick control and return it to program control, use:

SYS JS,SN,0

The sprite will continue moving in the current direction. For example, if the joystick is being pushed to the left when motion returns to program control, the sprite will continue to move to the left until its direction is changed by a SYS DI call.

Chase Mode

One more way to make a sprite move is to have it chase after another sprite. The Sprite Control System includes a SYS call format which allows you to do this. It looks like:

SYS CH,SN,*number of sprite to chase*

To make sprite 2 chase sprite 6, for example, you could use:

SYS CH,2,6

Sprite 2 will always move in the same direction as sprite 6, and will stop moving only when it's at the same position as sprite 6.

If a sprite in chase mode has delta values greater than 1, it may appear to jiggle when it stops moving. Try using different delta values and starting positions for both sprites to eliminate this problem.

Selecting the chase mode for a sprite turns off its program or joystick control. To disable the chase mode and return the sprite to program control, enter (either in program or immediate mode):

SYS JS,SN,0

Synchronized Sprite Motion

Sometimes it's useful to make two or more sprites move together. One application of this is to create objects made up of more than one sprite, as was done in Program 12-7, "DEMO#3."

The only problem with this is getting the sprites to start moving at the same time. If two sprites are to move together, for example, a SYS DI must be executed for each one. But after the SYS DI is executed for the first sprite, that sprite may start moving before the second SYS DI can be executed.

To solve this problem, the Control System provides a method for stopping and starting the motion of any or all sprites, all at the same time. To freeze one or more sprites, use the following SYS format:

SYS FR,*sprite number total*

To calculate the *sprite number total,* first decide which sprites you want to have stopped. Look at the numbers in the chart below which correspond to these sprites, and add the numbers together.

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

Freezing sprites 2 and 3 requires this line:

SYS FR,4+8

or

SYS FR,12

Once the motion of these sprites is stopped, any values, such as position, direction, and speed, can be set without the sprites going anywhere. After the desired values have been set, the sprites can be unfrozen with:

SYS UN,*sprite number total*

To unfreeze sprites 2 and 3, you'd use:

SYS UN,12

Both sprites will now start moving, with the new motion values in effect.

This feature helps you get sprites traveling together, but if you want several sprites to move side by side, a couple of other things must be done. First, the boundaries for the different sprites must be set differently. Let's say you want two sprites to move together. One displays the left half of an object, the other displays the right half. Each sprite is defined as a full 24 pixels wide. Here are the left and right boundaries that should be used.

Left sprite
Left boundary: 0
Right boundary: 319−24−24

Right sprite
Left boundary: 0+24
Right boundary: 319−24

All these boundary settings do is limit the movement of each sprite. The leftmost sprite can go all the way to the left edge, but it can only move as far right as position 271. It's

prevented from overlapping any of the rightmost sprite.

You must experiment to find combinations of starting positions and delta values which allow the sprites to stay synchronized. With some delta values, for example, synchronized sprites will be separated when they hit a boundary.

Another application of synchronized sprite motion is to display several sprites at one position, to get an overlay effect for color. Perhaps you want more colors than available with multicolor mode, or you want several colors without reducing the horizontal resolution from 24 to 12 pixels. Since the sprites should all start at the same position, any position can be used, and the boundaries would be the same for each sprite.

Enable/Disable

The display of a sprite can be turned off or on.

SYS EN,SN,0 Disable (default)
SYS EN,SN,1 Enable

When a sprite is disabled, it can continue to move, change shape, and so on. It just isn't displayed.

All sprites are initially disabled. The SYS BL call enables a sprite when it assigns a block number.

Disabling a sprite is an easy way (even easier than placing the sprite off the screen) to make a sprite disappear. Other than that, this SYS call has very few uses.

Miscellaneous Topics

Although you've seen how to use the Sprite Control System, from simple sprite movement to advanced synchronized sprites, there's still more you can do. This chapter presents two utility programs and gives additional tips on how to use the Sprite Control System in your own BASIC programs. Some of these tips, hints, and techniques are quite sophisticated, like relocating the Control System or relocating screen memory. Others are much simpler, such as detecting sprite-to-sprite collisions or protecting free memory. You'll even see how to use all three major utilities included in this book together.

The Merge Utility

Once in a while you may want to add the definitions in one file to those in another. Program 16-1, "Merge," lets you combine two or more sprite definition files to create one larger file. It's simple to use and can save you considerable redefining. With Merge, you can combine as many definition files (which you create with the Sprite Editor) as you want.

As with almost all other programs in this book, make sure you change the DN=8 to DN=1 if you're using tape. That variable value is assigned in line 110.

Program 16-1. Merge

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " SPRITE DEFINITION MERG
    E UTILITY"                                :rem 123
105 PRINT " BY CRAIG CHAMBERLAIN":PRINT       :rem 69
110 DN=8:SA=780:SX=781:SY=782:SP=783:REM DN=1 FOR
    {SPACE}TAPE                               :rem 180
120 DEF FNDP(N)=PEEK(N)+256*PEEK(N+1):BA=FNDP(49)+
    1000                                       :rem 8
130 DEF FNH(N)=INT(N/256):DEF FNL(N)=N-256*FNH(N)
                                              :rem 243
300 F$="":INPUT "{YEL} LOAD FILENAME{7}";F$:IF F$=
    "" OR LEN(F$)>12 GOTO 300                 :rem 38
310 LA=BA:GOSUB 630:NS=PEEK(BA):PRINT " LOADED DEF
    INITIONS 0 TO" NS                         :rem 169
400 EA=FNDP(SX):PRINT                         :rem 66
410 F$="":INPUT "{YEL} APPEND FILENAME{7}";F$:IF L
    EN(F$)>12 GOTO 410                        :rem 54
```

```
420 IF F$="" THEN PRINT:GOTO 500                    :rem 212
430 LA=EA-1:P=PEEK(LA):GOSUB 630:PRINT " APPENDED
    {SPACE}DEFINITIONS" NS+1 "TO";                   :rem 9
440 NS=NS+PEEK(LA)+1:PRINT NS:POKE LA,P:IF NS<256
    {SPACE}GOTO 400                                 :rem 139
450 PRINT " ERROR: TOO MANY DEFINITIONS":END
                                                    :rem 162
500 F$="":INPUT "{YEL} SAVE FILENAME{7}";F$:IF F$=
    "" OR LEN(F$)>12 GOTO 500                        :rem 57
510 POKE BA,NS:GOSUB 600:POKE SA,251:POKE 251,FNL(
    BA):POKE 252,FNH(BA)                            :rem 229
520 POKE SX,FNL(EA):POKE SY,FNH(EA):SYS 65496:IF P
    EEK(SP)AND1 GOTO 700                             :rem 69
530 PRINT " SAVED" EA-BA "BYTES":END                :rem 238
600 PRINT:POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466
    :F$=F$+".DEF"                                   :rem 187
610 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                            :rem 241
620 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                         :rem 159
630 GOSUB 600:POKE SA,0:POKE SX,FNL(LA):POKE SY,FN
    H(LA)                                            :rem 1
640 SYS 65493:IF PEEK(SP)AND1 GOTO 700              :rem 90
650 RETURN                                          :rem 123
700 P=PEEK(SA):PRINT " ERROR:";:IF P=4 THEN PRINT
    {SPACE}"FILE NOT FOUND":END                      :rem 96
710 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                    :rem 188
720 PRINT ST:END                                    :rem 222
```

When you run this program, it first asks for the name of a file to load. Enter the name of the first file (you don't have to type in the .DEF suffix placed on the end of the filename by the Editor) and wait a few moments.

Next, you'll need to enter the name of the file to be appended to the end of the file just loaded. After you type in the name of the second file, it loads, then is added to the end of the first file. Merge requests the name of another file to append. You can combine several if necessary.

When you're through appending files, just press RETURN instead of entering another filename. The program asks for one last filename—the one used to name the composite file. After that's typed in, the merged definitions are saved to tape or disk as a new file.

You can even use Merge as a quick and easy way to copy a definition file from disk to disk or tape to tape. Just run

Merge, load the file to be moved, append nothing, and save the file out to the new disk or tape.

Merge can also be used to copy a definition file from tape to disk or from disk to tape. To copy from tape to disk, assign the value 1 to the variable DN in line 110, and insert the statement DN=8 at the beginning of line 510. A file can then be transferred by loading the file, appending nothing, and then saving the file to disk. To copy from disk to tape, just switch the order of the numbers so that DN=8 appears in line 110 and DN=1 in line 510.

To keep things straight, refer to the short table below for the modifications you need to make to Merge for this kind of file copy/transfer process.

Copy Direction	Line 110	Line 510
Tape to Disk	DN=1	DN=8
Disk to Tape	DN=8	DN=1

The Extract Utility

Instead of merging several files together into one, Program 16-2, "Extract," does just the opposite. It pulls a sequence of definitions *from* a definition file. The extracted definitions are then saved as a new file. A major use of this utility is to break a definition file into two parts.

Make sure to change the DN=8 in line 110 to DN=1 if you're using tape.

Program 16-2. Extract

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " DEFINITION EXTRACTION
    {SPACE}UTILITY"                              :rem 53
105 PRINT " BY CRAIG CHAMBERLAIN":PRINT          :rem 69
110 DN=8:SA=780:SX=781:SY=782:SP=783            :rem 133
120 DEF FNH(N)=INT(N/256):DEF FNL(N)=N-256*FNH(N):
    LA=PEEK(49)+256*PEEK(50)+1000                :rem 146
300 F$="":INPUT " LOAD FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 300                            :rem 238
310 GOSUB 600:POKE SA,0:POKE SX,FNL(LA):POKE SY,FN
    H(LA)                                        :rem 252
320 SYS 65493:IF PEEK(SP)AND1 GOTO 700           :rem 85
330 NS=PEEK(LA):PRINT " LOADED DEFINITIONS 0 TO" N
    S:PRINT                                      :rem 162
400 FD=-1:INPUT " NUMBER OF FIRST DEFINITION";FD:I
    F FD<0 OR FD>NS GOTO 400                     :rem 241
```

```
410 FA=LA+1:IF FD THEN FOR K=1 TO FD:FA=FA+PEEK(FA
    )+1:NEXT                                    :rem 19
420 PRINT                                       :rem 35
450 LD=-1:INPUT " NUMBER OF LAST{2 SPACES}DEFINITI
    ON";LD:IF LD<FD OR LD>NS GOTO 450           :rem 25
460 EA=FA:FOR K=0 TO LD-FD:EA=EA+PEEK(EA)+1:NEXT:P
    RINT                                        :rem 237
500 F$="":INPUT " SAVE FILENAME";F$:IF F$="" OR LE
    N(F$)>12 GOTO 500                            :rem 1
510 FA=FA-1:POKE FA,LD-FD:GOSUB 600            :rem 242
520 POKE SA,251:POKE 251,FNL(FA):POKE 252,FNH(FA)
                                                :rem 229
530 POKE SX,FNL(EA):POKE SY,FNH(EA)             :rem 73
540 SYS 65496:IF PEEK(SP)AND1 GOTO 700          :rem 92
550 PRINT " SAVED" EA-FA "BYTES":END           :rem 244
600 PRINT:POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466
    :F$=F$+".DEF"                               :rem 187
610 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)):
    NEXT                                        :rem 241
620 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 65469
    :RETURN                                     :rem 159
700 P=PEEK(SA):PRINT " ERROR:";:IF P=4 THEN PRINT
    {SPACE}"FILE NOT FOUND":END                  :rem 96
710 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                                :rem 188
720 PRINT ST:END                                :rem 222
```

Once you run Extract, you're asked for the name of the file which contains the definitions to be extracted. Enter the filename and wait for the file to load.

