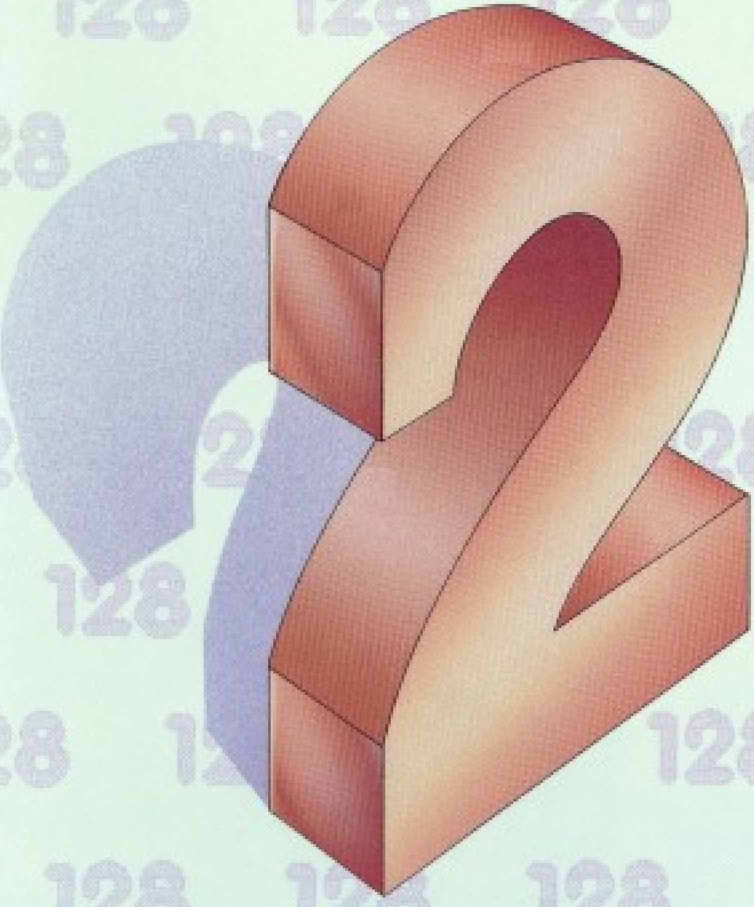
COMPUTE!'s



Second Book of Commodore 128

Over two dozen of our best applications, games, and tutorials for the Commodore 128 home computer—ready to type in and enjoy.

A **COMPUTE! Books** Publication $16.95

COMPUTE!'s SECOND BOOK OF Commodore 128

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

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-87455-077-7

Contents

Foreword

You'll find collected here some of the best material from *COMPUTE!* magazine and *COMPUTE!'s Gazette,* plus several never-before-published programs. *COMPUTE!'s Second Book of Commodore 128* offers dozens of programming tips, tutorials, applications, and games that will entertain you and inform you. No matter what your interest or computing background, the programs included here have been chosen to make your 128 more useful than ever.

If you like games, try "Puzzle Grid"; it's trickier than you might think (and it also gives you an opportunity to create your own puzzles). "Hex War" is a game of thought and strategy rather than quick reflexes. "Miami Ice" requires a combination of fast action and concentration. And "Pig$ for Buck$" will delight kids of all ages.

If programming is your interest, you'll find hints and tips to make using your disk drive easier and take less time—time that can be spent on programming. "Commodore 128 Hi-Res Text Manipulation" explores some techniques for adding text to your hi-res artwork. In "ESCaping with the 128," you'll be shown how to access 27 new screen-editing features with ESC-key sequences, and you'll learn how to use them in your own programs. And there's an article on machine language that examines some basic architectural features of the 128.

Graphics and music programs are fun to work with and have practical applications for your programs as well. "Artimation" is a way to use BASIC 7.0's hi-res graphics statements to create kinetic computer art. "Mastering 128 Sound and Music" explores the sound statements that can help you generate high-quality musical arrangements.

The CP/M operating system offers a lot to programmers. An introduction to CP/M mode tells you how to boot CP/M, discusses CP/M commands, and shows you how to create a PROFILE program. In addition, an article on public domain CP/M software clues you into gaining access to a whole new world of free and inexpensive software.

COMPUTE!'s Second Book of Commodore 128 is also strong in applications and utilities. You'll find a database program, a program that continuously displays a scrolling message on the screen, and a way to create a complete custom character set for the 128's 80-column screen. And you'll learn how to personalize your 128—changing the defaults so that the computer automatically turns on with your personal preferences of colors, tab settings, repeating keys, and cursor modes. There are utilities that enable you to redefine keys to print whatever characters you choose, that speed up the 128's action in 64 mode, and that automatically load and run any 64 program from disk when you boot your system.

Written clearly and concisely, *COMPUTE!'s Second Book of Commodore 128* will make your computer more valuable than ever. All the programs have been fully tested, and "The Automatic Proofreader" and "MLX" will assist you in typing in programs correctly the first time.

The programs in *COMPUTE!'s Second Book of Commodore 128* are ready to type in and run. If you prefer not to type in the programs, however, you can order a disk that includes all the programs from the book. Call toll-free 1-800-346-6767 (in New York 1-212-887-8525), or use the coupon in the back of the book.

Chapter 1

Games

Miami Ice

Jeff Kulczycki

Here's an action game that challenges both your driving skills and powers of concentration. A joystick is required.

Ah, Miami—sun city of the South. A sparkling metropolis blessed with a tropical climate, palm trees, beaches, revived art deco architecture, stylish pastels, and classy elegance. Almost paradise.

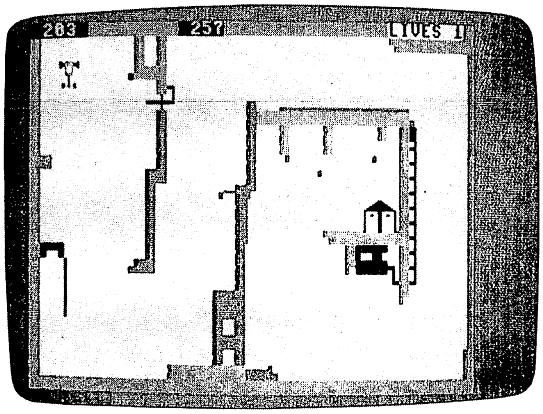
You wake up on another bright, sunny Miami morning, sip a glass of freshly squeezed orange juice, don your white linen suit and sunglasses, and stroll outside—then get the shock of your life.

What's Going On Here?

Overnight, a freak shift in the jet stream has piped a blistering cold front down from Ohio. The weathercaster had predicted a brief shower last evening, but that's not what happened. Instead, the Florida peninsula has been blasted by the worst ice storm in 400 years. The Everglades are frozen solid. The pink flamingos are blue. And the streets of Miami are coated with a shimmering layer of slippery ice.

As you start your car—the pampered engine coughs and sputters in the bitter cold—you wonder what it's going to be like driving to work. A Miami native, you've never driven on ice before. In fact, you've never even *seen* this much ice since your boss's retirement party last year, when the caterers made that life-size ice sculpture of Ponce de Leon. You've heard the horror stories told by tourists about winter driving conditions up North, but never thought it could happen to you. Not here in Miami.

The minute you pull out onto the street, your worst fears come true. When you step on the gas pedal, the wheels spin and the car accelerates sluggishly. When you turn the steering wheel, the car slides all over the road. And when you step on the brakes—well, forget it.



Driving through the streets of Miami isn't easy when they're covered with a layer of ice.

You realize, desperately, that you've got to make it to the parking garage across town without smashing your car to smithereens. It won't be easy. But at least there's one thing in your favor—you've got the whole road to yourself. Everyone else, it seems, had the good sense to stay home.

Out of Control

Using a joystick, you have to drive your car over ice-covered streets to reach the safety of a garage. The joystick button is the gas pedal, and pushing the stick right or left steers the car in the corresponding direction.

But here's the twist: The car doesn't respond instantly to your commands. It tends to slide in the same direction even after you've steered it toward another direction. Then, when you try to recover, you often overcorrect and start sliding in yet another new direction. It's an inertial nightmare, much like real winter driving.

When you hit a guardrail or some other obstruction, your car cracks up. You get three cars per game. If you reach the safety of the garage, the game isn't over. Instead, you advance to another screen whose streets are even harder to navigate.

The number of points you score depends on how soon you reach the garage. As an incentive to recklessness, a timer starts counting down when you begin each new screen. If you reach the garage, you score the number of points left on the timer. If the timer runs out, you can still reach the garage, but you won't get any points. However, you will advance to the next screen.

How to Play

"Miami Ice" is written completely in BASIC using BASIC 7.0's excellent sprite commands. Plug a joystick into port 2 and leave a disk in the drive. After each game, if your score ranks you among the top players, the program lets you enter your initials and then saves the high-score data to disk.

To complete each level, you merely have to steer your car into the parking garage from any angle. There are a total of four screens, and each screen displays the timer value in the upper left corner. Your current score can be seen immediately to the right.

Miami Ice

See instructions in article, and read Appendix C, "MLX," before typing in the following program listings.

Starting Address: 1C01
Ending Address: 4A40

```
1C01:4D 1C 0A 00 9F 32 2C 38 7E
1C09:2C 32 2C 22 48 49 2D 53 A0
1C11:43 4F 52 45 2C 53 2C 57 BB
1C19:22 3A A0 32 3A 9F 31 35 10
1C21:2C 38 2C 31 35 3A 84 31 E2
1C29:35 2C 41 24 2C 42 24 3A 5E
1C31:8B 42 24 B3 B1 22 46 49 6B
1C39:4C 45 20 45 58 49 53 54 24
1C41:53 22 A7 A0 31 35 3A 8D 0B
1C49:37 35 30 00 5D 1C 14 00 F3
1C51:E7 30 2C 31 36 3A E7 34 C0
1C59:2C 31 31 00 87 1C 1E 00 03
1C61:99 22 93 1C 11 11 11 11 23
1C69:11 11 11 1D 1D 1D 1D 1D 17
1C71:1D 1D 1D 1D 1D 1D 1D 1D A9
1C79:1D 1D 12 4D 49 41 4D 49 D1
1C81:20 49 43 45 22 00 AE 1C 63
1C89:28 00 99 22 1F 11 20 20 C8
1C91:20 20 20 20 20 20 20 20 C9
1C99:20 4A 4F 59 53 54 49 43 B5
1CA1:4B 20 49 4E 20 50 4F 52 C8
1CA9:54 20 32 22 00 F6 1C 32 C2
1CB1:00 99 22 90 11 90 20 20 C8
1CB9:20 20 20 20 20 20 20 20 F1
```

```
1CC1:2Ø 5B 4C 45 46 54 5D 2Ø 1D
1CC9:2Ø 54 55 52 4E 2Ø 4C 45 C7
1CD1:46 54 22 3A 99 22 2Ø 2Ø DF
1CD9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 12
1CE1:2Ø 5B 52 49 47 48 54 5D 41
1CE9:2Ø 54 55 52 4E 2Ø 52 49 F7
1CF1:47 48 54 22 ØØ 3D 1D 3C F7
1CF9:ØØ 99 22 2Ø 2Ø 2Ø 2Ø 2Ø CØ
1DØ1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 5B 46 D7
1DØ9:49 52 45 5D 2Ø 2Ø 41 43 42
1D11:43 45 4C 45 52 41 54 45 A1
1D19:22 3A 99 22 9E 11 2Ø 2Ø E1
1D21:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 5B
1D29:2Ø 2Ø 2Ø 52 45 41 44 49 A5
1D31:4E 47 2Ø 44 41 54 41 2E B8
1D39:2E 2E 22 ØØ 6B 1D 46 ØØ B6
1D41:8D 31 38 3Ø 3Ø 3A 99 22 58
1D49:91 1F 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø FB
1D51:2Ø 2Ø 2Ø 2Ø 5Ø 52 45 53 53
1D59:53 2Ø 42 55 54 54 4F 4E C3
1D61:2Ø 54 4F 2Ø 5Ø 4C 41 59 3C
1D69:22 ØØ 7D 1D 5Ø ØØ 8B CF 9F
1D71:28 32 29 B3 B1 31 32 38 9B
1D79:A7 38 3Ø ØØ 9Ø 1D 5A ØØ 49
1D81:48 59 B2 33 3A 53 43 B2 18
1D89:3Ø 3A 53 4E B2 31 ØØ DC FØ
1D91:1D 64 ØØ FE 25 3A 91 53 EB
1D99:4E 8D 37 36 3Ø 2C 31 3Ø 6D
1DA1:32 3Ø 2C 31 32 38 3Ø 2C 98
1DA9:31 35 35 3Ø 3A FE 26 3A C7
1DB1:99 22 13 22 A3 33 32 29 3D
1DB9:3B 22 9Ø 4C 49 56 45 53 72
1DC1:22 3B 48 59 3A E7 3Ø 2C 78
1DC9:31 36 3A 54 4D B2 34 3Ø 84
1DD1:3Ø 3A 54 B2 3Ø 3A 58 45 C8
1DD9:B2 3Ø ØØ E6 1D 6E ØØ 8D 18
1DE1:35 34 3Ø 3A ØØ 1D 1E 78 96
1DE9:ØØ 97 32 3Ø 34 31 2C 36 48
1DF1:32 3A FE Ø6 32 2C 58 2C 33
1DF9:59 3A FE Ø7 32 2C 31 2C 9Ø
1EØ1:32 2C 3Ø 2C 3Ø 2C 3Ø 2C E8
1EØ9:31 3A 97 32 3Ø 34 3Ø 2C 61
1E11:35 37 3A 58 45 B2 CE Ø3 18
1E19:28 32 29 ØØ 7Ø 1E 82 ØØ 1C
1E21:FE Ø6 31 2C 33 3Ø 23 3Ø 18
1E29:3A FE Ø7 31 2C 31 2C 39 ED
1E31:2C 3Ø 2C 3Ø 2C 3Ø 2C 31 C3
1E39:3A FE Ø8 31 2C 32 3A FE Ø4
1E41:Ø6 31 2C 34 3Ø 2C 36 35 69
```

```
1E49:3A 49 B2 34 3A 41 4E B2 B4
1E51:31 38 3Ø 3A 48 54 B2 31 Ø8
1E59:33 35 3A 54 48 B2 3Ø 3A BØ
1E61:58 45 B2 CE Ø3 28 32 29 A4
1E69:AA CE Ø3 28 31 29 ØØ 83 43
1E71:1E 8C ØØ 99 22 13 12 22 1D
1E79:3B 54 4D 3B 22 9D 2Ø 92 2Ø
1E81:22 ØØ 93 1E 96 ØØ 8B CF BE
1E89:28 32 29 B2 3Ø A7 31 35 6E
1E91:3Ø ØØ A3 1E AØ ØØ 8B CF 28
1E99:28 32 29 B2 33 A7 32 38 9B
1EA1:3Ø ØØ B3 1E AA ØØ 8B CF 8A
1EA9:28 32 29 B2 37 A7 33 31 C6
1EB1:3Ø ØØ E5 1E B4 ØØ 8B CF 31
1EB9:28 32 29 B2 31 32 38 A7 51
1EC1:FE Ø6 31 2C 41 4E 23 31 A2
1EC9:3A 54 48 B2 31 3A DA 31 C5
1ED1:2C 35 3Ø 3Ø 3Ø 2C 32 34 45
1ED9:2C 32 2C 31 3Ø 3Ø 3Ø 2C 2Ø
1EE1:33 2C 33 ØØ ØF 1F BE ØØ 9B
1EE9:97 32 3Ø 34 3Ø 2C 35 33 97
1EF1:AA 49 3A 8B CE Ø3 28 31 D9
1EF9:29 B2 33 A7 34 39 3Ø 3A 79
1FØ1:D5 8B CE Ø3 28 32 29 AF 23
1FØ9:31 A7 34 32 3Ø ØØ 26 1F 6Ø
1F11:C8 ØØ 8B 54 48 B1 31 A7 7D
1F19:91 B6 28 54 AB 32 3Ø 29 C7
1F21:89 34 31 3Ø ØØ 3Ø 1F D2 2C
1F29:ØØ 54 B2 54 AA 31 ØØ 4B 7D
1F31:1F DC ØØ 8B 48 54 B1 31 17
1F39:38 3Ø A7 8B 48 54 AB 31 69
1F41:38 3Ø B1 41 4E A7 33 36 9F
1F49:3Ø ØØ 66 1F E6 ØØ 8B 48 F4
1F51:54 B1 31 38 3Ø A7 8B 48 4F
1F59:54 AB 31 38 3Ø B3 41 4E 77
1F61:A7 33 37 3Ø ØØ 81 1F FØ 5F
1F69:ØØ 8B 48 54 B3 31 38 3Ø DB
1F71:A7 8B 48 54 AA 31 38 3Ø 6F
1F79:B3 41 4E A7 33 38 3Ø ØØ Ø1
1F81:9C 1F FA ØØ 8B 48 54 B3 ØF
1F89:31 38 3Ø A7 8B 48 54 AA BF
1F91:31 38 3Ø B1 41 4E A7 33 5D
1F99:39 3Ø ØØ A8 1F Ø4 Ø1 54 6A
1FA1:4D B2 54 4D AB 31 ØØ B7 6C
1FA9:1F Ø5 Ø1 8B 54 4D B3 3Ø Ø1
1FB1:A7 54 4D B2 3Ø ØØ CA 1F E3
1FB9:Ø6 Ø1 99 22 13 12 22 3B FØ
1FC1:54 4D 3B 22 9D 2Ø 92 22 BB
1FC9:ØØ DB 1F ØE Ø1 8B CF 28 C1
```

```
1FD1:32 29 B3 B1 33 A7 33 3Ø D3
1FD9:3Ø ØØ F6 1F 18 Ø1 41 4E 96
1FE1:B2 41 4E AA 34 35 3A 8B B4
1FE9:41 4E B1 33 36 3Ø A7 41 C8
1FF1:4E B2 34 35 ØØ ØE 2Ø 22 78
1FF9:Ø1 49 B2 49 AB 31 3A 8B 18
2ØØ1:49 B2 3Ø A7 49 B2 38 3A D2
2ØØ9:89 31 39 3Ø ØØ 1F 2Ø 2C 6D
2Ø11:Ø1 8B CF 28 32 29 B3 B1 8Ø
2Ø19:37 A7 33 34 3Ø ØØ 39 2Ø 9C
2Ø21:36 Ø1 41 4E B2 41 4E AB AC
2Ø29:34 35 3A 8B 41 4E B3 3Ø AB
2Ø31:A7 41 4E B2 33 31 35 ØØ 53
2Ø39:4A 2Ø 4Ø Ø1 8B 41 4E B2 6F
2Ø41:33 36 3Ø A7 41 4E B2 3Ø Ø2
2Ø49:ØØ 62 2Ø 4A Ø1 49 B2 49 A6
2Ø51:AA 31 3A 8B 49 B2 39 A7 62
2Ø59:49 B2 31 3A 89 31 39 3Ø 68
2Ø61:ØØ A4 2Ø 54 Ø1 8B CF 28 12
2Ø69:32 29 B2 31 32 38 A7 DA 13
2Ø71:31 2C 35 3Ø 3Ø 3Ø 2C 32 CB
2Ø79:34 2C 32 2C 31 3Ø 3Ø 3Ø C2
2Ø81:2C 33 2C 33 3A 54 48 B2 C3
2Ø89:54 48 AA 31 3A 54 B2 3Ø 27
2Ø91:3A 8B 54 48 B1 31 35 A7 45
2Ø99:54 48 B2 31 35 3A 89 31 56
2ØA1:39 3Ø ØØ AD 2Ø 5E Ø1 89 6B
2ØA9:31 39 3Ø ØØ EC 2Ø 68 Ø1 9Ø
2ØB1:48 54 B2 48 54 AA 28 28 CB
2ØB9:41 4E AA 28 33 36 3Ø AB 84
2ØC1:48 54 29 29 AD 31 3Ø 29 AE
2ØC9:3A FE Ø6 31 2C 48 54 23 Ø9
2ØD1:54 48 3A 8B 48 54 B3 33 7C
2ØD9:36 3Ø A7 32 36 3Ø 3A D5 16
2ØE1:3A 48 54 B2 3Ø 3A 89 33 B7
2ØE9:39 3Ø ØØ 12 21 72 Ø1 48 11
2ØF1:54 B2 48 54 AB 28 28 48 ED
2ØF9:54 AB 41 4E 29 AD 31 3Ø EE
21Ø1:29 3A FE Ø6 31 2C 48 54 C5
21Ø9:23 54 48 3A 89 32 36 3Ø 5Ø
2111:ØØ 51 21 7C Ø1 48 54 B2 18
2119:48 54 AB 28 28 48 54 AA 42
2121:28 33 36 3Ø AB 41 4E 29 36
2129:29 AD 31 3Ø 29 3A FE Ø6 CA
2131:31 2C 48 54 23 54 48 3A 9A
2139:8B 48 54 B1 3Ø A7 32 36 B3
2141:3Ø 3A D5 3A 48 54 B2 33 B4
2149:36 3Ø 3A 89 33 37 3Ø ØØ 69
2151:77 21 86 Ø1 48 54 B2 48 B9
```

```
2159:54 AA 28 28 41 4E AB 48 DA
2161:54 29 AD 31 3Ø 29 3A FE 7A
2169:Ø6 31 2C 48 54 23 54 48 25
2171:3A 89 32 36 3Ø ØØ 8Ø 21 8Ø
2179:9Ø Ø1 89 32 36 3Ø ØØ A5 BØ
2181:21 9A Ø1 54 48 B2 54 48 5E
2189:AB 31 3A 54 B2 3Ø 3A 8B DØ
2191:54 48 B3 31 A7 54 48 B2 6B
2199:31 3A 89 32 32 3Ø 3A D5 F3
21A1:32 32 3Ø ØØ C8 21 A4 Ø1 A4
21A9:97 32 3Ø 34 3Ø 2C 36 33 5F
21B1:3A 81 44 45 4C 41 59 B2 1B
21B9:31 A4 31 35 3Ø 3A 82 3A EØ
21C1:FE Ø7 2Ø 31 2C 3Ø ØØ E7 66
21C9:21 AE Ø1 DA 31 2C 32 3Ø E4
21D1:3Ø 3Ø 2C 31 3Ø 3Ø 2C 3Ø 9B
21D9:2C 31 3Ø 3Ø 3Ø 2C 31 2C 48
21E1:33 2C 31 3Ø 3Ø ØØ 14 22 BD
21E9:B8 Ø1 54 B2 3Ø 3A 48 59 D2
21F1:B2 48 59 AB 31 3A 99 22 4D
21F9:13 22 A3 33 32 29 22 4C BC
22Ø1:49 56 45 53 22 3B 48 59 45
22Ø9:3A 8B 48 59 B2 3Ø A7 34 C5
2211:36 3Ø ØØ 46 22 C2 Ø1 58 57
2219:45 B2 CE Ø3 28 32 29 3A 4D
2221:FE ØB 32 3A 97 32 3Ø 34 AB
2229:3Ø 2C 35 37 3A FE Ø6 31 B5
2231:2C 33 33 2C 35 35 3A 58 CC
2239:45 B2 CE Ø3 28 32 29 3A 6D
2241:89 31 33 3Ø ØØ 76 22 CC EA
2249:Ø1 FE ØB 32 3A 99 22 11 DF
2251:11 11 11 11 11 11 11 11 95
2259:1D 1D 1D 1D 1D 1D 1D 1D 9D
2261:1D 1D 1D 1D 1D 1D 1D 12 9A
2269:9Ø 47 41 4D 45 2Ø 4F 56 64
2271:45 52 92 22 ØØ BA 22 D6 67
2279:Ø1 9F 32 2C 38 2C 32 2C 32
2281:22 48 49 2D 53 43 4F 52 7D
2289:45 2C 53 2C 52 22 3A 84 BC
2291:32 2C 41 24 2C 42 24 3A 51
2299:AØ 32 3A 8B 53 43 B1 C5 8B
22A1:28 41 24 29 A7 81 49 B2 E9
22A9:31 A4 38 3A FE Ø7 49 2C 2D
22B1:3Ø 3A 82 3A 89 35 36 3Ø 4E
22B9:ØØ DC 22 EØ Ø1 8B CF 28 85
22C1:32 29 B3 B1 31 32 38 A7 65
22C9:34 38 3Ø 3A D5 3A 58 45 6D
22D1:B2 CE Ø3 28 32 29 3A 89 3A
22D9:39 3Ø ØØ Ø7 23 EA Ø1 97 95
```

```
22E1:35  33  32  38  30  2C  36  3A  30
22E9:FE  06  31  2C  34  30  23  30  F0
22F1:3A  FE  04  22  51  47  52  47  49
22F9:52  47  22  3A  8B  54  4D  B2  1C
2301:30  A7  35  32  30  00  19  23  E9
2309:EC  01  53  44  B2  B5  28  32  A3
2311:30  30  30  AD  54  4D  29  00  86
2319:64  23  F4  01  81  54  59  B2  CB
2321:31  A4  54  4D  A9  35  3A  99  B8
2329:22  13  12  22  3B  54  4D  AB  1B
2331:54  59  3A  99  22  13  12  22  7C
2339:3B  A3  31  33  29  3B  53  43  7F
2341:AA  54  59  3A  8B  54  4D  AB  B4
2349:54  59  B3  39  39  A7  8B  54  ED
2351:4D  AB  54  59  B1  39  30  A7  C3
2359:99  22  13  12  20  20  20  20  5A
2361:92  22  00  9C  23  FE  01  DA  35
2369:31  2C  33  30  30  30  AA  28  7C
2371:53  44  AC  54  59  29  2C  31  46
2379:3A  82  3A  53  43  B2  53  43  C8
2381:AA  54  4D  3A  99  22  13  12  0D
2389:20  20  30  20  92  22  3B  A3  27
2391:31  33  29  3B  22  12  22  3B  EE
2399:53  43  00  B7  23  08  02  FE  12
23A1:0B  31  3A  53  4E  B2  53  4E  68
23A9:AA  31  3A  8B  53  4E  B2  35  FF
23B1:A7  53  4E  B2  31  00  C0  23  C3
23B9:12  02  89  31  30  30  00  DB  EB
23C1:23  1C  02  99  22  13  12  22  1E
23C9:3B  54  4D  3B  22  13  12  22  C3
23D1:A3  31  33  29  3B  53  43  3A  17
23D9:8E  00  00  24  26  02  8F  20  22
23E1:2A  2A  2A  2A  2A  2A  2A  2A  28
23E9:2A  2A  20  48  49  20  53  43  0D
23F1:4F  52  45  20  2A  2A  2A  2A  97
23F9:2A  2A  2A  2A  2A  2A  00  43  05
2401:24  30  02  FE  04  22  4F  34  13
2409:53  43  43  46  47  42  42  41  A1
2411:52  20  41  42  22  3A  99  22  26
2419:93  11  11  20  20  20  20  20  75
2421:20  20  20  20  20  20  59  4F  0B
2429:55  52  20  53  43  4F  52  45  2B
2431:3A  20  22  3B  53  43  3A  41  F3
2439:42  B2  36  35  3A  4F  50  B2  CB
2441:30  00  6A  24  3A  02  99  22  60
2449:20  20  20  20  20  20  20  20  91
2451:20  20  C3  C3  C3  C3  C3  C3  DE
2459:C3  C3  C3  C3  C3  C3  C3  C3  A1
2461:C3  C3  C3  C3  C3  C3  91  22  A3
```

```
2469:00 AF 24 44 02 9F 32 2C 85
2471:38 2C 32 2C 22 48 49 2D DB
2479:53 43 4F 52 45 2C 53 2C F8
2481:52 22 3A 81 49 B2 31 A4 F6
2489:31 30 3A 84 32 2C 42 24 F0
2491:28 49 29 3A 84 32 2C 41 8F
2499:24 28 49 29 3A 82 3A A0 AA
24A1:32 3A F2 22 48 49 2D 53 27
24A9:43 4F 52 45 22 00 CA 24 D0
24B1:4E 02 81 55 B2 31 A4 31 FB
24B9:30 3A 8B 53 43 B1 C5 28 E3
24C1:42 24 28 55 29 29 A7 82 4E
24C9:00 21 25 58 02 55 B2 55 A4
24D1:AB 31 3A 81 45 B2 31 A4 97
24D9:55 AB 31 3A 41 24 28 45 B1
24E1:29 B2 41 24 28 45 AA 31 B2
24E9:29 3A 42 24 28 45 29 B2 3B
24F1:42 24 28 45 AA 31 29 3A 64
24F9:82 3A 42 24 28 55 29 B2 38
2501:C9 28 C4 28 53 43 29 2C 7B
2509:C3 28 C4 28 53 43 29 29 7D
2511:AB 31 29 3A 41 24 28 55 86
2519:29 B2 22 2D 2D 2D 22 00 1E
2521:4D 25 62 02 54 45 B2 C3 A8
2529:28 42 24 28 55 29 29 3A FA
2531:81 50 B2 31 A4 36 AB 54 63
2539:45 3A 42 24 28 55 29 B2 DA
2541:22 30 22 AA 42 24 28 55 DF
2549:29 3A 82 00 86 25 6C 02 AA
2551:99 22 11 11 22 3A 81 45 66
2559:B2 31 30 A4 32 A9 AB 31 5A
2561:3A 99 A3 31 31 29 3B 31 8C
2569:31 AB 45 3B 22 20 20 22 87
2571:3B 41 24 28 45 29 3B 22 18
2579:20 20 20 22 3B 42 24 28 55
2581:45 29 3A 82 00 AA 25 76 93
2589:02 99 A3 31 30 29 3B 31 90
2591:30 3B 22 20 20 22 3B 41 4A
2599:24 28 31 29 3B 22 20 20 7B
25A1:20 22 3B 42 24 28 31 29 5D
25A9:00 CD 25 80 02 99 22 13 E1
25B1:11 11 11 11 11 22 3A 81 03
25B9:49 B2 31 A4 31 31 AB 55 C0
25C1:3A 99 3A 82 3A 4E 4D 24 C8
25C9:B2 22 22 00 E3 25 8A 02 05
25D1:99 22 91 22 A3 31 36 AA BE
25D9:4F 50 29 3B C7 28 41 42 5C
25E1:29 00 09 26 94 02 8B CF D7
25E9:28 32 29 B2 37 A7 41 42 42
```

```
25F1:B2 41 42 AB 31 3A 8B 41 B3
25F9:42 B3 36 35 A7 41 42 B2 E5
2601:36 35 3A 89 36 35 30 00 7C
2609:2F 26 9E 02 8B CF 28 32 88
2611:29 B2 33 A7 41 42 B2 41 39
2619:42 AA 31 3A 8B 41 42 B1 92
2621:39 30 A7 41 42 B2 39 30 9E
2629:3A 89 36 35 30 00 67 26 85
2631:A8 02 8B CF 28 32 29 B2 CF
2639:31 32 38 A7 4E 4D 24 B2 CE
2641:4E 4D 24 AA C7 28 41 42 DA
2649:29 3A 41 42 B2 36 35 3A 18
2651:4F 50 B2 4F 50 AA 31 3A 6E
2659:FE 0B 31 3A 8B 4F 50 B2 9E
2661:33 A7 37 30 30 00 70 26 A3
2669:B2 02 89 36 35 30 00 B3 42
2671:26 BC 02 41 24 28 55 29 E9
2679:B2 4E 4D 24 3A 9F 32 2C 7F
2681:38 2C 32 2C 22 48 49 2D EF
2689:53 43 4F 52 45 2C 53 2C 0D
2691:57 22 3A 81 49 B2 31 A4 8D
2699:31 30 3A 98 32 2C 42 24 46
26A1:28 49 29 3A 98 32 2C 41 44
26A9:24 28 49 29 3A 82 3A A0 BE
26B1:32 00 C8 26 C6 02 99 22 26
26B9:13 22 3A 81 49 B2 31 A4 93
26C1:31 38 3A 99 3A 82 00 F6 68
26C9:26 D0 02 99 22 20 20 20 29
26D1:20 20 20 20 50 52 45 53 E5
26D9:53 20 42 55 54 54 4F 4E 56
26E1:20 54 4F 20 50 4C 41 59 CE
26E9:20 41 47 41 49 4E 22 3A 95
26F1:89 34 38 30 00 1C 27 E4 BD
26F9:02 8F 20 2A 2A 2A 2A 2A 4A
2701:2A 2A 20 43 4C 45 41 52 73
2709:20 48 49 2D 53 43 4F 52 0E
2711:45 53 20 2A 2A 2A 2A 2A F5
2719:2A 2A 00 7A 27 EE 02 F2 9A
2721:22 48 49 2D 53 43 4F 52 27
2729:45 22 3A 99 22 93 4D 41 BE
2731:4B 49 4E 47 20 48 49 2D 97
2739:53 43 4F 52 45 22 3A 9F D7
2741:32 2C 38 2C 32 2C 22 48 4C
2749:49 2D 53 43 4F 52 45 2C A0
2751:53 2C 57 22 3A 81 49 B2 7E
2759:31 A4 31 30 3A 98 32 2C 57
2761:22 30 30 30 30 30 30 22 9A
2769:3A 98 32 2C 22 2D 2D 2D 51
2771:22 3A 82 3A A0 32 3A 8E 24
```

```
2779:00 8F 27 F8 02 58 B2 36 2D
2781:32 3A 59 B2 31 33 35 3A C8
2789:E7 34 2C 31 36 00 B7 27 B9
2791:02 03 99 22 93 12 1C 20 34
2799:20 20 20 20 20 20 20 20 E7
27A1:20 20 20 20 20 20 20 20 EF
27A9:20 20 20 20 20 20 20 20 F7
27B1:20 20 20 BC 22 00 EB 27 F7
27B9:0C 03 99 22 12 20 92 20 7A
27C1:20 20 20 20 20 20 20 20 10
27C9:20 20 20 20 20 20 20 20 18
27D1:20 20 20 20 20 20 20 05 05
27D9:DB DB 1C BC 12 20 20 20 CD
27E1:20 20 20 20 20 20 BC 92 DB
27E9:22 00 21 28 16 03 99 22 02
27F1:12 20 92 20 20 20 20 20 87
27F9:20 20 20 20 20 20 20 20 48
2801:20 20 20 20 20 20 20 20 51
2809:20 20 20 20 05 AB 20 20 AE
2811:20 20 20 AB DB B3 1C BC DA
2819:12 20 20 20 BC 92 22 00 F4
2821:59 28 20 03 99 22 12 20 F5
2829:92 20 20 20 20 20 20 20 B2
2831:20 20 20 20 20 20 20 20 81
2839:20 20 20 20 20 20 20 20 89
2841:20 20 20 20 20 20 20 20 91
2849:98 12 20 20 20 92 20 20 1C
2851:1C BC 12 20 20 92 22 00 B2
2859:8D 28 2A 03 99 22 12 20 89
2861:92 20 20 20 20 20 20 20 EA
2869:20 20 20 20 20 20 20 20 B9
2871:20 20 20 20 20 20 20 20 C1
2879:20 20 20 20 20 20 20 20 C9
2881:20 20 20 20 20 20 12 20 B5
2889:20 92 22 00 C1 28 34 03 EC
2891:99 22 12 20 BC 92 20 20 0C
2899:20 20 20 20 20 20 20 20 E9
28A1:20 20 20 20 20 20 20 20 F1
28A9:20 20 20 20 20 20 20 20 F9
28B1:20 20 20 20 20 20 20 20 02
28B9:20 20 BC 12 20 92 22 00 6A
28C1:25 29 3E 03 99 22 12 20 80
28C9:20 20 20 20 20 20 20 20 1A
28D1:BC 92 20 20 20 20 20 20 0D
28D9:20 20 20 20 20 20 20 20 2A
28E1:20 20 20 20 20 20 20 20 32
28E9:20 20 20 20 20 20 20 12 2C
28F1:20 20 92 05 DB DB DB DB DE
28F9:1C 20 20 20 BC 12 20 20 F4
```

```
2901:20 20 BC 92 20 20 20 20 0E
2909:20 20 20 20 20 20 20 20 5B
2911:20 20 20 20 20 20 20 20 63
2919:20 20 20 20 90 AB 1C 12 07
2921:20 92 22 00 61 29 48 03 AF
2929:99 22 12 20 92 05 DB DB 51
2931:DB 1C 20 20 20 20 20 05 45
2939:DB 1C BC 12 20 20 20 20 1B
2941:20 20 20 20 20 20 20 20 93
2949:20 20 20 BC 92 20 20 20 F8
2951:20 20 20 20 20 20 20 20 A3
2959:90 AB 1C 12 20 92 22 00 13
2961:9F 29 52 03 99 22 12 20 E1
2969:92 20 20 05 DB 20 20 20 21
2971:20 20 AD DB DB 1C 12 BB 7E
2979:20 20 20 92 05 B3 20 20 68
2981:20 20 DB DB DB 1C BC 12 FF
2989:20 20 92 20 20 20 20 20 2A
2991:20 20 20 20 20 20 90 AB 50
2999:1C 12 20 92 22 00 DB 29 9D
29A1:5C 03 99 22 12 20 92 20 8E
29A9:20 20 20 20 20 20 20 20 FB
29B1:05 AB DB DB 1C 12 20 20 34
29B9:92 05 DB B3 20 20 20 20 2F
29C1:20 AB DB DB 1C BC 12 20 60
29C9:BC 92 20 20 20 20 20 20 07
29D1:20 20 20 20 AC 12 20 92 C2
29D9:22 00 19 2A 66 03 99 22 97
29E1:12 20 92 20 20 20 20 20 7B
29E9:20 20 20 20 20 05 AD BD 88
29F1:1C 12 20 92 05 DB B3 20 23
29F9:20 20 20 20 20 20 20 AB D7
2A01:DB 1C 12 20 20 92 20 20 3A
2A09:20 20 20 20 20 20 20 90 CD
2A11:AB 1C 12 20 20 92 22 00 16
2A19:55 2A 70 03 99 22 12 20 7A
2A21:92 20 20 20 20 20 20 20 AE
2A29:20 20 20 20 20 12 20 92 B7
2A31:05 B3 20 20 20 20 20 20 DC
2A39:20 20 20 20 AB 1C BC 12 05
2A41:20 92 20 20 20 20 20 20 32
2A49:20 20 20 90 AB 1C 12 20 D4
2A51:20 92 22 00 8D 2A 7A 03 AB
2A59:99 22 12 20 92 20 20 20 BC
2A61:20 20 20 20 20 20 20 20 B5
2A69:20 A3 20 20 20 20 20 20 9E
2A71:20 20 20 20 20 20 20 12 B7
2A79:20 92 20 20 20 20 20 20 6A
2A81:20 20 20 90 AB 1C 12 20 0D
```

```
2A89:20 92 22 00 C5 2A 84 03 B9
2A91:99 22 12 20 92 20 20 20 F4
2A99:20 20 20 20 20 20 20 20 ED
2AA1:20 20 20 20 20 20 20 20 F5
2AA9:20 20 20 20 20 20 20 12 EF
2AB1:20 92 20 20 20 20 20 20 A2
2AB9:20 20 20 90 AB 1C 12 20 45
2AC1:20 92 22 00 F9 2A 8E 03 A7
2AC9:99 22 12 20 92 20 20 20 2D
2AD1:20 20 20 20 20 20 20 20 26
2AD9:20 20 20 20 20 20 20 20 2E
2AE1:20 20 20 20 20 20 20 20 36
2AE9:20 20 20 20 20 20 20 20 3E
2AF1:20 20 BC 12 20 92 22 00 A6
2AF9:2F 2B 98 03 99 22 12 20 8D
2B01:92 20 20 20 20 20 20 20 90
2B09:20 20 20 20 20 20 20 20 5F
2B11:20 20 20 20 20 20 20 20 67
2B19:20 20 20 20 20 20 20 20 6F
2B21:20 20 20 20 20 20 90 AB E3
2B29:1C 12 20 92 22 00 65 2B 46
2B31:A2 03 99 22 12 20 92 20 45
2B39:20 20 20 20 20 20 20 20 8F
2B41:20 20 20 20 20 20 20 20 97
2B49:20 20 20 20 20 20 20 20 9F
2B51:20 20 20 20 20 20 20 20 A7
2B59:20 20 20 20 90 AB 1C 12 4B
2B61:20 92 22 00 9B 2B AC 03 96
2B69:99 22 12 20 92 20 20 20 CE
2B71:20 20 20 20 20 20 20 20 C7
2B79:20 20 20 20 20 20 20 20 CF
2B81:20 20 20 20 20 20 20 20 D7
2B89:20 20 20 20 20 20 20 20 DF
2B91:20 20 90 AB 1C 12 20 92 C8
2B99:22 00 D1 2B B6 03 99 22 05
2BA1:12 20 92 BB 20 20 20 20 F8
2BA9:20 20 20 20 20 20 20 A4 84
2BB1:20 20 20 20 20 20 20 20 08
2BB9:20 20 20 20 20 20 20 20 10
2BC1:20 20 20 20 20 20 20 20 18
2BC9:90 AB 1C 12 20 92 22 00 87
2BD1:0A 2C C0 03 99 22 12 20 1A
2BD9:20 92 BB 20 20 20 20 20 40
2BE1:20 20 20 20 20 12 20 92 72
2BE9:20 05 AB 20 20 20 20 20 EA
2BF1:20 20 20 20 20 20 20 20 48
2BF9:20 20 20 20 20 20 20 20 50
2C01:20 90 AB 1C 12 20 92 22 1D
2C09:00 42 2C CA 03 99 22 12 F8
```

```
2C11:20 20 20 92 20 20 20 20 90
2C19:20 20 20 20 20 20 12 20 55
2C21:92 05 DB B3 20 20 20 20 9C
2C29:20 20 20 20 20 20 20 20 81
2C31:20 20 20 B0 20 20 20 20 92
2C39:20 20 1C AC 12 20 92 22 50
2C41:00 7C 2C D4 03 99 22 12 60
2C49:20 20 20 92 BB 05 B2 B2 F0
2C51:DB AE 20 20 20 20 1C AC AF
2C59:12 20 92 05 DB DB DB 20 8B
2C61:20 20 20 20 20 20 20 20 B9
2C69:20 20 20 B0 DB B3 20 20 F6
2C71:20 20 20 1C 12 BE 20 20 93
2C79:92 22 00 B6 2C DE 03 99 8B
2C81:22 12 20 20 20 20 92 BB D7
2C89:05 DB DB DB AE 1C 20 AC 67
2C91:12 20 20 20 BC 92 05 AB E6
2C99:DB DB B2 B2 DB 20 20 20 18
2CA1:20 20 20 AB DB DB DB DB B2
2CA9:AE 20 20 1C 12 BE 20 20 13
2CB1:20 20 92 22 00 FB 2C E8 C7
2CB9:03 99 22 12 20 20 20 20 41
2CC1:20 20 20 20 20 20 20 20 1A
2CC9:20 20 20 20 20 20 20 20 22
2CD1:20 20 20 20 20 20 20 20 2A
2CD9:20 20 20 20 20 20 20 20 32
2CE1:20 20 20 92 22 3B 3A 97 89
2CE9:32 30 32 33 2C 32 32 34 A3
2CF1:3A 97 35 36 32 39 35 2C 64
2CF9:32 00 01 2D F2 03 8E 00 1F
2D01:16 2D FC 03 58 B2 32 36 A9
2D09:32 3A 59 B2 31 34 32 3A 5A
2D11:E7 34 2C 33 00 4A 2D 06 AE
2D19:04 99 22 93 81 12 20 20 0E
2D21:20 20 20 20 20 20 20 20 7B
2D29:20 20 20 20 20 20 20 20 83
2D31:20 20 20 20 20 20 20 20 8B
2D39:20 20 20 20 20 20 20 20 93
2D41:20 20 20 20 20 20 92 22 82
2D49:00 87 2D 10 04 99 22 81 78
2D51:12 20 92 20 20 20 20 20 F2
2D59:20 20 20 12 20 92 05 AB F1
2D61:81 12 20 92 05 C3 B1 BD 86
2D69:20 20 20 20 20 20 20 20 C3
2D71:20 20 20 20 20 20 20 20 CB
2D79:20 81 12 BB 20 20 20 20 24
2D81:20 20 20 92 22 00 EF 2D 3F
2D89:1A 04 99 22 12 20 92 20 9D
2D91:20 20 20 20 20 20 20 12 DD
```

```
2D99:20 92 05 AB 81 12 20 92 2B
2DA1:20 20 20 20 20 20 20 20 FB
2DA9:20 20 20 20 20 20 20 20 04
2DB1:20 20 20 20 20 20 20 20 0C
2DB9:20 20 20 12 B5 20 92 20 C4
2DC1:20 20 20 20 20 20 1C AC A0
2DC9:81 12 20 20 20 92 20 20 1B
2DD1:20 20 20 20 20 20 20 20 2C
2DD9:20 20 20 20 20 20 20 20 34
2DE1:20 20 20 20 20 20 20 20 3C
2DE9:20 12 B5 92 22 00 29 2E 4A
2DF1:24 04 99 22 12 20 92 20 0B
2DF9:20 20 20 20 20 20 20 05 39
2E01:A7 20 81 12 20 92 90 AE A5
2E09:20 20 20 20 20 20 20 20 65
2E11:20 20 20 20 20 20 20 20 6D
2E19:20 20 20 20 20 20 20 20 75
2E21:20 20 81 12 B5 92 22 00 23
2E29:66 2E 2E 04 99 22 12 20 E3
2E31:92 20 20 20 20 20 20 20 C6
2E39:20 05 A7 90 C3 DB BD 20 0E
2E41:20 20 20 20 05 AB 1C 12 DC
2E49:B8 92 05 AE 20 1C AF AF B2
2E51:AF AF AF AF AF AF AF AF AD
2E59:AF AF 20 20 20 20 20 81 C2
2E61:12 B5 92 22 00 A5 2E 38 D3
2E69:04 99 22 12 20 92 20 20 3F
2E71:20 20 20 20 20 20 05 A7 1F
2E79:20 1C 12 20 92 20 20 20 A6
2E81:20 20 20 05 AB 1C 12 20 5C
2E89:81 20 20 20 20 20 20 20 96
2E91:20 20 20 20 20 20 20 92 60
2E99:1C BB 20 20 20 20 81 12 8F
2EA1:B5 92 22 00 F3 2E 42 04 A2
2EA9:99 22 12 20 92 20 20 20 15
2EB1:20 20 20 20 20 05 A7 1C AC
2EB9:20 12 20 92 20 20 20 20 B9
2EC1:20 20 05 AB 1C 12 20 92 8D
2EC9:20 05 AB 81 12 20 92 05 40
2ED1:BD 20 AB 81 12 20 92 05 DD
2ED9:BD 20 20 AB 81 12 20 92 03
2EE1:05 BD 81 12 20 90 B6 92 C4
2EE9:20 20 20 20 81 12 B5 92 B6
2EF1:22 00 3D 2F 4C 04 99 22 C1
2EF9:12 20 92 20 20 20 20 20 9D
2F01:20 20 20 05 A7 1C 20 12 CB
2F09:20 92 20 20 20 20 20 20 04
2F11:05 AB 1C 12 20 92 20 05 12
2F19:AB 81 12 20 92 20 20 05 4C
```

```
2F21:AB 81 12 20 92 20 20 20 6F
2F29:05 AB 81 20 20 12 20 92 43
2F31:90 B3 20 20 20 20 81 12 61
2F39:B5 92 22 00 81 2F 56 04 D4
2F41:99 22 12 20 20 92 20 20 E4
2F49:20 20 20 20 20 20 05 B7 09
2F51:1C 12 20 92 20 20 20 20 51
2F59:20 20 05 AB 1C 12 20 92 27
2F61:20 20 1C BC 20 20 05 AB 5E
2F69:20 20 20 20 1C BC 20 20 1A
2F71:81 12 20 92 90 B3 20 20 F5
2F79:20 20 81 12 B5 92 22 00 7D
2F81:C1 2F 60 04 99 22 12 20 72
2F89:92 20 20 20 20 20 20 20 21
2F91:20 20 1C 12 20 20 92 20 73
2F99:20 20 20 20 20 05 AB 1C 9E
2FA1:12 20 92 20 20 20 20 20 47
2FA9:BC 20 20 20 20 20 20 20 56
2FB1:81 12 20 92 90 B3 20 20 36
2FB9:20 20 81 12 B5 92 22 00 BD
2FC1:00 30 6A 04 99 22 12 20 53
2FC9:92 20 20 20 20 20 20 20 61
2FD1:20 20 1C 12 20 92 20 20 98
2FD9:20 20 20 1C AC C3 12 20 CE
2FE1:AC 92 20 20 20 20 20 20 23
2FE9:20 20 20 20 20 20 20 81 A9
2FF1:12 20 92 90 B3 20 20 20 3B
2FF9:20 81 12 B5 92 22 00 40 C3
3001:30 74 04 99 22 12 20 92 DC
3009:20 20 20 20 20 20 20 20 69
3011:20 1C 12 20 92 20 20 20 42
3019:20 20 20 05 AB 1C 12 20 F7
3021:92 B4 20 20 20 20 20 20 DF
3029:20 20 20 20 20 20 20 81 EA
3031:12 20 92 90 B3 20 20 20 7C
3039:20 81 12 B5 92 22 00 80 45
3041:30 7E 04 99 22 12 20 92 9F
3049:20 20 20 20 20 20 20 20 A9
3051:20 1C 12 20 92 20 20 20 82
3059:20 20 20 05 AB 1C 12 20 38
3061:92 B4 20 20 20 20 20 20 20
3069:20 20 20 20 20 20 20 81 2B
3071:12 20 92 90 B3 20 20 20 BC
3079:20 81 12 B5 92 22 00 C0 C5
3081:30 88 04 99 22 12 20 92 62
3089:20 20 20 20 20 20 20 20 E9
3091:20 1C 12 20 92 20 20 20 C2
3099:20 20 20 05 AB 1C 12 20 78
30A1:92 B4 20 20 20 20 20 20 60
```

```
30A9:81 12 BB 20 20 20 20 20 AA
30B1:20 20 92 90 B3 20 20 20 04
30B9:20 81 12 B5 92 22 00 05 4A
30C1:31 92 04 99 22 12 20 90 A3
30C9:AC BB 92 20 20 20 20 20 A5
30D1:20 20 1C 12 20 92 20 20 9A
30D9:20 20 20 20 05 AB 1C 12 79
30E1:20 92 20 20 20 20 20 20 DE
30E9:20 20 20 81 12 20 1F AC 7A
30F1:BB AC 92 20 81 12 20 92 D6
30F9:90 B3 20 20 20 20 81 12 2C
3101:B5 92 22 00 4C 31 9C 04 8B
3109:99 22 12 20 9B 20 20 92 35
3111:90 A5 20 20 20 20 1C AC 91
3119:12 20 20 92 20 20 20 20 9B
3121:20 20 05 AB 1C 12 20 92 F2
3129:20 20 20 20 20 20 20 20 8B
3131:20 81 12 AC 1F 20 20 20 EA
3139:92 90 AE 81 12 20 92 90 BD
3141:B3 20 20 20 20 81 12 B5 6C
3149:92 22 00 8D 31 A6 04 99 1C
3151:22 12 20 9B AC BB 92 90 11
3159:A5 20 20 20 20 20 20 20 7E
3161:20 20 20 20 20 20 05 AB 19
3169:1C 12 20 92 20 20 20 20 6D
3171:20 20 20 20 20 20 05 A3 21
3179:A3 A3 90 AD 81 12 20 92 AA
3181:90 BD 20 20 20 20 81 12 38
3189:B5 92 22 00 C9 31 B0 04 28
3191:99 22 12 20 9B 20 20 92 BD
3199:90 A5 20 20 20 20 20 20 95
31A1:20 20 20 20 20 20 1C 12 ED
31A9:20 20 20 92 20 20 20 20 33
31B1:20 20 20 20 20 20 20 20 14
31B9:20 20 81 12 AC 92 20 20 95
31C1:20 20 20 12 B5 92 22 00 9D
31C9:03 32 BA 04 99 22 12 20 6B
31D1:9B AC BB 92 90 A5 20 20 49
31D9:20 20 20 20 20 20 20 20 3C
31E1:20 20 1C 12 20 20 20 92 55
31E9:20 20 20 20 20 20 20 20 4C
31F1:20 20 20 20 20 20 20 20 54
31F9:20 20 20 20 81 12 B5 92 CC
3201:22 00 3F 32 C4 04 99 22 0D
3209:12 20 9B AC BB 92 05 AE 9D
3211:20 20 20 20 20 20 20 20 75
3219:20 20 20 20 1C 12 20 96 9B
3221:D1 1C 20 92 20 20 20 20 84
3229:20 20 20 20 20 20 20 20 8D
```

```
3231:20 20 20 20 20 20 20 20 95
3239:81 12 B5 92 22 00 79 32 F8
3241:CE 04 99 22 12 20 9B AC 58
3249:BB 92 05 DB DB 20 20 20 4E
3251:20 20 20 20 20 20 20 20 B5
3259:1C 12 20 20 20 92 20 20 02
3261:20 20 20 20 20 20 20 20 C5
3269:20 20 20 20 20 20 20 20 CD
3271:20 20 81 12 B5 92 22 00 7B
3279:B6 32 D8 04 99 22 12 20 BA
3281:9B 20 20 92 05 DB DB AE E6
3289:20 20 20 20 20 20 20 20 ED
3291:20 B0 1C 12 20 96 D1 1C F1
3299:20 92 20 05 20 B4 20 20 3B
32A1:20 20 20 20 20 20 20 20 06
32A9:20 20 20 20 20 20 20 81 6F
32B1:12 20 92 22 00 F1 32 E2 AA
32B9:04 99 22 12 20 9B 20 20 BB
32C1:20 20 20 92 05 C3 C3 C3 ED
32C9:C3 C3 C3 81 12 20 20 20 03
32D1:20 20 20 20 1C B7 B7 BC 40
32D9:92 20 20 20 20 20 20 20 77
32E1:20 20 20 20 20 20 20 20 46
32E9:20 20 81 12 20 92 22 00 47
32F1:36 33 EC 04 99 22 12 20 B5
32F9:20 20 20 20 20 20 20 20 5E
3301:20 20 20 20 20 20 20 20 67
3309:20 20 20 20 20 20 20 20 6F
3311:20 20 20 20 20 20 20 20 77
3319:20 20 20 20 20 20 92 22 66
3321:3B 3A 97 32 30 32 33 2C A6
3329:32 32 34 3A 97 35 36 32 8F
3331:39 35 2C 38 00 3C 33 F6 D8
3339:04 8E 00 52 33 00 05 58 66
3341:B2 32 38 30 3A 59 B2 32 66
3349:30 30 3A E7 34 2C 31 36 84
3351:00 86 33 0A 05 99 22 93 C6
3359:1E 12 20 20 20 20 20 20 3B
3361:20 20 20 20 20 20 20 20 C7
3369:20 20 20 20 20 20 20 20 CF
3371:20 20 20 20 20 20 20 20 D7
3379:20 20 20 20 20 20 20 20 DF
3381:20 20 92 22 00 BF 33 14 ED
3389:05 99 22 12 20 92 20 20 E9
3391:20 20 20 20 20 20 20 20 F7
3399:20 20 20 20 20 20 20 20 FF
33A1:20 DF 12 20 20 20 20 92 A8
33A9:05 DD 90 B1 B1 B1 B1 B1 A0
33B1:B1 B1 B1 B1 B1 B1 B1 B1 18
```

```
33B9:1E 12 20 92 22 00 F7 33 15
33C1:1E 05 99 22 12 20 92 20 24
33C9:20 20 20 20 20 20 20 20 30
33D1:20 20 20 20 20 20 20 20 38
33D9:20 20 20 20 DF 12 20 20 06
33E1:92 05 DD 20 20 20 20 20 72
33E9:20 20 20 20 20 20 20 20 50
33F1:1E 12 20 92 22 00 2F 34 BC
33F9:28 05 99 22 12 20 92 20 61
3401:20 20 20 20 20 20 20 20 69
3409:20 20 20 20 20 20 20 20 71
3411:20 20 20 20 20 12 20 20 41
3419:92 05 DD 20 20 20 20 20 AB
3421:20 20 20 20 20 20 20 20 89
3429:1E 12 20 92 22 00 69 34 6A
3431:32 05 99 22 12 20 92 20 9F
3439:20 20 20 20 20 20 20 20 A1
3441:20 20 20 20 12 A9 20 92 D1
3449:20 20 20 20 20 20 20 DF 71
3451:12 20 92 05 DD 20 20 20 3D
3459:20 20 20 20 20 20 20 20 C1
3461:20 20 1E 12 20 92 22 00 56
3469:A3 34 3C 05 99 22 12 20 22
3471:20 20 20 20 20 20 20 20 D9
3479:20 20 20 20 20 20 92 A9 50
3481:20 20 20 20 20 20 05 D5 69
3489:C3 1E 12 20 92 05 CB 20 FF
3491:20 20 20 20 20 20 20 20 F9
3499:20 20 20 20 1E 12 20 92 2C
34A1:22 00 DD 34 46 05 99 22 B5
34A9:12 20 20 20 20 92 A9 20 E7
34B1:20 20 20 20 20 20 20 20 1A
34B9:20 20 20 20 20 20 20 20 22
34C1:05 DD 1E 12 A9 20 92 20 1C
34C9:20 20 20 20 20 20 12 20 16
34D1:92 20 20 20 20 20 20 12 65
34D9:20 92 22 00 15 35 50 05 5E
34E1:99 22 12 20 20 20 92 A9 34
34E9:20 20 20 20 20 20 20 20 52
34F1:20 20 20 20 20 20 20 20 5A
34F9:20 12 A9 20 20 20 92 20 F4
3501:20 20 20 20 20 20 12 20 4F
3509:92 20 20 20 20 20 20 12 9E
3511:20 92 22 00 4D 35 5A 05 6D
3519:99 22 12 20 20 92 A9 20 DB
3521:20 20 20 20 20 20 20 20 8B
3529:20 20 20 20 20 20 20 20 93
3531:12 A9 20 20 20 92 A9 20 D3
3539:20 20 20 20 20 20 12 20 87
```

```
3541:92 20 20 20 20 20 20 12 D6
3549:20 92 22 00 86 35 64 05 83
3551:99 22 12 20 20 92 20 20 01
3559:20 20 20 20 20 20 98 12 A6
3561:A9 1E 20 20 20 20 20 20 10
3569:20 20 20 20 20 92 A9 20 B0
3571:20 20 20 20 20 20 20 20 DB
3579:12 20 92 20 20 20 20 20 2B
3581:20 12 20 92 00 C2 35 6E 91
3589:05 99 22 12 20 92 05 DD 75
3591:1E 20 20 20 20 20 20 20 FA
3599:98 12 A9 20 20 20 20 1E EB
35A1:20 20 92 A9 20 20 20 20 F2
35A9:20 20 20 20 20 20 20 20 14
35B1:20 20 12 A9 20 92 20 20 BC
35B9:20 20 20 20 12 20 92 22 9A
35C1:00 FE 35 78 05 99 22 12 FE
35C9:20 92 05 DD 1E 20 20 20 39
35D1:20 20 20 20 12 20 98 20 BC
35D9:20 20 20 1E 20 92 A9 20 01
35E1:20 20 20 20 20 20 20 20 4C
35E9:20 20 20 20 20 12 A9 20 2F
35F1:20 92 20 20 20 20 20 20 F8
35F9:12 20 92 22 00 3A 36 82 C1
3601:05 99 22 12 20 92 05 DD EE
3609:1E 20 20 20 20 20 20 20 74
3611:12 20 98 20 20 1E 20 20 7D
3619:92 A9 20 20 20 20 20 20 21
3621:20 20 20 20 20 20 20 20 8D
3629:12 A9 20 20 20 92 20 20 BA
3631:20 20 20 20 12 20 92 22 14
3639:00 75 36 8C 05 99 22 12 77
3641:20 92 05 DD 1E 20 20 20 B2
3649:20 20 20 20 12 20 98 20 36
3651:20 1E 20 92 A9 20 20 20 B0
3659:20 20 20 20 20 20 20 20 C5
3661:20 20 20 12 A9 20 20 CD E6
3669:20 92 20 20 20 20 20 20 72
3671:12 20 92 00 AF 36 96 05 C9
3679:99 22 12 20 92 05 DD 1E 02
3681:20 20 20 20 20 20 20 12 DF
3689:20 20 20 92 A9 20 20 20 69
3691:20 20 20 20 20 20 20 20 FD
3699:20 20 20 12 A9 20 20 CD 1F
36A1:20 92 A9 20 20 20 20 20 DB
36A9:20 12 20 92 22 00 E9 36 F2
36B1:A0 05 99 22 12 20 92 05 40
36B9:DD 1E 20 20 20 20 20 20 84
36C1:20 12 20 20 92 A9 20 20 64
```

```
36C9:20 20 20 20 20 20 20 20 36
36D1:20 20 20 20 12 A9 20 20 F3
36D9:CD 20 92 A9 20 20 20 20 04
36E1:20 20 20 12 20 92 22 00 1B
36E9:23 37 AA 05 99 22 12 20 F4
36F1:92 05 DD 1E 20 20 20 20 68
36F9:20 20 20 12 20 92 A9 20 62
3701:20 20 20 20 20 20 20 20 6F
3709:20 20 20 20 20 12 A9 20 52
3711:20 CD 20 92 A9 20 20 20 5E
3719:20 20 20 20 20 12 20 92 C1
3721:22 00 5D 37 B4 05 99 22 CE
3729:12 20 92 05 DD 1E 20 20 13
3731:20 20 20 20 20 12 20 92 D9
3739:20 20 20 20 20 20 20 20 A7
3741:20 20 20 20 20 20 12 A9 1D
3749:20 20 20 20 92 A9 20 20 71
3751:20 20 20 20 20 20 20 12 B1
3759:20 92 22 00 97 37 BE 05 DC
3761:99 22 12 20 92 05 DD 1E EB
3769:20 20 20 20 20 20 20 20 D7
3771:20 20 20 20 20 20 20 20 DF
3779:20 20 20 20 98 12 A9 20 86
3781:1E 20 20 20 20 20 92 20 D3
3789:20 20 20 20 20 20 20 20 F7
3791:20 12 20 92 22 00 D5 37 B5
3799:C8 05 99 22 12 20 92 05 3E
37A1:CA C9 1E 20 20 20 20 20 8F
37A9:20 20 20 20 20 20 20 20 18
37B1:20 20 20 20 20 98 12 A9 6F
37B9:20 1E 20 20 20 92 05 DD F8
37C1:20 1E 12 20 92 20 20 20 81
37C9:20 20 20 20 20 20 20 12 2A
37D1:20 92 22 00 11 38 D2 05 4D
37D9:99 22 12 20 98 20 20 20 87
37E1:92 20 20 20 20 20 20 20 89
37E9:20 20 20 20 20 20 20 20 58
37F1:20 20 12 20 1E 20 20 20 8E
37F9:92 20 05 AB C3 1E 12 20 EF
3801:92 20 20 20 20 20 20 20 AA
3809:20 20 20 12 20 92 22 00 46
3811:4D 38 DC 05 99 22 12 20 BB
3819:98 20 20 20 92 20 20 20 59
3821:20 20 20 20 20 20 20 20 91
3829:20 20 20 20 20 1E 12 A9 FE
3831:20 20 20 20 92 05 C3 B3 A3
3839:20 1E 12 20 92 20 20 20 FA
3841:20 20 20 20 20 20 20 12 A3
3849:20 92 22 00 8A 38 E6 05 BA
```

```
3851:99 22 12 2Ø 98 2Ø 2Ø 2Ø Ø1
3859:2Ø 92 Ø5 C3 C3 C3 C9 2Ø 3C
3861:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø D1
3869:2Ø 1E 12 A9 2Ø 2Ø 92 Ø5 F9
3871:2Ø DD 2Ø D5 B3 2Ø 1E 12 37
3879:2Ø 92 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 86
3881:2Ø 2Ø 2Ø 2Ø 12 2Ø 92 22 68
3889:ØØ CB 38 FØ Ø5 99 22 12 E7
3891:2Ø 98 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
3899:2Ø 92 Ø5 DD 2Ø 2Ø 2Ø 2Ø 1F
38A1:2Ø 2Ø 2Ø 2Ø 1E AC 12 2Ø 18
38A9:2Ø 2Ø 2Ø 92 Ø5 C3 CB 2Ø 4E
38B1:12 2Ø 92 CA C3 1E 12 2Ø ØD
38B9:92 Ø5 C3 C3 C3 C3 C3 C3 E1
38C1:C3 C3 C3 C3 1E 12 2Ø 92 C5
38C9:22 ØØ 1Ø 39 FA Ø5 99 22 22
38D1:12 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 3B
38D9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 4A
38E1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 52
38E9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 5A
38F1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 62
38F9:92 22 3B 3A 97 32 3Ø 32 5F
39Ø1:33 2C 32 32 34 3A 97 35 7Ø
39Ø9:36 32 39 35 2C 35 ØØ 16 E9
3911:39 Ø4 Ø6 8E ØØ 2C 39 ØE FB
3919:Ø6 58 B2 31 31 3Ø 3A 59 26
3921:B2 31 36 35 3A E7 34 2C 59
3929:31 36 ØØ 62 39 18 Ø6 99 B7
3931:22 93 98 12 2Ø 2Ø 2Ø 2Ø AF
3939:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø AB
3941:2Ø 2Ø 2Ø 2Ø 2Ø C3 C3 C3 2D
3949:C3 C3 C3 C3 C3 2Ø 2Ø 2Ø 42
3951:92 B8 B8 B8 B8 B8 B8 B8 BØ
3959:B8 B8 B8 B8 12 BB 92 22 BF
3961:ØØ 94 39 22 Ø6 99 22 B5 D2
3969:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø B6 B4 9D
3971:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø E3
3979:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø DF AB
3981:12 2Ø 2Ø 2Ø 92 2Ø 2Ø 2Ø 8Ø
3989:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø FB
3991:B6 22 ØØ C7 39 2C Ø6 99 84
3999:22 B5 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 72
39A1:Ø5 BC BE 98 2Ø 2Ø 2Ø 2Ø Ø9
39A9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 1C
39B1:2Ø 2Ø 2Ø 2Ø DF 12 2Ø 2D F6
39B9:92 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 65
39C1:2Ø 2Ø 2Ø 2Ø B6 ØØ F9 39 35
39C9:36 Ø6 99 22 B5 2Ø 2Ø 2Ø BC
39D1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 44
```

```
39D9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 4C
39E1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø DF 12 C5
39E9:2D 92 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 7F
39F1:2Ø 2Ø 2Ø 2Ø 2Ø B6 22 ØØ A2
39F9:2B 3A 4Ø Ø6 99 22 B5 2Ø D9
3AØ1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 75
3AØ9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 7D
3A11:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 85
3A19:2Ø 12 2D 92 2Ø 2Ø 2Ø 2Ø D2
3A21:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø B6 2C
3A29:22 ØØ 5D 3A 4A Ø6 99 22 BD
3A31:12 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 9E
3A39:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø AD
3A41:2Ø 92 BB 2Ø 2Ø 2Ø 2Ø 2Ø C5
3A49:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø BD
3A51:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø C5
3A59:2Ø B6 22 ØØ 91 3A 54 Ø6 F3
3A61:99 22 12 2Ø 2Ø 2Ø 2Ø 2Ø 51
3A69:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø DD
3A71:2Ø 2Ø 2Ø 2Ø 92 BB 2Ø 2Ø E7
3A79:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ED
3A81:2Ø 2Ø 2Ø 2Ø 2Ø 12 2D 92 4A
3A89:2Ø 2Ø 2Ø 2Ø 2Ø B6 22 ØØ 3C
3A91:C7 3A 5E Ø6 99 22 9Ø B1 CC
3A99:B1 B1 B1 B1 B1 DB B1 B1 B6
3AA1:B1 B1 B1 B1 DB B1 B1 98 4E
3AA9:12 2Ø 2Ø 2Ø 92 BB 2Ø 2Ø 19
3AB1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 26
3AB9:2Ø 2Ø AC 12 2Ø 92 2Ø 2Ø A8
3AC1:2Ø 2Ø 2Ø B6 22 ØØ FF 3A Ø9
3AC9:68 Ø6 99 22 B5 2Ø 2Ø 2Ø D7
3AD1:2Ø 2Ø Ø5 12 A2 92 2Ø 2Ø DF
3AD9:2Ø 2Ø 2Ø 2Ø 12 A2 92 2Ø CC
3AE1:2Ø 98 12 2Ø 2Ø 2Ø 2Ø 2Ø B2
3AE9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 5E
3AF1:2Ø 2Ø 2Ø 2Ø AC 92 2Ø 2Ø 94
3AF9:2Ø 2Ø 2Ø B6 22 ØØ 35 3B AC
3BØ1:72 Ø6 99 22 B5 2Ø 2Ø 2Ø 16
3BØ9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 7F
3B11:2Ø 2Ø 2Ø 2Ø 12 2Ø 2Ø 2Ø 17
3B19:92 Ø5 DB DB DB B3 2Ø 2Ø 61
3B21:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 98 1Ø
3B29:12 BB BC 92 2Ø 2Ø 2Ø 2Ø 3A
3B31:2Ø B6 22 ØØ 6B 3B 7C Ø6 FØ
3B39:99 22 B5 2Ø 2Ø 2Ø 2Ø 2Ø 9F
3B41:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø B7
3B49:2Ø 2Ø Ø5 A3 A3 A3 2Ø B3 52
3B51:AD 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 8E
3B59:2Ø 2Ø 2Ø 98 12 BE 2Ø 92 D3
```

```
3B61:Ø5 D5 C9 2Ø 2Ø 2Ø 98 B6 74
3B69:22 ØØ Ø2 3C 86 Ø6 99 22 96
3B71:B5 2Ø 2Ø 2Ø 2Ø 1C A4 A4 3Ø
3B79:A4 A4 A4 A4 A4 A4 A4 98 E3
3B81:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø F7
3B89:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø FF
3B91:2Ø 2Ø 12 2Ø 92 1F AC 12 EØ
3B99:2Ø 92 Ø5 DD 2Ø 2Ø 2Ø 98 9D
3BA1:B6 B5 2Ø 2Ø 2Ø 2Ø 96 12 A7
3BA9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø
3BB1:2Ø 92 Ø5 B9 2Ø 2Ø 2Ø 2Ø FA
3BB9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 3Ø
3BC1:2Ø 2Ø 2Ø 2Ø 2Ø 98 12 2Ø FD
3BC9:2Ø 2Ø 92 Ø5 DD 98 2Ø 2Ø AC
3BD1:2Ø B6 B5 2Ø 2Ø 2Ø 2Ø 96 17
3BD9:12 2Ø BC BE BC BE BC BE FD
3BE1:BC BE 92 2Ø 2Ø 2Ø 2Ø 2Ø 9C
3BE9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 6Ø
3BF1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 68
3BF9:Ø5 DD 98 2Ø 2Ø 2Ø B6 22 9Ø
3CØ1:ØØ 3D 3C 9Ø Ø6 99 22 B5 E9
3CØ9:2Ø 2Ø 2Ø 2Ø 2Ø 12 2Ø 92 BB
3C11:9A DD 98 12 2Ø 92 2Ø 2Ø 2E
3C19:2Ø 2Ø 2Ø 96 12 2Ø C3 2Ø CF
3C21:C3 2Ø 92 Ø5 B8 2Ø 2Ø 2Ø CC
3C29:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø A1
3C31:2Ø 2Ø 2Ø 2Ø DD 2Ø 2Ø 2Ø 97
3C39:98 B6 22 ØØ 77 3C 9A Ø6 D7
3C41:99 22 B5 2Ø 2Ø 2Ø 2Ø 2Ø A9
3C49:12 2Ø 92 9A DD 98 12 2Ø 64
3C51:92 2Ø 2Ø 2Ø 2Ø 2Ø 12 2Ø E6
3C59:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø D1
3C61:2Ø 2Ø 2Ø 2Ø 2Ø DF 92 2Ø BD
3C69:2Ø 2Ø 2Ø 2Ø 2Ø Ø5 DD 2Ø FØ
3C71:2Ø 2Ø 98 B6 22 ØØ B1 3C 31
3C79:A4 Ø6 99 22 B5 2Ø 2Ø 2Ø A9
3C81:2Ø 2Ø 12 2Ø 92 9A DD 98 A9
3C89:12 2Ø 92 2Ø 2Ø 2Ø 2Ø 2Ø 49
3C91:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØA
3C99:2Ø 2Ø 2Ø 2Ø 2Ø DF 12 2Ø F4
3CA1:DF 92 2Ø 2Ø 2Ø 2Ø 2Ø Ø5 7B
3CA9:DD 2Ø 2Ø 2Ø 98 B6 22 ØØ Ø3
3CB1:1B 3D AE Ø6 99 22 B5 2Ø 1E
3CB9:2Ø 2Ø 2Ø 2Ø 12 2Ø 92 9A 21
3CC1:DD 98 12 2Ø 92 2Ø 2Ø 2Ø Ø9
3CC9:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 42
3CD1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 4A
3CD9:DF 12 2Ø DF 92 2Ø 2Ø 2Ø 3E
3CE1:2Ø Ø5 DD 2Ø 2Ø 2Ø 98 B6 D2
```

```
3CE9:B5  20  20  20  20  20  12  20  11
3CF1:9A  20  98  20  92  20  20  20  4A
3CF9:20  20  20  20  20  20  20  20  72
3D01:20  20  20  20  20  20  20  20  7B
3D09:20  DF  12  20  92  20  20  20  45
3D11:20  05  DD  20  20  20  98  B6  04
3D19:22  00  53  3D  B8  06  99  22  16
3D21:B5  20  20  20  20  20  12  20  4A
3D29:92  20  12  20  92  20  20  20  AE
3D31:20  20  20  20  20  20  20  20  AB
3D39:20  20  20  20  20  20  20  20  B3
3D41:20  20  12  20  92  20  20  20  8D
3D49:20  90  B8  20  20  20  98  B6  7A
3D51:22  00  87  3D  C2  06  99  22  25
3D59:B5  20  20  20  20  20  12  BB  1E
3D61:20  20  BC  92  20  20  20  20  96
3D69:20  20  20  20  20  20  20  20  E3
3D71:20  20  20  20  20  20  20  20  EB
3D79:12  20  92  20  20  20  20  20  3B
3D81:20  20  20  B6  22  00  BB  3D  49
3D89:CC  06  99  22  B5  20  20  20  CF
3D91:20  20  20  12  BB  20  20  AF  97
3D99:AF  AF  AF  AF  AF  AF  AF  AF  14
3DA1:AF  AF  20  92  20  20  20  20  EE
3DA9:20  20  20  20  12  20  92  20  98
3DB1:20  20  20  20  20  20  20  B6  C2
3DB9:22  00  ED  3D  D6  06  99  22  FA
3DC1:B5  20  20  20  20  20  20  20  07
3DC9:20  20  20  20  20  20  20  20  44
3DD1:20  20  20  20  20  20  20  20  4C
3DD9:20  20  20  20  20  20  12  20  38
3DE1:92  20  20  20  20  20  20  20  95
3DE9:20  B6  22  00  1F  3E  E0  06  20
3DF1:99  22  B5  20  20  20  20  20  5C
3DF9:20  20  20  20  20  20  20  20  74
3E01:20  20  20  20  20  20  20  20  7D
3E09:20  20  20  20  20  20  20  20  85
3E11:12  20  92  20  20  20  20  20  D4
3E19:20  20  20  B6  22  00  65  3E  37
3E21:EA  06  99  22  12  20  B8  B8  25
3E29:B8  B8  B8  B8  B8  B8  B8  B8  A5
3E31:B8  B8  B8  B8  B8  B8  B8  B8  AD
3E39:B8  B8  B8  B8  B8  B8  B8  B8  B5
3E41:B8  B8  B8  20  20  20  20  20  43
3E49:20  20  20  20  92  22  3B  3A  B1
3E51:97  32  30  32  33  2C  32  32  30
3E59:34  3A  97  35  36  32  39  35  E6
3E61:2C  31  32  00  6B  3E  F4  06  CA
3E69:8E  00  76  3E  08  07  49  B2  81
```

```
3E71:33 34 35 36 ØØ 91 3E 12 73
3E79:Ø7 87 2Ø 41 3A 8B 2Ø 41 F4
3E81:B2 32 35 36 2Ø A7 2Ø 8E 5C
3E89:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø ØØ E5
3E91:A7 3E 1C Ø7 97 2Ø 49 2C 61
3E99:41 3A 49 B2 49 AA 31 3A 2B
3EA1:89 31 38 31 3Ø ØØ CC 3E A2
3EA9:26 Ø7 83 3Ø 3Ø 3Ø 2C 3Ø 39
3EB1:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø A5
3EB9:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø AD
3EC1:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø B5
3EC9:3Ø 3Ø ØØ F1 3E 3Ø Ø7 83 CD
3ED1:3Ø 3Ø 3Ø 2C 3Ø 3Ø 34 2C 12
3ED9:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 12
3EE1:3Ø 3Ø 39 2C 3Ø 3Ø 3Ø 2C 3B
3EE9:3Ø 3Ø 3Ø 2C 3Ø 34 3Ø ØØ Ø6
3EF1:16 3F 3A Ø7 83 3Ø 3Ø 3Ø 6E
3EF9:2C 3Ø 3Ø 3Ø 2C 31 3Ø 36 5E
3FØ1:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 5D
3FØ9:2C 3Ø 31 38 2C 31 32 38 16
3F11:2C 3Ø 31 36 ØØ 3B 3F 44 EA
3F19:Ø7 83 3Ø 3Ø 3Ø 2C 31 36 CF
3F21:32 2C 3Ø 32 3Ø 2C 3Ø 3Ø AF
3F29:3Ø 2C 3Ø 34 33 2C 31 33 F3
3F31:33 2C 3Ø 3Ø 3Ø 2C 3Ø 31 21
3F39:31 ØØ 6Ø 3F 4E Ø7 83 32 18
3F41:32 35 2C 3Ø 3Ø 3Ø 2C 3Ø 79
3F49:31 3Ø 2C 32 33 32 2C 3Ø FF
3F51:3Ø 3Ø 2C 3Ø 31 38 2C 31 7Ø
3F59:36 38 2C 3Ø 3Ø 3Ø ØØ 85 51
3F61:3F 58 Ø7 83 3Ø 32 3Ø 2C 85
3F69:31 36 3Ø 2C 3Ø 3Ø 3Ø 2C A5
3F71:3Ø 3Ø 35 2C 31 32 38 2C 6C
3F79:3Ø 3Ø 3Ø 2C 3Ø 3Ø 31 2C B5
3F81:3Ø 3Ø 3Ø ØØ AA 3F 62 Ø7 48
3F89:83 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 91
3F91:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø ED
3F99:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø F5
3FA1:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø FD
3FA9:ØØ CF 3F 6C Ø7 83 3Ø 3Ø A1
3FB1:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 1F
3FB9:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 27
3FC1:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 2F
3FC9:3Ø 2C 3Ø 3Ø 3Ø ØØ F4 3F 1F
3FD1:76 Ø7 83 3Ø 3Ø 3Ø 2C 3Ø 8B
3FD9:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø CF
3FE1:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø D7
3FE9:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø DF
3FF1:3Ø 3Ø ØØ 19 4Ø 8Ø Ø7 83 BB
```

```
3FF9:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 34
4ØØ1:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3D
4ØØ9:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 45
4Ø11:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø ØØ 21
4Ø19:3E 4Ø 8A Ø7 83 3Ø 3Ø 3Ø F7
4Ø21:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 7F
4Ø29:2C 3Ø 3Ø 3Ø 2C 3Ø 38 34 9B
4Ø31:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 8F
4Ø39:2C 3Ø 38 34 ØØ 63 4Ø 94 C8
4Ø41:Ø7 83 3Ø 36 34 2C 3Ø 3Ø 72
4Ø49:3Ø 2C 3Ø 31 36 2C 3Ø 36 FE
4Ø51:34 2C 3Ø 31 3Ø 2C 31 36 DA
4Ø59:38 2C 31 37 3Ø 2C 31 37 66
4Ø61:31 ØØ 88 4Ø 9E Ø7 83 32 D9
4Ø69:33 32 2C 31 37 3Ø 2C 31 AC
4Ø71:37 31 2C 32 33 32 2C 3Ø 6D
4Ø79:36 34 2C 3Ø 31 3Ø 2C 31 7E
4Ø81:36 38 2C 3Ø 36 34 ØØ AD E3
4Ø89:4Ø A8 Ø7 83 3Ø 3Ø 3Ø 2C 3C
4Ø91:3Ø 31 36 2C 3Ø 3Ø 3Ø 2C CE
4Ø99:3Ø 3Ø 3Ø 2C 3Ø 38 34 2C FD
4ØA1:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C DD
4ØA9:3Ø 38 34 ØØ D2 4Ø B2 Ø7 DA
4ØB1:83 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø BB
4ØB9:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 18
4ØC1:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2Ø
4ØC9:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 28
4ØD1:ØØ F7 4Ø BC Ø7 83 3Ø 3Ø FA
4ØD9:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 49
4ØE1:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 51
4ØE9:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 59
4ØF1:3Ø 2C 3Ø 3Ø 3Ø ØØ 1C 41 99
4ØF9:C6 Ø7 83 3Ø 3Ø 3Ø 2C 3Ø DD
41Ø1:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø FA
41Ø9:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø Ø3
4111:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø ØB
4119:3Ø 3Ø ØØ 41 41 DØ Ø7 83 B2
4121:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 5F
4129:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 67
4131:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 6F
4139:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø ØØ 4B
4141:66 41 DA Ø7 83 3Ø 3Ø 3Ø 8Ø
4149:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø A9
4151:2C 3Ø 3Ø 31 2C 3Ø 3Ø 3Ø C1
4159:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 35 BE
4161:2C 31 32 38 ØØ 8B 41 E4 A5
4169:Ø7 83 3Ø 3Ø 3Ø 2C 3Ø 32 1E
4171:3Ø 2C 31 36 3Ø 2C 3Ø 3Ø 63
4179:3Ø 2C 3Ø 31 38 2C 32 33 42
```

```
4181:32 2C 30 30 30 2C 30 31 F4
4189:31 00 B0 41 EE 07 83 32 9B
4191:33 32 2C 30 30 30 2C 30 8D
4199:31 31 2C 31 36 31 2C 30 98
41A1:30 30 2C 30 34 32 2C 31 C4
41A9:33 33 2C 30 30 30 00 D5 33
41B1:41 F8 07 83 31 36 32 2C 1F
41B9:30 32 30 2C 30 31 38 2C 8C
41C1:31 32 38 2C 30 31 36 2C 12
41C9:31 30 36 2C 30 30 30 2C 49
41D1:30 30 30 00 FA 41 02 08 67
41D9:83 30 34 30 2C 30 30 30 66
41E1:2C 30 30 30 2C 30 30 39 4B
41E9:2C 30 30 30 2C 30 30 30 4A
41F1:2C 30 30 34 2C 30 30 30 92
41F9:00 1F 42 0C 08 83 30 30 2C
4201:30 2C 30 30 30 2C 30 30 74
4209:30 2C 30 30 30 2C 30 30 7C
4211:30 2C 30 30 30 2C 30 30 84
4219:30 2C 30 30 30 00 44 42 16
4221:16 08 83 30 30 30 2C 30 F0
4229:30 30 2C 30 30 30 2C 30 25
4231:30 30 2C 30 30 30 2C 30 2D
4239:30 30 2C 30 30 30 2C 30 35
4241:30 30 00 69 42 20 08 83 A6
4249:30 30 30 2C 30 30 30 2C 89
4251:30 30 30 2C 30 30 30 2C 91
4259:30 30 30 2C 30 30 30 2C 99
4261:30 30 30 2C 30 30 30 00 75
4269:8E 42 2A 08 83 30 30 30 F8
4271:2C 30 30 30 2C 30 30 34 D7
4279:2C 31 36 38 2C 30 36 34 6D
4281:2C 30 30 35 2C 31 36 39 4D
4289:2C 30 36 34 00 B3 42 34 C1
4291:08 83 30 30 35 2C 31 36 F6
4299:39 2C 30 36 34 2C 30 30 12
42A1:34 2C 31 38 34 2C 30 36 DD
42A9:34 2C 30 30 30 2C 31 38 29
42B1:34 00 D8 42 3E 08 83 30 D8
42B9:30 30 2C 30 30 30 2C 31 B6
42C1:38 34 2C 30 30 30 2C 30 C2
42C9:30 30 2C 31 36 38 2C 30 26
42D1:30 30 2C 30 30 30 00 FD 43
42D9:42 48 08 83 31 36 38 2C C9
42E1:30 30 30 2C 30 30 30 2C 22
42E9:30 33 32 2C 30 30 30 2C 2B
42F1:30 30 30 2C 30 33 32 2C 42
42F9:30 30 30 00 22 43 52 08 73
4301:83 30 30 30 2C 30 33 32 19
```

```
4309:2C 30 30 30 2C 30 30 30 6D
4311:2C 30 33 32 2C 30 30 30 F5
4319:2C 30 30 31 2C 30 33 33 96
4321:00 47 43 5C 08 83 30 30 86
4329:30 2C 30 30 31 2C 31 36 AE
4331:39 2C 30 30 30 2C 30 30 2B
4339:31 2C 30 33 33 2C 30 30 77
4341:30 2C 30 30 30 00 6C 43 91
4349:66 08 83 30 30 30 2C 30 43
4351:30 30 2C 30 30 30 2C 30 4F
4359:30 30 2C 30 30 30 2C 30 57
4361:30 30 2C 30 30 30 2C 30 5F
4369:30 30 00 91 43 70 08 83 9C
4371:30 30 30 2C 30 30 30 2C B3
4379:30 30 30 2C 30 30 30 2C BB
4381:30 30 30 2C 30 30 30 2C C3
4389:30 30 30 2C 30 30 30 00 9F
4391:B6 43 7A 08 83 30 30 30 81
4399:2C 30 30 30 2C 30 30 30 FD
43A1:2C 30 36 34 2C 30 30 30 07
43A9:2C 30 30 32 2C 30 38 30 3E
43B1:2C 30 30 30 00 DB 43 84 DD
43B9:08 83 30 31 30 2C 30 32 03
43C1:30 2C 30 30 30 2C 30 34 3B
43C9:33 2C 31 33 32 2C 30 30 21
43D1:30 2C 30 34 33 2C 32 32 A5
43D9:34 00 00 44 8E 08 83 30 8A
43E1:30 30 2C 30 37 34 2C 32 2A
43E9:32 34 2C 30 30 30 2C 30 E9
43F1:38 32 2C 31 36 38 2C 30 D4
43F9:30 30 2C 30 32 30 00 25 A4
4401:44 98 08 83 31 33 38 2C FD
4409:30 30 30 2C 30 30 34 2C 55
4411:30 30 32 2C 31 33 32 2C AD
4419:30 30 30 2C 30 30 30 2C 5D
4421:31 36 39 00 4A 44 A2 08 A7
4429:83 30 30 30 2C 30 30 30 3B
4431:2C 30 34 30 2C 30 30 30 18
4439:2C 30 30 30 2C 30 39 36 B7
4441:2C 30 30 30 2C 30 30 30 A7
4449:00 6F 44 AC 08 83 30 31 E0
4451:36 2C 30 30 30 2C 30 30 CB
4459:30 2C 30 30 30 2C 30 30 D0
4461:30 2C 30 30 30 2C 30 30 D8
4469:30 2C 30 30 30 00 94 44 0D
4471:B6 08 83 30 30 30 2C 30 95
4479:30 30 2C 30 30 30 2C 30 79
4481:30 30 2C 30 30 30 2C 30 81
4489:30 30 2C 30 30 30 2C 30 89
```

```
4491:30 30 00 B9 44 C0 08 83 92
4499:30 30 30 2C 30 30 30 2C DD
44A1:30 30 30 2C 30 30 30 2C E5
44A9:30 30 30 2C 30 30 30 2C ED
44B1:30 30 30 2C 30 30 30 00 C9
44B9:DE 44 CA 08 83 30 30 30 0A
44C1:2C 30 30 30 2C 30 32 31 2D
44C9:2C 30 30 30 2C 30 30 30 30
44D1:2C 30 32 31 2C 30 30 30 88
44D9:2C 30 30 30 00 03 45 D4 F8
44E1:08 83 30 30 34 2C 30 30 3B
44E9:30 2C 30 30 31 2C 30 34 6D
44F1:32 2C 31 36 30 2C 30 30 EA
44F9:31 2C 30 34 33 2C 32 33 51
4501:34 00 28 45 DE 08 83 31 4E
4509:37 30 2C 30 34 33 2C 32 BC
4511:33 34 2C 31 37 30 2C 30 DD
4519:34 32 2C 31 36 30 2C 30 DD
4521:30 31 2C 30 30 34 00 4D 38
4529:45 E8 08 83 30 30 30 2C 98
4531:30 30 31 2C 30 32 31 2C A1
4539:30 30 30 2C 30 30 30 2C 7F
4541:30 32 31 2C 30 30 30 2C 28
4549:30 30 30 00 72 45 F2 08 94
4551:83 30 30 30 2C 30 30 30 65
4559:2C 30 30 30 2C 30 30 30 C1
4561:2C 30 30 30 2C 30 30 30 C9
4569:2C 30 30 30 2C 30 30 30 D1
4571:00 97 45 FC 08 83 30 30 39
4579:30 2C 30 30 30 2C 30 30 F2
4581:30 2C 30 30 30 2C 30 30 FA
4589:30 2C 30 30 30 2C 30 30 03
4591:30 2C 30 30 30 00 BC 45 88
4599:06 09 83 30 30 30 2C 30 A7
45A1:30 30 2C 30 30 30 2C 30 A3
45A9:30 30 2C 30 30 30 2C 30 AB
45B1:30 30 2C 30 30 30 2C 30 B3
45B9:30 30 00 E1 45 10 09 83 86
45C1:30 30 30 2C 30 30 30 2C 08
45C9:30 30 30 2C 30 31 36 2C 20
45D1:30 30 30 2C 30 30 30 2C 18
45D9:30 39 36 2C 30 30 30 00 F6
45E1:06 46 1A 09 83 30 30 30 42
45E9:2C 30 34 30 2C 30 30 30 D2
45F1:2C 30 30 30 2C 31 36 39 73
45F9:2C 30 30 34 2C 30 30 32 A4
4601:2C 31 33 32 00 2B 46 24 D6
4609:09 83 30 32 30 2C 31 33 EB
4611:38 2C 30 30 30 2C 30 38 98
```

```
4619:32 2C 31 36 38 2C 30 30 56
4621:30 2C 30 37 35 2C 32 32 3B
4629:34 00 50 46 2E 09 83 30 0B
4631:30 30 2C 30 34 37 2C 31 72
4639:36 30 2C 30 30 30 2C 30 40
4641:34 32 2C 31 33 32 2C 30 F7
4649:30 30 2C 30 31 30 00 75 42
4651:46 38 09 83 30 32 30 2C 3F
4659:30 30 30 2C 30 30 32 2C A5
4661:30 38 30 2C 30 30 30 2C AB
4669:30 30 30 2C 30 36 34 2C D1
4671:30 30 30 00 9A 46 42 09 A3
4679:83 30 30 30 2C 30 30 30 8F
4681:2C 30 30 30 2C 30 30 30 EB
4689:2C 30 30 30 2C 30 30 30 F3
4691:2C 30 30 30 2C 30 30 30 FB
4699:00 BF 46 4C 09 83 30 30 8A
46A1:30 2C 30 30 30 2C 30 30 1D
46A9:30 2C 30 30 30 2C 30 30 25
46B1:30 2C 30 30 30 2C 30 30 2D
46B9:30 2C 30 30 30 00 E4 46 04
46C1:56 09 83 30 30 30 2C 30 F9
46C9:30 30 2C 30 30 30 2C 30 CD
46D1:30 31 2C 30 33 33 2C 30 3A
46D9:30 30 2C 30 30 31 2C 31 E2
46E1:36 39 00 09 47 60 09 83 B9
46E9:30 30 30 2C 30 30 31 2C 34
46F1:30 33 33 2C 30 30 30 2C 5B
46F9:30 30 30 2C 30 33 32 2C 52
4701:30 30 30 2C 30 30 30 00 1F
4709:2E 47 6A 09 83 30 33 32 D3
4711:2C 30 30 30 2C 30 30 30 7D
4719:2C 30 33 32 2C 30 30 30 06
4721:2C 30 30 30 2C 30 33 32 95
4729:2C 30 30 30 00 53 47 74 33
4731:09 83 30 30 30 2C 30 33 F3
4739:32 2C 30 30 30 2C 30 30 B7
4741:30 2C 31 36 38 2C 30 30 7F
4749:30 2C 30 30 30 2C 31 38 D0
4751:34 00 78 47 7E 09 83 30 CC
4759:30 30 2C 30 30 30 2C 31 60
4761:38 34 2C 30 30 30 2C 30 6C
4769:30 34 2C 31 38 34 2C 30 D0
4771:36 34 2C 30 30 35 00 9D A4
4779:47 88 09 83 31 36 39 2C 28
4781:30 36 34 2C 30 30 35 2C D7
4789:31 36 39 2C 30 36 34 2C 17
4791:30 30 34 2C 31 36 38 2C 8C
4799:30 36 34 00 C2 47 92 09 B5
```

```
47A1:83 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø B9
47A9:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 16
47B1:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 1E
47B9:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 26
47C1:ØØ E7 47 9C Ø9 83 3Ø 3Ø E3
47C9:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 47
47D1:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 4F
47D9:3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 57
47E1:3Ø 2C 3Ø 3Ø 3Ø ØØ ØC 48 7E
47E9:A6 Ø9 83 3Ø 3Ø 3Ø 2C 3Ø 4C
47F1:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø F7
47F9:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø FF
48Ø1:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø Ø9
48Ø9:3Ø 3Ø ØØ 31 48 BØ Ø9 83 6B
4811:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 5D
4819:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 65
4821:3Ø 3Ø 3Ø 2C 3Ø 32 3Ø 2C 75
4829:3Ø 3Ø 3Ø 2C 3Ø 3Ø 3Ø ØØ 49
4831:56 48 BA Ø9 83 3Ø 38 35 69
4839:2C 3Ø 3Ø 3Ø 2C 3Ø 3Ø 31 A8
4841:2C 3Ø 38 35 2C 3Ø 36 34 11
4849:2C 3Ø 3Ø 35 2C 3Ø 38 35 1D
4851:2C 3Ø 38 3Ø ØØ 7B 48 C4 51
4859:Ø9 83 3Ø 32 31 2C 3Ø 38 4B
4861:35 2C 3Ø 38 34 2C 3Ø 38 ØC
4869:35 2C 3Ø 38 35 2C 3Ø 38 1C
4871:35 2C 31 3Ø 36 2C 31 35 CA
4879:3Ø ØØ AØ 48 CE Ø9 83 31 8D
4881:36 39 2C 31 3Ø 31 2C 31 E3
4889:35 3Ø 2C 3Ø 38 39 2C 31 79
4891:3Ø 31 2C 31 35 3Ø 2C 3Ø 12
4899:38 39 2C 31 3Ø 36 ØØ C5 4D
48A1:48 D8 Ø9 83 31 35 3Ø 2C DØ
48A9:31 36 39 2C 31 3Ø 36 2C 2D
48B1:31 35 3Ø 2C 31 36 39 2C F1
48B9:31 3Ø 36 2C 31 35 3Ø 2C 63
48C1:31 36 39 ØØ EA 48 E2 Ø9 E6
48C9:83 31 3Ø 36 2C 31 35 3Ø 92
48D1:2C 31 36 39 2C 31 3Ø 36 DB
48D9:2C 31 35 3Ø 2C 31 36 39 42
48E1:2C 31 3Ø 36 2C 31 35 3Ø FE
48E9:ØØ ØF 49 EC Ø9 83 31 36 25
48F1:39 2C 31 3Ø 36 2C 31 35 4D
48F9:3Ø 2C 31 36 39 2C 31 3Ø 44
49Ø1:36 2C 31 35 3Ø 2C 31 36 FD
49Ø9:39 2C 3Ø 3Ø 3Ø ØØ 34 49 7F
4911:F6 Ø9 83 3Ø 3Ø 3Ø 2C 3Ø 9F
4919:3Ø 3Ø 2C 3Ø 3Ø 3Ø 2C 3Ø 23
4921:3Ø 3Ø 2C 3Ø 3Ø 31 2C 3Ø 2F
```

```
4929:30 30 2C 30 36 34 2C 30 73
4931:30 37 00 59 49 00 0A 83 21
4939:30 36 34 2C 30 36 34 2C A9
4941:30 30 37 2C 30 36 34 2C 90
4949:30 30 30 2C 30 34 39 2C B9
4951:30 30 30 2C 30 30 34 00 7B
4959:7E 49 0A 0A 83 30 30 30 CC
4961:2C 30 31 30 2C 30 33 32 F9
4969:2C 30 30 30 2C 30 31 34 DF
4971:2C 30 30 30 2C 31 32 38 F1
4979:2C 30 30 30 00 A3 49 14 6C
4981:0A 83 30 30 32 2C 31 33 DA
4989:36 2C 31 39 32 2C 30 36 D4
4991:36 2C 30 34 32 2C 31 33 6B
4999:31 2C 30 30 30 2C 31 36 A3
49A1:38 00 C8 49 1E 0A 83 30 4E
49A9:30 30 2C 30 34 38 2C 30 F3
49B1:34 33 2C 30 33 32 2C 30 9E
49B9:30 32 2C 31 36 38 2C 30 A4
49C1:31 32 2C 30 31 36 00 ED 52
49C9:49 28 0A 83 30 33 32 2C 63
49D1:31 32 38 2C 30 30 30 2C 22
49D9:30 30 30 2C 30 34 30 2C 38
49E1:30 30 33 2C 30 30 30 2C 90
49E9:30 36 34 00 12 4A 32 0A D0
49F1:83 30 34 38 2C 30 30 30 0F
49F9:2C 30 30 30 2C 30 30 30 6A
4A01:2C 30 31 36 2C 30 31 36 FB
4A09:2C 31 33 31 2C 30 30 30 2C
4A11:00 3B 4A 3C 0A 83 31 31 73
4A19:36 2C 31 36 30 2C 30 30 20
4A21:30 2C 31 31 36 2C 30 30 05
4A29:30 2C 30 30 30 2C 30 31 AD
4A31:36 2C 30 30 30 2C 32 35 C0
4A39:36 00 00 00 00 00 00 00 E8
```