The next two prompts ask for the numbers of the first and last definitions in the sequence that you want to pull out of this file. Type in the appropriate numbers.

Finally, the program requests a filename to use in saving the extracted definitions. After you enter the filename, the program saves the specified range of definitions as a new file. The original file has not been changed, however. The definitions you extracted are still there.

To completely break up a definition file, then, you have to perform two extractions. If the original file contains definitions 0–5, for instance, and you want two three-definition files, first extract definitions 0–2, save those to a new filename, then extract definitions 3–5, saving them out under a different name. Then, if you want, you can scratch the original file from disk or overwrite it on tape.



Sprite-to-Sprite Collisions

In games it's often necessary to know when a sprite has collided with another sprite or some other screen data. There's an easy way, using a PEEK and the AND operator, to tell if a sprite has been involved in a collision.

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

The general procedure is to PEEK the location, assign the value to a variable, and then use the variable with the AND operator and sprite numbers to check individual bits. Something like this:

P=PEEK(53278):IF P AND N THEN ***sprite was involved in collision***

The value for N, the bit value, is determined from the table below. (This is the same table used for freezing and unfreezing sprites.)

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

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

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

Actual program statements would be placed after each THEN. In the example below, execution jumps to line 300 if sprite 5 is involved in a collision.

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

The bits in location 53278 are cleared after the location is PEEKed so that new collisions can be detected. So far, the PEEKed value has been assigned to a variable so that it can be checked for multiple collisions. If you're going to be watching only one sprite to see when it collides, however, you can eliminate the variable assignment.

100 IF PEEK(53278) AND 32 THEN 300

If you don't care which sprite is involved in a collision, and only need to detect when *any* sprite collision occurs, the AND operation can also be eliminated. This is the method used in Program 12-5, "DEMO#2."

100 IF PEEK(53278) THEN 300

It's important to understand that location 53278 tells you only which sprites were involved in collisions. It does not tell you which sprite collided with which. For example, if location 53278 reports that sprites 1, 2, 3, and 4 have all been involved in collisions since the last time the location was PEEKed, several combinations of collisions are possible. Perhaps sprite 1 collided with sprite 2, and sprite 3 collided with sprite 4. Maybe it was sprites 1 and 3 and sprites 2 and 4. Or maybe sprites 1 and 4 collided, and sprites 2 and 3 collided. Another possibility is that all four sprites collided with each other.

The only way to determine which sprites hit is to compare their positions. A method for doing this is discussed a bit later.

Sprite-to-Screen-Data Collisions

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

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

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

P=PEEK(53279):IF P AND 64 THEN 500

As with priority, the multicolor bit pair 01 is a special exception. No collision is detected when a sprite overlaps anything displayed by the bit pair 01 in character or bitmapped graphics mode.

POKEing and PEEKing

You were earlier promised that you would be able to animate sprites with the Control System without having to use a single POKE statement. That promise has been kept. But you can still use POKE statements if you want to. For instance, location 53287 holds the color for sprite 0. To change this color you can use either of the following two statements:

SYS CO,0,*color number*

or

POKE 53287,*color number*

Both statements do the same thing. Other sprite characteristics, such as size and position, can also be set by POKE statements. In most cases, however, the SYS calls in the Sprite Control System are more convenient to use.

On the other hand, it can be very useful to PEEK hardware locations which control sprites. This lets you determine things like current color or position. The following statements, for example, return the current coordinates of a sprite.

```
X=PEEK(53248+2*SN)+256*SGN(PEEK(53264)AND2↑SN)-24
Y=PEEK(53249+2*SN)-50
```

Reading Definitions from DATA

As mentioned earlier, definitions can be stored as data in a program so that you have one less file to load whenever you run a program using sprites. The procedure is very simple.

First, use the Data feature in the Sprite Editor to convert the definitions to DATA statements. Then use Program E-1, "Merge Utility," found in Appendix E to merge the DATA statements with your program.

The last step is to replace the call to the definition loading subroutine at 58500 (line 58500 is part of "SCS.BAS," the BASIC file you must include as part of your own program when you want to use the Sprite Control System) with a call to the subroutine at 58400. This subroutine reads the definitions into memory instead of loading them from tape or disk.

Before you call the subroutine at 58500, you have to specify a filename *and* a loading address. But with the subroutine at 58400, only the loading address has to be set. In your own program, the line would probably look like:

LA=16384:GOSUB 58400:REM READ DEFINITIONS

The data for the bunny shape definition is already included in Program 12-3, "DEMO#1." To see how the subroutine at 58400 works, just modify line 140 in Program 12-3 to:

```
140 LA=16384:GOSUB 58400:REM READ DEFINITIONS
```

Run the program and you'll see the same end result. Notice, however, that if you're using tape, you have to load only one additional file, "SCS.OBJ," not two as you did before. Reading definitions from DATA is especially efficient when you have to use tape.

When using this method, be careful if the program contains DATA statements for other purposes.

One drawback to using this method is that DATA statements make a program grow quickly. For large definition files, then, this method is impractical.

Protecting Free Memory

No matter which technique is used to load definitions, they end up stored in free memory. This presents no problem unless your program uses string variables. Free memory is used when strings are assigned. If your program assigns string variables, you should periodically call the free memory function to reorganize free memory.

K=FRE(0)

Relocating Screen Memory

You'll remember that the number of the lowest definition block available can be determined by the expression INT((PEEK(49)+256*PEEK(50))/64)+16. If the value of this expression is greater than 255, you have a small problem. It means there are no definition blocks available. This can happen when your BASIC program is larger than 8K.

There is a way to get around this problem. It's a bit involved, but by following the instructions below, you shouldn't have any trouble.

What you'll do is use a different bank of memory for definition blocks. The Commodore 64's memory is divided into four sections, called *banks.* Each bank consists of 16K bytes, which is enough for 256 definition blocks. The default bank when you turn the computer on is bank 0. When there is no

room in this bank for definition blocks, you can switch to bank 1, 2, or 3 instead.

Bank	Memory Locations
0	0–16383
1	16384–32767
2	32768–49151
3	49152–65535

The only difficulty with using another bank is that screen memory must be located in the same bank used for definition blocks. If you switch to bank 2, for example, you must move the screen memory from bank 0 to bank 2.

Bank 2. This bank is a good choice because the RAM under the BASIC ROM can be used to hold sprite definitions. This gives you 128 definition blocks, numbered from 128 to 255. Another advantage of bank 2 is that the VIC-II chip sees an image of the character set in this bank, so you don't have to define your own character set.

However, when you use bank 2, you'll run into some problems finding a place for screen memory. No matter where it's put, the top of free memory is going to have to be lowered. To preserve as much free memory as possible, start the screen memory at location 35840.

To relocate screen memory, use the following statement.

SYS RL,*address of screen memory*/256

If you want to relocate the screen memory to start at 35840, for example, you'd use:

SYS RL,35840/256

or

SYS RL,140

Do this *after* the call to the subroutine at 58000, but before the Sprite Control System is installed by SYS IN.

To lower the top of free memory, POKE location 56 with the same number used above, and then use a CLR to reset the memory pointers. In the previous example, with the screen memory set to start at 35840, you should enter:

POKE 56,140:CLR

Place this at the beginning of your program, *before* any variables are assigned.

Since you're no longer using bank 0, you don't have to set the LOAD address to start at the beginning of the next

bank. Instead of using the statement LA=16384 to set the LOAD address, use this:

```
LA=PEEK(49)+256*PEEK(50)+1000
```

The screen memory should not be placed at 35840 if some other utilities have been installed at the top of free memory earlier. To check if the top of free memory has already been lowered, examine the value in location 56 before you change it.

```
PRINT PEEK(56)
```

Normally, this will give you the value 160, which is fine. If you get a number less than 144, however, you should not start the screen at 35840. Instead, subtract 4 from this value, AND the value with the number 252, and use the result in the SYS RL and POKE 56 statements. For instance, if you PEEK location 56 and receive 141 as an answer, these statements should be used to relocate screen memory and lower the top of free memory.

```
SYS RL,141-4AND252:POKE 56,141-4AND252:CLR
```

Don't use bank 2 if the value in location 56 is less than 132.

Bank 3. If you use bank 3, the memory under the Kernal ROM can be used to hold definitions. This gives you blocks 128–254. You must not use block 255 in bank 3. Doing so will make the computer crash.

Another area of memory in bank 3 that can be used to store definitions is the RAM under the I/O chips and color memory. This RAM cannot be easily POKEd from BASIC, so it's rarely used. The SYS DF routine, however, banks this RAM in when it defines a block. The result is that in bank 3, blocks 64 through 127 are also available.

Screen memory can be placed at either location 51200 or location 52224. Neither of these is in free memory, so the top of free memory does not have to be lowered. This is a major advantage of using bank 3—no free memory is used.

The only disadvantage to using bank 3 is that the VIC-II chip does not see an image of the character ROM, so you have to supply your own. The best place to put the character set is in the RAM under the I/O chips, starting at location 53248. If you do this, blocks 64 through 95 will no longer be available.

Bank 1. This offers a full 16K of RAM in the middle of free memory. Even after subtracting 1K for screen memory

and 2K for a character set, you should still have plenty of room for definition blocks. If you use this bank, put the screen memory at location 29696, and a character set at 30720. Since you're using the middle of free memory, the top of memory does not have to be lowered.

Definition blocks up to block 207 are available. To find the number of the lowest available block, use this statement after the definition file has been loaded:

```
PRINT INT((LA-16384)/64)+16
```

If this number is negative, all the blocks from 0 to 207 can be used. If the number is greater than 207, don't use bank 1.

Reset screen memory. At the end of your program, you'll want to reset screen memory back to bank 0. This can be done by the following SYS call.

```
SYS RS
```

Do this after the Sprite Control System has been removed.

You'll also need to use SYS RS if you have relocated the screen memory and pressed RUN/STOP–RESTORE to remove the Control System.

If you've changed the top of memory, you may also want to reset it. POKE location 56 with the value that it contained before you changed it, and then perform a CLR.

Here's a summary of the steps to take to relocate the screen memory. Bank 2 is usually the best choice, because it supports 128 definition blocks (blocks 128–255) and no character set has to be defined.

To use bank 2:

- Remember the contents of location 56.
- POKE 56,140:CLR to lower the top of free memory.
- Load the SCS.OBJ and definition files.
- Use SYS RL,140 to relocate screen memory
- Use SYS IN to install the Sprite Control System, and then proceed as usual.
- After SYS RE, use SYS RS to reset the screen.
- Restore the original contents of location 56.

When you select bank 3, blocks 64–254 can be used. The only drawback is that no character set is supplied. If one is needed, it should be put at location 53248, in which case only blocks 96–254 can be used for definitions.