Hex War

Todd Heimarck

You float high above a distant planet, controlling robot armies below. Can you take control of the priceless mining turf planetside, or will your opponent's robot crews prevail? To win at this thoughtfully designed, engaging strategy game, you'll need foresight and conceptual skills rather than a quick hand. A joystick is required.

"Hex War" is a two-player strategy game that can be played five different ways, and there are limitless variations. But the basic premise is always the same: You and an opponent move armies on a field of hexagons, attempting to capture territory.

The goal of the first two games is simple: capture the other player's capital city. In game 1, the capital cities are far apart; you must devote some of your armies to defending your own capital while attempting to breach the walls of the other capital. Game 2 puts the capitals near each other, so offense and defense tend to merge in this scenario. Most of the action takes place within a small area of the battlefield.

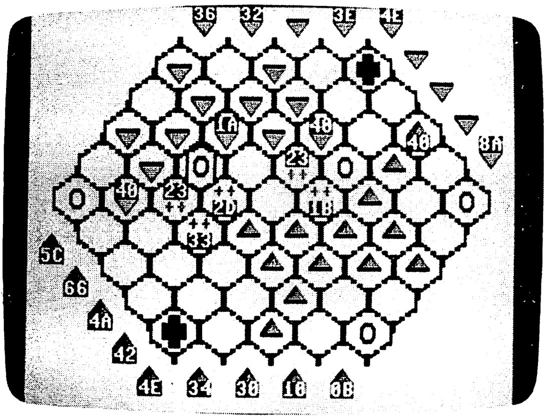
Games 3 and 4 spread the action over a wider area. In the third game, your objective is to occupy 8 of the 12 cities on the game board. Six cities occupy the periphery, and 6 are in the center of the playing field. Game 4 requires actual control of 6 cities; you must have an army in the city, one that's not involved in a battle, before you're credited with control (this version will probably take the longest time to play).

Although the first four scenarios encourage a commitment to battle, you employ different tactics in the fifth. The goal here is to acquire 40 of the 61 hexes, so you need some free armies to move around. As soon as you claim 40 hexes, you win the game.

Typing It In

Hex War is written in BASIC, with some important information in DATA statements. Type in the program, and be sure to save a copy. After you've saved the game, type RUN to begin playing.

When you first run Hex War, the computer pauses to set up the screen; then it displays a menu of five choices. The five different games are explained in detail below. If you're new to the game, press the 1 key to choose game 1. There will be another short pause while the variables are initialized, and then you'll see a playing field with 61 hex shapes, containing four armies on each side.



"Hex War" is an absorbing strategy game with many variations.

Hexes and Hexadecimal

A chess board has 64 squares arranged in a rectilinear grid. Hex War gives you a playing field of 61 hexagons (almost as many spaces as a chess board), but they're part of a six-sided honeycomb field. If you've played war games before, you may recognize the hexes. Plug in the joystick before playing (use port 2). At first, the cursor movement may seem unusual. The cursor travels not up-down/left-right, but northeast-southwest/northwest-southeast. To make the movement less confusing, turn your joystick 45 degrees clockwise so that what was up becomes northeast, and so forth.

Each hex has six neighbors, so an army can move in six possible directions. To travel left and right, you'll have to push the joystick twice (for example, up and right on the joystick to move one hex to the right, which counts as one movement).

Army strengths are listed in hexadecimal (base 16) numbers, so the four armies labeled 40 actually have strengths of 64 (the hexadecimal value 40 equals 64 in our everyday decimal numbering system). At the beginning of a turn, any army has exactly three movement points. It requires one point to move an army into a neutral or enemy-controlled zone. To move *through* the same zone also requires a point. To move into and through a friendly hex requires a total of one point. This means you can move a single army through two neutral or enemy hexes in any one turn, but the same army can move through up to three friendly zones during a turn.

Select an army by moving the cursor onto it. Click the joystick button once; then position the cursor on a neighboring hex and click again. If you want to stop, click again. Two plus signs (++) will appear, signaling that no more movement can occur. Otherwise, position the cursor on another neighboring hex and click.

Zones of Control

Each army controls the six contiguous hexes surrounding its resident hex. If you enter an enemy's zone of control, you forfeit any additional moves and must prepare for battle. In addition, an army that begins the turn in a zone of control cannot move until the battle is resolved.

Robots Vs. Robots

In this game, you aren't really on the planet, but parked high above it in a remote mothership. You've landed some robots to explore the area, and they've encountered robots belonging to another explorer. Your robots, or *bots* as you call them, follow your orders to advance toward the other bots. Each bot has a mining laser which can stop or disable the other bots. Also, your bots have disruptor beams which can daze another bot, temporarily confusing it. When two bot groups come close to each other, they shoot lasers and disruptors until one army of bots is disabled.

Three things can happen to a robot that suffers a hit. If the robot suffers a direct hit by a laser in its logic unit, it is vaporized. It is destroyed forever and never reappears in play.

The second thing that can happen is injury. If the laser beam is deflected, the robot is out of commission until it can be transported back to a botspital. An injured bot is frozen in place until the battle is finished, after which the victorious army carts away the injured bots to be repaired and reused.

Thus, winning a battle means you evacuate both the friendly injured and the enemy injured. After all of the injured bots recover, *they join the force in whose botspital they were healed*. In effect, injured bots eventually become members of the army that won the battle in which they were damaged.

The third possibility is confusion: The robot is temporarily disoriented for two turns. When that time has passed, the robot is ready again.

Reprogramming Bots

Moving the cursor onto an army of robots brings up a status window in the upper left corner of the screen. The number in reverse video is unimportant; it's the army number (which may change as the game progresses).

The four numbers underneath, however, are significant. The first is the army's active strength (in decimal). The second is the number of injured robots that will be transported to the botspital of whichever side wins the battle. The third—on the line below—is the number of disrupted robots who will be available for combat in the next turn. The fourth number is the number of robots that can join the active force two turns from now.

If one side is able to reduce the other player's active force to zero, two things happen. The winner sends all injured bots away to be repaired. The winning side also collects all enemy bots (injured or dazed) and sends them to the reinforcement center to be reprogrammed. Eventually, all these bots will be available to the winner of this particular battle for future engagements.