To use bank 3:

- Define a character set at location 53248 if one is needed.
- Follow the standard procedure to use the Sprite Control System, but insert the statement SYS RL,200 before SYS IN to relocate screen memory.
- At the end of the program (after SYS RE), use SYS RS to reset the screen.

Bank 1 supports the most blocks, from INT((LA−16384)/64)+16 to 207, but like bank 3, it does not contain a character set.

To use bank 1:

- Define a character set at location 30720 if one is necessary.
- Follow the standard procedure, but perform a SYS RL,116 before SYS IN.
- Use SYS RS after SYS RE at the end of the program.

Whenever any of these three banks is used, the LOAD address is no longer set as LA=16384, but is calculated by the formula:

LA=PEEK(49)+256*PEEK(50)+1000

If the screen has been relocated to one bank and you want to place it back in bank 0, just use SYS RL,4. Be sure that the Sprite Control System is not installed when you do this.

Relocating the Sprite Control System

The machine language routines for the Sprite Control System are stored in memory from locations 49664 to 51199. The Control System also uses locations 2 and 251–254, and memory in the tape buffer. You cannot use the Sprite Control System with another utility that uses any of this memory.

Both the Bitmapped Graphics and Sidplayer utilities use some of this memory. In order to use the Control System with one or more of these other utilities, then, you have to have a way to relocate the Sprite Control System. It's possible.

Those portions of the Control System which conflict with the other utilities can be moved to free memory. This lets you use the Sprite Control System with the bitmapped graphics extensions or Sidplayer. You can even use all three utilities at the same time. (More on that in a bit.)

To relocate the Sprite Control System to free memory, as-

sign the LOAD address and call the subroutine at 58100, instead of calling the subroutine at 58000. The subroutine at 58100 loads the SCS.OBJ file, moves it to free memory starting at LA, and advances LA to point to the first byte of free memory after the SCS.OBJ file. The line in your BASIC program might look like this:

```
13Ø LA=16384:GOSUB 581ØØ
```

When you load the definitions, just set the filename and call the subroutine at 58500. Do not reassign LA. The LOAD address should be set only once.

```
14Ø F$="BUNNY":GOSUB 585ØØ
```

Using the Control System with the Bitmapped Graphics Extensions

When using the Sprite Control System with the bitmapped graphics extensions to BASIC, the procedure to relocate the Control System is complicated by the fact that the bitmap is placed in bank 3. Thus, besides relocating the Control System, you must also relocate screen memory.

Begin with the bitmapped graphics extensions to BASIC already installed. Then you can start placing the Control System in memory. First, assign LA:

LA=PEEK(49)+256*PEEK(50)+1000

This is used instead of LA=16384 because screen memory will be relocated.

Next, call the subroutine at 58100.

GOSUB 58100

Now, specify the filename of the definition file and call the subroutine at 58500 to load the definitions.

F$=*filename*:GOSUB 58500

The Bitmapped Graphics utility requires that screen memory start at location 50176, and since 50176/256 is 196, use the following to relocate screen memory.

SYS RL,196

You're now ready to use the graphics statements and Control System SYS calls. There are, however, a few restrictions. The first is that whenever you use the statement GRAPHICS,

you must immediately follow it with this POKE statement:

POKE 53265,PEEK(53265)AND127

Location 53265 controls various features of the VIC-II graphics chip, including bitmap mode. The GRAPHICS statement changes this location when it turns on the mode. The only problem is that the location also controls raster scan interrupts. Bit 7 of location 53265 must be kept clear if the raster scan interrupt used by the Sprite Control System is to work reliably. The above statement insures that bit 7 is clear.

A second restriction is that you *must not use* the TEXT statement. Doing so while the Sprite Control System is installed can cause the computer to crash.

The final restriction is that the RAM under the Kernal is used to store the bitmap, so the corresponding definition blocks are not available. You may use only blocks 64–127 and blocks 253 and 254.

Since no character set is used in bitmapped graphics, you don't have to worry about defining one.

If you want to draw bitmapped graphics shapes with the SHAPE statement, merge the lines in the "SHP.BAS" file with your program, and load the shape file before you load the sprite definitions.

The following program lines summarize the steps you should take to use the Bitmapped Graphics and Sprite Control System utilities together.

```
200 LA=PEEK(49)+256*PEEK(50)+1000:GOSUB 58100:REM
    {SPACE}LOAD AND RELOCATE SCS.OBJ
210 F$="FILENAME":GOSUB 58500:REM LOAD DEFINITIONS
220 SYS RL,196:REM RELOCATE SCREEN MEMORY
230 GRAPHICS 0:POKE 53265,PEEK(53265)AND127:REM CA
    N USE ANY MODE
240 SYS IN:REM INSTALL SPRITE CONTROL SYSTEM
250 ...
```

"FILENAME" in line 210 should be changed to match the definition file's name you're calling.

If you're also using bitmapped graphics shapes (the SHAPE statement), insert the following line:

```
205 F$="FILENAME":GOSUB 56500:REM LOAD SHAPES
```

End the program in the normal way.

```
560 ...
570 SYS RE:REM REMOVE SPRITE CONTROL SYSTEM
580 SYS RS:REM RESET SCREEN
590 END
```

Using the Sprite Control System with Sidplayer

It's much easier to use the Sprite Control System with the Sidplayer. You should have no problems as long as you follow this process.

Your program must contain the Sidplayer subroutine lines starting at 57000, as well as the lines beginning at 58000 used by the Sprite Control System. The lines from 57000 can be found in Program 11-1, "SID.BAS."

Your BASIC program should begin by assigning the LOAD address.

LA=16384

Load and relocate the "SID.OBJ" file with:

GOSUB 57100

Next, load the "SCS.OBJ" file. It's not necessary to relocate the file this time.

GOSUB 58000

Now load the song and sprite definition files.

F$=*filename*:GOSUB 57500:REM LOAD SONG
F$=*filename*:GOSUB 58500:REM LOAD DEFINITIONS

The last step is to use the SYS HK call to install Sidplayer *before* you use the SYS IN call to install the Sprite Control System. This order is important.

SYS HK:SYS IN

Proceed as normal from this point. At the end of the program, the utilities must be removed in the reverse order.

SYS RE:SYS DP

There are a few important notes about using these two utilities together. The first is that both utilities are interrupt-driven. Both utilities take an amount of processing time away from BASIC, so your programs may run noticeably slower.

Using the utilities in tandem increases the chance of a crash-causing error. If the music or sprite motion ever stops suddenly for no apparent reason, it means that music and motion processing took more than one frame. If this is ever a

problem, try using fewer special options between notes in the music, or reduce the number of moving sprites.

Finally, when the two utilities are used together and your computer is operating on 50 Hz power (in Europe, for example), the tempo of the music will be slightly decreased.

Using All Three Utilities Together

If you want, you can use the Bitmapped Graphics, Sidplayer, and Sprite Control System utilities simultaneously. To do so, start with the bitmapped graphics extensions already installed. Assign the LOAD address, load and relocate the SID.OBJ file, load and relocate the SCS.OBJ file, and then load the shape, song, and definition files. Make sure that you use SYS HK and SYS RL,196 before SYS IN, and be sure to clear bit 7 of location 53265 whenever you use the GRAPHICS statement. At the end of the program, use SYS RE, SYS RS, and SYS DP, in that order.

The following demonstration is not intended to be a typical application of the three utilities, but it does show you how to get all three going at the same time.

Program 16-3. Three at Once

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D. Remember to install the bitmapped graphics extensions to BASIC before entering this program.

```
100 PRINT "{CLR}{DOWN} ALL ABOUT THE COMMODORE 64:
    VOLUME TWO"                                  :rem 86
110 PRINT TAB(10) "GRAND DEMONSTRATION":PRINT
                                                  :rem 6
120 DN=8:SA=780:SX=781:SY=782:SP=783            :rem 134
200 LA=PEEK(49)+256*PEEK(50)+1000               :rem 248
210 GOSUB 57100:REM LOAD AND RELOCATE SID.OBJ
                                                 :rem 89
220 GOSUB 58100:REM LOAD AND RELOCATE SCS.OBJ
                                                :rem 100
230 F$="COMMODORE":GOSUB 57500:REM LOAD SONG
                                                 :rem 85
240 F$="BUNNY":GOSUB 58500:REM LOAD DEFINITION
                                                :rem 240
300 GRAPHICS 6:POKE 53265,PEEK(53265)AND127:REM TU
    RN ON BITMAPPED GRAPHICS                    :rem 135
310 SYS HK:REM INSTALL SIDPLAYER                  :rem 8
320 SYS RL,196:REM RELOCATE SCREEN MEMORY         :rem 4
330 SYS IN:REM INSTALL SPRITE CONTROL SYSTEM
                                                 :rem 62
```

```
340 SYS DF,254,0:REM DEFINE BLOCK 254 WITH DEFINIT
    ION 0                                    :rem 103
350 SYS FR,255:REM FREEZE ALL SPRITES        :rem 217
360 FOR SN=0 TO 7:REM LOOP WITH SPRITE NUMBER
                                             :rem 156
370 SYS CO,SN,1:REM SET COLOR TO WHITE        :rem 2
380 SYS PO,SN,50+SN*25,20+SN*22:REM SET POSITION
                                             :rem 91
390 SYS DI,SN,3:REM SET DIRECTION            :rem 255
400 SYS MO,SN,2,2:REM SET SPEED               :rem 51
410 SYS BO,SN,1:REM TURN ON BOUNCE            :rem 15
420 SYS BL,SN,254:REM ASSIGN BLOCK 254 TO SPRITE (
    AND ENABLE)                              :rem 229
430 NEXT SN                                  :rem 119
440 POKE SX,LO:POKE SY,HI:SYS PL:POKE SS,7:REM STA
    RT SONG PLAYING                           :rem 73
450 SYS UN,255:REM START SPRITES MOVING      :rem 169
460 SETPEN 1,RND(0)*14+2:DRAW RND(0)*40,RND(0)*25
    {SPACE}TO RND(0)*40,RND(0)*25             :rem 76
470 IF PEEK(SS)AND7 GOTO 460:REM DRAW LINES UNTIL
    {SPACE}SONG ENDS                         :rem 215
480 SYS RE:REM REMOVE SPRITE CONTROL SYSTEM
                                             :rem 251
490 SYS RS:REM RESET SCREEN                  :rem 162
500 SYS DP:REM REMOVE SIDPLAYER              :rem 193
510 PRINT:PRINT " WOW! WHAT A SHOW!"          :rem 35
520 END                                      :rem 110
57000 POKE SA,1:POKE SX,DN:POKE SY,1:SYS 65466:F$=
      "SID.OBJ":GOSUB 59000                  :rem 106
57010 POKE SA,0:SYS 65493:IF PEEK(SP)AND1 GOTO 591
      00                                     :rem 126
57020 SS=49152:HK=49435:PL=49458:DP=49629:RETURN
                                             :rem 77
57100 GOSUB 57000:POKE SX,LA-256*INT(LA/256):POKE
      {SPACE}SY,INT(LA/256):SYS 51042        :rem 218
57110 LA=LA+1398:RETURN                      :rem 111
57500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".MUS":GOSUB 59000                   :rem 61
57510 HI=INT(LA/256):LO=LA-256*HI            :rem 137
57520 POKE SA,0:POKE SX,LO:POKE SY,HI:SYS 65493:IF
       PEEK(SP)AND1 GOTO 59100                :rem 49
57530 LA=PEEK(SX)+256*PEEK(SY):RETURN        :rem 29
58000 POKE SA,1:POKE SX,DN:POKE SY,1:SYS 65466:F$=
      "SCS.OBJ":GOSUB 59000                  :rem 116
58010 POKE SA,0:SYS 65493:IF PEEK(SP)AND1 GOTO 591
      00                                     :rem 127
58020 IN=49664:RE=49667:RL=49670:RS=49673:DF=49676
      :BL=49679:CO=49682:WI=49685            :rem 172
58030 HE=49688:PO=49691:DI=49694:WR=49697:BO=49700
      :MO=49703:LB=49706:RB=49709            :rem 148
```

```
58040 TB=49712:BB=49715:FR=49718:UN=49721:PR=49724
      :MU=49727:EN=49730:JS=49733          :rem 149
58050 AU=49736:CH=49739:RETURN              :rem 14
58100 GOSUB 58000:SYS 51123,LA:LA=LA+1191:RETURN
                                           :rem 11
58400 POKE 49749,LA-256*INT(LA/256):POKE 49750,INT
      (LA/256)                             :rem 182
58410 READ J:POKE 49751,J:FOR K=0 TO J:READ J:POKE
       LA,J:LA=LA+1                        :rem 16
58420 IF J THEN FOR LA=LA TO LA+J-1:READ I:POKE LA
      ,I:NEXT LA                           :rem 92
58430 NEXT K:RETURN                        :rem 168
58500 POKE SA,1:POKE SX,DN:POKE SY,0:SYS 65466:F$=
      F$+".DEF":GOSUB 59000                :rem 24
58510 POKE SA,0:POKE SX,LA-256*INT(LA/256):POKE SY
      ,INT(LA/256)                         :rem 78
58520 SYS 65493:IF PEEK(SP)AND1 GOTO 59100 :rem 44
58530 LA=LA+1:POKE 49749,LA-256*INT(LA/256):POKE 4
      9750,INT(LA/256)                     :rem 167
58540 POKE 49751,PEEK(LA-1):LA=PEEK(SX)+256*PEEK(S
      Y):RETURN                            :rem 31
59000 FOR K=1 TO LEN(F$):POKE 584+K,ASC(MID$(F$,K)
      ):NEXT                               :rem 88
59010 POKE SA,LEN(F$):POKE SX,73:POKE SY,2:SYS 654
      69:RETURN                             :rem 6
59100 P=PEEK(SA):PRINT " ERROR: ";:IF P=4 THEN PRI
      NT "FILE NOT FOUND":END              :rem 200
59110 IF P=5 THEN PRINT "DEVICE NOT PRESENT":END
                                           :rem 36
59120 PRINT ST:END                         :rem 70
```

When this program runs, it loads the files SID.OBJ, SCS.OBJ, COMMODORE.MUS, and BUNNY.DEF, in that order. If you're using disk, all of these files must be on the same disk. If you are using tape, the files should be stored on the same tape in that order.

Final Comments

The Sprite Control System makes it possible to animate sprites in ways that are otherwise not possible in BASIC. If applications don't seem readily apparent, it's probably because you're not used to having such capabilities available.

The two biggest areas of potential use are probably educational programs and games. An effective use of animation can greatly enhance educational software, and action games and adventures are made more feasible by the Sprite Control System.

With the bitmapped graphics extensions to BASIC, Sidplayer, and the Sprite Control System, you now have a complete sound and graphics package. In fact, you have a whole new computer. Have fun in discovering your new Commodore 64!

Sprite Control System Command Index

Sprite Editor
Editing Commands
Change color 411
Copy 413–14
Data 415–17
Delete 411
Edit 412–13
Erase 411
Flip 410
Height 411–12
Insert 411
Invert 410
Key effect 412
Load 415
Multicolor 412
Quit 417
Save 414
Scrolling 411
Width 411–12

Sprite Control System
SYS Calls
AU (autoshape) 435
BB (bottom boundary) 428–29
BL (assign block) 422–23
BO (bounce) 426–27
CH (chase) 440–41
CO (color) 423
DF (define block) 422
DI (direction) 425–26
EN (enable/disable) 443
FR (freeze) 441–43
HE (height) 423
IN (install) 421

JS (joystick) 438–40
LB (left boundary) 428–29
MO (speed and motion) 427–28
MU (multicolor) 437
PO (position) 423–25
PR (set priority) 437–38
RB (right boundary) 428–29
RE (remove) 429
RL (relocate screen memory) 453–54
RS (reset screen memory) 455
TB (top boundary) 428–29
UN (unfreeze) 442–43
WI (width) 423
WR (wraparound) 426–27

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. Many 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 the letter *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 it appears 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 "How to Type In Programs" (Appendix B).

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 (in machine language), while others may 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 RUN/STOP key may seem dead, and the screen may go blank. But don't panic. No damage has been done. To regain control, turn off your computer and 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. *However, the error is still in the DATA statements.*

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 should at least know how to backspace. Do you know how to enter reverse-video, lowercase, and control characters? It's all explained in your manual.

In order to insure accurate entry of each program line, we have included a checksum program. Please read "The Automatic Proofreader" (Appendix D) before typing in any of the programs in this book.

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

Some of the programs in this book contain special control characters (cursor controls, color keys, reverse-video, etc.). 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 (for example, {10 N}), you should type the key as many times as indicated. In that case, 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.

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 INST/DEL key; you can still use INST/DEL to back up and edit the line. Once you type another quote, you are out of quote mode.

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.

In order to insure accurate entry of each program line, we have included a checksum program. Please read "The Automatic Proofreader" (Appendix D) before typing in any of the programs in this book.

Refer to the following table when entering cursor and color control keys:

When You Read:	Press:	See:	When You Read:	Press:	See:
{CLR}	SHIFT CLR/HOME		{1}	COMMODORE 1	
{HOME}	CLR/HOME		{2}	COMMODORE 2	
{UP}	SHIFT ↑ CRSR ↓		{3}	COMMODORE 3	
{DOWN}	↑ CRSR ↓		{4}	COMMODORE 4	
{LEFT}	SHIFT ← CRSR →		{5}	COMMODORE 5	
{RIGHT}	← CRSR →		{6}	COMMODORE 6	
{RVS}	CTRL 9		{7}	COMMODORE 7	
{OFF}	CTRL 0		{8}	COMMODORE 8	
{BLK}	CTRL 1		{F1}	f1	
{WHT}	CTRL 2		{F2}	SHIFT f1	
{RED}	CTRL 3		{F3}	f3	
{CYN}	CTRL 4		{F4}	SHIFT f3	
{PUR}	CTRL 5		{F5}	f5	
{GRN}	CTRL 6		{F6}	SHIFT f5	
{BLU}	CTRL 7		{F7}	f7	
{YEL}	CTRL 8		{F8}	SHIFT f7	

Using the Machine Language Editor: MLX

Charles Brannon

Remember the last time you typed in the BASIC loader for a long machine language program? You typed in hundreds of numbers and commas. Even then, you couldn't be sure if you typed it in right. So you went back, proofread, tried to run the program, crashed, went back again, proofread, corrected a few typing errors, ran again, crashed again, rechecked your typing . . .

Frustrating, wasn't it?

Now, "MLX" comes to the rescue. MLX makes it easy to enter all those long machine language programs with a minimum of fuss. It lets you enter the numbers from a special list that looks similar to DATA statements, and 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 numbers on the wrong line. In short, MLX will make proofreading obsolete.

Tape or Disk Copies

In addition, MLX will generate a ready-to-use tape or disk copy of your machine language program. You can then use the LOAD command to read the program into the computer, just like you would with a BASIC program. Specifically, you enter:

LOAD "*program name*",1,1 (for tape)
LOAD "*program name*",8,1 (for disk)

To start the program, you need to enter a SYS command that tranfers control from BASIC to your machine language program. The starting SYS will always be given in the article which presents the machine language program in MLX format.

Using MLX

Type in and save MLX (you'll want to use it in the future). When you're ready to type in the machine language program,

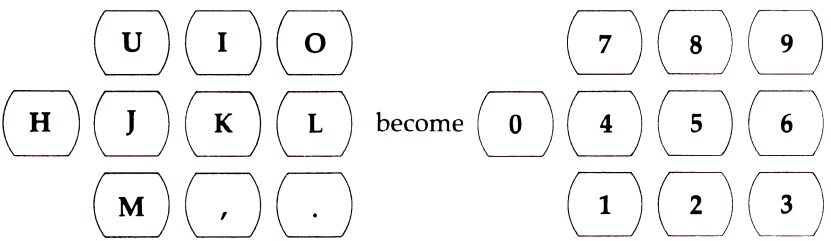
run MLX. MLX will ask you for two numbers: the starting address and the ending address. Then you'll get a prompt showing the specified starting address; that tells you to type in the corresponding first line of the program.

Subsequent prompts will ask you to type in subsequent lines from the MLX 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 sound a buzzer and prompt you to reenter the entire line. If you enter the line correctly, a pleasant bell tone will sound and you may go on to enter the next line.

A Special Editor

You are not using the normal 64 BASIC editor with MLX. For example, it will accept only numbers as input. If you make a typing error, 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 automatically prints the comma and goes on to accept the next number. If you enter less than three digits, you can press either the space bar or RETURN key to advance to the next number. The checksum automatically appears in reverse video for emphasis.

To make it even easier to enter these numbers, MLX redefines part of the keyboard as a numeric keypad (lines 581–584). The keypad can be used only to enter the data; you must use the regular number keys to enter the starting and ending addresses.



When testing it, I've found MLX to be an extremely easy way 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 your machine language program all in one session, you can then save the completed and bug-free program to tape or disk. Follow the instructions displayed on the screen. If you get any error messages while saving, you probably have a bad disk, a full disk, or a typo in MLX. Sorry, MLX 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 completed portion, and then reload your work from tape or disk when you want to continue. MLX recognizes these 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 prompt. Use the Save command to store what you've been working on. It will write the tape or disk file as if you've finished. Remember what address you stop on. Then, the next time you run MLX, answer all the prompts as you did before and insert the disk or tape containing the stored file. When you get the entry prompt, press SHIFT-L to reload the file into memory. You'll then use the New Address command (SHIFT-N) 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 continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing, or else the checksums 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 stop the display by pressing any key.

Tricky Stuff

You can use the Save and Load commands to make copies of the complete machine language program. 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 MLX Save command: When you load them, the message "FOUND *program*" may appear twice. The tape will load just fine, however.

Programmers will find MLX to be an interesting program which protects the user from most typing mistakes. Some screen formatting techniques are also used. Most interesting is the use of ROM Kernal routines for loading and saving blocks of memory. To use these routines, just POKE the starting address (low byte/high byte) into memory locations 251 and 252, and POKE the ending address into locations 254 and 255. Any error code for the Save or Load 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. Be sure to save MLX; it will be used for future applications in other COMPUTE! books.