Reinforcements and Mergers

At the start of the game, you'll see some armies positioned outside the hex field. These are reinforcements and reserves in transit to the battle. Player 1's reinforcements enter at the bottom right corner; player 2's enter at the top left. The line of new armies moves counterclockwise; the army next to the entry point is the next to enter the battlefield.

However, the reinforcements cannot enter the battlefield if an army (friendly or enemy) is blocking their way. Keep your armies off your own reinforcement point, and try to block your opponent's armies from this area if you can. If the entry hex is owned but not occupied by your opponent, you'll lose some reinforcements.

After completing a turn, you are credited with additional reinforcements according to how much territory you own. Passing over a hex allows you to claim it; the hex changes color to indicate ownership. Each piece of property provides enough ore and energy to build a new robot, available for use two turns in the future. The numbers in the line of reinforcements are updated after you move to show additional robots being built.

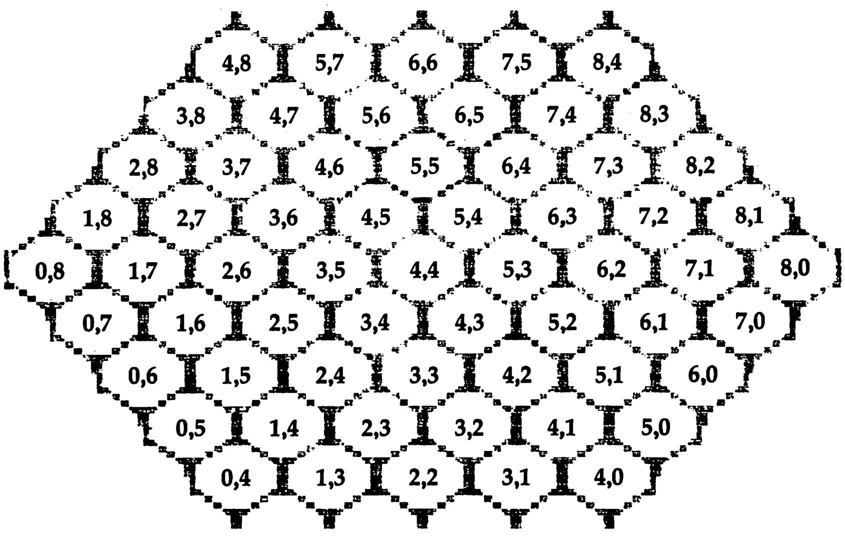
Winning a battle also provides additional armies in the line of reinforcements. As mentioned above, a victorious army captures any dazed enemy bots, which are reprogrammed and available in three turns. At the same time, the winner evacuates injured bots of both sides. Transportation and repair take five turns for friendly bots; seven for enemy bots. The two additional turns are needed for reprogramming the opponent's forces.

If you're losing a battle, the number of injured robots (displayed in the status window) will begin to rise. Remember that, if your opponent reduces your active strength to zero, he or she will capture all of your injured bots; they'll be reprogrammed and added to future reinforcements. To prevent this from happening, you're allowed to bring in a second army for merging. Simply move another army on top of the army with which you want to merge. There's just one rule: One or both of the armies must have a strength less than 32 decimal (1F or less in hex).

Customizing the Scenarios

The five built-in scenarios provide plenty of variety, but if you'd like to add more challenges, here are some suggestions. But first, a note about the logical organization of the grid. The variables T and B, CT and CB, and HT and HB are used to locate the coordinates on the playing field (see figure). The first number is T (or HT or CT); the second is B (or HB or CB). These coordinates are also used in the three-dimensional MAP array (where level 0 of the array is the army number, 1 is the current owner, and 2 keeps track of whether or not a city is located there). They're also part of the ARMY array. By varying the starting position, number of armies, reinforcement strengths, and location of cities, you could simulate historic battles.

T and B Coordinates



Note: The first number is the T coordinate; the second, the B coordinate.

To add or subtract cities from the field, change the value of CN in line 50. You'll also have to change the DATA statements in lines 270 and 280. The numbers there are the T and B coordinates of the cities.

The strengths and locations of the armies can be changed as well. The DATA statements starting at line 1540 determine the strength (64) and T/B coordinates for the armies at the beginning of the game. If you wish to start with more armies (or fewer), you'll have to change the inner FOR-NEXT loop (with the index of K) in line 1500. In that same line, change NX(J) to one number higher than the number of armies on each side. For example, if you want six armies apiece, change NX(J) to 7. The subroutine at line 1600 sets up the reinforcements; if you don't like the random patterns, change the formula here.

Variables defined in lines 70–90 control the play of the game. PN determines which player goes second; it can be either zero or one. Variable ME controls the maximum merge strength. If you'd like to be able to merge any two armies, change it to a high value (512, for example). To remove the merge option altogether, change ME to zero.

The movement points are defined by MM in line 80. Movement across friendly territory takes one point; across neutral or hostile territory, two points. Increasing MM will give your armies greater mobility. The three variables KA, KB, and KC affect the outcome of individual battles. KA determines how many bots are vaporized, KB controls the number injured, and KC affects how many are dazed. If you make the fractions smaller (1/24, for example), the battles end more quickly. The subroutine starting at line 2600 resolves current battles.

Hex War

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
MM 10 IF PEEK(46)<>64 THEN GRAPHIC1:GRAPHIC0:FAST:
      FORJ=0TO15:COLOR4,J+1:K=J*128:FORL=KTOK+127:
      BANK14:J1=PEEK(53248+L):BANK0:POKE12288+L,J1
      :NEXTL,J
FB 20 BANK15:POKE217,4:POKE2604,28:SLOW
PD 30 DIMJ,K,HT,HB,CT,CB,J1,J2,A,B,C,D,E
PK 40 DIM ARMY(31,6,1),BTL(64,1,3),MAP(9,9,2),FQ(2
      0,1),NX(1),C(2)
BQ 50 CN=12:DIM CIT(CN,1)
BC 60 A=RND(-TI/97):P0$="{BLU}{OFF}Ɛ2 PƷ{DOWN}
      {2 LEFT}Ɛ6ƷƐ*Ʒ£":P1$="{YEL}{RVS}£Ɛ*Ʒ{OFF}
      {DOWN}{2 LEFT}{PUR}Ɛ2 YƷ"
EP 70 PN=1:ME=31
CE 80 MM=3: REM MAX MOVES
JG 90 KA=1/48:KB=1/48:KC=1/32
BX 100 BANK15:POKE217,4:POKE2604,28
SM 110 FORJ=1TO4:READA
FQ 120 FORK=ATOA+7:READB:POKEK,B:NEXTK,J
RA 130 DATA 12936,240,240,63,15,3,3,3,3
AS 140 DATA 12984,15,15,252,240,192,192,192,192
PP 150 DATA 12840,3,3,3,3,15,63,240,240
BJ 160 DATA 12944,192,192,192,192,240,252,15,15
RG 170 FORJ=0TO63:READK:POKE3584+J,K:NEXT
CC 180 DATA0,255,0,15,195,240,63,0
KM 190 DATA252,48,0,12,48,0,12,48
AE 200 DATA0,12,48,0,12,48,0,12
GD 210 DATA48,0,12,48,0,12,48,0
```

```
JA 22Ø DATA12,48,Ø,12,48,Ø,12,48
EG 23Ø DATAØ,12,48,Ø,12,48,Ø,12
KB 24Ø DATA48,Ø,12,48,Ø,12,63,Ø
BK 25Ø DATA252,15,195,24Ø,Ø,255,Ø,Ø
PQ 26Ø FORJ=1TOCN:FORK=ØTO1:READ CIT(J,K):NEXTK:MA
       P(CIT(J,Ø),CIT(J,1),2)=1:NEXTJ: REM T&B OF
       {SPACE}CITIES
ER 27Ø DATA 8,4,Ø,4,8,Ø,Ø,8,4,Ø,4,8
JQ 28Ø DATA 5,5,3,3,6,3,2,5,5,2,3,6
XK 3ØØ GOSUB6ØØ:GOSUB32ØØ:GOSUB6ØØ:REM GAME
JS 31Ø CHAR1,6,11:PRINT"{BLU}{RVS}{2 SPACES}L{M}
       {2 SPACES}L{Y}{2 SPACES}MN{4 SPACES}{G}{M}
       {2 SPACES}NM{2 SPACES}OM{2 SPACES}"
EP 32Ø CHAR1,6,12:PRINT"{PUR}{RVS}{2 SPACES}{G}{M}
       {2 SPACES}L{P}{2 SPACES}NM{4 SPACES}NM
       {2 SPACES}O{M}{2 SPACES}{G}M{2 SPACES}
       {HOME}{RED}{OFF}";
SM 33Ø PRINTSPC(1Ø);"PLEASE WAIT A MOMENT"
FJ 34Ø GOSUB15ØØ
ME 4ØØ DO
GR 41Ø POKE2Ø8,Ø
QX 42Ø GOSUB19ØØ:GOSUB6ØØ:GOSUB171Ø:REM FIND BATTL
       ES
QG 43Ø COLOR4,7-2*PN:GOSUB8ØØ:REM JOYSTICK
CG 44Ø FAST:COLOR4,9:GOSUB21ØØ:REM BATTLES AGAIN
JJ 45Ø COLOR4,3:GOSUB26ØØ:REM RESOLVE
EB 46Ø COLOR4,1:GOSUB21ØØ:REM POST-BATTLE
AA 47Ø COLOR4,16:GOSUB22ØØ:REM SPLIT PRISONERS
JP 48Ø GOSUB 29ØØ:REM REINFORCEMENTS
HB 49Ø SLOW:COLOR4,8:GOSUB34ØØ
HS 5ØØ PN=1-PN
DS 51Ø LOOP
JA 6ØØ COLOR4,2:COLORØ,2:COLOR5,16:SCNCLR:PRINT:PR
       INT:SPRITE1,Ø
CB 61Ø FORJ=1TO5:PRINTSPC(13-2*J);:FORK=1TOJ+3:PRI
       NT"ER{2 SPACES}";:NEXTK:PRINT"ER"
CS 62Ø PRINTSPC(12-2*J);:FORK=1TOJ+4:PRINT"W
       {2 SPACES}Q";:NEXTK:PRINT:NEXTJ
QX 63Ø FORJ=1TO5:PRINTSPC(J*2);:FORK=1TO1Ø-J:PRINT
       "R{2 SPACES}E";:NEXTK:PRINT
SS 64Ø PRINTSPC(J*2+1);:FORK=1TO9-J:PRINT"QW
       {2 SPACES}";:NEXTK:PRINT"QW"
PX 65Ø NEXTJ
CD 66Ø C$="UI{DOWN}{2 LEFT}JK":D$="{RVS}{V}{C}
       {DOWN}{2 LEFT}{F}{D}"
MJ 67Ø COLOR5,3:FORJ=1TO12:GOSUB71Ø:NEXT
HQ 68Ø J=1:COLOR5,7:IFGN=1THENC$=D$:GOSUB71Ø:J=2:C
       OLOR5,5:GOSUB71Ø
```

```
RB 690 IFGN=2THENC$=D$:GOSUB710:J=3:COLOR5,5:GOSUB
       710
SH 700 PRINT"{HOME}";:RETURN
KB 710 K=CIT(J,0):L=CIT(J,1):X=(K-L)*2+19:Y=(12-(K
       +L))*2+3:CHAR1,X,Y,C$:RETURN
FF 800 IFNX(PN)<2THENRETURN
AX 805 HT=4:HB=4:GOSUB1000:SPRITE1,1,1,0
KE 810 MV=0:CT=0:CB=0:PK=0:REM PICKED UP OR NOT
HS 820 K=0:FORJ=1TONX(PN)-1:IF(ARMY(J,0,PN)>0)AND(
       ARMY(J,6,PN)<1)THENK=1:J=NX(PN)-1
KX 830 NEXTJ:IFK=0 THEN RETURN
SB 840 GETG$:IFG$=CHR$(13)THENRETURN
HD 850 J=JOY(2):IFJ=0THEN840:ELSE IF(JAND128)THEN1
       100:ELSE IF(JAND1)=0THEN840
KK 860 J=(J-1)/2:IFJAND1THENB1=HB+J-2:T1=HT:ELSET1
       =HT+1-J:B1=HB
HD 870 IF (T1<0)OR(T1>8)THEN840
FP 880 IF (B1<0)OR(B1>8)THEN840
KC 890 S1=T1+B1:IF(S1<4)OR(S1>12)THEN840
EM 900 HB=B1:HT=T1:GOSUB1000:WINDOW1,1,8,4,1:QN=MA
       P(HT,HB,0):IFQN=0THENPRINT"{2 HOME}";:GOTO8
       40:ELSE Q1=MAP(HT,HB,1)-1
JF 910 COLOR5,7-2*Q1:PRINTUSING"{RVS}##";QN;:PRINT
       "******{OFF}{5}";
RH 920 FORJ=0TO3:PRINTUSING"####";ARMY(QN,J,Q1);:N
       EXT
DD 930 PRINT"{2 HOME}";:GOTO840
PA 1000 SX=172+16*(HT-HB):SY=264-16*(HT+HB)
KC 1010 MOVSPR1,SX,SY
SA 1020 IF MAP(HT,HB,2)=1THENSPRITE1,1,3,0:RETURN
KP 1030 SPRITE1,1,1,0:RETURN
CX 1100 IFJOY(2)<>0THEN1100
FE 1110 IFPK=1THEN1200:REM PICKEDUP, CHECK IF OK
FD 1120 IF((MAP(HT,HB,1)<>PN+1)OR(MAP(HT,HB,0)=0))
        THENGOTO810:REM NO ONE HOME
XE 1130 AN=MAP(HT,HB,0):IFARMY(AN,6,PN)<>0THEN810:
        REM ARMY AN IS ENGAGED
SD 1140 PK=1:CT=HT:CB=HB:CS=ARMY(AN,0,PN):REM T&B,
         CURRENT STRENGTH
DP 1150 SOUND1,5000,10
MK 1160 GOTO840
XP 1200 J=((HT=CT)AND(HB=CB))
AQ 1210 IFJAND(MV=0)THEN810
RE 1220 IFJAND(MV>0)THEN1420
MH 1230 AS=ARMY(MAP(HT,HB,0),0,PN):IF((AS>ME)AND(C
        S>ME))OR((MAP(HT,HB,1)-1=1-PN)AND(AS>0)) T
        HEN840
FB 1240 DT=ABS(CT-HT):DB=ABS(CB-HB):TL=DB+DT:IF NO
        T((TL=1)OR((CT+CB=HT+HB)AND(DT=1)))THEN840
```

```
FS 1250 MG=MAP(HT,HB,0): IF MG=0THEN1300
AE 1260 FORJ=0TO3:ARMY(MG,J,PN)=ARMY(MG,J,PN)+ARMY
        (AN,J,PN):ARMY(AN,J,PN)=0:NEXTJ
SM 1270 ARMY(MG,6,PN)=1:MAP(CT,CB,0)=0
AD 1280 CS=ARMY(MG,0,PN):AN=MG:MV=MM+1
RH 1290 GOTO 1380
FF 1300 N8=MAP(HT,HB,1)-1:MV=MV+1:IF(N8<>PN)THENMV
        =MV+1
DK 1310 MAP(CT,CB,0)=0
SA 1320 MAP(HT,HB,0)=AN:MAP(HT,HB,1)=PN+1:ARMY(AN,
        4,PN)=HT:ARMY(AN,5,PN)=HB:IF MV>=MMTHEN AR
        MY(AN,6,PN)=1
KR 1330 K=0:FORJ=-1TO1STEP2:J1=HT+J:J2=HB+J:J3=HB-
        J:IF(J1<0)OR(J1>8)THEN1340:ELSE IF(MAP(J1,
        HB,0)>0)THENIF(MAP(J1,HB,1)=2-PN)THENK=1:J
        =1:GOTO1360
BR 1340 IF(J2<0)OR(J2>8)THEN1350:ELSE IF(MAP(HT,J2
        ,0)>0)THENIF(MAP(HT,J2,1)=2-PN)THENK=1:J=1
        :GOTO1360
KE 1350 IF(J3<0)OR(J3>8)OR(J1<0)OR(J1>8)THEN1360:E
        LSE IF(MAP(J1,J3,0)>0)THENIF(MAP(J1,J3,1)=
        2-PN)THENK=1:J=1
HK 1360 NEXTJ: REM ZOC
AS 1370 IFK=1THEN ARMY(AN,6,PN)=1:MV=MM+1
EG 1380 A=PN:J=CT:K=CB:C=0:D=0:GOSUB1830
QP 1390 J=HT:K=HB:C=CS:D=ARMY(AN,6,PN):GOSUB1830
XJ 1400 CT=HT:CB=HB
MQ 1410 IFMV<MMTHEN840
HA 1420 ARMY(AN,6,PN)=1:J=HT:K=HB:C=CS:D=1:GOSUB18
        30
SG 1430 GOTO810
FC 1500 RESTORE1540:FORJ=0TO1:NX(J)=5:FORK=1TO4:RE
        ADA,B,C
QM 1510 ARMY(K,0,J)=A:ARMY(K,4,J)=B:ARMY(K,5,J)=C:
        MAP(B,C,0)=K:MAP(B,C,1)=J+1
BJ 1520 NEXTK,J
EK 1530 REM STRENGTH, T-POS, B-POS
JD 1540 DATA 64,2,8,64,3,7,64,5,6,64,6,6:REM BLUE
QP 1550 DATA 64,2,2,64,3,2,64,5,1,64,6,0:REM VIOLE
        T
JK 1600 REM SET RANDOM REINFORCEMENTS
AS 1610 FORJ=0TO1:FORK=0TO20
QM 1620 A=INT(RND(1)*K*3):FORL=1TO5:A=A+INT(RND(1)
        *21-8):NEXTL:IFA<16THENA=0:ELSEA=(A+K*8)AN
        D254
HA 1630 FQ(K,J)=A:NEXTK,J
RG 1640 RETURN
EH 1700 REM ARMIES->MAP
SC 1710 FORJ=0TO8:FORK=0TO8
```

```
FX 1720 A=MAP(J,K,1):IFATHENA=A-1:GOSUB1800
HH 1730 NEXTK,J
PH 1740 FORA=0TO1:E=13+A*12:F=A*22:DX=2-4*A:D=0
PF 1750 FORJ=0TO8:C=FQ(J,A):GOSUB1840
HF 1760 E=E+DX*2:IFJ>3THENF=F+DX:E=E-DX
XC 1770 NEXTJ,A
CA 1780 RETURN
QC 1800 B=MAP(J,K,0)
PH 1810 C=ARMY(B,0,A)
BK 1820 D=ARMY(B,6,A)
FJ 1830 E=(J-K+10)*2-1:F=(13-J-K)*2+1: REM T&B TO
        {SPACE}X/Y
SC 1840 CHAR1,E,F:IFATHENPRINTP1$:ELSEPRINTP0$
MH 1850 IFC=0THENRETURN
PP 1860 COLOR5,(7-2*A):CHAR1,E,F+A
HR 1870 PRINTCHR$(18);RIGHT$(HEX$(C),2);CHR$(146)
RD 1880 IFDTHENF=F+1-A:G=1024+E+F*40:POKEG,43:POKE
        G+1,43:REM ++
XK 1890 RETURN
GC 1900 SW=0:E=NX(PN)-1:IFE<1THENRETURN
XJ 1910 FORJ=1TOE-1:IFARMY(J,0,PN)<1THENBEGIN
SE 1920 T=ARMY(J,4,PN):B=ARMY(J,5,PN):IFMAP(T,B,0)
        =JTHENMAP(T,B,0)=0
BP 1930 FORK=JTOE:FORL=0TO6:ARMY(K,L,PN)=ARMY(K+1,
        L,PN):ARMY(K+1,L,PN)=0:NEXTL
JQ 1940 T=ARMY(K,4,PN):B=ARMY(K,5,PN):MAP(T,B,0)=K
PH 1950 NEXTK
MA 1960 NX(PN)=NX(PN)-1:J=E:SW=1:BEND
FJ 1970 NEXTJ:IF SW THEN1900
MG 2000 FORJ=1TOE:ARMY(J,0,PN)=ARMY(J,0,PN)+ARMY(J
        ,2,PN)
SD 2010 ARMY(J,2,PN)=ARMY(J,3,PN):ARMY(J,3,PN)=0
QS 2020 ARMY(J,6,PN)=0
PP 2030 NEXTJ:K=NX(1-PN):FOR J=1TOK:ARMY(J,6,1-PN)
        =0:NEXT
XF 2040 GOSUB2400
DS 2050 IFBP>0 THEN FORJ=0TO1:FORK=1TOBP:A=BTL(K,J
        ,0):ARMY(A,6,J)=ARMY(A,6,J)+1:NEXTK,J
CA 2060 RETURN
KG 2100 GOSUB2400
BX 2110 A=NX(0):IFNX(1)>ATHENA=NX(1)
EX 2120 FORJ=0TO1:FORK=1TOA:ARMY(K,6,J)=0:NEXTK,J
GP 2130 GOSUB2050
SH 2140 RETURN
PB 2200 FORJ=0TO1:A=1-J:B=NX(J)-1
DD 2210 FORK=1TOB
EM 2220 IF ARMY(K,0,J)<1 THEN BEGIN
```

```
JC 2230 FQ(2,A)=FQ(2,A)+ARMY(K,2,J)+ARMY(K,3,J):IF
         FQ(2,A)>255 THEN C=FQ(2,A)-255:FQ(3,A)=FQ
        (3,A)+C:FQ(2,A)=255
JX 2240 FQ(6,A)=FQ(6,A)+ARMY(K,1,J):IF FQ(6,A)>255
         THEN C=FQ(6,A)-255:FQ(7,A)=FQ(7,A)+C:FQ(6
        ,A)=255
JQ 2250 IF (MAP(ARMY(K,4,J),ARMY(K,5,J),Ø)=K)AND(M
        AP(ARMY(K,4,J),ARMY(K,5,J),1)=J+1) THEN MA
        P(ARMY(K,4,J),ARMY(K,5,J),Ø)=Ø
RB 2260 FORL=ØTO6:ARMY(K,L,J)=Ø:NEXTL
BB 2270 BEND
FR 2280 IF ARMY(K,6,J)<1 THEN BEGIN:REM EVACUATE I
        NJURED
AE 2290 FQ(4,J)=FQ(4,J)+ARMY(K,1,J): ARMY(K,1,J)=Ø
DJ 2300 IF FQ(4,J)>255 THEN C=FQ(4,J)-255:FQ(5,J)=
        FQ(5,J)+C:FQ(4,J)=255
KA 2310 BEND
XP 2320 NEXTK,J:RETURN
SJ 2400 BP=Ø
KG 2410 FORJ=ØTO8:J1=(J-4)*(4-J>Ø):J2=8-(J>4)*(4-J
        ):FORK=J1TOJ2
JE 2420 A=MAP(J,K,Ø)
HQ 2430 R=MAP(J,K,1)
BG 2440 IF (A=Ø)OR(R=Ø) THEN2490
KQ 2450 IF ARMY(A,Ø,R-1)<1 THEN2490
DK 2460 T=J+1:B=K:GOSUB25ØØ
HK 2470 B=B-1:GOSUB25ØØ
BP 2480 T=T-1:GOSUB25ØØ
RH 2490 NEXTK,J:RETURN
RR 2500 IF(T<Ø)OR(B<Ø)OR(T>8)OR(B>8)THEN RETURN
MA 2510 PA=MAP(T,B,Ø):IF PA=Ø THEN RETURN
FG 2520 IF MAP(T,B,1)=R THEN RETURN
FX 2530 IF ARMY(PA,Ø,2-R)<1 THEN RETURN
MA 2540 BP=BP+1:BTL(BP,R-1,Ø)=A:BTL(BP,2-R,Ø)=PA:R
        ETURN
CF 2600 IFBP=ØTHENRETURN
FC 2610 FORJ=1TOBP
XM 2620 FORK=ØTO1:A=1-K
QX 2630 AN=BTL(J,K,Ø)
BF 2640 AS=ARMY(AN,Ø,K):HT=ARMY(AN,6,K):CT=INT(AS/
        HT)+1
GH 2650 BTL(J,A,1)=INT(CT*KA+1)
JC 2660 BTL(J,A,2)=INT(CT*KB+1)
MB 2670 BTL(J,A,3)=INT(CT*KC+1)
XC 2680 NEXTK,J
MD 2700 FORJ=1TOBP:JØ=BTL(J,Ø,Ø):J1=BTL(J,1,Ø)
BR 2710 GOSUB31ØØ
CG 2720 ARMY(JØ,Ø,Ø)=ARMY(JØ,Ø,Ø)-A*BTL(J,Ø,1)
SK 2730 ARMY(J1,Ø,1)=ARMY(J1,Ø,1)-B*BTL(J,1,1)
```

```
MS 2740 GOSUB3100
BC 2750 C=A*BTL(J,0,2):ARMY(J0,0,0)=ARMY(J0,0,0)-C
        :ARMY(J0,1,0)=ARMY(J0,1,0)+C
FJ 2760 C=B*BTL(J,1,2):ARMY(J1,0,1)=ARMY(J1,0,1)-C
        :ARMY(J1,1,1)=ARMY(J1,1,1)+C
QD 2770 GOSUB3100
AX 2780 C=A*BTL(J,0,3):ARMY(J0,0,0)=ARMY(J0,0,0)-C
        :ARMY(J0,3,0)=ARMY(J0,3,0)+C
FF 2790 C=B*BTL(J,1,3):ARMY(J1,0,1)=ARMY(J1,0,1)-C
        :ARMY(J1,3,1)=ARMY(J1,3,1)+C
XP 2800 NEXTJ
CB 2810 RETURN
GQ 2900 A=1-PN:B=0
GG 2910 FORJ=0TO8:FORK=0TO8
PA 2920 IFMAP(J,K,1)=PN+1THENB=B+1
KD 2930 NEXTK,J
JB 2950 FQ(1,PN)=FQ(1,PN)+B: IF FQ(1,PN)>255 THEN
        {SPACE}B=FQ(1,PN)-255:FQ(2,PN)=FQ(2,PN)+B:
        FQ(1,PN)=255
GQ 2960 T=4:B=PN*8
SJ 2970 IF MAP(T,B,0)<>0 THEN RETURN
JK 2980 IF MAP(T,B,1)=PN+1 THEN FQ(0,A)=0:FQ(1,A)=
        0:GOTO3060
XS 2990 J=NX(A):IFJ>31THENRETURN
HR 3000 J1=FQ(0,A):IF J1<1 THEN3060
EP 3010 NX(A)=NX(A)+1
XE 3020 MAP(T,B,0)=J:MAP(T,B,1)=A+1
BK 3030 ARMY(J,0,A)=J1
QR 3040 FORK=1TO3:ARMY(J,K,A)=0:NEXTK
PM 3050 ARMY(J,4,A)=T:ARMY(J,5,A)=B
KH 3060 FORK=0TO19:FQ(K,A)=FQ(K+1,A):NEXTK
PK 3070 FQ(20,A)=0
BA 3080 RETURN
FS 3100 A=0:FORM=1TO6:IFRND(1)<.5THENA=A+1
AH 3110 NEXTM:B=6-A:RETURN
PS 3200 WINDOW6,7,33,16,1
EJ 3210 PRINT"{7}{2 M}{2 L}{RVS}{2 K}{J}{H}{G} SCE
        NARIO {M}{N}{L}{OFF}{3 K}{J}{2 G}{M}"
FB 3220 PRINT"{LEFT}{G}{M}{2 SPACES}1> CAPTURE CAP
        ITAL/FAR{2 SPACES}{G}";
QX 3230 PRINT"{M}{2 SPACES}2> CAPTURE CAPITAL/NEAR
         {G}";
GQ 3240 PRINT"{M}{2 SPACES}3> OCCUPY{3 SPACES}8/12
         CITIES {G}";
AR 3250 PRINT"{M}{2 SPACES}4> CONTROL{2 SPACES}6/1
        2 CITIES {G}";
DJ 3260 PRINT"{M}{2 SPACES}5> OCCUPY{2 SPACES}40/6
        1 HEXES{2 SPACES}{G}{M}";:PRINTSPC(26);"
        {G}"
```