(Note: If you have a copy of MLX from another COMPUTE! publication and you're using tape, change line 763 of your copy so that it matches line 763 as listed below. Tape users who have a copy of MLX which does not have a line 763 must *use this version of MLX.)*

The Machine Language Editor: MLX

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
10 REM LINES CHANGED FROM MLX VERSION 2.00 ARE 750
   ,765,770 AND 860                              :rem 50
20 REM LINE CHANGED FROM MLX VERSION 2.01 IS 300
                                                 :rem 147
30 REM LINE CHANGED FROM MLX VERSION 2.02 IS 763
                                                 :rem 162
100 PRINT"{CLR}{6}";CHR$(142);CHR$(8);:POKE53281,1
    :POKE53280,1                                  :rem 67
101 POKE 788,52:REM DISABLE RUN/STOP             :rem 119
200 PRINT"{2 DOWN}{PUR}{BLK} MACHINE LANGUAGE EDIT
    OR VERSION 2.03{5 DOWN}"                     :rem 239
210 PRINT"{5}{2 UP}STARTING ADDRESS?{8 SPACES}
    {9 LEFT}";                                   :rem 143
215 INPUTS:F=1-F:C$=CHR$(31+119*F)               :rem 166
220 IFS<256OR(S>40960ANDS<49152)ORS>53247THENGOSUB
    3000:GOTO210                                 :rem 235
225 PRINT:PRINT:PRINT                            :rem 180
```

```
230 PRINT"[5]{2 UP}ENDING ADDRESS?{8 SPACES}
    {9 LEFT}";:INPUTE:F=1-F:C$=CHR$(31+119*F)
                                          :rem 20
240 IFE<256ORE>40960ANDE<49152)ORE>53247THENGOSUB
    3000:GOTO230                          :rem 183
250 IFE<STHENPRINTC$;"{RVS}ENDING < START
    {2 SPACES}":GOSUB1000:GOTO 230        :rem 176
260 PRINT:PRINT:PRINT                     :rem 179
300 PRINT"{CLR}";CHR$(14):AD=S            :rem 56
310 A=1:PRINTRIGHT$("0000"+MID$(STR$(AD),2),5);":"
    ;                                     :rem 33
315 FORJ=ATO6                             :rem 33
320 GOSUB570:IFN=-1THENJ=J+N:GOTO320      :rem 228
390 IFN=-211THEN 710                      :rem 62
400 IFN=-204THEN 790                      :rem 64
410 IFN=-206THENPRINT:INPUT"{DOWN}ENTER NEW ADDRES
    S";ZZ                                 :rem 44
415 IFN=-206THENIFZZ<SORZZ>ETHENPRINT"{RVS}OUT OF
    {SPACE}RANGE":GOSUB1000:GOTO410       :rem 225
417 IFN=-206THENAD=ZZ:PRINT:GOTO310       :rem 238
420 IF N<>-196 THEN 480                   :rem 133
430 PRINT:INPUT"DISPLAY:FROM";F:PRINT,"TO";:INPUTT
                                          :rem 234
440 IFF<SORF>EORT<SORT>ETHENPRINT"AT LEAST";S;"
    {LEFT}, NOT MORE THAN";E:GOTO430      :rem 159
450 FORI=FTOTSTEP6:PRINT:PRINTRIGHT$("0000"+MID$(S
    TR$(I),2),5);":";                     :rem 30
451 FORK=0TO5:N=PEEK(I+K):PRINTRIGHT$("00"+MID$(ST
    R$(N),2),3);",";                      :rem 66
460 GETA$:IFA$>""THENPRINT:PRINT:GOTO310  :rem 25
470 NEXTK:PRINTCHR$(20);:NEXTI:PRINT:PRINT:GOTO310
                                          :rem 50
480 IFN<0 THEN PRINT:GOTO310              :rem 168
490 A(J)=N:NEXTJ                          :rem 199
500 CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT                    :rem 200
510 PRINTCHR$(18);:GOSUB570:PRINTCHR$(146);:rem 94
511 IFN=-1THENA=6:GOTO315                 :rem 254
515 PRINTCHR$(20):IFN=CKSUMTHEN530        :rem 122
520 PRINT:PRINT"LINE ENTERED WRONG : RE-ENTER":PRI
    NT:GOSUB1000:GOTO310                  :rem 176
530 GOSUB2000                             :rem 218
540 FORI=1TO6:POKEAD+I-1,A(I):NEXT:POKE54272,0:POK
    E54273,0                              :rem 227
550 AD=AD+6:IF AD<E THEN 310              :rem 212
560 GOTO 710                              :rem 108
570 N=0:Z=0                               :rem 88
580 PRINT"[£]";                           :rem 81
581 GETA$:IFA$=""THEN581                  :rem 95
```

```
582 AV=-(A$="M")-2*(A$=",")-3*(A$=".")-4*(A$="J")-
    5*(A$="K")-6*(A$="L")                          :rem 41
583 AV=AV-7*(A$="U")-8*(A$="I")-9*(A$="O"):IFA$="H
    "THENA$="Ø"                                    :rem 134
584 IFAV>ØTHENA$=CHR$(48+AV)                       :rem 134
585 PRINTCHR$(2Ø);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N67Ø                                           :rem 229
59Ø IFA>128THENN=-A:RETURN                         :rem 137
6ØØ IFA<>2Ø THEN 63Ø                               :rem 1Ø
61Ø GOSUB69Ø:IFI=1ANDT=44THENN=-1:PRINT"{OFF}
    {LEFT} {LEFT}";:GOTO69Ø                        :rem 62
62Ø GOTO57Ø                                        :rem 1Ø9
63Ø IFA<48ORA>57THEN58Ø                            :rem 1Ø5
64Ø PRINTA$;:N=N*1Ø+A-48                           :rem 1Ø6
65Ø IFN>255 THEN A=2Ø:GOSUB1ØØØ:GOTO6ØØ            :rem 229
66Ø Z=Z+1:IFZ<3THEN58Ø                             :rem 71
67Ø IFZ=ØTHENGOSUB1ØØØ:GOTO57Ø                     :rem 114
68Ø PRINT",";:RETURN                               :rem 24Ø
69Ø S%=PEEK(2Ø9)+256*PEEK(21Ø)+PEEK(211)           :rem 149
691 FORI=1TO3:T=PEEK(S%-I)                         :rem 67
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT            :rem 2Ø5
7ØØ PRINTLEFT$("{3 LEFT}",I-1);:RETURN             :rem 7
71Ø PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}"          :rem 236
715 PRINT"{2 DOWN}(PRESS {RVS}RETURN{OFF} ALONE TO
     CANCEL SAVE){DOWN}"                           :rem 1Ø6
72Ø F$="":INPUT"{DOWN} FILENAME";F$:IFF$=""THENPRI
    NT:PRINT:GOTO31Ø                               :rem 71
73Ø PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                               :rem 228
74Ø GETA$:IFA$<>"T"ANDA$<>"D"THEN74Ø               :rem 36
75Ø DV=1-7*(A$="D"):IFDV=8THENF$="Ø:"+F$:OPEN15,8,
    15,"S"+F$:CLOSE15                              :rem 212
76Ø T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                        :rem 3
762 POKE781,ZK-PEEK(782)*256:POKE78Ø,LEN(T$):SYS65
    469                                            :rem 1Ø9
763 POKE78Ø,1:POKE781,DV:POKE782,Ø:SYS65466:rem 68
765 K=S:POKE254,K/256:POKE253,K-PEEK(254)*256:POKE
    78Ø,253                                        :rem 17
766 K=E+1:POKE782,K/256:POKE781,K-PEEK(782)*256:SY
    S65496                                         :rem 235
77Ø IF(PEEK(783)AND1)OR(191ANDST)THEN78Ø           :rem 111
775 PRINT"{DOWN}DONE.{DOWN}":GOTO31Ø               :rem 113
78Ø PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN72Ø                                :rem 171
781 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO72Ø                                    :rem 1Ø3
79Ø PRINT"{CLR}{RVS}*** LOAD ***{2 DOWN}"          :rem 212
```

```
795 PRINT"{2 DOWN}(PRESS {RVS}RETURN{OFF} ALONE TO
     CANCEL LOAD)"                                  :rem 82
800 F$="":INPUT"{2 DOWN} FILENAME";F$:IFF$=""THENP
    RINT:GOTO310                                    :rem 144
810 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                                :rem 227
820 GETA$:IFA$<>"T"ANDA$<>"D"THEN820                :rem 34
830 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$            :rem 157
840 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                         :rem 2
841 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                             :rem 107
845 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 70
850 POKE780,0:SYS65493                              :rem 11
860 IF(PEEK(783)AND1)OR(191ANDST)THEN870            :rem 111
865 PRINT"{DOWN}DONE.":GOTO310                      :rem 96
870 PRINT"{DOWN}ERROR ON LOAD.{2 SPACES}TRY AGAIN.
    {DOWN}":IFDV=1THEN800                           :rem 172
880 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO800                                     :rem 102
1000 REM BUZZER                                     :rem 135
1001 POKE54296,15:POKE54277,45:POKE54278,165
                                                    :rem 207
1002 POKE54276,33:POKE 54273,6:POKE54272,5          :rem 42
1003 FORT=1TO200:NEXT:POKE54276,32:POKE54273,0:POK
     E54272,0:RETURN                                :rem 202
2000 REM BELL SOUND                                 :rem 78
2001 POKE54296,15:POKE54277,0:POKE54278,247
                                                    :rem 152
2002 POKE 54276,17:POKE54273,40:POKE54272,0:rem 86
2003 FORT=1TO100:NEXT:POKE54276,16:RETURN           :rem 57
3000 PRINTC$;"{RVS}NOT ZERO PAGE OR ROM":GOTO1000
                                                    :rem 89
```

The Automatic Proofreader

Charles Brannon

"The Automatic Proofreader" will help you type in program listings without typing mistakes. It is a short error-checking program that hides itself in memory. When activated, it lets you know immediately after typing a line from a program listing if you have made a mistake. Please read these instructions carefully before typing any programs in this book.

Preparing the Proofreader

1. Using the listing below, type in the Proofreader. Be very careful when entering the DATA statements—don't type an *l* instead of a *1*, an *O* instead of a *0*, extra commas, etc.
2. Save the Proofreader on tape or disk at least twice *before running it for the first time.* This is very important because the Proofreader erases part of itself when you first type RUN.
3. After the Proofreader is saved, type RUN. It will check itself for typing errors in the DATA statements and warn you if there's a mistake. Correct any errors and save the corrected version. Keep a copy in a safe place—you'll need it again and again, every time you enter a program from this book, *COMPUTE!'s Gazette,* or *COMPUTE!* magazine.
4. When a correct version of the Proofreader is run, it activates itself. You are now ready to enter a program listing. If you press RUN/STOP–RESTORE, the Proofreader is disabled. To reactivate it, just type SYS 886 and press RETURN.

Using the Proofreader

Many listings in this book have a *checksum number* appended to the end of each line, for example, *:rem 123. Don't enter this statement when typing in a program.* It is just for your information. The rem makes the number harmless if someone does type it in. It will, however, use up memory if you enter it, and it will confuse the Proofreader, even if you entered the rest of the line correctly.

When you type in a line from a program listing and press RETURN, the Proofreader displays a number at the top of your screen. *This checksum number must match the checksum*

number in the printed listing. If it doesn't, it means you typed the line differently from the way it is listed. Immediately recheck your typing. Remember, don't type the rem statement with the checksum number; it is published only so that you can check it against the number which appears on your screen.

The Proofreader is not picky about spaces. It will not notice extra spaces or missing ones. This is for your convenience, since spacing is generally not important. But occasionally proper spacing *is* important, so be extra careful with spaces.

Due to the nature of a checksum, the Proofreader will not catch all errors. Since 1 + 3 + 5 = 3 + 1 + 5, the Proofreader cannot catch errors of transposition. Thus, the Proofreader will not notice if you type GOTO 385 where you mean GOTO 835. In fact, you could type in the line in any order and the Proofreader wouldn't notice. The Proofreader should help you catch most typing mistakes, but keep this in mind if a program that checks out with the Proofreader still seems to have errors.

There's another thing to watch out for: If you enter the line by using abbreviations for commands, the checksum will not match up. But there is a way to make the Proofreader check it. After entering the line, list it. This eliminates the abbreviations. Then move the cursor up to the line and press RETURN. It should now match the checksum. You can check whole groups of lines this way.

Special Tape SAVE Instructions

When you're through typing in a listing, you must disable the Proofreader before saving the program on tape. Disable the Proofreader by pressing RUN/STOP–RESTORE (hold down the RUN/STOP key and sharply hit the RESTORE key). This procedure is not necessary for disk SAVEs, *but you must disable the Proofreader this way before a tape SAVE.*

Saving to tape erases the Proofreader from memory, so you'll have to load and run it again if you want to type another listing. Saving to disk does not erase the Proofreader.

Hidden Perils

The Proofreader's home in memory is not a very safe haven. Since the cassette buffer is wiped out during tape operations,

you need to disable the Proofreader with RUN/STOP–RESTORE before you save your program. This applies only to tape use. Disk users have nothing to worry about.

Not so for 64 owners with tape drives. What if you type in a program in several sittings? The next day, you come to your computer, load and run the Proofreader, then try to load the partially completed program so you can add to it. But since the Proofreader is trying to hide in the cassette buffer, it is wiped out!

What you need is a way to load the Proofreader after you've loaded the partial program. The problem is, a tape LOAD to the buffer destroys what it's supposed to load.

If you intend to type in a program in more than one sitting or wish to make a safety SAVE, follow this procedure:

1. Load and run the Proofreader.
2. Disable it by pressing RUN/STOP–RESTORE.
3. Type the following three lines in direct mode (without line numbers):

```
A$="PROOFREADER.T":B$="{10 SPACES}":FORX=1TO4:A$=A
  $+B$:NEXTX

FORX=886TO1018:A$=A$+CHR$(PEEK(X)):NEXTX

OPEN1,1,1,A$:CLOSE1
```

After you enter the last line, you will be asked to press RECORD and PLAY on your cassette recorder. Put this program at the beginning of a new tape.

You now have a new version of the Proofreader. Turn your computer off and on, then load the program you were working on. Put the cassette containing the Proofreader into the tape unit and type:

OPEN1:CLOSE1

You can now start the Proofreader by typing SYS 886. To test this, entering PRINT PEEK (886) should return the number 173. If it does not, repeat the steps above, making sure that A$ ("PROOFREADER.T") contains 13 characters and that B$ contains 10 spaces.

You can now reload the Proofreader into memory whenever the LOAD or SAVE command destroys it, restoring your personal typing helper.

The Automatic Proofreader

```
100 PRINT"{CLR}PLEASE WAIT...":FORI=886TO1018:READ
    A:CK=CK+A:POKEI,A:NEXT
110 IF CK<>17539 THEN PRINT"{DOWN}YOU MADE AN ERRO
    R":PRINT"IN DATA STATEMENTS.":END
120 SYS886:PRINT"{CLR}{2 DOWN}PROOFREADER ACTIVATE
    D.":NEW
886 DATA 173,036,003,201,150,208
892 DATA 001,096,141,151,003,173
898 DATA 037,003,141,152,003,169
904 DATA 150,141,036,003,169,003
910 DATA 141,037,003,169,000,133
916 DATA 254,096,032,087,241,133
922 DATA 251,134,252,132,253,008
928 DATA 201,013,240,017,201,032
934 DATA 240,005,024,101,254,133
940 DATA 254,165,251,166,252,164
946 DATA 253,040,096,169,013,032
952 DATA 210,255,165,214,141,251
958 DATA 003,206,251,003,169,000
964 DATA 133,216,169,019,032,210
970 DATA 255,169,018,032,210,255
976 DATA 169,058,032,210,255,166
982 DATA 254,169,000,133,254,172
988 DATA 151,003,192,087,208,006
994 DATA 032,205,189,076,235,003
1000 DATA 032,205,221,169,032,032
1006 DATA 210,255,032,210,255,173
1012 DATA 251,003,133,214,076,173
1018 DATA 003
```

Program Merge Utility

Craig Chamberlain

Being able to easily merge two or more programs together can be extremely useful. In Part 3 of this book, for example, there are a series of programs that must be merged with "SID.BAS." The directions told you to load SID.BAS before you began entering each of the programs. When you finished entering the program lines, SID.BAS would be appended to the end—you could then save the entire program. But what if you forgot to load SID.BAS and entered all the lines? Without a merge program you might have to retype the entire program.

That's why we've included a simple merge utility. It's easy to use and can merge two or more programs together. Notice that this is a true *merge*, not an append. This means that if one program contains lines numbered 1, 2, 3, 7, 8, 10, and a second contains lines 3, 4, 5, 6, 9, 11, the merged program will have lines 1 through 11. Since both original programs have a line 3 you must be careful which program is loaded first. The last line 3 entered will be the one which appears in the merged program. (Please note that this merge program will not work if the bit-mapped graphics extensions to BASIC are resident in memory.)

Using the Merge

Enter Program E-1, and save it before running. Use "The Automatic Proofreader," from Appendix D, to insure that it's entered correctly. When you run Program E-1, the following options will be displayed on the screen:

F1 CLEAR BUFFER
F3 START ADDING TO BUFFER
F5 STOP ADDING TO BUFFER
F7 DUMP BUFFER (MERGE)

Once you see this menu, you're ready to merge. Wait for the READY. prompt, then load the first file from disk or tape. Once it's loaded, press f3 and the border of the screen turns black. List the program, then press the f5 key. Now load the file you wish to merge with the first. Once that's loaded, press f7; the two programs are now merged. Save the new program. It's possible to merge more than two files together simply by

repeating the procedure. You must press f3 *before* and f5 *after* listing each program (except the last program to be merged, where only f7 is necessary).

(Note: Once you press the f3 key, you should press only the keys necessary to list the program to the screen. Do *not* press the CTRL key to slow down the listing or any other key. Also, once you press the f7 key to merge the programs, do *not* touch the keyboard while the screen scrolls.)

To see how easy the merge is to use, we've provided three short programs that when merged together offer a short demonstration. Individually, they will do nothing—only after you've merged them will they work. Enter and save programs E-2, E-3, and E-4. If you're using tape, save each program in order on the same tape.

To merge the programs, load and run Program E-1. Then follow these steps:

1. **Load Program E-2**
2. **Press f3**
3. **List the program**
4. **Press f5**
5. **Load Program E-3**
6. **Press f3**
7. **List the program**
8. **Press f5**
9. **Load Program E-4**
10. **Press f7**
11. **Press RUN/STOP-RESTORE**
12. **Run the program**

If the program runs correctly, you can save it just as you would any other program. You must press *RUN/STOP-RESTORE to disable the merge before running any program.*

If you wish to do several merges, simply skip steps 11 and 12 above and save your merged program. Then press f1 to clear the buffer. Now you can begin the process over again.

Program E-1. Merge Utility

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix D.

```
100 PRINT CHR$(147):PRINT " DANDY PROGRAM MERGING
    {SPACE}UTILITY":REM V1.0                :rem 223
110 PRINT " BY CRAIG CHAMBERLAIN":PRINT     :rem 65
300 PRINT " F1 CLEAR BUFFER":PRINT " F3 START ADDI
    NG TO BUFFER"                           :rem 18
```

```
310 PRINT " F5 STOP ADDING TO BUFFER":PRINT " F7 D
    UMP BUFFER (MERGE)":PRINT                    :rem 42
320 PRINT " RUN/STOP RESTORE TO REMOVE UTILITY":PR
    INT                                          :rem 96
330 BP=208:EP=255:PRINT " PLEASE WAIT...":PRINT
                                                 :rem 242
500 FOR K=679 TO 707:READ P:POKE K,P:NEXT:FOR K=82
    8 TO 998:READ P:POKE K,P:NEXT                :rem 33
510 POKE 65530,134:POKE 65531,234:POKE 252,BP:SYS
    {SPACE}679                                   :rem 93
520 FOR K=679 TO 761:READ P:POKE K,P:NEXT:POKE 690
    ,BP:POKE 718,BP:POKE 749,BP                  :rem 148
530 POKE 733,PEEK(806):POKE 734,PEEK(807):POKE 754
    ,PEEK(788):POKE 755,PEEK(789)                :rem 146
540 POKE 758,BP+1:POKE 759,EP:POKE 56334,PEEK(5633
    4)AND254                                     :rem 216
550 POKE 655,167:POKE 656,2:POKE 788,223:POKE 789,
    2:POKE 56334,PEEK(56334)OR1                  :rem 196
560 POKE 252,BP+1:POKE 254,BP+1:NEW              :rem 191
800 DATA 165,1,72,41,252,120,133,1,160,0,132,251,1
    32,253,185,60,3,145,251                      :rem 231
801 DATA 200,192,171,208,246,104,133,1,88,96
                                                 :rem 120
810 DATA 165,203,201,5,240,26,201,6,240,88,201,3,2
    40,46,201,4,208,13,169,0,133                 :rem 219
811 DATA 251,133,253,173,246,2,133,252,133,254,96,
    169,2,205,39,3,240,20,162,2                  :rem 198
812 DATA 189,37,3,157,220,2,189,243,2,157,37,3,202
    ,208,241,142,249,2,96,165,251                :rem 62
813 DATA 197,253,208,6,165,252,197,254,240,43,238,
    248,2,208,21,152,160,0,145                   :rem 170
814 DATA 253,230,253,208,29,230,254,165,254,205,24
    7,2,208,20,169,2,44,169,14                   :rem 165
815 DATA 141,249,2,173,221,2,141,38,3,173,222,2,14
    1,39,3,96,166,198,236,137,2                  :rem 211
816 DATA 176,47,160,0,177,251,157,119,2,232,230,25
    1,208,2,230,252,165,251,197                  :rem 213
817 DATA 253,208,20,165,252,197,254,208,14,169,0,1
    41,248,2,133,251,173,246,2                   :rem 166
818 DATA 133,252,208,5,236,137,2,208,213,134,198,9
    6                                            :rem 223
820 DATA 8,165,1,72,41,252,120,133,1,32,0,192,104,
    133,1,40,173,249,2,141,32,208                :rem 10
821 DATA 76,72,235,72,132,2,168,165,1,72,41,252,12
    0,133,1,32,77,192,104,133,1                  :rem 192
822 DATA 88,173,249,2,141,32,208,164,2,104,76,202,
    241,173,248,2,240,13,165,1                   :rem 151
823 DATA 72,41,252,133,1,32,116,192,104,133,1,76,4
    9,234,192,2,208,255,0,14                     :rem 44
```

Program E-2. Merge Demo 1