```
SQ 3270 PRINT"{2 M}{2 L}{RVS}{2 K}{J}{H}{G}
        {10 SPACES}{M}{N}{L}{OFF}{3 K}{J}{2 G}"
GE 3280 GETKEYA$:GN=VAL(A$):IFGN<1ORGN>5THEN3280
DP 3290 CHAR1,2,1+GN,"ZZ":SLEEP1:PRINT"{2 HOME}"
PX 3300 RETURN
GS 3400 A=0:ON GN GOSUB 3430,3450,3480,3490,3580
JD 3410 IFA=0THENRETURN:ELSEEN$=C$:QQ=A:GOSUB600:G
        OSUB1710:A=QQ
KD 3420 PRINT"{HOME}PLAYER";A;" WINS":PRINTEN$:PRI
        NT"(PRESS ANY KEY)":POKE208,0:GETKEYA$:RUN
BB 3430 IF MAP(CIT(2,0),CIT(2,1),1)=1THENA=2:C$="B
        LUE CAPTURED THE CAPITAL":RETURN
CR 3440 GOTO3460
SG 3450 IF MAP(CIT(3,0),CIT(3,1),1)=1THENA=2:C$="B
        LUE CAPTURED THE CAPITAL":RETURN
KB 3460 IF MAP(CIT(1,0),CIT(1,1),1)=2THENA=1:C$="V
        IOLET CAPTURED THE CAPITAL"
FP 3470 RETURN
BC 3480 L=8:GOTO3500
JB 3490 L=6
FG 3500 C(1)=0:C(2)=0
PK 3510 FORJ=1TO12:T=CIT(J,0):B=CIT(J,1)
MB 3520 R=MAP(T,B,1):C(R)=C(R)+1
DB 3530 IF GN=4THEN AN=MAP(T,B,0):IF R>0 THEN IF (
        AN=0)OR(ARMY(AN,6,R-1)>0)THEN C(R)=C(R)-1
KJ 3540 NEXTJ
CB 3550 IF C(1)=> L THEN A=2:C$="BLUE HAS CAPTURED
        "+STR$(C(1))+" CITIES":RETURN
QF 3560 IF C(2)=> L THEN A=1:C$="VIOLET HAS CAPTUR
        ED"+STR$(C(2))+" CITIES"
AB 3570 RETURN
RK 3580 C(1)=0:C(2)=0
PP 3590 FORJ=0TO8:FORK=0TO8
KM 3600 R=MAP(J,K,1):C(R)=C(R)+1
SR 3610 NEXTK,J
CE 3620 IF C(1)=>40THENA=2:C$="BLUE OCCUPIES"+STR$
        (C(1))+" HEXES":RETURN
RK 3630 IF C(2)=>40THENA=1:C$="VIOLET OCCUPIES"+ST
        R$(C(2))+" HEXES"
DF 3640 RETURN
```

Pig$ for Buck$

Bruce Willis and Dave Zeigler

Both children and adults will have hours of fun with this innovative and amusing game. The object is to catch each of your squealing and elusive pigs so that you can take them all to market. And, as you'll see, keeping your overalls clean—a must as far as meticulous Farmer Brown is concerned—means you'll have to stay clear of the mud puddles. A joystick is required.

As soon as we set eyes on the sprite and sound commands of Commodore's BASIC 7.0, we wanted to see if it was possible to write an arcade-style game almost entirely in BASIC. The result was "Pig$ for Buck$," a nonviolent game that's fun for children and adults. Just one word of warning: The game is extremely habit-forming. And it's somewhat difficult: No one has yet made it past level 11.

Farmer Brown raises pigs, and now he must catch them to take them to market. But the price of pigs is dropping, so he must get there quickly to insure a profit.

Farmer Brown is somewhat vain—he must have clean overalls to wear to market. The pigsty, however, is naturally slippery and full of mud puddles. Stepping in the mud or touching the electric fence will certainly cause him to slip and fall. His clean overalls will get muddy. But if he does fall, all is not lost. His clothesline contains four clean pairs of overalls, and he'll be able to buy up to four more pairs at the market.

Occasionally, a pig will manage to squeeze out of a hole in the fence. Farmer Brown's assistant, positioned outside the fence, will eventually catch the loose pig and return it to the sty. Farmer Brown must be very cautious while leading a pig near the fence because the helpful, but mischievous, farmhand loves to see his employer fall in the mud. He just might push a pig back through the fence right into Farmer Brown's path.

The object of the game is to catch all the pigs in the pen as quickly as possible. The bank-account total is added to your score, and the sooner you sell your pigs, the more profit you make. When all of your overalls have become muddy, the game ends.

Typing It In

The main part of the game is written in BASIC, but also included are one short machine language routine and three sprite-definition files. (There are 24 sprites used in the game.)

First, type in Program 1, which is written entirely in BASIC. Be sure to save a copy when you're through typing. "The Automatic Proofreader," found in Appendix B, will insure that you type it in correctly. "MLX," a machine language entry program (Appendix C), is required to type in the four remaining files. After loading and running MLX, you'll be asked for a starting and ending address. Here are the correct values:

Program 2
Starting address: 1E06
Ending address: 1F05

Programs 3, 4, and 5
Starting address: 0E00
Ending address: 0FFF

Be sure to save each program with its respective filename:

Program 2	**PIG.ML**
Program 3	**PIG.SPR1**
Program 4	**PIG.SPR2**
Program 5	**PIG.SPR3**

Also, be sure to save each of the five programs to the same disk.

Note that several lines in Program 1 are packed with commands. This was done to maintain speed during the main program loop. The decimal points used in many commands are not errors. They can be used in place of a zero and are slightly faster than variables or constants. Programs 1 and 2 make use of the FAST command, which blanks the screen during DATA POKEing and screen setup.

To start play, load Program 1 and type RUN. (The machine language program and sprite files will be loaded automatically.) Most of the screen is taken up by the pigsty. At the bottom of the screen is a clothesline, the bank-account balance, your present score, and the high score. The current level is displayed at the top of the screen. The selling price of a pig appears at the top right when one is sold.

Not an Easy Business

When your joystick is plugged into port 2, chase the pigs and catch them by pressing the fire button as contact is made. But beware: If you collide with a pig while leading another pig out of the pen, you'll trip and fall in the mud. If you see that a collision is unavoidable, press the fire button to release the pig you are leading.

Lead each pig out of the gate in the upper left corner and go catch the next one. The selling price of a pig is set as soon as you lead it out of the gate. The price starts at $100 at the beginning of a round and drops $6 every ten seconds to a low of $60. The selling price is added to your present score, along with bonus points if the pig is caught before the selling price hits $60.

When all the pigs have been captured, you'll automatically purchase as many piglets as you can afford. Piglets—including the food needed to raise them—cost $60 dollars each. You can buy a maximum of seven per round. You'll then be given the option to purchase overalls to replace the ones taken from the clothesline during play. The store has four pairs in stock at the beginning of a game, and they sell for $25 a pair. When the last pair of pants from the clothesline gets muddy, the game ends, even if there are more pants in stock at the store.

When all pigs have been caught and sold, you advance to the next level. As the levels increase, so do the size and number of mud puddles on the screen. At higher levels, the speed of the farmer will vary.

At the end of a game, you have the option to play again. The high score for the current session of play is displayed at the bottom right.

Program 1. Pig$ for Buck$—Main Program

Be sure to save all five programs to the same disk.

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
SG 1Ø  FAST:TRAP97Ø:GRAPHIC1:GRAPHICØ:GOTO58Ø
MP 2Ø  GOSUB3ØØ:FORI=2TOP:MOVSPRI,2ØØ,125:NEXT:TI$=
       "ØØØØØØ"
FE 3Ø  SPRSAVS$(5,F1),F:SYSM1,Ø:MOVSPRF,24,88
ED 4Ø  FORI=2TOP:IFI=P1THEN5Ø:ELSEJ=RND(F)*359+F:C=
       INT(J/18Ø):SPRSAVS$(C,F1),I:SYSM1,I-F:SPRSAV
       S$(C,F3),I:SPRITEI,F,11,Ø:MOVSPRI,J #S
```

```
AM 5Ø  NEXT:SPRITEF,F,15,,,,F:POKE768Ø,F:I=BUMP(F1)
       :I=BUMP(F)
PK 6Ø  FORI=2TOP:IF(BUMP(F1)ANDF)OR((BUMP(F)ANDF)AN
       DP1>.)THEN31Ø:ELSEIFI=P1THEN9Ø
BP 7Ø  IFRSPPOS(I,F)<Y1THENC=F3:GOTO1ØØ:ELSEIFRSPPO
       S(I,F)>Y2THENC=F4:GOTO1ØØ:ELSEIFRSPPOS(I,.)>
       X2THENC=F1:GOTO1ØØ:ELSEIFRSPPOS(I,.)<X1THENC
       =F:GOTO1ØØ
RB 8Ø  GOTO13Ø
EX 9Ø  IFRSPPOS(P1,.)<25ORRSPPOS(P1,.)>344THEN37Ø:E
       LSE13Ø
BK 1ØØ  J=RND(F)*16Ø+SM(C):IFJ>36ØTHENJ=J-36Ø
QA 11Ø  C=INT(J/18Ø):SPRSAVS$(C,F1),I:SYSM1,I-F:SPR
        SAVS$(C,F3),I:MOVSPRI,J #RND(F)*F1+F
MF 12Ø  IFRND(F)>F2THENSOUNDF1,2ØØØ,5,.,1ØØ,1ØØ,F3:
        SOUNDF,13ØØ,12,.,758,19,F1,37Ø:GOTO13Ø
JQ 13Ø  J=JOY(F1)
RQ 14Ø  J1=J:IFJ>.ANDJ<9THEN18Ø:ELSEMOVSPRF,. #.:PO
        KEM2,.:IFP1>.THENMOVSPRP1,. #.
GQ 15Ø  IFRND(F)>F2THENSOUNDF1,2ØØØ,5,.,1ØØ,1ØØ,F3:
        SOUNDF,2ØØØ,1Ø,.,6ØØ,15Ø,F1,22Ø:IFL>1ØTHENF
        S=RND(F)*F3+F
FX 16Ø  IFJ>8THEN21Ø
FQ 17Ø  NEXT:GOTO6Ø
DC 18Ø  IFRSPPOS(F,.)<24ORRSPPOS(F,.)>344THENJ=O(J)
        :IFRSPPOS(F,.)<4ØØANDRSPPOS(F,.)>25THENJ=3
XE 19Ø  IFJ=.THEN6Ø:ELSEMOVSPRF,A(J) #FS:POKEM2,F:I
        FP1>.THENMOVSPRP1,A(J) #FS
DR 2ØØ  SPRSAVS$(J,.),F:SYSM1,.:SPRSAVS$(J,F),F:NEX
        T:GOTO6Ø
DF 21Ø  IFP1>.THEN27Ø:ELSEJ=BUMP(F):IF(JANDF)<>FTHE
        NNEXT:GOTO6Ø
CS 22Ø  GOSUB28Ø:I=F1:C=RSPPOS(F,.):J=RSPPOS(F,F)
EP 23Ø  DOUNTILI>P:IF(ABS(C-RSPPOS(I,.))<2Ø)AND(ABS
        (J-RSPPOS(I,F))<2Ø)THENBEGIN
HD 24Ø  P1=I:SOUND1,7ØØØ,4Ø,.,4ØØ,4ØØ,2,22Ø:SPRSAVS
        $(F,F1),P1:SYSM1,P1-F:SPRSAVS$(F,F3),P1:MOV
        SPRI,C+26,J-5:EXIT:BEND
 AM 25Ø  I=I+F:LOOP
KH 26Ø  I=BUMP(F):GOTO4Ø
HA 27Ø  P1=.:GOTO4Ø
HH 28Ø  POKE768Ø,.:POKE7681,.:FORI=1TO8:MOVSPRI,. #
        .:NEXT:RETURN
CS 29Ø  PRINT"{HOME}E2}":FORI=1TO4:READC,J,M$:CHAR,
        C,J,M$:NEXT:IFL=12THENCHAR,31,6:SYS7937:RET
        URN
QC 3ØØ  J=P*2:X1=5Ø+J:X2=29Ø-J:Y1=65+J:Y2=19Ø-J:RET
        URN
```

```
DH 310 GOSUB280:P1=Ø:POKE2040,56:SPRSAVS$(5,2),F:V
       OL15:SOUNDF1,9000,5,F,2000,900,F:FORI=1TO80
       :NEXT:FORI=6TO8:IFI=6THENSOUNDF,9000,35,F,2
       000,200,Ø
HA 320 SPRSAVS$(I,2),F:FORC=1TO50:NEXTC,I:SOUNDF,3
       00,15,F,200,6:VOL3:SLEEPF
MG 330 IFOL=ØTHEN790:ELSECHAR,2*OL-F,23,"{6} ":CHA
       R,2*OL-F,24,"{6} ":OL=OL-F:GOTO30
EG 340 FORI=1TOOL:CHAR,2*I-1,23,"{BLU}£":CHAR,2*I
       -1,24,"{BLU}<":NEXT:RETURN
PP 350 PRINT"{HOME} {RVS}{BLK}{4 SPACES}PIG$ FOR B
       UCK${8 SPACES}LEVEL:";:PRINTUSING"###";L;:P
       RINT"{3 SPACES}{OFF}";
AM 360 CHAR,10,24:PRINT"{2}";:PRINTUSING"#$######.
       ##";BA;:PRINTTAB(24);:PRINTUSING"######";PT
       ;:PRINTTAB(33);:PRINTUSING"######";HS;:RETU
       RN
BK 370 GOSUB280:PS=100-TI/100:IFPS<60THENPS=60
HE 380 PT=PT+INT(PS):BA=BA+PS:IFP1<>PTHENMOVSPRP1,
       RSPPOS(P,.),RSPPOS(P,F):SPRITEP,.
GK 390 SPRITEP,Ø:CHAR,20,Ø,CHR$(7)+"{RVS}{PUR}PIG
       {SPACE}SOLD FOR "
RA 400 IFPS>60THENPT=PT+(10*L)
XA 410 IFPT>HSTHENHS=PT
DP 420 PRINTUSING"$##.##";PS;:PRINT"{OFF}";:SLEEP1
       :GOSUB350
KX 430 P=P-F:P1=Ø:IFP>1THENPOKE2040,56:SPRSAVS$(2,
       Ø),F:MOVSPRF,24,88:GOTO40
GE 440 L=L+F:IFL<5THEN450:ELSEIF(LANDF)AND(L<12)TH
       ENFS=F1:ELSEFS=F3:GOTO460
KS 450 GOSUB290
HQ 460 P=INT(BA/60):IFP>7THENP=7
AM 470 PRINT"{HOME}"CHR$(27)"Q""{RVS}{RED}YOU HAVE
       ";:PRINTUSING"#$#####.##";BA;:PRINT";YOU MU
       ST BUY"P"{LEFT} PIGS.":BA=BA-P*60:P=P+F:GOS
       UB360:SLEEP3
XM 480 I=INT(BA/25)
SR 490 IFI=ØOROL=4THEN560:ELSEPRINT"{HOME}"CHR$(27
       )"Q""{RVS}{RED}PAIRS OF PANTS IN STORE=
       {LEFT}"OS"{LEFT}; QTY. TO BUY?{OFF}";
PX 500 GETA$:IFA$<"Ø"ORA$>"4"THEN500
FS 510 C=VAL(A$)
GE 520 IFC>ITHENPRINT"{HOME}"CHR$(27)"Q{RVS}YOU CA
       N'T AFFORD"C"{LEFT} PAIRS OF PANTS!";:SLEEP
       1:GOTO490
FE 530 IFOL+C>4THENPRINT"{HOME}"CHR$(27)"Q{RVS}THE
        LINE CAN'T HOLD"OL+C"{LEFT} PAIRS OF PANTS
       !";:SLEEP1:GOTO490
```

```
FX 54Ø  IFOS-C<ØTHENPRINT"{HOME}"CHR$(27)"Q{RVS}STO
        RE DOESN'T HAVE"C"{LEFT} PAIRS OF PANTS!";:
        SLEEP1:GOTO49Ø
MR 55Ø  BA=BA-25*C:OL=OL+C:OS=OS-C:GOSUB34Ø:GOTO56Ø
GH 56Ø  GOSUB35Ø:GOSUB3ØØ:GOTO2Ø
AR 57Ø  FORI=1TO8:SPRSAVI,S$(I,C):NEXT:RETURN
XA 58Ø  COLORØ,14:COLOR4,6:SPRCOLOR11,1Ø:VOL3
MQ 59Ø  DIMF,J,J1,I,P1,C,F1,F2,F3,F4,S$(8,3),M2,M1,
        FS,A(8),Y1,Y2,X1,X2,SM(4),O(8):SM(1)=15:SM(
        2)=195:SM(3)=1Ø5:SM(4)=285:F=1:F1=2:M1=787Ø
        :M2=7681:F2=.5:F3=3:F4=4
RM 6ØØ  FORI=2TO8:A(I)=(I-F)*45:NEXT
PX 61Ø  FORI=2TO4:O(I)=3:O(I+4)=7:NEXT
MD 62Ø  BLOAD"PIG.SPR1":C=Ø:GOSUB57Ø:BLOAD"PIG.SPR2
        ":C=F:GOSUB57Ø:BLOAD"PIG.SPR3":C=2:GOSUB57Ø
XK 63Ø  S$(Ø,F1)=S$(F,F1):S$(Ø,F3)=S$(F1,F1):S$(F,F
        1)=S$(F3,F1):S$(F,F3)=S$(4,F1)
HA 64Ø  BLOAD"PIG.ML":SYS7686:POKE54291,2:POKE54287
        ,2Ø:POKE54292,Ø:POKE54286,Ø
HS 65Ø  IFPEEK(26Ø4)=28THEN7ØØ:ELSEBANK14
KM 66Ø  FORI=ØTO3:READC:POKE25Ø+I,C:NEXT:POKE195,8:
        POKE196,.:SYS79Ø4
PH 67Ø  BANK15:POKE26Ø4,28
KD 68Ø  READC:IFC=-1THEN7ØØ
JM 69Ø  FORI=ØTO7:READJ:POKE12288+C*8+I,J:NEXT:GOTO
        68Ø
JX 7ØØ  S=2:P=3:OL=4:OS=4:FS=2:L=1:PT=Ø:BA=Ø:PRINT"
        {CLR}{DOWN}E2]Q********Q********Q********
        *Q********Q";
PC 71Ø  FORI=1TO3:FORJ=1TO6:IFI=3ANDJ=6THEN72Ø:ELSE
        PRINT"B"TAB(39)"B";
XS 72Ø  NEXT:IFI<>3THENPRINT"Q"TAB(39)"Q";
SX 73Ø  NEXT
SH 74Ø  PRINT"Q********Q********Q********Q******
        **Q";
HP 75Ø  FORX=1TO8:SPRITEX,Ø:NEXT
FA 76Ø  PRINT"{RED}E1Ø @] {RVS}E5]{3 SPACES}BANK
        {6 SPACES}PRESENT{3 SPACES}HIGH{2 SPACES}
        {OFF}";
FK 77Ø  PRINT"{RED}EK]{OFF}"TAB(9)"{RVS}EK]{OFF}
        {RVS}E5]{3 SPACES}ACCT.{5 SPACES}SCORE
        {5 SPACES}SCORE {OFF}";
DK 78Ø  PRINT"{RED}EK]"TAB(9)"{RVS}EK]{OFF}{HOME}
        {3 DOWN}Q{DOWN}{LEFT} {DOWN}{LEFT} {DOWN}
        {LEFT} {DOWN}{LEFT} ";:GOSUB35Ø:RESTORE88Ø:
        GOSUB29Ø:GOSUB29Ø:SLOW:GOSUB34Ø:GOTO2Ø
EA 79Ø  CHAR,5,9,"{RVS}{PUR}{2 SPACES}END OF GAME!
        {2 SPACES}PLAY AGAIN?{2 SPACES}{OFF}"
```

```
BJ 8ØØ POKE2Ø8,Ø:GETKEYA$:IFA$<>"Y"THENPRINT"{CLR}
       {GRN}";:FORC=1TO8:SPRITEC,Ø:NEXT:END
PP 81Ø IFBA>HSTHENHS=BA
MQ 82Ø FAST:GOTO7ØØ
QE 83Ø DATAØ,2Ø8,Ø,48,28,1Ø2,1Ø2,1Ø2,126,126,126,1
       26,126,6Ø,255,255,231,231,231,231,231,Ø
BP 84Ø DATA65,Ø,131,199,255,255,255,255,255,73,128
       ,224,248,252,252,254,254,255
KF 85Ø DATA74,255,127,127,63,63,31,7,1,75,255,254,
       254,252,252,248,224,128
GR 86Ø DATA83,254,252,248,248,248,252,254,254,85,1
       ,7,31,63,63,127,127,255
JQ 87Ø DATA88,255,255,255,255,255,227,193,Ø,9Ø,127
       ,127,63,31,31,31,63,127,-1
XF 88Ø DATA8,6,"UI{DOWN}{2 LEFT}ZS{DOWN}{2 LEFT}JK
       ",21,6,"UI{DOWN}{2 LEFT}JK",33,5,"UAI{DOWN}
       {3 LEFT}J{RVS} {OFF}K",1,19,"I{DOWN}{LEFT}
       {RVS} {OFF}I"
HJ 89Ø DATA7,14,"UI{DOWN}{2 LEFT}JK",17,12,"UI
       {DOWN}{3 LEFT}U{RVS}{2 SPACES}{OFF}{DOWN}
       {3 LEFT}JXK",26,14,"UI{DOWN}{2 LEFT}Z{RVS}
       {SPACE}{OFF}{DOWN}{2 LEFT}JK",34,11,"UI
       {DOWN}{2 LEFT}JK"
CF 9ØØ DATA9,7,"{RVS} {OFF}I{DOWN}{2 LEFT}XK",2Ø,5
       ,"UI{DOWN}{2 LEFT}J{RVS} {OFF}",32,5,"U
       {RVS} {OFF}{DOWN}{2 LEFT}JX",35,11,"AI
       {DOWN}{2 LEFT}{RVS} {OFF}K"
KG 91Ø DATA22,7,"{RVS}{2 SPACES}{OFF}I{DOWN}
       {3 LEFT}JXK",8,13,"UI{DOWN}{2 LEFT}{RVS}
       {OFF}K",19,13,"AI{DOWN}{3 LEFT}{RVS}
       {2 SPACES}{OFF}K",34,12,"Z{RVS}{2 SPACES}
       {DOWN}{3 LEFT}{3 SPACES}{OFF}{DOWN}{3 LEFT}
       JXK"
PF 92Ø DATA1,2,"{RVS} {OFF}K{DOWN}{2 LEFT}K",18,4,
       "UI{DOWN}{2 LEFT}Z{RVS}{2 SPACES}{OFF}
       {DOWN}{3 LEFT}JX{RVS} {OFF}",3Ø,5,"U{RVS}
       {2 SPACES}{DOWN}{3 LEFT}{3 SPACES}{DOWN}
       {3 LEFT}{OFF}JK",27,14,"AI{DOWN}{LEFT}S
       {DOWN}{2 LEFT}{RVS} {OFF}K"
PH 93Ø DATA35,5,"AI{DOWN}{2 LEFT}XK",9,14,"{RVS}
       {DOWN}{2 LEFT}{2 SPACES}{OFF}I{DOWN}
       {3 LEFT}{RVS}{2 SPACES}{OFF}S{DOWN}{3 LEFT}
       JXK",25,13,"U{RVS} {OFF}I{DOWN}{3 LEFT}J
       {RVS}{2 SPACES}{OFF}",36,14,"{RVS} {OFF}I
       {DOWN}{2 LEFT}{RVS}{2 SPACES}{OFF}{DOWN}
       {2 LEFT}JK"
BE 94Ø DATA1Ø,7,"{RVS} {OFF}I{DOWN}{3 LEFT}{RVS}
       {2 SPACES}{OFF}K",36,5,"{RVS} {OFF}I{DOWN}
       {2 LEFT}{RVS}{2 SPACES}{OFF}I{DOWN}{3 LEFT}
```

```
          J{RVS} {OFF}K",16,14,"{RVS}{2 SPACES}{OFF}
          {DOWN}{2 LEFT}JXXK",28,15,"{RVS} {OFF}I
          {DOWN}{2 LEFT}XK"
SR 95Ø DATA11,8,"K",25,6,"UI{DOWN}{3 LEFT}{RVS}
          {3 SPACES}{DOWN}{3 LEFT} {OFF}XK",6,12,"UI
          {DOWN}{2 LEFT}{RVS}{3 SPACES}{OFF}{DOWN}
          {3 LEFT}J{RVS} {OFF}",34,14,"{RVS}
          {2 SPACES}{DOWN}{2 LEFT}{2 SPACES}{OFF}
          {DOWN}{3 LEFT}U{RVS}{2 SPACES}{OFF}X{DOWN}
          {4 LEFT}JK"
RM 96Ø DATA24,2,"J{RVS} {OFF}K",1,19,"{RVS} {OFF}I
          {DOWN}{LEFT}{RVS} {OFF}I",16,15,"{RVS}
          {2 SPACES}{OFF}{DOWN}{3 LEFT}U{RVS} {OFF}S
          {DOWN}{3 LEFT}J{RVS} {OFF}S{DOWN}{2 LEFT}JX
          XI{DOWN}{LEFT}JI{DOWN}{LEFT}JI",33,17,"Z
          {RVS}{2 SPACES}{DOWN}{3 LEFT}{3 SPACES}
          {DOWN}{3 LEFT}{OFF}JXK"
KK 97Ø PRINT"{HOME}"CHR$(27)"Q""{WHT}{RVS}LINE "EL
          "{2 SPACES}"ERR$(ER)"E2}PRESS STOP KEY";
EG 98Ø GOTO98Ø
```

Program 2. Pig$ for Buck$—PIG.ML

See instructions in article and read Appendix C, "MLX," before typing in the following program listings.

Starting address: 1E06
Ending address: 1F05
Filename: PIG.ML

```
1EØ6:A9 ØØ 8D ØØ 1E 8D Ø1 1E 1Ø
1EØE:8D Ø4 1E 8D Ø5 1E 78 A9 E9
1E16:21 8D 14 Ø3 A9 1E 8D 15 EE
1E1E:Ø3 58 6Ø AD ØØ 1E C9 Ø1 E5
1E26:DØ 35 EE Ø4 1E AD Ø4 1E Ø4
1E2E:C9 1Ø DØ 2B A9 ØØ 8D Ø4 8C
1E36:1E AD F9 Ø7 C9 39 DØ 11 82
1E3E:A2 Ø7 18 BD F8 Ø7 69 48 6B
1E46:9D F8 Ø7 CA DØ F4 4C 5D 6D
1E4E:1E A2 Ø7 38 BD F8 Ø7 E9 7Ø
1E56:48 9D F8 Ø7 CA DØ F4 AD DE
1E5E:Ø1 1E C9 Ø1 FØ Ø8 A9 ØØ E6
1E66:8D 12 D4 4C 9B 1E EE Ø5 85
1E6E:1E AD Ø5 1E C9 Ø7 DØ 25 D8
1E76:A9 ØØ 8D Ø5 1E AD F8 Ø7 2A
1E7E:C9 38 DØ ØE 18 69 48 8D 2D
1E86:F8 Ø7 A9 81 8D 12 D4 4C F8
1E8E:9B 1E 38 E9 48 8D F8 Ø7 37
1E96:A9 ØØ 8D 12 D4 4C 65 FA 18
1E9E:ØØ ØE 4Ø ØE 8Ø ØE CØ ØE 13
```

```
1EA6:ØØ ØF 4Ø ØF 8Ø ØF CØ ØF 7Ø
1EAE:ØØ 2Ø 4Ø 2Ø 8Ø 2Ø CØ 2Ø 23
1EB6:ØØ 21 4Ø 21 8Ø 21 CØ 21 8Ø
1EBE:85 C3 18 65 C3 AA BD 9E EA
1EC6:1E 85 FA BD 9F 1E 85 FB 2B
1ECE:BD AE 1E 85 FC BD AF 1E ØE
1ED6:85 FD A9 ØØ 85 C3 A9 4Ø 59
1EDE:85 C4 AØ ØØ A6 C3 FØ ØE 57
1EE6:B1 FA 91 FC C8 DØ F9 E6 21
1EEE:FB E6 FD CA DØ F2 A6 C4 B3
1EF6:FØ Ø8 B1 FA 91 FC C8 CA 7Ø
1EFE:DØ F8 6Ø ØØ ØØ ØØ ØØ ØØ ED
```

Program 3. Pig$ for Buck$—PIG.SPR1

Starting address: 0E00
Ending address: 0FFF
Filename: PIG.SPR1

```
ØEØØ:ØØ FF ØØ Ø3 FF CØ ØØ FF 4F
ØEØ8:ØØ ØØ 7D ØØ ØF 96 FØ 3F C7
ØE1Ø:BE FC FF BE FF F3 BE C5 C9
ØE18:F2 AA 85 F2 AA 8Ø 52 AA DE
ØE2Ø:8Ø 52 AA 8Ø Ø2 AA 8Ø Ø2 2C
ØE28:AA 8Ø Ø2 82 8Ø Ø2 82 8Ø B3
ØE3Ø:ØF C2 8Ø ØØ Ø2 8Ø ØØ Ø2 A8
ØE38:8Ø ØØ Ø3 FØ ØØ Ø3 FC ØØ ØA
ØE4Ø:Ø3 FF ØØ ØF FF CØ Ø3 D7 AF
ØE48:ØØ Ø3 55 ØØ Ø1 55 ØØ ØØ 2D
ØE5Ø:BC ØØ ØØ BC ØØ Ø3 BC ØØ 1C
ØE58:ØF AF ØØ ØF AF CØ Ø5 AB ØF
ØE6Ø:5Ø Ø5 A8 5Ø ØØ A8 ØØ ØØ A2
ØE68:AA ØØ ØØ AA ØØ Ø2 AA ØØ E1
ØE7Ø:Ø2 8A ØØ Ø2 8A ØØ Ø2 8F 38
ØE78:ØØ Ø3 CF CØ Ø3 FØ ØØ ØØ 37
ØE8Ø:Ø3 FF ØØ ØF FF CØ Ø3 D7 EF
ØE88:ØØ Ø3 55 ØØ Ø1 55 ØØ ØØ 6D
ØE9Ø:BC ØØ ØØ BC ØØ Ø3 BC ØØ 5C
ØE98:ØF AF ØØ ØF AF CØ Ø5 AB 4F
ØEAØ:5Ø Ø5 A8 5Ø ØØ A8 ØØ ØØ E2
ØEA8:AA ØØ ØØ AA ØØ Ø2 AA ØØ 22
ØEBØ:Ø2 8A ØØ Ø2 8A ØØ Ø2 8F 78
ØEB8:ØØ Ø3 CF CØ Ø3 FØ ØØ ØØ 77
ØECØ:Ø3 FF ØØ ØF FF CØ Ø3 D7 3Ø
ØEC8:ØØ Ø3 55 ØØ Ø1 55 ØØ ØØ AD
ØEDØ:BC ØØ ØØ BC ØØ Ø3 BC ØØ 9C
ØED8:ØF AF ØØ ØF AF CØ Ø5 AB 8F
ØEEØ:5Ø Ø5 A8 5Ø ØØ A8 ØØ ØØ 23
ØEE8:AA ØØ ØØ AA ØØ Ø2 AA ØØ 62
```

```
ØEFØ:Ø2 8A ØØ Ø2 8A ØØ Ø2 8F B8
ØEF8:ØØ Ø3 CF CØ Ø3 FØ ØØ ØØ B7
ØFØØ:ØØ FF ØØ Ø3 FF CØ ØØ D7 29
ØFØ8:ØØ ØØ 7D ØØ ØF 55 FØ 3F C4
ØF1Ø:BE FC FF BE FF F3 BE C5 CB
ØF18:F2 AA 85 F2 AA 8Ø 52 AA EØ
ØF2Ø:8Ø 52 AA 8Ø Ø2 AA 8Ø Ø2 2E
ØF28:AA 8Ø Ø2 82 8Ø Ø2 82 8Ø B5
ØF3Ø:ØF C2 8Ø ØØ Ø2 8Ø ØØ Ø2 AA
ØF38:8Ø ØØ Ø3 FØ ØØ Ø3 FC ØØ ØC
ØF4Ø:Ø3 FF ØØ ØF FF CØ Ø3 5F 39
ØF48:ØØ Ø1 57 ØØ Ø1 55 ØØ ØØ EE
ØF5Ø:F8 ØØ ØØ F8 ØØ 5F FB ØØ EF
ØF58:5F EB CØ ØØ AB FØ ØØ A8 Ø3
ØF6Ø:5Ø ØØ A8 5Ø ØØ A8 ØØ Ø2 65
ØF68:A8 ØØ Ø2 A8 ØØ ØA AA 8Ø A3
ØF7Ø:ØA Ø2 AØ 3A ØØ AØ 3F ØØ CC
ØF78:28 ØØ ØØ 3C ØØ ØØ FØ ØØ 5Ø
ØF8Ø:Ø3 FF ØØ ØF FF CØ Ø3 5F 79
ØF88:ØØ Ø1 57 ØØ Ø1 55 ØØ ØØ 2F
ØF9Ø:F8 ØØ ØØ F8 ØØ 5F FB ØØ 3Ø
ØF98:5F EB CØ ØØ AB FØ ØØ A8 43
ØFAØ:5Ø ØØ A8 5Ø ØØ A8 ØØ Ø2 A5
ØFA8:A8 ØØ Ø2 A8 ØØ ØA AA 8Ø E3
ØFBØ:ØA Ø2 AØ 3A ØØ AØ 3F ØØ ØD
ØFB8:28 ØØ ØØ 3C ØØ ØØ FØ ØØ 9Ø
ØFCØ:Ø3 FF ØØ ØF FF CØ Ø3 5F B9
ØFC8:ØØ Ø1 57 ØØ Ø1 55 ØØ ØØ 6F
ØFDØ:F8 ØØ ØØ F8 ØØ 5F FB ØØ 7Ø
ØFD8:5F EB CØ ØØ AB FØ ØØ A8 83
ØFEØ:5Ø ØØ A8 5Ø ØØ A8 ØØ Ø2 E5
ØFE8:A8 ØØ Ø2 A8 ØØ ØA AA 8Ø 24
ØFFØ:ØA Ø2 AØ 3A ØØ AØ 3F ØØ 4D
ØFF8:28 ØØ ØØ 3C ØØ ØØ FØ ØØ DØ
```

Program 4. Pig$ for Buck$—PIG.SPR2

Starting address: 0E00
Ending address: 0FFF
Filename: PIG.SPR2

```
ØEØØ:ØØ FF ØØ Ø3 FF CØ ØØ FF 4F
ØEØ8:ØØ ØØ 7D ØØ ØF 96 FØ 3F C7
ØE1Ø:BE FC FF BE FF 53 BE CF 51
ØE18:52 AA 8F Ø2 AA 8F Ø2 AA 5C
ØE2Ø:85 Ø2 AA 85 Ø2 AA 8Ø Ø2 EA
ØE28:AA 8Ø Ø2 82 8Ø Ø2 82 8Ø B3
ØE3Ø:Ø2 83 FØ Ø2 8Ø ØØ Ø2 8Ø F4
```

```
ØE38:ØØ ØF CØ ØØ 3F CØ ØØ ØØ 2D
ØE4Ø:Ø3 FF ØØ ØF FF CØ Ø3 D7 AF
ØE48:ØØ Ø3 55 ØØ Ø1 55 ØØ ØØ 2D
ØE5Ø:BC ØØ ØØ BC ØØ Ø3 BF D4 F6
ØE58:ØF AF D4 3F A8 ØØ 14 A8 8C
ØE6Ø:ØØ 14 A8 ØØ ØØ AA ØØ ØØ 41
ØE68:AA ØØ ØØ AA 8Ø Ø2 A2 8Ø 56
ØE7Ø:ØA 82 8Ø ØA Ø2 BØ 28 Ø3 Ø9
ØE78:FØ 3C ØØ ØØ ØF ØØ ØØ ØØ 94
ØE8Ø:Ø3 FF ØØ ØF FF CØ Ø3 D7 EF
ØE88:ØØ Ø3 55 ØØ Ø1 55 ØØ ØØ 6D
ØE9Ø:BC ØØ ØØ BC ØØ Ø3 BF D4 37
ØE98:ØF AF D4 3F A8 ØØ 14 A8 CC
ØEAØ:ØØ 14 A8 ØØ ØØ AA ØØ ØØ 81
ØEA8:AA ØØ ØØ AA 8Ø Ø2 A2 8Ø 96
ØEBØ:ØA 82 8Ø ØA Ø2 BØ 28 Ø3 49
ØEB8:FØ 3C ØØ ØØ ØF ØØ ØØ ØØ D4
ØECØ:Ø3 FF ØØ ØF FF CØ Ø3 D7 3Ø
ØEC8:ØØ Ø3 55 ØØ Ø1 55 ØØ ØØ AD
ØEDØ:BC ØØ ØØ BC ØØ Ø3 BF D4 77
ØED8:ØF AF D4 3F A8 ØØ 14 A8 ØD
ØEEØ:ØØ 14 A8 ØØ ØØ AA ØØ ØØ C1
ØEE8:AA ØØ ØØ AA 8Ø Ø2 A2 8Ø D6
ØEFØ:ØA 82 8Ø ØA Ø2 BØ 28 Ø3 89
ØEF8:FØ 3C ØØ ØØ ØF ØØ ØØ ØØ 15
ØFØØ:ØØ FF ØØ Ø3 FF CØ ØØ D7 29
ØFØ8:ØØ ØØ 7D ØØ ØF 55 FØ 3F C4
ØF1Ø:BE FC FF BE FF 53 BE CF 53
ØF18:52 AA 8F Ø2 AA 8F Ø2 AA 5E
ØF2Ø:85 Ø2 AA 85 Ø2 AA 8Ø Ø2 EC
ØF28:AA 8Ø Ø2 82 8Ø Ø2 82 8Ø B5
ØF3Ø:Ø2 83 FØ Ø2 8Ø ØØ Ø2 8Ø F6
ØF38:ØØ ØF CØ ØØ 3F CØ ØØ ØØ 2F
ØF4Ø:Ø3 FF ØØ ØF FF CØ Ø3 5F 39
ØF48:ØØ Ø1 57 ØØ Ø1 55 ØØ ØØ EE
ØF5Ø:F8 ØØ ØØ F8 ØØ Ø3 FB ØØ 7E
ØF58:Ø3 EB CØ ØF EB CØ 17 A9 36
ØF6Ø:4Ø 14 A9 4Ø ØØ A8 ØØ Ø2 81
ØF68:A8 ØØ Ø2 A8 ØØ Ø2 AA ØØ Ø3
ØF7Ø:Ø2 8A ØØ Ø2 8A ØØ Ø3 CA 77
ØF78:ØØ ØF CF ØØ ØØ 3F ØØ ØØ 51
ØF8Ø:Ø3 FF ØØ ØF FF CØ Ø3 5F 79
ØF88:ØØ Ø1 57 ØØ Ø1 55 ØØ ØØ 2F
ØF9Ø:F8 ØØ ØØ F8 ØØ Ø3 FB ØØ BE
ØF98:Ø3 EB CØ ØF EB CØ 17 A9 76
ØFAØ:4Ø 14 A9 4Ø ØØ A8 ØØ Ø2 C1
ØFA8:A8 ØØ Ø2 A8 ØØ Ø2 AA ØØ 43
ØFBØ:Ø2 8A ØØ Ø2 8A ØØ Ø3 CA B7
ØFB8:ØØ ØF CF ØØ ØØ 3F ØØ ØØ 91
```

```
ØFCØ:Ø3 FF ØØ ØF FF CØ Ø3 5F B9
ØFC8:ØØ Ø1 57 ØØ Ø1 55 ØØ ØØ 6F
ØFDØ:F8 ØØ ØØ F8 ØØ Ø3 FB ØØ FE
ØFD8:Ø3 EB CØ ØF EB CØ 17 A9 B6
ØFEØ:4Ø 14 A9 4Ø ØØ A8 ØØ Ø2 Ø2
ØFE8:A8 ØØ Ø2 A8 ØØ Ø2 AA ØØ 83
ØFFØ:Ø2 8A ØØ Ø2 8A ØØ Ø3 CA F7
ØFF8:ØØ ØF CF ØØ ØØ 3F ØØ ØØ D1
```

Program 5. Pig$ for Buck$—PIG.SPR3

Starting address: 0E00
Ending address: 0FFF
Filename: PIG.SPR3

```
ØEØØ:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ 1C
ØEØ8:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ 24
ØE1Ø:ØØ ØØ ØØ ØØ ØØ ØØ Ø1 ØØ 2E
ØE18:ØØ Ø2 ØØ ØØ Ø3 CØ ØØ Ø3 D2
ØE2Ø:4Ø Ø8 Ø3 FØ Ø4 Ø7 FØ Ø5 FØ
ØE28:FF CØ Ø3 FF ØØ Ø3 FF ØØ EØ
ØE3Ø:Ø3 FF ØØ Ø3 FE ØØ Ø1 FE F6
ØE38:ØØ Ø1 FA ØØ Ø1 Ø2 ØØ ØØ Ø4
ØE4Ø:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ 5C
ØE48:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ 64
ØE5Ø:Ø1 ØØ ØØ Ø1 ØØ ØØ Ø1 EØ DF
ØE58:ØØ Ø1 AØ Ø4 Ø1 F8 Ø4 Ø3 FF
ØE6Ø:F8 Ø5 FF CØ Ø3 FF ØØ Ø3 61
ØE68:FF ØØ Ø3 FE ØØ Ø3 FE ØØ DE
ØE7Ø:Ø1 FE ØØ Ø3 F2 ØØ Ø2 ØC A4
ØE78:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ 94
ØE8Ø:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ 9C
ØE88:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ A4
ØE9Ø:ØØ ØØ ØØ ØØ ØØ Ø1 ØØ ØØ BØ
ØE98:ØØ 8Ø ØØ Ø7 8Ø ØØ Ø5 8Ø D3
ØEAØ:ØØ 1F 8Ø 1Ø 1F CØ 2Ø Ø3 D4
ØEA8:FF AØ ØØ FF CØ ØØ FF CØ B3
ØEBØ:ØØ FF CØ ØØ 7F CØ ØØ 7F 63
ØEB8:8Ø ØØ 5F 8Ø ØØ 4Ø 8Ø ØØ ØB
ØECØ:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ DC
ØEC8:ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ E4
ØEDØ:8Ø ØØ ØØ 8Ø ØØ Ø7 8Ø ØØ 52
ØED8:Ø5 8Ø ØØ 1F 8Ø 2Ø 1F CØ ØD
ØEEØ:2Ø Ø3 FF AØ ØØ FF CØ ØØ 59
ØEE8:FF CØ ØØ FF CØ ØØ 7F CØ FA
ØEFØ:ØØ 7F 8Ø ØØ 4F 8Ø ØØ 3Ø A9
ØEF8:4Ø ØØ ØØ ØØ ØØ ØØ ØØ ØØ 35
ØFØØ:ØØ FF ØØ Ø3 FF CØ ØØ D7 29
```

```
ØFØ8:ØØ ØØ 7D ØØ ØF 55 FØ 3F C4
ØF1Ø:BE FC FF BE FF F3 BE CF D5
ØF18:F2 AA 8F F2 AA 8F 52 AA 5E
ØF2Ø:85 52 AA 85 Ø2 AA 8Ø Ø2 Ø1
ØF28:AA 8Ø Ø2 82 8Ø Ø2 82 8Ø B5
ØF3Ø:Ø2 82 8Ø Ø2 82 8Ø Ø2 82 BC
ØF38:8Ø ØF C3 FØ 3F C3 FC ØØ E4
ØF4Ø:ØØ FF ØØ Ø3 FF CØ ØØ D7 69
ØF48:ØØ ØØ 7D ØØ ØF 55 FØ 3F Ø5
ØF5Ø:BE FC 5F BE F5 53 BE C5 25
ØF58:Ø2 AA 8Ø Ø2 AA 8Ø Ø2 AA 58
ØF6Ø:8Ø Ø2 AA 8Ø Ø2 AA 8Ø Ø2 5A
ØF68:AA 8Ø Ø2 82 8Ø Ø2 82 8Ø F5
ØF7Ø:ØA ØØ AØ ØA ØØ AØ ØF ØØ E8
ØF78:FØ 3F ØØ FC ØØ ØØ ØØ ØØ AE
ØF8Ø:ØØ ØØ ØØ ØØ FF ØØ 53 FF 45
ØF88:C5 5Ø D7 Ø5 FØ 7D ØF FF 84
ØF9Ø:55 FF 3F BE FC ØF BE FØ BF
ØF98:Ø2 AA 8Ø Ø2 AA 8Ø Ø2 AA 98
ØFAØ:8Ø Ø2 AA 8Ø Ø2 AA 8Ø Ø2 9A
ØFA8:AA 8Ø Ø2 AA 8Ø ØA 82 AØ F8
ØFBØ:3A ØØ AC 3E ØØ BC 3E ØØ D4
ØFB8:BC ØØ ØØ ØØ ØØ ØØ ØØ ØØ 35
ØFCØ:ØØ ØØ ØØ 5Ø ØØ Ø5 5Ø FF 98
ØFC8:Ø5 F3 FF CF FØ D7 ØF FC 65
ØFDØ:7D 3F 3F 55 FC 3F BE FC 1A
ØFD8:ØF BE FØ Ø2 AA 83 C2 AA FF
ØFEØ:8Ø ØE AB 8C Ø2 BA 8Ø 3E 3B
ØFE8:AA BC FE EA BF FE AA BF 29
ØFFØ:FE AB BF EA EA AB FA BE DA
ØFF8:AF FA FF AF 3B FF EC ØØ 5C
```

Puzzle Grid

Philip Schielke

Solving these 25-piece puzzles is not as easy as you might think. Once you've tried your hand with the five built-in puzzles, you might want to experiment with creating your own. Runs only in 128 mode.