```
10 REM MERGE DEMO ONE
30 PRINT "{CLR}YOU MUST MERGE THIS PROGRAM WITH ME
   RGE{2 SPACES}DEMO TWO FOR IT TO WORK"
50 GOSUB 200:GOTO 70
70 REM THIS LINE WILL BE DELETED BY MERGE DEMO TWO
210 PRINT CHR$(147)
230 PRINT TAB(X) "THE MERGE WORKS!"
```

Program E-3. Merge Demo 2

```
20 REM MERGE DEMO TWO
40 PRINT "{CLR}YOU MUST MERGE THIS PROGRAM WITH ME
   RGE{2 SPACES}DEMO ONE FOR IT TO WORK"
60 PRINT "THE MERGE WORKED!"
70 REM REPLACE LINE 70
80 PRINT "{7}":END
200 REM SUBROUTINE
220 FOR X= 1 TO 20
240 PRINT "{YEL}";:IF X/2=INT(X/2) THEN PRINT"
    {BLK}";
```

Program E-4. Merge Demo 3

```
10 REM THE MERGE IS COMPLETE
20 REM
30 REM
40 REM
250 NEXT X
260 RETURN
```

Index

ADR envelope. *See* envelopes, nonsustaining
ADSR envelope. *See* envelopes, sustaining
"ALBUMLEAF.MUS" 316–18
animation. *See also* Sprite Control System
  principles of 379–80
  use of bitmapped graphics 380–81
  use of character graphics 380
  use of sprite graphics 381
"ANTIMUNCH" 154
argument variable. *See* DEF statement
ATN functions 25, 26. *See also* transcendental functions
attack 185–86
attack rate 282. *See also* envelopes
"The Automatic Proofreader" 481
  instructions for use 478–80
"BASIC Doodle" 91
"BISTRO.MUS" 327–28
bitmap, relocation of 170–74
bitmapped graphics 97, 106–22
  illustration of coordinate system 107
  mixing text and graphics 164–66
  multicolor modes 119–22
  POKE equivalents for graphics statements 175–77
  SYS equivalents for graphics statements 175–77
Bitmapped Graphics Utility
  command 98
  command index 181
  function 98, 168–70
  line drawing 112–15
  multicolor modes 119–22
  statements 98, 106–17, 123, 149, 153–54, 164
bit mask 85, 86, 88. *See also* WAIT statement
"BLUES" 265–66
"BMG Demonstration" 106
"BMGLOADER Disk Version" 99
"BMGLOADER Tape Version" 100
"BMG.OBJ" 100–05
  explanation of use 98–99
  instructions for making copies 389
"BRAID" 152
"BRAID.SHP" 152–53
"BRASS.MUS" 296–97
"BROTHERHOOD" 127
buffer 71–73, 79. *See also* telecommunications
  illustration of 72
"BUNNY.DEF" 392
  instructions for use 392–93
"CALLIOPE.MUS" 326
chaining, definition of 82, 84
"CHDEMO1" 165–66
"CHDEMO2" 166
"CHSETA.SHP" 159–63
CLOSE statement 42, 43–44, 45, 57
  syntax of 42, 45
CMD statement
  summary of 49–50
  syntax of 47
  use of 47–48
coda. *See* repetition
command number 39, 44, 59, 61–62
  with an RS-232 device 73, 79
commands, use of in programs 81–83
  summary of 83–84
"COMMODORE.MUS" 196–200
conditions, numerical values for 3–6
  summary of 6–8
CONT command 81
  use of in program mode 81, 83–84. *See also* commands, use of in programs
conversion of mathematical formulas 23–24
COS function 25, 26. *See also* transcendental functions
"COURANTE.MUS" 345–46
"Cross File Merge" 374–75
  explanation of 374–75
cutoff frequency. *See* SID chip, filter controls
da capo. *See* repetition
dal segno. *See* repetition
decay 185–86
decay rate 282. *See also* envelopes
DEF statement 27–32
  purpose of a variable 28–29
  summary of 33–35
  syntax of 27–28, 33
  use of function calls 28–32
"DEMO#1" 390–91
  instructions for use 392
"DEMO#2" 393–95
  instructions for use 395
"DEMO#3" 398–400
dense. *See* random function
DEVICE NOT PRESENT error 43, 45, 70. *See also* file input/output
device number 39, 44, 59. *See also* OPEN statement, syntax of

with an RS-232 device 73, 79
"Directory Reader" 65
disk directory
reading of 64–67, 70
use of wild cards 65–67, 70
disk drive commands 61, 62, 63. *See also* disk files
disk files 58–71. *See also* sequential files
program files (PRG) 67, 69, 70–71
relative files (REL) 68, 69, 71
replacement of a file 62
summary of 68–71
USR files 67, 69, 70
dotted notes. *See* music theory, duration of notes
double dots. *See* music theory, duration of notes
dynamics. *See* music theory, volume
"Editor" 226–45. *See also* Sidplayer Editor
instructions for use 246
"EDITOR.OBJ" 247–48
instructions for making copies 389
eighth note. *See* music theory, duration of notes
envelopes 185–86, 282–83, 333, 338, 342. *See also* music theory
illustration of 282, 284
nonsustaining 283
sustaining 283
"ETAL.MUS" 204–206
EXP function 24–25, 26. *See also* transcendental functions
syntax of 24
exponent. *See* scientific notation
"EXTRACT," Bitmapped Graphics 179
explanation of 178–79
"Extract," Sidplayer Editor 372–73
explanation of 371–73
"Extract," Sprite Control System 447–48
explanation of 447
instructions for use 448
"FAMILY.SHP" 127–28
file creation 38
FILE DATA error 43, 44, 58. *See also* file input/output
file input/output 37–45, 57
error messages 42, 43
filename 39
FILE NOT OPEN error 43, 45. *See also* file input/output
file number 39, 44, 59. *See also* OPEN statement, syntax of
range of 48, 50
with an RS-232 device 73, 79
FILE OPEN error 42, 45. *See also* file input/output
filters 290–91, 342. *See also* music theory
illustrations of 291
"Flags" 118–19
flats. *See* pitch
floating-point variables 12, 13, 14, 15
frequency 185
"FSONATINA.MUS" 275–79
fundamental frequency. *See* filters
GET# statement 64, 70
syntax of 64, 70
"GOBBLER.DEF" 396–97
instructions for use 395
grace notes. *See* music theory, duration of notes
graphics character set, use with a printer 48
'GSONATINA.MUS" 279–81
"GUY.DEF" 400–401
instructions for use 401
half note. *See* music theory, duration of notes
harmonics. *See* filters
"HOLST.MUS" 339
IF-THEN statement 7
simplification of 3
input 37, 38, 69
statements 38
summary of statements 44–45
INPUT# statement 40–41, 44
syntax of 40, 44
I/O devices
kinds of 37, 38, 51
identification numbers 39, 42–43, 57
integer variables
computation of 13–14, 15
in FOR-NEXT loops 13, 15
range of 12, 15
summary of 14–15
syntax of 12–14, 15
use of 11–14, 15
use of with negative numbers 12–13
use of with positive numbers 12
"JOKE.MUS" 346–47
"K.C.O.MUS" 319
"KOTO.MUS" 297
LIST command. *See also* commands, use of in programs
use of in program mode 81, 83
"Lister" 365–67
explanation of 364, 367–69
LOAD command. *See also* commands, use of in programs
secondary address 81, 82, 84

syntax of 81–82, 84
use of in program mode 81, 82
LOG function. *See also* transcendental functions
syntax of 24
logical lines 19, 21
machine language
explanation of 88–90
summary of 93–94
"Machine Language Doodle" 91–92
"The Machine Language Editor: MLX" 474–77
instructions for use 471–74
mantissa. *See* scientific notation
mathematical calculations, using BASIC
limitations of 9
precision of 8
summary of 11
"MAZE" 169
"MERGE," Bitmapped Graphics 180–81
explanation of 180
"Merge," Sidplayer Editor 369–71
explanation of 369, 371
"Merge," Sprite Control System 445–46
explanation of 445
instructions for use 446–47
"Merge Demo 1" 485
instructions for use 483
"Merge Demo 2" 485
instructions for use 483
"Merge Demo 3" 485
instructions for use 483
"Merge Utility" 483–84
explanation of 482
instructions for use 482–83
mode. *See* SID chip, filter controls
mode number 107–108. *See also* bit-mapped graphics
modem 71, 73–77, 78
baud rate 73
command character 75
control character 73, 74
duplex 75
handshaking 75
word length 73
"Modem" 77
explanation of 77–78
"MODERNART" 177–78
"Munchkins" 125
"MUNCHKIN.SHP" 126
music theory 206–25, 271–73, 282–83, 290–91, 305–49
accidentals 309–11
detuning 320–21, 343
duration of notes 217–19, 225, 226, 311–14
dynamics 306–307
illustrations of grand staff 207, 210
key 213–16, 226
key changes 307–308
measures 219–20, 226
notation 206–207
portamento 314–15, 342–43
repetition 298–99
rests 221–22, 226
ring modulation 333–37
slurs 223, 226
summary of 225–26
synchronization 324–25, 344
tempo 220–21, 226
tempo changes 305–306
ties 222–23, 226
time signatures 308–309
transposing 322–24, 343–44
trills 314
vibrato 315, 320, 343
voice 224–25, 226
volume 223–24, 226
NEW command 62–63. *See also* commands, use of in programs; disk drive commands
syntax of 62–63, 69
use of in program mode 81, 83
noise waves 273. *See also* waveforms
illustration of 273
nonrelocatable LOAD 82, 84
NOT INPUT FILE error 43, 45, 56–57, 58. *See also* file input/output
NOT OUTPUT FILE error 43, 45, 56, 58. *See also* file input/output
octave. *See* pitch
ON-GOTO statement
examples of 6, 7
simplified use of 6, 7
OPEN statement 39, 40–41
syntax of 39, 44, 48, 50, 51, 57, 59, 67, 68, 73
use of with a printer 46
with an RS-232 device 73, 79
output 37, 38, 69
statements 38, 41
summary of statements 44–45
OVERFLOW error. *See* scientific notation
parallel communication, definition of 71, 78
"Parts" 20
physical lines 19, 21
"PINBALL" 170
"PIPERS.MUS" 321–22
pi symbol, location of the keyboard 25
pitch 185, 208–13, 225, 309–11, 314–15, 320, 322–25, 333–38, 342–44. *See also* music theory

"PLAYER," disk version 188–89
explanation of 187, 190–91
"PLAYER," tape version 189–90
explanation of 187, 190–91
POS function 19–20, 21.
print formatting functions 19–21
summary of 21–22
printing
to a printer 45, 46, 48
to the screen 45, 46
PRINT statement 3–4, 7, 45
PRINT# statement 41, 45, 49
syntax of 41, 45
"PROMENADE.MUS" 340–41
pulse waves 272. *See also* waveforms
illustration of 272
quarter note. *See* music theory, duration of notes
radians 25, 26
random function (RND) 15–17
kinds of 16
summary of 17
Register Storage Area 92. *See also* SYS command
relational operator 3
relative positioning. *See* Shapedit, features of
release 185–86
release rate 282. *See also* envelopes
relocatable LOAD 82, 84
RENAME command 62, 69. *See also* disk drive commands
syntax of 62, 69
repetition 298–99, 302–3. *See also* music theory
resolution 107–8, 109. *See also* bitmapped graphics
resonance. *See* SID chip, filter controls
rounding function, of a number 26–27
adding it to BASIC 27
RS-232 communication. *See also* modems
definition of 71
summary of 78–79
RUN command, use of in program mode 81, 83. *See also* commands, use of in programs.
SAVE command, use of in program mode 83, 84. *See also* commands, use of in programs.
sawtooth waves 271. *See also* waveforms
illustration of 271
scientific notation, explanation of 9–10, 11
"SCIPIO.MUS" 356–59
SCRATCH command 61–62, 69. *See also* disk files
"SCS.BAS" 382–83
"SCS.OBJ" 384–88
instructions for making copies 389
secondary address. *See* command number
seed value. *See* random function
sequential file 58–59, 60, 67, 68, 69. *See also* disk files; tape files
reading of 59–60
writing of 59–60
serial communication, definition of 71, 78
Shapedit 130–37
combining sprites with bitmapped graphics 167
features of 137–46
instructions for use 130
reference chart 145
use of to redefine custom characters 166–67
shape tables 123–29
plotting process 150–51
using shapes in BASIC programs 146–48
sharps. *See* pitch
"SHP.BAS" 124
"SID.BAS" 351–52
SID chip 185, 186–87, 344, 348
filter controls 290–96
illustration of 186
"SIDDEMO" 360–62
"SID.OBJ" 191–96
instructions for making copies 389
Sidplayer Editor
command index 375
commands 255–59, 273–75, 283–87, 290–94, 299–302, 306–307, 310, 312–13, 315, 320–21, 323, 324–25, 333, 342–44, 355–56, 362
DROP routine 354
error checking features 262–63
function key summary 260
HOOK routine 353–54
illustration of instrument parameters 349
joystick editing 267–68
keyboard note entry 266–67
merging with BASIC programs 351–63
PLAY routine 353–54
relocating of 363–64
use of with advanced music theory 305–49
using the program 249–68
SIN function 25, 26. *See also* transcendental functions
"SITAR.MUS" 297–98
sixteenth note. *See* music theory, duration of notes

SPC function 20, 21, 22. *See also* print formatting functions
sprite collision 4
Sprite Control System
autoshape features 434–36
available definition blocks 433–34
chase mode 440–41
command index 463
disabling the display of a sprite 443
enabling the display of a sprite 443
explanation of 381–82
instructions for use 389–90
joystick control of a sprite 438–40
multiple sprites 430
relocation of screen memory 452–56
relocation of Sprite Control System 456–57
setting priority of each sprite 437–38
synchronization of sprite motion 441–43
SYS calls 421–30
use of in BASIC programs 419–43, 445–63
use of multicolor mode 437
use of with Bitmapped Graphics 457–59
use of with Bitmapped Graphics and Sidplayer Editor 460–62
use of with Sidplayer Editor 459–60
"The Sprite Editor" 403–10. *See also* Sprite Control System
joystick editing 410
editing commands 410–17
explanation of 381–82
sprites 380, 381. *See also* Sprite Control System
animation of multiple sprites 430
designing of 403–18
illustration of direction numbers 426
SQR function 22–23, 26. *See also* transcendental functions
syntax of 23
square waves 271–72. *See also* waveforms
illustration of 272
string 59. *See also* disk files
with an RS-232 device 73, 79
STRING TOO LONG error 53, 58. *See also* tape files, input
sustain 185–86
SYS command 88, 92, 93–94
TAB function 20–21, 22. *See also* print formatting functions
use of to justify text 20–21
TAN function 25, 26. *See also* transcendental functions
tape files 50–57
advanced feature of 56
importance of CLOSE statement 57, 58
input 51, 52, 53
output 51, 53
prevention of running off the end of a file 54–55, 58
summary of 57–58
telecommunications
definition of 71
demodulation 71, 78
explanation of 71–77
modulation 71, 78
tempo terminology. *See* music theory, tempo
"Theme and Variation" 287–89
timbre. *See* filters
"TOWERS" 148
"TOWER.SHP" 149
"TOWER II" 154–55
"TPI#14.MUS" 328–32
transcendental functions
summary of 26
use of in mathematical-oriented applications 22–25
triangle waves 271. *See also* waveforms
illustration of 271
trigonometry. *See* transcendental functions
triplets. *See* music theory, duration of notes
"TULIPS" 128–29
"TULIPS.SHP" 129
"TWIST" 151–52
type. *See* integer variables, use of
underflow error. *See* scientific notation
uppercase character set, use with a printer 48
USR function 93, 94
VALIDATE command 63, 70. *See also* disk drive commands
WAIT statement 84, 88. *See also* Sidplayer Editor, merging with BASIC programs
summary of 88
syntax of 88
use of in program mode 84–88
waveform 185, 333, 337–38. *See also* music theory
types of 271–73, 337–38
"WEATHER" 155–56
"WEATHER.SHP" 157–59
whole note. *See* music theory, duration of notes
"WSOLDIER.MUS" 201–204
"YOY.MUS" 347

To order your copy of the *All About the Commodore 64, Volume Two Disk,* call our toll-free US order line: 1-800-334-0868 (in NC call 919-275-9809) or send your prepaid order to:

All About the Commodore 64, Volume Two Disk
COMPUTE! Publications
P.O. Box 5406
Greensboro, NC 27403

All orders must be prepaid (check, charge, or money order). NC residents add 4.5% sales tax.

Send _____ copies of the *All About the Commodore 64, Volume Two Disk* at $12.95 per copy

Subtotal $_____

Shipping & Handling: $1.00/disk* $_____

Sales tax (if applicable) $_____

Total payment enclosed $_____

*Outside US and Canada, add $3.00 per disk for shipping and handling. All payments must be in US funds.

□ Payment enclosed
Charge □ Visa □ MasterCard □ American Express

Acct. No. _____ Exp. Date _____

Name _____

Address _____

City _____ State _____ Zip _____

Please allow 4-5 weeks for delivery.

75 3106 B

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!'s Gazette** for Commodore.

For Fastest Service
Call Our **Toll-Free** US Order Line
800-334-0868
In NC call 919-275-9809

COMPUTE!'s Gazette

P.O. Box 5406
Greensboro, NC 27403

My computer is:
☐ Commodore 64 ☐ VIC-20 ☐ Other_____

☐ $24 One Year US Subscription
☐ $45 Two Year US Subscription
☐ $65 Three Year US Subscription

Subscription rates outside the US:

☐ $30 Canada
☐ $65 Air Mail Delivery
☐ $30 International Surface Mail

Name _____

Address _____

City _____ State _____ Zip _____

Country _____

Payment must be in US funds drawn on a US bank, international money order, or charge card. Your subscription will begin with the next available issue. Please allow 4–6 weeks for delivery of first issue. Subscription prices subject to change at any time.

☐ Payment Enclosed ☐ Visa
☐ MasterCard ☐ American Express

Acct. No. _____ Expires ___/___

The *COMPUTE!'s Gazette* subscriber list is made available to carefully screened organizations with a product or service which may be of interest to our readers. If you prefer not to receive such mailings, please check this box ☐.

753199

COMPUTE! Books

Ask your retailer for these **COMPUTE! Books** or order directly from **COMPUTE!**.

Call toll free (in US) **800-334-0868** (in NC 919-275-9809) or write COMPUTE! Books, P.O. Box 5406, Greensboro, NC 27403.

Quantity	Title	Price*	Total
_____	COMPUTE!'s Commodore Collection, Volume 1	**$12.95**	_____
_____	Commodore Peripherals: A User's Guide	**$ 9.95**	_____
_____	Creating Arcade Games on the Commodore 64	**$14.95**	_____
_____	Machine Language Routines for the Commodore 64	**$14.95**	_____
_____	Mapping the Commodore 64	**$14.95**	_____
_____	COMPUTE!'s First Book of VIC	**$12.95**	_____
_____	COMPUTE!'s Second Book of VIC	**$12.95**	_____
_____	COMPUTE!'s Third Book of VIC	**$12.95**	_____
_____	COMPUTE!'s First Book of VIC Games	**$12.95**	_____
_____	COMPUTE!'s Second Book of VIC Games	**$12.95**	_____
_____	Creating Arcade Games on the VIC	**$12.95**	_____
_____	Programming the VIC	**$24.95**	_____
_____	VIC Games for Kids	**$12.95**	_____
_____	Mapping the VIC	**$14.95**	_____
_____	The VIC and 64 Tool Kit: BASIC	**$16.95**	_____
_____	Machine Language for Beginners	**$14.95**	_____
_____	The Second Book of Machine Language	**$14.95**	_____
_____	Computing Together: A Parents & Teachers Guide to Computing with Young Children	**$12.95**	_____
_____	BASIC Programs for Small Computers	**$12.95**	_____

*Add $2.00 per book for shipping and handling.
Outside US add $5.00 air mail or $2.00 surface mail.

Shipping & handling: $2.00/book _____
Total payment _____

All orders must be prepaid (check, charge, or money order).
All payments must be in US funds.
NC residents add 4.5% sales tax.
☐ Payment enclosed.
Charge ☐ Visa ☐ MasterCard ☐ American Express

Acct. No._____ Exp. Date_____

Name_____

Address_____

City_____ State _____ Zip_____

*Allow 4–5 weeks for delivery.
Prices and availability subject to change.
Current catalog available upon request.

A Brand New Computer

The Commodore 64 is a powerful personal computer. Its graphics and sound capabilities are among the best. But these features haven't always been the easiest to use. Sometimes they seem so complicated that you give up before you start.

That all changes as soon as you have *All About the Commodore 64, Volume Two* in hand. Its three major utilities can transform your computer into a machine you won't recognize. Bitmap graphics, synthesized music, and animated sprites are now easy to use—and to insert into your own BASIC programs.

You needn't have read Volume One, though it may help you to better understand BASIC. In fact, Part 1, "Advanced BASIC," picks up where Volume One left off. You'll learn about numbers and mathematics, functions, file input/output, and the powerful WAIT statement. But the heart of this book is its extraordinary machine language utilities. With the convenience of BASIC, but with the speed and power of machine language, you'll quickly be enhancing your own programs with dazzling graphics displays and fascinating sound and music.

Part of what *All About the Commodore 64, Volume Two* can do for your computer includes:

- Give you powerful new graphics commands that allow you to draw on and fill the bitmap screen.
- Simplify shape creation. "Shapedit," an easy-to-use shape editor, lets you create even the most complex drawing.
- Turn your 64 into a sophisticated music machine. "Sidplayer" and an elaborate Editor let you create, or copy from sheet music, the most detailed scores.
- Play music without slowing a BASIC program down.
- Create sprites with the Editor, save them to disk or tape, even merge them with BASIC programs.
- Achieve continuous sprite motion, whether controlled by the computer, or by you, that keeps going while BASIC is doing something else.

Each utility, by itself, can make your computer do impressive things. The programs are integrated, too, so you can use all three together.

All About the Commodore 64, Volume Two is the book which shows you how to make the best use of your computer's special features. Isn't it about time you got control over the power waiting beneath your keyboard?

ISBN 0-942386-45-0



This was brought to you

from the archives of

http://retro-commodore.eu