The object of "Puzzle Grid" is to unscramble a 25-piece puzzle in the time allotted. You'll have about 16 minutes. Sounds like a job for a five-year-old? It may be harder than you think. For one thing, the puzzle pieces are square, not irregularly shaped like jigsaw pieces. You'll have to rely on your memory of the picture alone and not how the shapes of the pieces fit together.

Also, you lose 50 seconds for each piece that you place incorrectly. You can't choose the pieces by trial and error, or you'll quickly run out of time. And, finally, the picture of the puzzle you're to solve is displayed for only a short time before it disappears. You must try to remember the picture as you're figuring out the puzzle.

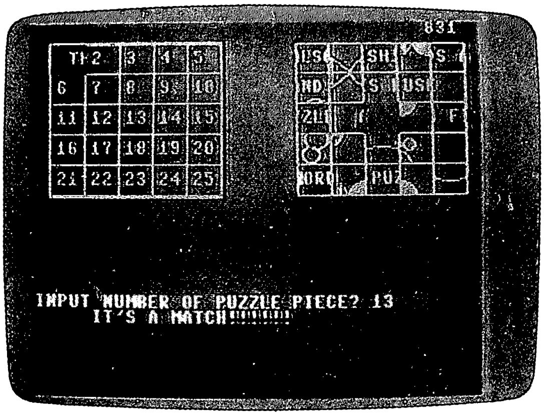
The computer randomly chooses from five puzzles that are built into the program. The same one will not be selected twice in a row.

How to Play

After an introductory screen, a box will appear in the upper left corner. In this box, a picture is drawn. You'll have a few seconds to study it. Then, the picture will be scrambled, and your job is to piece it together again. Try to memorize it as the computer scrambles it (the message *Scrambling* will appear on the screen). The mixed-up puzzle pieces will then appear inside a square grid on the right-hand side of your screen.

When the picture of the completed puzzle is erased from the left-hand side of the screen, it is replaced by an empty grid of small numbered blocks. This is the puzzle completion area where you will piece the puzzle together. You're asked which block (1–25) of the empty puzzle grid you want to fill.

Then you're asked which puzzle piece from the right side of the screen fits that position. The numbers of the scrambled pieces also range from 1 to 25, in the same pattern as the blocks in the completion grid on the left. To make the game more difficult, though, the scrambled pieces on the right-hand side are *not* numbered. Their position numbers, however, do match the numbered positions of the blocks in the completion area. If you choose a piece correctly, it will vanish from the scrambled puzzle area and appear in its correct spot in the completion grid. If you choose a piece incorrectly, you lose 50 points (50 seconds) and must try again. Your score, or time remaining, is constantly updated and displayed in the upper right corner.



```
INPUT NUMBER OF PUZZLE PIECE? 13
      IT'S A MATCH!!!!!!!
```

Solving "Puzzle Grid" will give your memory a workout.

If you complete the puzzle within the time limit, your score will be displayed (your score is the amount of time remaining in seconds), and you'll be asked if you want to play again. If time runs out, you'll be told the number of pieces you chose correctly and will also have a chance at another game.

Adding Your Own Puzzles

Puzzle Grid has five ready-made puzzles, and you can add up to five more of your own. But, first, you need to see how the program fits together. Line 60 selects the random puzzle. In the first part of the line is the statement

G=INT(RND(1)*5)+1

The 5 indicates the number of puzzles the computer can choose from. Be sure to add 1 to that number for each puzzle you add.

The program is designed to contain one puzzle every 50 lines, beginning at line 410. The last of the built-in puzzles starts at line 610. This means that you can start your first puzzle at line 660, and you'll have 50 lines in which to draw it. Your next puzzle will start at line 710, and so forth. The last line of your puzzle must be a RETURN statement, since each puzzle is a separate subroutine. Remember, the program is equipped to handle a maximum of ten puzzles, so line 909 is the last line available for programming your creations.

Each of your puzzles must fit into a box which has these parameters:

Corner	Location
Upper left	12,12
Upper right	12,132
Lower left	92,12
Lower right	92,132

SSHAPE and GSHAPE

The five puzzles in the program are created with hi-res graphics statements like CIRCLE, BOX, DRAW, and CHAR. The heart of Puzzle Grid, however, is the use of the graphics statements SSHAPE and GSHAPE. (As a matter of fact, the idea for the program sprang from reading about these two statements.)

SSHAPE takes a specified area of the hi-res graphics screen and stores it in a string variable. Then GSHAPE places that area onto another part of the screen. You can use SSHAPE and GSHAPE like a stamp to copy one part of the screen and put it anywhere else on the screen as many times as you wish.

The syntax for SSHAPE is

SSHAPE A$,*X1,Y1,X2,Y2*

In this statement, A$ is the string where the shape is stored; *X1,Y1* are the *x* and *y* coordinates that form the upper left corner of the area you want to store; and the *X2,Y2* pair designates the lower right corner.

GSHAPE's syntax is

GSHAPE A$,*X,Y,mode*

Again, A$ is the string where the shape is stored; *X* and *Y* are the screen coordinates of the point where you want the shape to begin; and *mode* is a number from 0 through 4. When mode 0 is specified, the shape is left as it is. If 1 is chosen, the shape will be inverted, or flipped upside-down. Options 2–4 allow you to OR, AND, or XOR the shape (discussion of these options is outside the scope of this article).

After you've typed in the program and solved the puzzles a few times, you might enjoy designing your own. Here is the method I followed. First, a puzzle was drawn inside the box area discussed above. Then, with SSHAPE, it was put into array PU$( ). Each puzzle piece is a 24 × 16–pixel square. A simple scramble routine was used to scramble the pieces. And finally, GSHAPE was used to place each piece in a different position on the right side of the screen.

Puzzle Grid

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
PJ 10 GOSUB1160
DC 20 GRAPHIC2,1,18:WIDTH1
DC 30 SSHAPECL$,0,0,24,16
MR 40 F=25:DIMPU$(F),U$(F),SC(F),XC(F),YC(F)
AF 50 GOSUB1020:GOSUB950
HG 60 G=INT(RND(1)*5)+1:IFG=PEEK(128)THEN60
RQ 70 POKE128,G
XP 80 ON G GOSUB 410,460,510,560,610,660,710,760,8
      10,860
XS 90 FORT=1TO5:FORR=1TO5:W=(R-1)*5+T:SSHAPEPU$(W)
      ,XC(W),YC(W),T*24+12,R*16+12:NEXT:NEXT
QQ 100 FORT=1TO4:SPRITET,0:NEXT
XQ 110 REM SCRAMBLE
KS 120 PRINT"{CLR}{20 DOWN}{5 SPACES}SCRAMBLING
AR 130 FORT=1TO25:U$(T)="N":NEXT
AX 140 FORT=1TO25
AK 150 G=INT(RND(1)*25)+1:IFU$(G)="Y"THEN150
CC 160 U$(G)="Y":SC(T)=G:NEXT
HC 170 REM PUT SCRAMBLED PUZZLE ON SCREEN
GQ 180 COLOR1,8
RE 190 FORT=1TO5:FORR=1TO5:W=(T-1)*5+R:GSHAPEPU$(S
       C(W)),XC(W)+176,YC(W)
```

```
AP 200 SOUND1,SC(W)*1000+3000,2,,,,1:NEXT:NEXT
FP 210 FL=1:GOSUB980:FL=0:COLOR1,16:GOSUB980
EM 220 SLEEP5
BQ 230 BOX0,12,12,132,92,,1:FL=0:GOSUB910:GOSUB980
       :GOSUB950
KR 240 SC=1000:CO=VAL(TI$)
RK 250 PRINT"{CLR}":FORPL=1TO18:PRINT:NEXT
QC 260 INPUT"INPUT NUMBER OF SQUARE TO FILL";FS
XR 270 TN=VAL(TI$)-CO:SC=SC-TN:CO=CO+TN:IFSC<=0THE
       N1070
FE 280 SC$=STR$(SC):CHAR1,34,0,SC$
QM 290 IFFS<1ORFS>25THEN250
MQ 300 PRINT"{CLR}":FORPL=1TO18:PRINT:NEXT
SD 310 INPUT"INPUT NUMBER OF PUZZLE PIECE";PP
GS 320 TN=VAL(TI$)-CO:SC=SC-TN:CO=CO+TN:IFSC<=0THE
       N1070
CD 330 SC$=STR$(SC):CHAR1,34,0,SC$
XA 340 IFPP<1ORPP>25THEN310
QX 350 IFSC(PP)<>FSTHENGOSUB1050
GQ 360 IFSC(PP)<>FSTHEN250
SX 370 PRINT"{5 SPACES}IT'S A MATCH!!!!!!":PC=PC+1
       :GSHAPECL$,XC(PP)+176,YC(PP)
FD 380 COLOR1,8:FL=1:GOSUB980:COLOR1,16
QP 390 GSHAPEPU$(FS),XC(FS),YC(FS):IFPC=25THEN1270
HQ 400 GOTO250
PF 410 REM CIRCLE
MS 420 CIRCLE1,72,54,7,5:CIRCLE1,72,54,11,8:CIRCLE
       1,72,54,22,16:CIRCLE1,72,54,27,20
FF 430 CIRCLE1,73,51,35,25:CIRCLE1,70,55,54,36:DRA
       W 1,15,15 TO 130,15 TO 125,87
BB 440 RETURN
HC 460 REM TRIANGLE AND CIRCLES
MD 470 DRAW1,41,81 TO 123,81 TO 79,14 TO 41,81
CQ 480 CIRCLE1,36,36,20,18:PAINT1,36,36:CIRCLE1,84
       ,60,10,10:CIRCLE1,24,76,10,3
KQ 490 DRAW1,72,86 TO 79,86:WIDTH2:DRAW 1,120,22 T
       O 120,54:WIDTH1:RETURN
XQ 510 CHAR1,4,2,"THIS PUZZLE":CHAR1,5,3,"USES ONL
       Y":CHAR1,3,4,"WORDS AND NO":CHAR1,4,6,"PICT
       URES OR":CHAR1,4,7,"SYMBOLS{2 SPACES}OR"
EH 520 CHAR1,5,9,"DRAWINGS":CHAR1,3,10,"IN
       {8 SPACES}IT"
JH 530 RETURN
XF 560 CHAR1,3,2,"THIS PUZZLE":CHAR1,2,4,"USES WOR
       DS AND":CHAR1,4,6,"ALSO FISH!"
XS 570 CIRCLE 1,96,76,20,9:DRAW1,76,76 TO 62,68 TO
        62,84 TO 76,76:DRAW1,116,77 TO 108,77
XC 580 WIDTH2:DRAW 1,109,73 TO 108,72:WIDTH1
```

```
RQ 590 CIRCLE1,115,66,4,3:CIRCLE1,118,55,6,4:CIRCL
       E1,122,47,8,5
XA 600 CIRCLE1,36,76,12,7:DRAW1,24,76 TO 18,70 TO
       {SPACE}18,82 TO 24,76:PAINT1,36,76:RETURN
DD 610 CHAR1,2,2,"CITY AT NIGHT":CIRCLE 1,57,37,11
       ,9,125,330:CIRCLE1,62,33,11,9,160,285:PAINT
       1,47,37
SP 620 DRAW1,17,27 TO 19,32 TO 14,29 TO 20,29 TO 1
       5,32 TO 17,27:SSHAPEST$,14,27,20,32:RESTORE
       650
AB 630 FORT=1TO9:READXX,YY:GSHAPEST$,XX,YY:NEXT
AH 640 FORT=1TO7:READXX,YY,XZ,YZ:BOX1,XX,YY,XZ,YZ:
       NEXT
DP 645 FORT=54TO86STEP2:DRAW1,13,T TO 35,T:NEXT:PA
       INT1,60,90:FORT=51TO90STEP3:DRAW1,99,T TO 1
       16,T:NEXT:RETURN
DC 650 DATA20,39,28,34,58,32,87,37,95,31,81,22,112
       ,36,122,30,118,42,13,52,35,92,35,64,51,92,5
       1,79,72,92,72,60,87,92,87,84,99,92,99,50,11
       6,92,116,85,132,92
XF 910 REM NUMBERS
DM 920 GRAPHIC2,0,18:IFFL=1THENOV=22:ELSE OV=0
AB 930 FORT=1TO5:FORR=1TO5:X=((T-1)*5)+R:X$=STR$(X
       ):CHAR1,(R)*3-2+OV,(T)*2,X$:NEXT:NEXT
GA 940 RETURN
GM 950 REM BOX
FS 960 BOX1,11,11,133,93
MC 970 RETURN
MA 980 REM LITTLE BOXES
AD 990 IFFL=1THENOH=176:ELSEOH=0
RE 1000 FORT=1TO5:FORR=1TO5:X=(R-1)*5+T:BOX1,XC(X)
        +OH,YC(X),T*24+12+OH,R*16+12:NEXT:NEXT
DS 1010 RETURN
XM 1020 REM SET COORDINATES ROUTINE
PB 1030 Q=12:FORT=1TO5:QQ=12:FORR=1TO5:W=(R-1)*5+T
        :XC(W)=Q:YC(W)=QQ:QQ=R*16+12:NEXT:Q=T*24+1
        2:NEXT
DA 1040 RETURN
PQ 1050 PRINT"{5 SPACES}{RVS}INCORRECT{OFF} YOU LO
        SE 50 POINTS"
ME 1060 SOUND1,30000,60,0,30000,0,1:FORX=1TO60:NEX
        T:SOUND1,1500,60,0,1500,0,1:FORX=1TO60:NEX
        T:SC=SC-50:RETURN
SP 1070 REM RAN OUT OF POINTS
AX 1080 GRAPHIC1:FORX=1TO25
EH 1090 G=INT(RND(1)*25)+1:XX=INT(RND(1)*270):YY=I
        NT(RND(1)*180)
PA 1100 GSHAPEPU$(G),XX,YY:NEXT:FORX=1TO500:NEXT:G
        RAPHIC0
```

```
PR 1110 PRINT"{CLR}{2 DOWN}{5 SPACES}YOU HAVE RUN
        {SPACE}OUT OF TIME":PRINT"{DOWN}{5 SPACES}
        YOU GOT";PC;"PIECES CORRECT"
FP 1120 SOUND1,30000,380,1,0,50,1:SOUND2,30000,380
        ,2,29000,50,1:FORX=1TO2500:NEXT
CE 1130 INPUT"WOULD YOU LIKE TO PLAY AGAIN (Y/N)";
        YN$
DX 1140 IFYN$="Y"THENRUN
JB 1150 PRINT"{DOWN}THANKS FOR PLAYING!!!":END
DF 1160 COLOR0,3:COLOR4,7:COLOR1,16:PRINT"{WHT}":R
        ESTORE 1260
CS 1170 PRINT"{CLR}{2 DOWN}"SPC(15)"WELCOME TO":SL
        EEP1:PRINT"{2 DOWN}"SPC(14)"{RVS}*PUZZLE G
        RID*{OFF}":SLEEP1
BR 1180 SLEEP1
DH 1190 GRAPHIC1,1:CHAR1,1,0,"PUZZLE GRID"
XS 1200 FORT=1TO4:SSHAPESH$(T),4+((T-1)*23),0,4+(T
        *23),20
XD 1210 SPRSAVSH$(T),T:SPRITET,1,1,1,1,1,0:MOVSPRT
        ,T*47#4:NEXT:GRAPHIC0
FH 1220 SLEEP3:FORT=1TO4:MOVSPRT,0#0:READF:MOVSPRT
        ,F,120:SOUND1,4870,25:SLEEP2:NEXT
XR 1230 TEMPO11:PLAY"V1O4IC"
QM 1240 PLAY"V1T8O4CV2T8O2FV1O4CV2O2FV1O3AV2O2AV1O
        3AV2O2AV1O3FV2O2F"
ES 1250 RETURN
FR 1260 DATA 92,137,183,229
QP 1270 GRAPHIC0:SCNCLR:RESTORE1260
HD 1280 PRINT"{4 DOWN}"SPC(14)"{YEL}YOU HAVE WON"
BX 1290 FORT=1TO4:SPRITET,1:MOVSPRT,T*72#6:NEXT:SL
        EEP3
XM 1300 FORT=1TO4:MOVSPRT,0#0:READF:MOVSPRT,F,120:
        SOUND1,4870,25:SLEEP1:NEXT
BF 1310 TEMPO25:PLAY"V1O4QCEGO5HCO4QGO5WC"
JH 1320 PRINT"{8 DOWN}"SPC(11)"WITH A SCORE OF";SC
SX 1330 PRINT"{2 DOWN}":GOTO1130
```

Chapter 2

Applications

Database 128

Allen Vaughan

For a useful, multifunctional program that will help you manage any information you need to store, retrieve, and manipulate, try "Database 128." It's quick and easy to use.

Databases are useful tools for many applications. You'll find that "Database 128" includes options for sorting alphabetically in ascending order, searching individual fields, deleting records, and editing records. It has a housekeeping option that allows you to view the disk directory for entries, scratch files on disks, and format new disks. When you save a file, the save routine checks to see whether a file of that name already exists. If it does exist, the old file is automatically renamed so that you can refer back to it later or use it as a backup in case something goes wrong with the save. There is also a disk error-checking routine that keeps you informed of the disk status.

Written entirely in BASIC 7.0, and using the FAST command, the program runs surprisingly fast. And because the 128 banks its memory, there are over 64,000 bits of memory reserved for the data, allowing for a most impressive data collection.

The Menus

Database 128 actually has three menus: the main menu appears when you first run the program, the housekeeping menu displays the disk options, and the printer menu allows you to choose from two print formats (either mailing labels or a list). To make a choice from either menu, or from the VIEW option, just press the key that corresponds to your choice.

Database Options

When you first use Database 128, you will need to create a new file. Do so with the **CREATE** option. The computer will ask for the number of fields (individual units of information within a file). Enter the number of fields that will be required for the file you're creating. An address file, for example, may have four fields: One for the person's name; one for the street address; one for the city, state, and zip code; and one for the phone number.

The screen will then clear and prompt you to enter the title for each field and its maximum length. You must be sure to give a maximum number of characters for each field, or you'll get a message and will have to start from the beginning. Once you've defined the last field, the computer will display the maximum number of entries that particular configuration will allow and ask if it is acceptable.

Use the **ADD** option to enter data into your database. To exit this option, simply hit RETURN on the first field of the next entry.

DELETE allows you to remove a record from memory. The 128 prompts you to enter numbers individually (#) or to list all (A). If you choose #, you'll need to enter the appropriate number of the record to be removed. The A response will go through the entire file, and you will have the opportunity to keep each entry or remove it.

FIND is a search routine. When you choose it, the display clears, and the names of the different fields are shown. Select the field you want to search. You'll be prompted to enter the string which is to be found.

MODIFY gives basically the same prompts as DELETE, but instead of removing an entry, you can change the data in the file.

READ is used to retrieve data from a disk file. If data is in memory at the time READ is selected, you are warned that a READ command will erase that data and are asked if you want to save that data first. Database 128 will read only disk files that it has written.

SORT will arrange the entries in alphanumeric order. The computer displays the names of the different fields and prompts you to select the field to be sorted. The display will then clear because the program goes into FAST mode to speed up the sort routine.

VIEW allows you to look at the entries in the file. While you're in this option, you can advance to the next entry, back up to the last entry, print the entry onscreen, go to the FIND routine, or exit to the main menu.

WRITE saves the file in memory to disk. It labels its files with a *DB-* prefix in front of the filename that you've assigned. This allows the program to identify its own files during the DIRECTORY and READ options.

The Print Option

When you select **PRINT** from the main menu, you are transferred to the print menu. Here, you can choose either to print a mailing list or to dump the contents of the file to the printer. If you choose the mailing labels option, only the first three fields are printed. However, if the list option is selected, all fields are printed. With both options you are prompted to enter the starting entry number to be printed. At that time, you can abort the print by pressing RETURN, or you can enter the number of the first entry to be printed. If you enter a 1, the entire file will be printed.

The Housekeeping Menu

The HOUSEKEEPING option takes you to yet another menu, the housekeeping menu, where you'll find the following options:

DIRECTORY displays only the files on the disk that can be read by Database 128.

EXIT TO BASIC takes you out of the database and back to BASIC 7.0.

FORMAT makes a disk ready for use. It assigns the disk the name DATABASE and the ID number 40. To change either the disk name or the ID number, list the program and make the changes in line 410.

MAIN MENU returns you to the main menu.

SCRATCH prompts you for the filename that's to be scratched and then executes the SCRATCH command.

As with all of your important files, be sure to make backup copies of all your Database 128 files.

Database 128

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
QG 10 REM DATABASE 128
JQ 12 COLOR4,1:COLOR0,1:COLOR5,6:DCLOSE:CLOSE4:GOT
      068
MP 16 D$=CHR$(0):MR$=D$:DR$=D$:S=0:B1$=CHR$(10):PW
      =0:CW=0:B$=CHR$(32)
```

```
EJ 18 NC=Ø:NL=Ø:PG=Ø:F1=Ø:F2=Ø:F3=Ø:L$=D$:RL=Ø:SB$
      =D$:CR$=CHR$(13):HN$=D$:ID$=D$
RJ 2Ø A$=D$:C$=D$:T%=Ø:I$=D$:CK=Ø:I=Ø:J=Ø:K=Ø:L=Ø:
      M=Ø:N=Ø:RW=5:SF=Ø:Z=Ø:E$="EOF"
KH 22 MEM=64249:EN=Ø:EM$=D$:ET=Ø:ES=Ø:A1$=D$:A2$=D
      $:A3$=D$:RETURN
RF 24 DIM F$(F+1),T%(F+1),L%(F+1):RETURN
EJ 26 DIM REC$(R+1,F+1),ML$(9,4),PC(1Ø),TT$(5),HC$
      (9),K%(R+1):RETURN
JX 28 REM *** GET ***
QJ 3Ø GOSUB2ØØØØ
MQ 31 PRINT"{2 UP}": GETKEY A$
MB 32 RETURN
FB 34 REM *** CREATE ***
HC 36 IFCK<>ØTHENGOSUB394
FG 4Ø FAST: CLR:GOSUB16:SLOW:INPUT"{CLR}{DOWN}HOW
      {SPACE}MANY FIELDS IN EACH RECORD? Ø
      {4 LEFT}";F:IFF=ØTHEN68
CB 42 FAST: GOSUB24:SLOW:FORI=1TOF
MC 44 PRINT"{HOME}{4 DOWN}{RVS}FIELD #";I:PRINT"
      {DOWN}TITLE ?{29 SPACES}"
AF 46 PRINT"LENGTH{31 SPACES}{HOME}"
MP 47 PRINT"{5 DOWN}";TAB(6);:INPUTF$(I):PRINTTAB(
      6);:INPUTL%(I)
DA 48 IF F$(I)="" OR L%(I)=Ø THEN PRINT "EACH FIEL
      D MUST HAVE A TITLE AND LENGTH":I=F:SLEEP3:G
      OTO4Ø
QJ 49 NEXT I
HH 5Ø REM *** COMPUTE # RECORDS ***
MC 52 FORJ=ØTOF:RL=RL+L%(J):NEXTJ:RL=RL+3*(F+1)+5:
      R=INT((MEM-12*(F+1)-21ØØ)/RL)
SG 54 PRINT"{DOWN} YOUR SELECTIONS WILL ALLOW APPR
      OX"
XS 56 PRINTR;"RECORDS.{2 SPACES}{RVS}A{OFF}CCEPT O
      R {RVS}R{OFF}EJECT?"
HG 58 GOSUB3Ø:IFA$="R"THEN38
PG 6Ø IFA$="A"THENGOSUB26:CK=1:GOTO68
DK 62 GOTO58
HG 64 REM *** MENU ***
PC 68 SLOW: PRINT"{CLR} {RVS}{8 SPACES}DATABASE 12
      8 MENU{14 SPACES}"
EP 7Ø PRINT"{4 DOWN}{5 SPACES}{RVS}A{OFF}DD
      {16 SPACES}{RVS}P{OFF}RINT"
SG 72 PRINT"{DOWN}{5 SPACES}{RVS}C{OFF}REATE
      {13 SPACES}{RVS}R{OFF}EAD"
DA 74 PRINT"{DOWN}{5 SPACES}{RVS}D{OFF}ELETE
      {13 SPACES}{RVS}S{OFF}ORT"
GH 76 PRINT"{DOWN}{5 SPACES}{RVS}F{OFF}IND
      {15 SPACES}{RVS}V{OFF}IEW"
```

```
BE 78 PRINT"{DOWN}{5 SPACES}{RVS}H{OFF}OUSEKEEPING
      {7 SPACES}{RVS}W{OFF}RITE"
HC 80 PRINT"{DOWN}{5 SPACES}{RVS}M{OFF}ODIFY
MP 90 PRINT"{5 DOWN} {RVS}{6 SPACES}PRESS THE APPR
      OPRIATE KEY{7 SPACES}"
DS 92 PRINT"{2 SPACES}THERE ARE";X;"RECORDS IN MEM
      ORY"
QX 94 IF R>ØTHENPRINT"{2 SPACES}SPACE FOR";R-X;"MO
      RE RECORDS{UP}"
FF 96 GOSUB3Ø:IFA$="A"THENGOSUB35Ø:GOTO124
BA 98 IFA$="M"THENGOSUB35Ø:GOTO244
EP 1ØØ IFA$="D"THENGOSUB35Ø:GOTO272
BF 1Ø2 IFA$="C"THEN36
QE 1Ø4 IFA$="R"THEN17Ø
RB 1Ø6 IFA$="P"THENGOSUB35Ø:GOSUB356 :GOTO68
CQ 1Ø8 IFA$="V"THENGOSUB35Ø:GOTO192
EC 11Ø IFA$="W"THENGOSUB35Ø:GOTO144
BJ 112 IFA$="S"THENGOSUB35Ø:GOTO3Ø4
CB 116 IFA$="F"THEN218
DB 118 IFA$="H"THEN46Ø
XB 12Ø GOTO96
SA 122 REM *** ADD RECORDS ***
BC 124 FORI=X+1TOR:PRINT"{CLR}{DOWN}PRESS THE
       {RVS}RETURN{OFF} KEY AFTER EACH ENTRY{DOWN}
       "
MM 126 PRINT"PRESS {RVS}RETURN{OFF} WITHOUT ANY EN
       TRY TO STOP{2 DOWN}"
XX 128 PRINT"{RVS}RECORD NUMBER ";I;"{DOWN}"
FS 13Ø FORN=1TOF
DF 132 PRINTF$(N);"{3 SPACES}>{3 LEFT}";:INPUTREC$
       (I,N):IFREC$(I,N)=""THENREC$(I,N)=">"
GQ 134 IFLEN(REC$(I,N))>L%(N)THENGOSUB14Ø:GOTO132
BF 136 IFREC$(I,1)=">"THENX=I-1:CK=1:GOTO68
SE 138 NEXTN:K%(I)=I:NEXTI:X=R:CK=1:GOTO68
AF 14Ø PRINT"CANNOT EXCEED{RVS}";L%(N);" CHARACTER
       S":RETURN
XA 142 REM *** SAVE ***
GF 144 PRINT"{CLR}{DOWN}ENTER NAME OF CURRENT FILE
        TO BE SAVED"
SE 146 PRINT"(12 CHARACTERS MAX).{2 SPACES}ANY EXI
       STING FILE"
CG 148 PRINT"WITH THE SAME NAME WILL BE SCRATCHED.
       "
BX 149 PRINT"JUST PRESS {RVS}X{OFF} AND <RETURN> T
       O ABORT.{2 DOWN}"
RK 15Ø PRINT"{2 SPACES}";NF$:INPUT"{UP}";NF$:IFNF$
       ="" OR LEFT$(NF$,1)="X"THEN68
MK 152 SCRATCH "DB- "+LEFT$(NF$,8)+" OLD":PRINT"
       {DOWN}SCRATCHING OLD FILE"
```

```
SA 153 RENAME "DB- "+NF$ TO "DB- "+LEFT$(NF$,8)+"
       {SPACE}OLD":PRINT"{DOWN}RENAMING CURRENT FI
       LE"
AD 156 DOPEN#5,"DB- "+NF$+",S",W:PRINT "{DOWN}SAVI
       NG NEW FILE AS ";NF$:GOSUB414
KB 158 PRINT#5,R;CR$;F;CR$;X:FORN=1TOF:PRINT#5,F$(
       N);CR$;L%(N):NEXTN
RX 160 FORI=1TOX:PRINT"{DOWN}SAVING RECORD #";I;"
       {2 UP}"
XE 162 FORN=1TOF:PRINT#5,REC$(I,N):NEXTN:NEXTI:PRI
       NT
DX 164 FORI=1TOX:PRINT"{DOWN}SAVING POINTERS";I;"
       {2 UP}":PRINT#5,K%(I):NEXTI
EH 166 PRINT#5,E$:PRINT#5:DCLOSE:CK=0:PRINT"
       {2 DOWN}":GOSUB414:SLEEP2:GOTO 68
FK 168 REM *** LOAD ***
FA 170 IFCK<>0THENGOSUB394
CQ 172 CLR:GOSUB 16:PRINT"{CLR}{DOWN}ENTER NAME OF
        FILE TO BE LOADED":PRINT"PRESS <RETURN> TO
        ABORT {2 DOWN}":INPUT NF$:IFNF$=""THEN68
KM 174 DOPEN#5,"DB- "+NF$+",S":GOSUB414:IF DS=62 T
       HEN SLEEP1:GOTO 68
XP 176 INPUT#5,R,F,X:GOSUB24:GOSUB26:FORN=1TOF:INP
       UT#5,F$(N),L%(N):NEXTN
ED 178 FORI=1TOX:PRINT"{DOWN}READING RECORD #";I;"
       {2 UP}"
AC 180 FORN=1TOF:INPUT#5,REC$(I,N):NEXTN:NEXTI:PRI
       NT
FB 182 FORI=1TOX:PRINT"{DOWN}READING POINTERS";I;"
       {2 UP}":INPUT#5,K%(I):NEXTI
FB 184 INPUT#5,E$:IF E$<>"EOF"THEN GOSUB 414
MK 186 DCLOSE:PRINT"{2 DOWN}":GOSUB414:SLEEP2:GOTO
       68
HQ 190 REM *** VIEW ***
DB 192 INPUT"{CLR}{3 DOWN}VIEW STARTING WITH WHICH
        RECORD";L$:I=VAL(L$):IFL$=CHR$(13)THENI=1
SJ 194 IFI=0THEN68
JX 196 IFI>XTHEN68
RG 198 PRINT"{CLR}{2 DOWN}{RVS}RECORD NUMBER{OFF}"
       ;I;" {RVS}IN FILE{OFF} ";NF$;"{2 DOWN}"
AR 200 FORN=1TOF:PRINT F$(N);": {RVS}";REC$(K%(I),
       N):NEXTN
SG 202 PRINT"{DOWN} {RVS}N{OFF}EXT, {RVS}L{OFF}AST
       , {RVS}P{OFF}RINT, {RVS}F{OFF}IND, {RVS}E
       {OFF}XIT TO MENU"
MP 204 GOSUB30:IFA$="N"THENI=I+1:GOTO194
EQ 206 IFA$="L"THENI=I-1:GOTO194
BM 208 IFA$="P"THENGOSUB 600:I=I+1:GOTO194
```

```
XM 21Ø IFA$="F"THEN218
KG 212 IFA$="E"THEN68
JB 214 GOTO2Ø4
AH 216 INPUT"{DOWN}JUMP TO RECORD NUMBER";I:GOTO19
       4
HG 218 PRINT"{CLR}{4 SPACES}{RVS}FIND RECORDS WITH
        COMMON ITEMS {DOWN}"
BB 22Ø FORN=1TOF:PRINT" {RVS}";N;"{OFF} ";F$(N):NE
       XTN
BK 222 INPUT"{DOWN}WHICH FIELD IS TO BE SEARCHED?
       {SPACE}Ø {4 LEFT}";SF:IFSF=ØTHEN68
SQ 224 IFSF<1ORSF>FTHENPRINT"{3 UP}":GOTO222
QE 226 PRINT"{DOWN}ENTER {RVS}COMMON ITEM{OFF} ":P
       RINT"{DOWN}(THE ENTIRE STRING IS NOT REQUIR
       ED)"
PF 228 PRINT"{DOWN}{RVS}";F$(SF);"{OFF} ";:INPUTT$
EM 23Ø FORI=1TOX:PRINT"{DOWN}SEARCHING RECORD";I;"
       {2 UP}"
KD 232 IFT$=LEFT$(REC$(K%(I),SF),LEN(T$))THEN236
EC 234 GOTO24Ø
BH 236 PRINT"{CLR} RECORD #";I;"{DOWN}":FORN=1TOF:
       PRINTF$(N);": {RVS}";REC$(K%(I),N):NEXTN
SG 238 PRINT"{DOWN} {RVS}N{OFF}EXT RECORD":PRINT"
       {DOWN} {RVS}P{OFF}RINT RECORD":GOSUB3Ø
RJ 239 IFA$="P"THENGOSUB1ØØØ
CK 24Ø NEXTI:GOTO68
XB 242 REM *** MODIFY ***
DR 244 PRINT"{CLR}{DOWN}MODIFY WHICH RECORD? ENTER
        {RVS}#{OFF}{2 SPACES}OR {RVS}A{OFF}LL
       {2 DOWN}":INPUTMR$:IFMR$=D$THEN68
JM 246 IFMR$="A"THENMR$=D$:GOTO254
KX 248 I=VAL(MR$):MR$=D$
GC 25Ø IFI>XTHENGOSUB244
MH 252 GOSUB256:GOTO68
PG 254 FORI=1TOX:GOSUB256:NEXTI:GOTO68
EJ 256 PRINT"{CLR}{DOWN}TO MODIFY RECORD NUMBER";I
       ;", MAKE CHANGES"
PB 258 PRINT"AS EACH FIELD IS DISPLAYED,THEN {RVS}
       RETURN{OFF}{DOWN}"
BR 26Ø FORN=1TOF:PRINTF$(N)":":PRINT"{3 SPACES}
       {RVS}";REC$(K%(I),N)
PR 261 IF LEN(REC$(K%(I),N))>36 THEN PRINT"{UP}";
BR 262 PRINT"{UP} ";:INPUTREC$(K%(I),N)
BM 264 IFLEN(REC$(K%(I),N))>L%(N)THENGOSUB14Ø:GOTO
       26Ø
QD 266 IFREC$(K%(I),N)=""THENREC$(K%(I),N)=">"
DR 268 NEXTN:CK=1:RETURN
GG 27Ø REM *** DELETE ***
```

```
XS 272 PRINT"{CLR}{DOWN}DELETE WHICH RECORD? ENTER
        {RVS} # {OFF} OR {RVS}A{OFF}LL{2 DOWN}"
KD 274 INPUTDR$:IFDR$=D$THEN68
JD 276 IFDR$="A"THENDR$=D$:GOTO282
EJ 278 I=VAL(DR$):DR$=D$:IFI>XTHENGOSUB348:GOTO274
GF 280 GOSUB284:GOTO68
FF 282 FORI=1TOX:GOSUB284:NEXTI:GOTO68
DX 284 PRINT"{CLR}{2 DOWN}TO DELETE RECORD NUMBER"
       ;I;", PRESS"
QK 286 PRINT"{RVS}SHIFT{OFF} {RVS}D{OFF}, PRESS
       {RVS}SPACE BAR{OFF} TO ADVANCE{DOWN}"
CH 288 FORN=1TOF:PRINTF$(N);"{3 SPACES}{RVS}";REC$
       (K%(I),N):NEXTN
RC 290 GOSUB30:IFA$="D"THEN294
XA 292 CK=1:RETURN
MP 294 PRINT"{2 DOWN}DELETING RECORD";I:PRINT"
       {DOWN}{RVS}RECORDS MAY NOW BE OUT OF ORDER"
MX 295 FORZZ=1TO1000:NEXT
JF 296 FORN=1TOF:REC$(K%(I),N)=REC$(X,N):REC$(X,N)
       ="":NEXTN
JB 298 FORJ=1TOX:IFK%(J)=XTHENK%(J)=K%(X):K%(X)=0:
       X=X-1:GOTO292
QP 299 NEXT J
HX 302 REM *** SORT ***
PH 304 PRINT"{CLR}{DOWN}{RVS}{4 SPACES}SORT RECORD
       S IN ASCENDING ORDER{3 SPACES}{DOWN}"
XX 306 FOR N=1TOF:PRINT" {RVS}";N;"{OFF} ";F$(N):N
       EXTN
FD 308 INPUT"{DOWN}WHICH FIELD IS TO BE SORTED? 0
       {SPACE}{4 LEFT}";SF:IFSF=0THEN68
JE 310 IFSF>FTHENPRINT"{3 UP}":GOTO 308
SR 312 M=X
DD 314 M=INT(M/2):IFM=0THENCK=1:SLOW:GOTO68
DA 315 FAST
FB 316 J=1:K=X-M
MJ 318 I=J
KB 320 L=I+M
HQ 324 IFREC$(K%(I),SF)<=REC$(K%(L),SF)THEN328
SB 326 T%(N)=K%(I):K%(I)=K%(L):K%(L)=T%(N):I=I-M:I
       FI>0THEN320
GH 328 J=J+1:IFJ>KTHEN314
CP 330 GOTO318
GE 332 REM *** EXIT ***
SC 334 PRINT"{CLR}{2 DOWN} {RVS}YOU HAVE NOT SAVED
        YOUR CHANGES!"
JR 336 PRINT"{2 DOWN} DO YOU REALLY WANT TO QUIT?
       {SPACE}{RVS}Y{OFF} OR {RVS}N{OFF}
HK 338 GOSUB30:IFA$="Y" THEN344
DS 340 GOTO68
```

```
EJ 341 PRINT"{CLR}":GOTO336
MH 344 PRINT"{CLR}{2 DOWN}DATABASE TERMINATED":END
MD 348 PRINT"NO SUCH RECORD EXISTS":RETURN
PK 35Ø IFR>ØTHENRETURN
XD 352 PRINT"{CLR}{DOWN}NO FILES IN MEMORY":SLEEP
       {SPACE}3:GOTO68
SE 356 REM *** PRINT MENU ***
MM 358 PRINT"{CLR}{RVS}{14 SPACES}PRINT MENU
       {15 SPACES}"
QJ 36Ø PRINTTAB(12)"{6 DOWN}{RVS}L{OFF}IST"
JQ 362 PRINTTAB(12)"{DOWN}{RVS}M{OFF}AILING LABELS
       "
HF 364 PRINTTAB(12)"{DOWN}{RVS}E{OFF}XIT TO MAIN M
       ENU"
DR 365 PRINT"{1Ø DOWN}{RVS}{39 SPACES}"
QA 366 GOSUB3Ø:IFA$="L"THEN5ØØ
PF 368 IFA$="M"THEN 55Ø
ER 37Ø IFA$="E"THEN 68
AS 372 GOTO372
DR 392 REM *** WARNING ***
SD 394 PRINT"{CLR}{DOWN}{RED}{RVS}THIS WILL DESTRO
       Y THE FILE IN MEMORY!{3 SPACES}{GRN}
QR 396 PRINT"{2 DOWN}SAVE THE FILE FIRST? {RVS}Y
       {OFF} OR {RVS}N{OFF}":GOSUB3Ø:IFA$="N"THENR
       ETURN
RX 398 GOTO142
MC 4ØØ REM *** NEW DISK ***
GK 4Ø2 PRINT"{CLR}{DOWN} ARE YOU SURE? {RVS}Y{OFF}
        OR {RVS}N{OFF}
BQ 4Ø4 GOSUB3Ø:IFA$="N"THEN68
KG 4Ø6 IFA$<>"Y"THEN68
PC 41Ø HEADER "DATABASE",I4Ø:PRINT"{CLR}{3 DOWN}
       {6 SPACES}FORMATTING....."
KD 411 GOTO68
XA 412 REM *** DISK ERROR ***
MH 413 IF TT=1 THEN END
CH 414 PRINTDS$:IFDS>Ø1ANDDS<2Ø THENSLEEP2:GOTO68:
       ELSE RETURN
KF 422 PRINT"{CLR}":DIRECTORY"DB-*"
RA 456 PRINTTAB(25)"PRESS {RVS}ANY KEY{OFF}":GOSUB
       3Ø:GOTO68
MS 46Ø REM *** HOUSEKEEPING MENU ***
QA 462 PRINT"{CLR} {RVS}{8 SPACES}HOUSEKEEPING MEN
       U{14 SPACES}"
EM 464 PRINT"{4 DOWN}{5 SPACES}{RVS}D{OFF}IRECTORY
       "
SE 468 PRINT"{DOWN}{5 SPACES}{RVS}E{OFF}XIT TO BAS
       IC"
```

```
SK 470 PRINT"{DOWN}{5 SPACES}{RVS}F{OFF}ORMAT DISK
       "
SK 472 PRINT"{DOWN}{5 SPACES}{RVS}M{OFF}AIN MENU"
SH 474 PRINT"{DOWN}{5 SPACES}{RVS}S{OFF}CRATCH FIL
       E"
AE 476 PRINT"{DOWN}
GQ 478 PRINT"{6 DOWN} {RVS}{6 SPACES}PRESS THE APP
       ROPRIATE KEY{7 SPACES}":GOSUB 30
XH 480 IF A$="D" THEN422
XG 482 IF A$="E" THENPRINT"{CLR}":GOTO336
BH 484 IF A$="F" THEN402
QS 486 IF A$="M" THEN68
JH 488 IF A$="S" THEN525
DG 500 REM *** PRINT LIST ***
JM 505 L=0
QJ 510 INPUT"{CLR}{3 DOWN}PRINT STARTING WITH WHIC
       H RECORD";L$:I=VAL(L$):IFL$=CHR$(13)THENI=1
RA 512 OPEN 4,4:PRINT#4
HG 514 IFI=0ORI>XTHENCLOSE4:GOTO68
SJ 516 FORN=1TOF:PRINT#4,REC$(K%(I),N):NEXTN:PRINT
       #4:I=I+1:GOTO514
ER 525 REM *** SCRATCH FILE ***
DR 526 INPUT"{CLR}{2 SPACES}SCRATCH WHICH FILE";DF
       $
ME 528 IFDF$ =""THEN 68
BF 532 PRINT"{DOWN}{RED}SCRATCH{2 SPACES}{RVS}"DF$
       "{OFF} :ARE YOU SURE? {GRN}":GOSUB30
EK 533 IFA$<>"Y"THEN68
JS 534 SCRATCH "DB- "+DF$,D0 :PRINT"{DOWN}"DS$:SLE
       EP 3:GOTO68
HM 550 REM *** PRINT MAILING LIST ***
QS 555 L=0
AQ 560 INPUT"{CLR}{3 DOWN}PRINT STARTING WITH WHIC
       H RECORD";L$:I=VAL(L$):IFL$=CHR$(13)THENI=1
BH 562 OPEN 4,4:PRINT#4
AM 564 IFI=0ORI>XTHENCLOSE4:GOTO68
RJ 566 FORN=1TO3:PRINT#4,REC$(K%(I),N):NEXTN:PRINT
       #4:PRINT#4:PRINT#4:I=I+1:GOTO564
CX 600 REM *** PRINT ENTRY ***
FX 604 OPEN 4,4:PRINT#4,"{CLR}{2 DOWN} RECORD NUMB
       ER ";I;"{2 SPACES}IN FILE{2 SPACES}";NF$;"
       {2 DOWN}"
AE 606 FORN=1TOF:PRINT#4, F$(N);":{2 SPACES}";REC$
       (K%(I),N):NEXTN:PRINT#4:CLOSE4:RETURN
QG 20000 PRINT"{BELL}":RETURN
```

SpeedScript-80 for the 128

Todd Heimarck

Now 128 owners with an 80-column monitor can run the popular SpeedScript *word processor originally written for the Commodore 64. If you already have a copy of* SpeedScript, *version 3.0 or higher, very little typing is required because this program patches into the original.*

Without a doubt, the *SpeedScript* word processor is the most popular program ever published by COMPUTE! Publications. Version 1, including both a VIC and a 64 program, was printed in the January 1984 issue of *COMPUTE!'s Gazette.* The upgraded and improved version 2.0, with a help screen and custom characters, was included on the inaugural *Gazette Disk* in May of the same year. Version 2.1 can be found in *COMPUTE!'s Second Book of Commodore 64.*

Version 3.0 (with separate programs for the 64, VIC, Apple, and Atari) appeared over several months in *COMPUTE!* magazine in the spring of 1985. The machine language source code for version 3.1 was published separately in the *SpeedScript* books for Commodore, Atari, and Apple. Version 3.2 was included as a bonus on the January 1986 *COMPUTE! Disk.* Individual disks containing *SpeedScript* 2.0, 3.0, 3.1, and 3.2 are still available.

SpeedScript for the 128

The ideal 128 version would take full advantage of the machine's features: 40- or 80-column output, access to the numeric keypad and other keys (perhaps an alternate character set toggled by the ALT key), and two large text areas of about 60K each.

As you may have guessed, the program given here is not the full-featured 128 version. That's the bad news. The good news is that, if you already have a copy of *SpeedScript* for the 64, version 3.0 or higher (from the March 1985 *COMPUTE!,* the *SpeedScript* book, or the January 1986 *COMPUTE! Disk),* you'll have to type in only a few hundred characters to upgrade to a full 80 columns. Compare that to the roughly 7 to 8K you'd have to enter for a brand-new program.

"SpeedScript-80" patches into the main program to provide an 80-column screen display. It must be run on a 128—in 64 mode—because it takes advantage of the 128's 80-column chip (yes, the 80-column screen can be accessed in 64 mode). It won't run on standard 64s because they lack the 80-column chip.

Installing SpeedScript-80

You'll need to begin with a working version of *SpeedScript* 3. If you're not sure which version you own, look at the top (command) line. If there's no number, you have version 1 or 2. If it says 3.0, 3.1, or 3.2, you have the correct version.

First, go into 64 mode by typing GO 64. Load and run "MLX," the machine language entry program found in Appendix C, and type in the following programs (save them as five separate files):

Patch 1
Starting address: 289E
Ending address: 2935

Patch 2
Starting address: 2A4E
Ending address: 2A5D

Patch 3
Starting address: 315D
Ending address: 31A4

Patch 4
Starting address: 3445
Ending address: 346C

Patch 5
Starting address: C000
Ending address: C137

After saving these five programs to disk, go back into 128 mode (turn your computer off and then on). Enter the machine language monitor with the command MONITOR (or press F8). Insert the disk containing *SpeedScript* into your drive, and type the following monitor load command, substituting the appropriate filename for SPEEDSCRIPT:

```
L "SPEEDSCRIPT",8,02801
```

SpeedScript will be loaded into the 128's memory at address $02801. Normally, *SpeedScript* loads at $0801, but that part of memory is not available for use by programs in 128 mode. Now load the first four patches:

```
L "PATCH1",8
L "PATCH2",8
L "PATCH3",8
L "PATCH4",8
```

The 80-column patches are inserted into the program, and you can now save the results:

```
S "TEMP",8,02801,04009
```

This is only a temporary file, which you can scratch when you've finished creating SpeedScript-80. You're almost there. Now type X (to eXit to BASIC) and enter GO 64. From 64 mode, type these lines:

```
LOAD "PATCH5",8,1
NEW
LOAD "TEMP",8
SYS 49152
```

Insert the disk on which you want to save SpeedScript-80 before entering SAVE "SPEEDSCRIPT80",8. This file is the new 80-column version of *SpeedScript;* the patches are no longer needed.

How to Run It

You have to follow specific instructions to load and run SpeedScript-80:

1. Turn on your 128 in 128 mode.
2. Type GO 64, press RETURN, and then answer Y to the "ARE YOU SURE?" question. The computer will switch to 64 mode.
3. From 64 mode, LOAD "SPEEDSCRIPT80",8 and type RUN.
4. Switch the display from 40 to 80 columns.

Note: If you hold down the Commodore key to go straight into 64 mode when you turn on the machine, SpeedScript-80 won't work correctly. Starting out in 128 mode forces the computer to initialize the 80-column chip. Among other things, the character set is loaded into 80-column memory.

How It Works

A Commodore 128 in 64 mode is not a perfect replication of a Commodore 64. It has some extra capabilities, like access to the 80-column chip.

The 8563 chip in the 128 provides the 80-column screen in 128 mode and has two handles in 64 mode: You can PEEK and POKE locations $D600 and $D601. Address $D600 controls which internal 8563 register is PEEKed or POKEd, while $D601 contains the value read from or written to the register.

The 8563 80-column chip has its own 16K of dedicated memory, and you can reach it only through the two memory locations at $D600–$D601. Between the two control locations and the 16K of memory, though, are 36 internal 8563 chip registers. To POKE the value 1 (we'll use the *A* character) to the first memory location for the 80-column screen (address $0000 in the 8563's 16K bank), you have to perform the following POKEs:

1. Store $12 in $D600. Register 18 ($12) holds the high byte of the address where we'll POKE.
2. Store a 0 in $D601, which is sent to register 18 ($12), as we set up in step 1.
3. Store $13 in $D600. Register 19 ($13) holds the low byte of the address where we'll POKE (note that for the 8563 chip the high byte comes before the low byte, just the opposite of a typical 8502 machine language address).
4. Store a 0 in $D601. Now, registers $12–$13 point to location $0000. This is not $0000 in the 128's memory map; it's a location in the 8563's private memory.
5. Store $1F in $D600. Register 31 ($1F) will hold the character to be POKEd to screen memory.
6. Finally, store a 1, the screen code of the letter we're putting in the top left corner, in $D601. From register 31, it will be transferred to the 80-column screen memory address in registers $12–$13.

For machine language programmers who want to try this, there's one more thing to keep in mind. After storing the register number in $D600, you have to wait for the high bit ($80) of $D600 to be set before storing a value in $D601. This can be done with the BIT instruction followed by a BPL. When the high bit is turned on, you can store the appropriate value in $D601.

Screen memory for the 80-column screen starts at $0000, and it takes six POKEs to get a character there. *SpeedScript* is built to deal with a screen size of 40 × 25 characters, and a lot of time goes to updating the screen. Theoretically, the 80-column screen should take more time.

FAST 64 Mode

You can make the 128 work twice as fast as usual in 128 mode by entering the FAST command. The 40-column screen is disabled, but the computer's speed doubles from 1 megahertz (1,000,000 instruction cycles per second) to 2 megahertz (2,000,000 instruction cycles per second).

Within 64 mode, you can access the FAST mode by sacrificing the 40-column screen. Since you're POKEing the 80-column chip, this isn't a problem. The 8563 has twice as much screen memory, so it should take twice as long to update the display. But if the computer works twice as fast, the disadvantage is canceled out. Twice as much screen memory slows things down, and twice the speed returns things to normal.

From BASIC, you can go into FAST mode by typing POKE 53296,1. To go back to normal, POKE 53296,0. This works in both 64 mode and 128 mode (to be safe, issue a BANK 15 statement before you use this POKE on the 128). Of course, BASIC 7.0 has the FAST command, so the POKE isn't really necessary in 128 mode.

Slight Changes to the Command Set

Three commands are no longer available when you run SpeedScript-80: CTRL-L (change letter color), CTRL-B (change background color), and CTRL-X (exchange two transposed characters). An RGB monitor is required to see 80 columns in color. And the routine to fill attribute memory (something like color memory, but it also controls flashing and underlining) would have required an additional patch. The CTRL-X option has been deleted because a small section of memory is needed for one of the new 80-column subroutines.

In addition, because there is more information on the screen, the disk directory command (CTRL-$) lists the filenames in two columns. This means you can see up to 50 filenames on a single screen. Unfortunately, the way the screen wraps around puts half the number of blocks per file on the right-hand edge of the screen. To fix this would require another patch, which would take away one more *SpeedScript* command. Also, the characters have to be POKEd to 80-column memory, so there's no easy way to scroll the screen when you have more than 50 files on a disk. If you attempt to display a long directory, the extra characters go past screen memory into attribute memory. Again, writing a screen scroll routine would have meant another patch.

Apart from these four changes, all *SpeedScript* commands remain the same. Documents created in 40-column *SpeedScript* can be loaded, edited, and saved with SpeedScript-80. And the printing and formatting commands are unchanged.

SpeedScript-80

See instructions in article, and read Appendix C, "MLX," before typing in the following program listings.

Patch 1

Starting address: 289E
Ending address: 2935

```
289E:4C B6 08 8E 00 D6 D0 07 B0
28A6:48 A9 1F 8D 00 D6 68 2C 9A
28AE:00 D6 10 FB 8D 01 D6 60 F4
28B6:A9 00 A2 12 20 A1 08 E8 D1
28BE:A9 50 20 A1 08 AD 11 20 4F
28C6:85 FB AD 12 20 85 FC A2 63
28CE:01 A0 00 B1 FB 99 3C 03 A4
28D6:C8 29 7F C9 1F F0 13 C0 06
28DE:50 D0 F0 88 B1 FB 29 7F 81
28E6:C9 20 F0 05 88 D0 F5 A0 A6
28EE:4F C8 84 3B A0 00 B9 3C 12
28F6:03 20 A6 08 C8 C4 3B D0 C6
28FE:F5 18 98 65 FB 85 FB A5 4D
2906:FC 69 00 85 FC E0 01 D0 C7
290E:03 8C 10 20 20 28 09 E8 A5
2916:E0 19 F0 03 4C CF 08 A5 C4
291E:FB 8D 1B 20 A5 FC 8D 1C 8F
2926:20 60 C0 50 F0 08 A9 20 D8
292E:20 A6 08 C8 D0 F4 60 00 E2
```

Patch 2

Starting address: 2A4E
Ending address: 2A5D

```
2A4E:A9 00 A8 20 96 11 20 28 EF
2A56:09 A9 00 4C 96 11 00 00 57
```

Patch 3
Starting address: 315D
Ending address: 31A4

```
315D:A9  FD  8D  30  D0  A9  08  20  26
3165:96  11  A9  20  A2  18  20  A1  E5
316D:08  A9  8F  20  A6  08  A0  08  D0
3175:A9  FF  A2  1E  20  A1  08  88  03
317D:D0  FA  60  29  7F  C9  20  90  99
3185:0F  C9  40  90  08  E9  40  C9  25
318D:20  90  02  69  1F  20  A6  08  C9
3195:60  A2  12  20  A1  08  A9  00  95
319D:E8  4C  A1  08  00  00  00  00  3C
```

Patch 4
Starting address: 3445
Ending address: 346C

```
3445:A9  FC  8D  30  D0  A9  00  4C  EF
344D:96  11  20  CD  BD  A0  00  B9  50
3455:00  01  F0  06  20  A6  08  C8  F0
345D:D0  F5  60  30  20  A9  00  20  82
3465:96  11  A9  20  20  70  11  00  79
```

Patch 5
Starting address: C000
Ending address: C137

```
C000:A9  12  85  FB  A9  1E  85  FC  19
C008:A0  00  B1  FB  29  7F  F0  14  0D
C010:C9  20  B0  02  A9  2A  C9  40  7E
C018:90  0A  38  E9  40  C9  20  90  04
C020:03  18  69  20  91  FB  C8  D0  37
C028:E1  A6  FC  E0  20  F0  05  E8  A9
C030:86  FC  D0  D6  A0  00  84  FB  C5
C038:A9  08  85  FC  B1  FB  C9  20  42
C040:F0  0F  C8  D0  F7  A6  FC  E8  61
C048:86  FC  E0  15  D0  EE  4C  76  0B
C050:C0  84  02  84  FD  A5  FC  85  E1
C058:FE  A0  01  B1  FD  C9  D2  D0  4A
C060:10  C8  B1  FD  C9  FF  D0  09  2B
C068:A9  11  91  FD  88  A9  80  91  92
C070:FD  A4  02  4C  42  C0  A9  4F  D6
C078:8D  E3  14  A9  14  8D  E4  14  8B
C080:A9  06  8D  FA  14  A9  A6  8D  DB
C088:FC  14  A9  08  8D  FD  14  A9  79
C090:20  8D  B4  16  8D  85  18  A9  D9
C098:45  8D  B5  16  A9  14  8D  B6  A7
```

```
CØAØ:16 A9 5D 8D 86 18 A9 11 15
CØA8:8D 87 18 A9 A6 8D 7C Ø9 DD
CØBØ:A9 Ø8 8D 7D Ø9 AØ Ø4 A9 ØF
CØB8:78 99 8B ØE 99 ØA 16 A9 F9
CØCØ:58 99 9B ØE 99 1A 16 88 13
CØC8:A9 EA 99 8B ØE 99 ØA 16 C6
CØDØ:99 9B ØE 99 1A 16 88 1Ø AB
CØD8:F1 A9 4F 8D B8 1D 8D Ø6 DB
CØEØ:1E A9 14 8D B9 1D 8D Ø7 9B
CØE8:1E A9 22 8D 8C 1B A9 16 3B
CØFØ:8D 8D 1B AØ ØB B9 1C C1 43
CØF8:99 24 16 88 1Ø F7 AØ Ø8 45
C1ØØ:B9 28 C1 99 E5 14 88 1Ø DC
C1Ø8:F7 A9 EA 8D F9 14 8D FA 5E
C11Ø:14 A9 28 8D FC 14 A9 Ø9 7A
C118:8D FD 14 6Ø B9 45 2Ø FØ 7E
C12Ø:Ø6 2Ø D2 FF C8 DØ F5 6Ø DE
C128:A9 36 85 Ø1 98 18 69 43 ØA
C13Ø:A8 ØØ ØØ ØØ ØØ ØØ ØØ ØØ Ø8
```

Marquee

Keith Nonemaker

A message that scrolls across the screen can be quite an attention-grabber. This useful program converts messages up to 250 characters long into large sprite characters, which then glide smoothly over the screen.

"Marquee" continuously displays a scrolling message on the screen of your 128. This could prove a useful advertising tool—you might place the screen in the window of a small business. Or try using it as a message board at home—you could leave messages for the others in your family.

Setting It Up

Marquee is written entirely in BASIC; type it in and DSAVE it before running it. First, you're asked where you want the message to appear on the screen (how far from the top of the screen). You're also prompted to input the scrolling speed. To use the default values, simply press RETURN. Defaults may be changed permanently by altering lines 140 and 180.

The second screen allows you to choose colors for the message, background, and border. Again, defaults are provided, but you can change them permanently by altering lines 310, 330, and 350.

The third screen asks you to type the message to be printed. It can contain up to 250 characters. Letters, numbers, and any punctuation mark except the asterisk can be used. Some graphics characters will display properly, but avoid characters that require the bottom line of the character matrix.

Next, as the sprite data is calculated, you'll see a timer count down to zero. Data creation requires about four seconds for each unique character, which is rather time-consuming. However, duplicate characters are created almost instantaneously, so don't be surprised if the timer seems to jump ahead suddenly now and then.

Finally, the screen is cleared and the scrolling message begins. The message will continue until the RUN/STOP key is depressed. Even then, about five characters can be "caught" and will continue to move even as you proceed with other programming. If you want to eliminate those characters, use the RUN/STOP–RESTORE combination.

Marquee

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
QX 100 DIM C1$(256),G$(256),J(256)
QM 110 PRINT"{CLR}{3 DOWN}{16 SPACES}MARQUEE"
SA 120 PRINT:PRINT:PRINT:PRINT:PRINT
SD 130 INPUT"ENTER DISTANCE OF DISPLAY FROM TOP
       {6 SPACES}(RANGE: 50-200; DEFAULT: 100)";H$
BM 140 H=VAL(H$):IF H=0 THEN H=100
CM 150 IF H<50 THEN H=50
BX 160 IF H>200 THEN H=200
EC 170 PRINT:INPUT"ENTER SPEED{29 SPACES}(RANGE: 4
       -7; DEFAULT: 5)";SP$
DS 180 SP=VAL(SP$):IF SP=0 THEN SP=5
DH 190 IF SP<4 THEN SP=4
MK 200 IF SP>7 THEN SP=7
XE 210 PRINT"{CLR}{8 SPACES}COLOR MENU"
HX 220 PRINT"{3 SPACES}1) BLACK{4 SPACES}9) ORANGE
       "
JM 230 PRINT"{3 SPACES}2) WHITE{3 SPACES}10) BROWN
       "
AK 240 PRINT"{3 SPACES}3) RED{5 SPACES}11) LIGHT R
       ED"
EA 250 PRINT"{3 SPACES}4) CYAN{4 SPACES}12) DARK G
       RAY"
JJ 260 PRINT"{3 SPACES}5) PURPLE{2 SPACES}13) MEDI
       UM GRAY"
RQ 270 PRINT"{3 SPACES}6) GREEN{3 SPACES}14) LIGHT
        GREEN"
RC 280 PRINT"{3 SPACES}7) BLUE{4 SPACES}15) LIGHT
       {SPACE}BLUE"
QJ 290 PRINT"{3 SPACES}8) YELLOW{2 SPACES}16) LIGH
       T GRAY "
RK 300 PRINT:INPUT"ENTER BACKGROUND COLOR
       {18 SPACES}(DEFAULT 12)";CL$(0)
QS 310 CL(0)=VAL(CL$(0)):IF CL(0)=0 THEN CL(0)=12
HD 320 PRINT:INPUT"ENTER BORDER COLOR{22 SPACES}(D
       EFAULT 14)";CL$(4)
PD 330 CL(4)=VAL(CL$(4)):IF CL(4)=0 THEN CL(4)=14
QM 340 PRINT:INPUT"ENTER MARQUEE COLOR{21 SPACES}(
       DEFAULT 1)";CL$(2)
EB 350 CL(2)=VAL(CL$(2)):IF CL(2)=0 THEN CL(2)=1
DJ 360 PRINT"{CLR}ENTER YOUR MESSAGE:"
```

```
QB 370 PRINT "{RVS} {OFF}";
AJ 380 GETKEY B$
BK 390 IF B$=CHR$(17) OR B$=CHR$(145) OR{7 SPACES}
       B$=CHR$(157) OR B$=CHR$(29)THEN 380
PH 400 PRINT "{LEFT}";B$;
JD 410 IF B$=CHR$(34) THEN PRINT CHR$(34)CHR$(20);
MR 420 IF B$=CHR$(13) AND A$<>"" THEN 460
RC 430 IF B$=CHR$(13) THEN 360
SH 440 IF B$<>CHR$(20) THEN A$=A$+B$:GOTO370
GC 450 L=LEN(A$):A$=LEFT$(A$,L-1):GOTO 370
KM 460 PRINT "{CLR}{2 DOWN}";A$:A$=A$+"{5 SPACES}"
KB 470 WINDOW Ø,Ø,39,1,1:L=LEN(A$):BANK 14
RJ 480 A=A+1
DF 490 PRINT "{CLR}COUNTDOWN . .";4*L-4*A+4
QA 500 C$=MID$(A$,A,1)
SM 510 N=ASC(C$):IF N>64 THEN N=N-64
BA 520 IF J(N)=1 THEN 750
GK 530 CM=Ø:J(N)=1
QH 540 IF N=44 OR N=59 THEN CM=1
MM 550 FOR F=1 TO 8
ME 560 B=PEEK(53247+8*N+F):R$=CHR$(B)
GB 570 C1$(N)=C1$(N)+R$
MK 580 NEXT F
RP 590 D$=""
AM 600 FOR D=1+CM TO 21+CM
JD 610 C2=ASC(MID$(C1$(N),D,1))
GE 620 C6=(C2 AND 1)*7+(C2 AND 2)*56/2+(C2 AND 4)*
       192/4
GJ 630 IF D=6 THEN{2 SPACES}PRINT "{CLR}COUNTDOWN
       {SPACE}. .";4*L-4*A+3
FD 640 IF D=12 THEN{2 SPACES}PRINT "{CLR}COUNTDOWN
        . .";4*L-4*A+2
XB 650 IF D=18 THEN{2 SPACES}PRINT "{CLR}COUNTDOWN
        . .";4*L-4*A+1
RA 660 C5=(C2 AND 4)*1/4+(C2 AND 8)*14/8+(C2 AND 1
       6)*112/16+(C2 AND 32)*128/32
JX 670 C4=(C2 AND 32)*3/32+(C2 AND 64)*28/64+(C2 A
       ND 128)*224/128
QK 680 D$=D$+CHR$(C4)+CHR$(C5)+CHR$(C6)
PX 690 NEXT D
SQ 700 E$="":F$=""
KB 710 FOR D=1TO21 STEP 3
RC 720 E$=MID$(D$,D,3):F$=F$+E$+E$+E$
KB 730 NEXT D
MP 740 G$(N)=F$
DD 750 IF A<L THEN 480
QB 760 COLOR Ø,CL(Ø):COLOR 4,CL(4)
FE 770 WINDOW Ø,Ø,39,24,1
XF 780 FOR K=1TO8:MOVSPR K,270#SP:NEXT K
```

```
KX 790 SN=0:Q=L-3
BS 800 Q=Q+1:IF Q>L THEN Q=1
BD 810 N=ASC(MID$(A$,Q,1))
JS 820 IF N>64 THEN N=N-64
DX 830 SN=SN+1:IF SN>8 THEN SN=1
SX 840 SO=SN-6:IF SO<1 THEN SO=SO+8
JG 850 MOVSPR SN,320,H
KC 860 SPRSAV G$(N),SN
HS 870 IF MID$(A$,Q,1)="," OR MID$(A$,Q,1)=";"
       {2 SPACES}THEN MOVSPR SN,+0,+6
SR 880 SPRITE SN,1,CL(2),1,1,1:SPRITE SO,0
FC 890 FOR CT=1 TO (7-SP)*15:NEXT CT
FR 900 GOTO 800
```

80-Column Character Editor

Harry Rivera

Create a complete custom character set for the 128's 80-column screen with this useful program. When you're satisfied with the new characters, you can save them to disk to work with other programs. Also included is a boot program for loading the custom characters into memory.

The 128's 80-column screen is an enigma. The 80-column chip has its own private 16K of memory for screen memory, color memory, and two character sets. To change the character set, you might think you could just find the memory and POKE new values to it. But it's not that simple.

The 16K used by the 80-column screen is outside the realm of the normal memory of the 128. The only way to read or write values there is to POKE certain numbers to 80-column chip registers via locations 54784 and 54785 ($D600 and $D601 in hexadecimal). These two locations act as a gateway to 80-column memory.

It's possible to design your own characters for the 80-column screen, but it's a little more difficult than doing it in 40-column mode. Program 1, "80-Column Character Editor," solves the problem and lets you concentrate on creating a new character set. When you're finished, the character set can be stored on a disk and easily loaded into memory.

A Better-Looking Character Set

If you hook up a 128 to a monochrome or RGBI monitor, its 640 × 200 dot resolution is the same as you'll find on an IBM PC with a color/graphics board (as a matter of fact, you can use an IBM-compatible color monitor with the 128). If you look closely at the characters of an IBM and the Commodore 128, you'll notice that they're not the same. The IBM's characters appear to be more stylish or fancier.

I originally wrote 80-Column Character Editor to create an IBM-style character set for my 128. It's also good for adding foreign language characters and accents, like é and ö, or for creating specialized graphics.

Two programs accompany this article: one in BASIC, the other in machine language. Program 1 is the character editor, which allows you to create, edit, and save a custom character set. It's written in BASIC, with a machine language routine contained in DATA statements. There are no special instructions for typing it except that you should be in 128 mode. If you use "The Automatic Proofreader," Appendix B, you'll be able to avoid typing mistakes.

The second program, "Boot80," is a very short machine language program. You may use the "MLX" machine language editor, Appendix C, to enter it. Answer the prompts as follows:

Starting address: 0C00
Ending address: 0C77

Because it's so short, you may prefer to type it in with the built-in machine language monitor. If you do this, ignore the final number on each line (the rightmost number is a checksum used by MLX). After entering the numbers, save the routine to disk using the monitor Save command:

```
S "BOOT80", 8, 00C00, 00C78
```

The Menu of Commands

The program is very easy to use. All command keys are listed on the screen, so you don't have to refer back to this article if you forget them. After you have entered and saved the program, type RUN to start the editor. The full character set, including uppercase and lowercase letters, will appear on the screen along with the list of commands and a blank grid.

Using the arrow keys, just move the cursor to the character you want to change. Next, press RETURN (or E for Edit) to place that character on the grid. The cursor will move to the grid. Using the space bar, you can turn any of the dots in the grid on or off. After constructing your new character, press RETURN again to save it in the character set.

If you don't like the character you make on the grid and want to start over or move to another character, press ESC and that character will be discarded. Pressing the SHIFT–CLR/HOME combination clears the grid. The B key puts that character into a memory buffer, and pressing C retrieves the character shape you saved to the buffer. This is useful when you want to copy the same character design into two or more characters.

To save the character set, press the @ (*at* sign) key. A message will appear on the bottom of the screen: Pressing ESC cancels the save; any other key saves the complete character set to disk using the filename CHAR.SET. If you need to load a character set to change or finish it, press L. The same message will appear, and you can proceed in the same way.

Once you've made a character set, loading it into memory is a cinch. First, you should have saved the character set on the disk under the name CHAR.SET. You should also have a copy of Program 2 on the same disk, saved with the name BOOT80. To load and initialize the character set, enter **BLOAD"BOOT80"** followed by **SYS 3072**. An even faster way to load and SYS is simply to type **BOOT "BOOT 80"**. It takes just a second or two to set up the new character set.

How It Works

As mentioned above, the chip that controls the 80-column screen is quite different from the familiar VIC-II chip of the Commodore 64 or 128. The 80-column screen is driven by the 8563 VDC video chip, which offers a number of advanced features. First is its ability to display 80 columns and 16 colors. It's capable of working while the 128 is in FAST mode (2 megahertz clock speed), whereas the VIC-II can work only at 1 MHz. The 8563 also allows you to have characters from both the uppercase/graphics and lowercase/uppercase set on the screen at the same time, as well as underlined and flashing characters. One of the 8563's main advantages is that it has its own video RAM and character memory (which is copied from ROM when the computer is turned on). None of the Commodore's 128K of memory is used for the character set. The only drawback is that the 16K of memory from this chip must be addressed indirectly.

Here's a layout of the 80-column internal RAM:

$0000–$07CF	**Video RAM**
$0800–$0FCF	**Attribute RAM**
$2000–$3FFF	**Character RAM**

These addresses are in hexadecimal. We're interested in locations $2000–$3FFF, where the character-set definitions are found. There are 16 bytes available for each character, even though only the first eight are actually used. The question is, How do you transfer the new character definitions into this section of VDC memory?

Reading and Writing

The operations of the 8563 chip are controlled by the values in the chip's 37 internal registers. However, the chip has only two addresses in the 128's normal memory space: 54784 and 54785 ($D600 and $D601). Remember that the 128 must be set for a bank where the I/O space is visible, such as bank 15. To place a value in a register, you must store the register number (0–36) in $D600, then wait until bit 7 of $D600 is set to 1. Then you store the value you want to place in the register into $D601.

In order to write to or read from any location in the VDC chip's memory, you must first put the address of the desired memory location into registers 18–19 ($12–$13) in high-byte/low-byte order (just the opposite of what you usually do in machine language, where the low byte comes first). Then read from or write to register 31 ($1F). When you read from this register, the value there is the contents of the memory location addressed in locations 18–19. When you write to register 31, the value you send to the register is forwarded to the memory location addressed in registers 18–19. This program fragment shows the proper machine language procedure for writing a byte.

```
        LDX  #$12     ;Tell VDC you are writing
        STX  $D600    ; to register 18
WAIT1   BIT  $D600    ;Wait until VDC is ready
        BPL  WAIT1
        LDA  #$20     ;High byte of address ($2000)
        STA  $D601    ;Send it to register 18
        INX           ;Now tell the VDC you will
        STX  $D600    ; be writing to register 19
WAIT2   BIT  $D600    ;Wait again
        BPL  $WAIT2
        LDA  #$00     ;Low byte of address
        STA  $D601    ;Send it to register 19
        LDX  #$1F     ;Now set VDC for writing
        STX  $D600    ; to register 31
```

```
WAIT3  BIT   $D600    ;Wait again
       BPL   WAIT3
       LDA   #byte    ;Load byte to write
       STA   $D601    ;Store byte in VDC RAM
```

The procedure for reading a location is almost the same. Just replace the last two instructions (LDA *#byte*: STA $D601) with

LDA $D601 ;Read that location

Note that if you're reading or writing a sequential series of memory locations, you have to set up the address in registers 18–19 only once. Each time a value is written to or read from register 31, the value in 18–19 is automatically incremented, so each successive read or write of register 31 will read or write the next higher memory address.

This same technique could be adapted to POKE values directly to screen or attribute memory (which is similar to the VIC-II's color memory, but it also controls flashing, underlining, and other character attributes).

Program 1. 80-Column Character Editor

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
HA 1Ø  FAST:COLOR6,1:GRAPHIC1,1:DIM B(8,8),BF(8,8):
       CP=Ø:U$=CHR$(142)
MK 2Ø  FORI=ØTO192:READX:POKEDEC("DØØ")+I,X:S=S+X:N
       EXT:IFS<>24972THENPRINTCHR$(14)"ERROR IN DAT
       A"+CHR$(142):END:ELSESYSDEC("D46"):GOTO14Ø
JM 3Ø  DATA 142,Ø,214,44,Ø,214,16,251,141,1,214,96,
       162,18,169,Ø,32,Ø,13,232,169,Ø,32,Ø,13,162,3
       1,169,1,32,Ø,13,162,18,76,Ø,13
QP 4Ø  DATA 134,251,132,252,162,18,165,252,32,Ø,13,
       232,165,251,32,Ø,13,162,31,142,Ø,214,44,Ø,21
       4,16,251,173,1,214,133,25Ø,96
HS 5Ø  DATA 169,Ø,16Ø,2Ø8,133,218,132,219,133,25Ø,1
       69,32,133,251,16Ø,Ø,162,14,169,218,32,116,25
       5,145,25Ø,2ØØ,192,8,144,242,24,165,25Ø
HJ 6Ø  DATA 1Ø5,8,133,25Ø,144,2,23Ø,251,24,165,218,
       1Ø5,8,133,218,144,22Ø,23Ø,219,165,219,2Ø1,22
       4,144,212,96
HA 7Ø  DATA 169,Ø,16Ø,32,133,218,132,219,162,18,169
       ,32,32,2Ø4,2Ø5,232,169,Ø,32,2Ø4,2Ø5,16Ø,Ø,16
       2,14,169,218,32,116,255,32,2Ø2,2Ø5
XC 8Ø  DATA 2ØØ,192,8,144,241,169,Ø,32,2Ø2,2Ø5,136,
       2Ø8,25Ø,24,165,218,1Ø5,8,133,218,144,224,23Ø
       ,219,165,219,2Ø1,48,144,216,96
AH 9Ø  POKEDEC("D15"),ADAND255
QS 1ØØ POKEDEC("DØF"),AD/256
KD 11Ø POKEDEC("D1C"),BI
JQ 12Ø SYSDEC("DØC")
SS 13Ø RETURN
PB 14Ø PRINT"{CLR}{WHT}";TAB(7)U$;"* C O M M O D O
        R E{3 SPACES}1 2 8{5 SPACES}C H A R A C T
       {SPACE}E R{3 SPACES}E D I T O R *"
BJ 15Ø PRINTU$;TAB(8)"{2 DOWN}[7]@ A B C D E F G H
        I J K L M N O P Q R S T U V W X Y Z [ £ ]
        ↑ ←"
PG 16Ø PRINTU$;TAB(8)"{2 SPACES}! ";CHR$(34)" # $
       {SPACE}% & ' ( ) * + , - . / Ø 1 2 3 4 5 6
       {SPACE}7 8 9 : ; < = > ?
SQ 17Ø PRINTU$;TAB(8)"* A B C D E F G H I J K L M
       {SPACE}N O P Q R S T U V W X Y Z + [-] - ↑
       {SPACE}[*]"
FK 18Ø PRINTU$;TAB(8)"{2 SPACES}[K] [I] [T] [@]
       [G] [+] [M] [£] £ [N] [Q] [D] [Z] [S] [P]
        [A] [E] [R] [W] [H] [J] [L] [Y] [U] [O] @
       {SPACE}[F] [C] [X] [V] [B]{DOWN}"
XJ 19Ø PRINTTAB(8)"{N}@ A B C D E F G H I J K L M
       {SPACE}N O P Q R S T U V W X Y Z [ £ ] ↑ ←
       "
DE 2ØØ PRINTTAB(8)"{2 SPACES}! ";CHR$(34)" # $ % &
        ' ( ) * + , - . / Ø 1 2 3 4 5 6 7 8 9 : ;
       {SPACE}< = > ?
FS 21Ø PRINTTAB(8)"* A B C D E F G H I J K L M N O
        P Q R S T U V W X Y Z + [-] - ↑ [*]"
PG 22Ø PRINTTAB(8)"{2 SPACES}[K] [I] [T] [@] [G]
       [+] [M] [£] £ [N] [Q] [D] [Z] [S] [P] [A]
        [E] [R] [W] [H] [J] [L] [Y] [U] [O] @ [F]
       {SPACE}[C] [X] [V] [B]"
FJ 23Ø SYS52332,,14,5Ø
MD 24Ø PRINTCHR$(14)"{CYN}<ENTER> - [4]SAVE CHARAC
       TER":PRINTTAB(52)"{CYN}<CLR> - [4]CLEAR CHA
       RCTER":PRINTTAB(5Ø)"{CYN}<SPACE> - [4]TOGGL
       E BIT":PRINTTAB(52)"{CYN}<ESC> - [4]ABORT"
MA 25Ø PRINTTAB(56)"{DOWN}[3]Q - {RED}QUIT"
FD 26Ø SYS52332,,14,Ø
MX 27Ø PRINTCHR$(14)"{3 SPACES}{CYN}E - [4]EDIT CH
       ARACTER":PRINT"{3 SPACES}{CYN}S - [4]SWITCH
        CHARACTER SETS":PRINT"{3 SPACES}{CYN}@ -
       [4]SAVE CHARACTER SET":PRINT"{3 SPACES}
       {CYN}L - [4]LOAD CHARACTER SET"
```

```
HP 280 PRINTCHR$(14)"{DOWN}{3 SPACES}{CYN}C - E4}C
       OPY CHARACTER FROM BUFFER":PRINT"{3 SPACES}
       {CYN}B - E4}COPY CHARACTER TO BUFFER"
EK 290 SYS52332,,13,1:FORX=1TO8:PRINTTAB(36)"{WHT}
       ........":NEXT:FORX=1TO8:FORY=1TO8:B(X,Y)=Ø
       :NEXTY,X:PS=1:LN=1:CH=1:CL=1:OF=Ø
BH 300 IFOF=ØTHENSYS52332,,2+CL,6+(CH*2):ELSE SYS5
       2332,,7+CL,6+(CH*2)
QS 310 PRINTCHR$(27)"F"CHR$(27)"S";
CF 320 GETKEYA$
CR 330 IFA$="{UP}"THENCL=CL-1:IFCL<1THENCL=4
XR 340 IFA$="{DOWN}"THENCL=CL+1:IFCL>4THENCL=1
PA 350 IFA$="{LEFT}"THENCH=CH-1:IFCH<1THENCH=32
RS 360 IFA$="{RIGHT}"THENCH=CH+1:IFCH>32THENCH=1
GD 370 IFA$=CHR$(13)ORA$="E"THENGOSUB59Ø:GOSUB54Ø
AG 380 IFA$="@"THENGOSUB67Ø
BG 390 IFA$="L"THENGOSUB7ØØ
CM 400 IFA$="S"THENBEGIN:IFOF=ØTHENOF=2Ø48:ELSEOF=
       Ø:BEND
DB 410 IFA$="Q"THENPRINT"{17 DOWN}"U$:END
CP 420 GOTO3ØØ
DM 430 GETKEYA$
MR 440 IFA$="{DOWN}"THENLN=LN+1:IFLN>8THENLN=1
FC 450 IFA$="{UP}"THENLN=LN-1:IFLN<1THENLN=8
BS 460 IFA$="{RIGHT}"THENPS=PS+1:IFPS>8THENPS=1
CF 470 IFA$="{LEFT}"THENPS=PS-1:IFPS<1THENPS=8
MS 480 IFA$=" "THEN BEGIN:IFB(PS,LN)=Ø THEN B(PS,L
       N)=1:PRINT"{RVS}{YEL} {OFF}{LEFT}";: ELSE B
       (PS,LN)=Ø:PRINT"{WHT}.{LEFT}";:BEND
KK 490 IFA$=CHR$(13)THEN56Ø
GS 500 IFA$=CHR$(27)THENSYS 52332,,13,1:GOTO58Ø
EF 510 IFA$="{CLR}"THENSYS52332,,13,1:FORX=1TO8:PR
       INTTAB(36)"{WHT}........":NEXT:FORX=1TO8:FO
       RY=1TO8:B(X,Y)=Ø:NEXTY,X:
AE 520 IFA$="C"THENGOSUB79Ø
BQ 530 IFA$="B"THENGOSUB73Ø
BC 540 SYS52332,,LN+12,PS+35:PRINTCHR$(27)"U";
HG 550 GOTO43Ø
FX 560 BS=DEC("2ØØØ"):FD=BS+(8*(((CL-1)*32)+(CH-1)
       )-1)+1+OF:FORAD=FDTOFD+7:BT=Ø:RV=Ø:FORB=7TO
       ØSTEP-1:IFB(8-B,AD-FD+1)>ØTHENBT=BT+2↑B:ELS
       ERV=RV+2↑B
HM 570 NEXTB:POKEAD,BT:POKEAD+1Ø24,RV:NEXTAD:SYSDE
       C("D81"):SYS52332,,12,1
SJ 580 FORX=1TO8:PRINTTAB(36)"{WHT}........":NEXT:
       FORX=1TO8:FORY=1TO8:B(X,Y)=Ø:NEXTY,X:RETURN
CM 590 IFOF>ØTHENBS=DEC("2ØØØ"):FD=BS+(16*(((CL-1)
       *32)+(CH-1))-1)+1+OF+2Ø48:FORY=1TO8:FORX=1T
       O8:B(X,Y)=Ø:NEXTX,Y:GOTO61Ø
```

```
CC 600  BS=DEC("2000"):FD=BS+(16*(((CL-1)*32)+(CH-1
        ))-1)+1+OF:FORY=1TO8:FORX=1TO8:B(X,Y)=0:NEX
        TX,Y
PE 610  FORAD=FDTOFD+7:SYSDEC("D25"),,ADAND255,AD/2
        56:IFPEEK(250)>0THEN630
SB 620  NEXTAD:GOTO650
PF 630  FORB=7TO0STEP-1:IF(PEEK(250)AND2↑B) >0THENB
        (8-B,AD-FD+1)=1
AK 640  NEXTB:GOTO620
GD 650  FORY=1TO8:FORX=1TO8:SYS52332,,12+Y,35+X:IF
        {SPACE}B(X,Y)=1 THEN PRINT"{RVS}{YEL} {OFF}
        ";: ELSE PRINT"{WHT}.";
BA 660  NEXTX,Y:RETURN
ME 670  SYS52332,,23,15:PRINTCHR$(14)CHR$(7)"{6}PRE
        SS <ESC> TO ABORT OR ANY OTHER KEY TO SAVE.
        ..";:GETKEYA$
RJ 680  IFA$=CHR$(27)THENSYS52332,,23,1:PRINTA$"Q";
        :RETURN
SJ 690  SCRATCH"CHAR.SET":BSAVE"CHAR.SET",P8192 TO
        {SPACE}P12288:SYS52332,,23,1:PRINTCHR$(27)"
        Q";:RETURN
EG 700  SYS52332,,23,15:PRINTCHR$(14)CHR$(7)"{6}PRE
        SS <ESC> TO ABORT OR ANY OTHER KEY TO LOAD.
        ..";:GETKEYA$
SK 710  IFA$=CHR$(27) THEN SYS52332,,23,1:PRINTA$"Q
        ";:RETURN
CC 720  BLOAD"CHAR.SET",P8192:SYSDEC("D81"):SYS5233
        2,,22,1:PRINTCHR$(27)"Q";:RETURN
AK 730  IFOF=0THENBS=DEC("2000"):FD=BS+(16*(((CL-1)
        *32)+(CH-1))-1)+1+OF:FORY=1TO8:FORX=1TO8:BF
        (X,Y)=0:NEXTX,Y:GOTO750
BH 740  BS=DEC("2000"):FD=BS+(16*(((CL-1)*32)+(CH-1
        ))-1)+1+OF+2048:FORY=1TO8:FORX=1TO8:BF(X,Y)
        =0:NEXTX,Y
RF 750  FORAD=FDTOFD+7:SYSDEC("D25"),,ADAND255,AD/2
        56:IFPEEK(250)>0THEN770
KA 760  NEXTAD: RETURN
HX 770  FORB=7TO0STEP-1:IF(PEEK(250)AND2↑B) >0THENB
        F(8-B,AD-FD+1)=1
CP 780  NEXTB:GOTO760
QS 790  FORY=1TO8:FORX=1TO8:B(X,Y)=BF(X,Y):NEXTX,Y
FQ 800  FORY=1TO8:FORX=1TO8:SYS52332,,12+Y,35+X:IFB
        (X,Y)=1THENPRINT"{RVS}{YEL} {OFF}";:ELSEPRI
        NT"{WHT}.";
XJ 810  NEXTX,Y:RETURN
```

Program 2. Boot80

See instructions in article, and read Appendix C, "MLX," before typing in the following program listing.

Starting address: 0C00
Ending address: 0C77

```
ØCØØ:4C 1Ø ØC 43 48 41 52 2E 12
ØCØ8:53 45 54 FF ØØ FF ØØ FF A5
ØC1Ø:A9 ØC AA 2Ø 68 FF A9 Ø1 EE
ØC18:A2 Ø8 AØ ØØ 2Ø BA FF A9 2D
ØC2Ø:Ø8 A2 Ø3 AØ ØC 2Ø BD FF AB
ØC28:A9 ØØ A2 ØØ AØ 2Ø 2Ø D5 Ø5
ØC3Ø:FF 2Ø E7 FF A9 ØØ AØ 2Ø FB
ØC38:85 DA 84 DB A2 12 A9 2Ø E8
ØC4Ø:2Ø CC CD E8 A9 ØØ 2Ø CC 3E
ØC48:CD AØ ØØ A2 ØE A9 DA 2Ø 86
ØC5Ø:74 FF 2Ø CA CD C8 CØ Ø8 6E
ØC58:9Ø F1 A9 ØØ 2Ø CA CD 88 BA
ØC6Ø:DØ FA 18 A5 DA 69 Ø8 85 ØF
ØC68:DA 9Ø EØ E6 DB A5 DB C9 93
ØC7Ø:3Ø 9Ø D8 6Ø ØØ ØØ ØØ ØØ E5
```

Chapter 3

Art and Music

Sound Designer

Mark W. Pemburn

Here's an easy way to experiment with sound on your 128 and instantly save anything you like. Menus and a simulated audio control board make this an especially efficient program.

If you've spent any time experimenting with the 128's BASIC 7.0, you've probably noticed how easy it has become to use graphics and sound with instructions like DRAW, CIRCLE, BOX, PLAY, FILTER, and so on. "Sound Designer" was written to illustrate the use of these and other 7.0 instructions as well as to further simplify the use of the SOUND statement.

The SOUND statement controls the 128's Sound Interface Device (SID) chip, a sophisticated audio synthesizer circuit which is capable of generating a variety of sound waveforms, filtering them, pulse modulating them, jamming, squeezing, and otherwise manipulating them into all kinds of sounds.

SOUND has eight variable parameters, three of them essential (voice, frequency, and duration) and the rest optional (direction of sweep, minimum sweep frequency, step value, waveform, and pulse width). In the program, they're abbreviated VO, FR, DU, DI, MN, SP, WF, and PW. Here are the ranges for each of these parameters:

VO	**1–3**
FR	**0–65535 Hz (hertz)**
DU	**0–32767 jiffies (1 jiffy = 1/60 second)**
DI	**Up (0), down (1), or oscillate (2)**
MN	**0–65535 Hz**
SP	**Any value not larger than the main frequency minus the minimum frequency (see "Sound Parameters," below)**
WF	**0 (triangle), 1 (sawtooth), 2 (square), and 3 (noise)**
PW	**0–4095 (for use with square waves only)**

This program was spawned when I was attempting to generate a specific sound by moving these parameters up and down using the examples found in the 128 user's manual and elsewhere. It turned out to be a tedious trial-and-error process. Numbers alone do not reflect the nature of a sound. What's needed is an analog-type display that can be tested until the desired sound is achieved. This is where the graphic instructions come in handy.

Using the Program

To use Sound Designer, type it in, save a copy to disk, and type RUN. The first time you run the program, remove the GOTO statement in line 1 by placing a REM statement at the beginning of the line. This will allow you to create a CATALOG for your sounds by executing the commands in lines 2 and 3. Then, after the program has been run once, remove the REM from the beginning of line 1 so that it reads GOTO10.

The main menu presents these options:

+ TO VIEW THE BOARD
− TO VIEW TEXT
V TO VIEW VARIABLES
↑ TO VIEW CATALOG
*** TO SAVE SOUND**
£ TO LOAD SOUND
Q TO QUIT

CLR/HOME TO VIEW THIS MENU

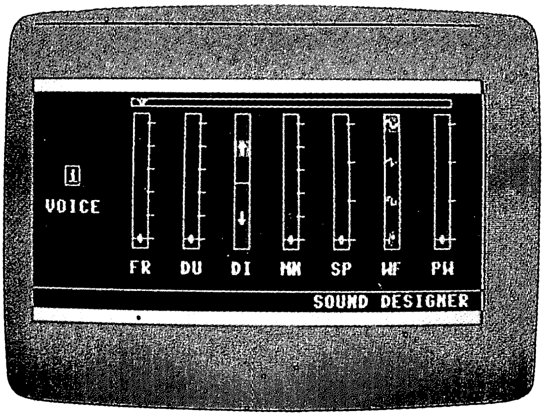
PRESS SPACE BAR TO HEAR SOUND

Press V to view the variables you can use to create the sound in your program. You may use any of these options at any time, but bear in mind that SAVE and LOAD will prompt you for a filename. If you decide not to enter a filename, the RETURN key will take you back to the main menu. If you wish to save a sound, use any valid filename, and the variables will be saved to disk as a sequential file.

The control board's analog-type display lets you easily test sounds.

The Control Board

Pressing the + key whisks the menu away and displays the control board. On the far left is a box containing the number 1 with the word *VOICE* displayed beneath. This is the default voice number, but it can be changed to 2 or 3. The other seven controls allow you to change the sound parameters within the ranges described above. Use the left- and right-cursor keys to position the green pointer over each control, and the up- and down-cursor keys to change each of the parameters. To test each change, press the space bar. (Note that



you'll not hear any sound when all controls are at zero, and you'll get only a feeble sound when the marker for waveform is on the symbol for square wave and the value of PW, pulse width, is zero. Other than this, you have a broad palette of sounds from which to choose.) Press — to return to the menu from the BOARD.

Saving and Loading Sounds

Once you have a sound you want to save, press the minus (—) key to return to the text screen and the asterisk (*) to get the input screen. Enter an appropriate filename for the sound, press RETURN, and that's it. To verify your save, press the up-arrow key (↑), and the catalog screen will appear and display the filename. Note that the area containing the catalog titles is a "window," a special screen area made possible by the WINDOW statement. This statement defines a special area of the screen within which all printing and scrolling will take place after the statement is issued. If you halt the program at this point, the window will remain in place and the text will stay within its borders. To return to the main screen and erase the window, press the unshifted CLR/HOME key twice.

You may call a sound that's been saved by pressing the British pound (£) key and entering a filename. (A misspelled filename will return the message FILE NOT FOUND ON THIS DISK, and you'll be returned to the menu. Check the catalog for spelling and try again.)

Remember that values saved by the SOUND program are stored in a sequential file and, true to the name of this type of file, these values can only be retrieved in the same order that they went in.

By studying the listing, which is mostly in BASIC, you can see how useful the 128's sound and graphics instructions can be.

Sound Parameters

Philip I. Nelson

The 128's SOUND statement is extremely versatile. But its versatility can make it look intimidating at first, since SOUND can take as few as three or as many as eight different parameters (controlling values). Here's a brief explanation of what each SOUND parameter does.

Every SOUND statement must be followed by at least three parameters: a *voice* value that picks one of the 128's three voices, a *frequency* value that chooses a pitch for the sound (whether it sounds high or low), and a *duration* value that controls how long the sound lasts. Since the duration is expressed in sixtieths of a second (or jiffies), a duration of 60 lasts for one second, a duration of 3600 lasts one minute, and so on. Here's a simple SOUND statement that uses only three parameters:

SOUND 1, 2000, 10

This example uses voice 1, sets the frequency to 2000, and chooses a duration of 10 jiffies (1/6 second). By including additional parameters, you can make the sound move up or down automatically and also select different waveforms for greater variety. All of the additional parameters are optional: If you leave them out, the 128 won't signal an error of any kind.

The fourth value in a SOUND statement represents the sound's *sweep direction*. You have three choices for the sweep: The sound can sweep upward from a low pitch to a high one, it can sweep downward from a high pitch to a low one, or it can oscillate, meaning that it sweeps up, then down, then back up, and so on (like a police or ambulance siren).

Whenever you specify a sweep direction, the 128 uses the main frequency value (the second parameter) as the upper limit of the sound sweep. To set the minimum or bottom limit for the sweep, you must supply a fifth parameter, the *minimum frequency*. The sound sweeps up or down between this frequency and the main frequency. Since it sets the bottom limit, the minimum frequency must always be smaller than the value you choose for the main frequency.

The sixth parameter, *step*, is very important when sound sweeps are involved. Like the STEP number in a FOR-NEXT loop, this value controls the size of the steps in a sound sweep. Larger values make the sweep move faster, and smaller ones make the sweep more gradual. Watch out for the "impossible step" error, which applies here just as it does in FOR-NEXT loops. In order for the sweep to work correctly, the step value must be smaller than the difference between the main frequency and the minimum frequency. For instance, if the main frequency is 6000 and the minimum is 2000, the largest sensible step value is 4000 (6000−2000).

Thus, to set up a sound sweep, you must supply three extra values: a direction for the sweep, a minimum frequency to set the sweep's lower limit, and a step value to control how fast the sound warbles between the upper and lower frequencies.

The seventh parameter, *waveform*, chooses one of the 128's four basic waveforms: triangle, sawtooth, pulse, and noise. Don't confuse waveforms with voices. Since the SID chip has three separate voices (tone generators), the 128 can make as many as three different sounds at once, like a three-fingered piano chord. If you don't want to produce simultaneous sounds, you don't need to use more than one voice. The waveform determines what *kind* of sound a given voice makes—its *timbre*. If you don't specify a waveform, the 128 chooses one for you automatically. By supplying a waveform value, you can change the character of the sound.

The pulse wave (often called a square wave) is different from the other three waveforms. By changing the width of the pulse wave, you can make it sound thin and hollow, strong and full, or anything in between those two extremes. The eighth parameter, *pulse width*, controls the width of the pulse wave, and is meaningful only when you choose the pulse waveform.

Though SOUND statements are very flexible, there are certain things they can't do. For instance, SOUND has no built-in means for using the SID filter, controlling ring modulation or synchronization, or creating advanced effects such as envelope following. You can learn more about those subjects in *COMPUTE!'s 128 Programmer's Guide*, available from COMPUTE! Books.

Sound Designer

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
JR 1 GOTO10:REM FOR THE FIRST RUN, REMOVE LINE 1 T
     O CREATE A CATALOG FILE
CF 2 DOPEN#1,"CATALOG",D0,U8,W:IF DS<>0 THEN STOP
HR 3 PRINT#1,"************":DCLOSE#1
DD 10 DIM W$(5),L$(9)
GA 20 X=93:Y=163
GG 30 GOSUB2190:REM LOAD SPRITE DATA
AX 40 FAST:VO=1:FR=0:DU=0:DI=0:MN=0:SP=0:WF=0:PW=0
QM 50 GRAPHIC 1,1
QG 60 COLOR 0,1
FK 70 COLOR 1,12
CP 80 COLOR 4,12
KH 90 CIRCLE 1,82,100,2,2,270,90:CIRCLE 1,86,100,2
      ,2,90,270:REM * SINE *
BJ 100 DRAW 1,100,100 TO 104,98 TO 104,102 TO 107,
       100:REM * SAWTOOTH *
FA 110 DRAW 1,120,100 TO 120,98 TO 123,98 TO 123,1
       02 TO 126,102 TO 126,100:REM ****** SQUARE
       {SPACE}*
BQ 120 DRAW 1,140,100 TO 141,102 TO 142,97 TO 143,
       103 TO 144,96 TO 145,100:REM ****** NOISE *
MX 130 A=0
XB 140 FOR S=79 TO 139 STEP 20:A=A+1:SSHAPE W$(A),
       S,96,S+10,104:NEXT
KE 150 CHAR 0,9,17,"FR DU DI MN SP WF PW"
EC 160 A=0:FOR S=72 TO 216 STEP 24:A=A+1:SSHAPE L$
       (A),S,134,S+16,144:NEXT
ME 170 GRAPHIC 1,1
RG 180 REM * DRAW.BOARD *
FP 190 FOR B=1 TO 37:DRAW 1,0,B TO 320,B:NEXT
ED 200 FOR B=154 TO 199:DRAW 1,0,B TO 320,B:NEXT
BD 210 COLOR 1,16
KM 220 DRAW 1,0,38 TO 320,38
MR 230 DRAW 1,0,142 TO 320,142
KA 240 DRAW 1,0,153 TO 320,153
SA 250 CHAR 1,25,18,"SOUND DESIGNER"
DD 260 BOX 1,22,78,32,88
GD 270 A=0:FOR S=51 TO 125 STEP 20:A=A+1
GK 280 IF A>2 THEN TF=1
JQ 290 GSHAPE W$(A),250+TF,S:NEXT
JP 300 FOR SQ=70 TO 286 STEP 36
QQ 310 BOX 1,SQ,50,SQ+11,120
RF 320 NEXT
HM 330 BOX 1,68,42,298,46
RQ 340 FOR J=80 TO 188 STEP 36
```

```
RD 350 IF J=152 THEN J=188
HJ 360 FOR F=55 TO 120 STEP 12
AB 370 DRAW 1,J,F TO J+5,F:NEXT:NEXT
JG 380 DRAW 1,142,86 TO 152,86
JS 390 CHAR 1,18,8,"↑"
EA 400 DRAW 1,147,101 TO 147,107
HD 410 DRAW 1,148,101 TO 148,107
EK 420 DRAW 1,145,105 TO 150,105
PX 430 DRAW 1,146,106 TO 149,106
MA 440 DRAW 1,250,51 TO 250,119
GH 450 FOR D=224 TO 296 STEP 72
JD 460 FOR E=55 TO 120 STEP 20
AG 470 DRAW 1,D,E TO D+5,E:NEXT:NEXT
EG 480 CHAR 1,1,12,"VOICE"
QR 490 CHAR 1,3,10,"1"
GA 500 A=0:FOR S=68 TO 284 STEP 36:A=A+1:GSHAPE L$
       (A),S,125:NEXT
XM 510 REM * INSTALL MARKERS *
KF 520 SPRITE 1,0,6
DR 530 MOVSPR 1,X+2,92
BS 540 FOR MK=2 TO 8:SPRITE MK,0,11:MOVSPR MK,X+2+
       ((MK-2)*36),Y:NEXT MK
AD 550 SPRITE 4,0,7,1:SPRITE 7,0,7,1
DH 560 MOVSPR 4,X+74,Y-49:MOVSPR 7,X+182,Y-62
MH 570 SLOW
DX 580 GRAPHIC 0,0
SR 590 GOSUB 1360
DB 600 REM * KEYBOARD INPUT *
FC 610 GETKEY CUR$
EC 620 IF CUR$="+" THEN BEGIN:GRAPHIC 1,0:FOR S=1
       {SPACE}TO 8:SPRITE S,1:NEXT:BEND
AR 630 IF CUR$="-" THEN GOSUB 1850
KH 640 IF CUR$="V" THEN GOSUB 2170
FD 650 IF CUR$="↑" THEN PRINT"{CLR}":GOSUB 1930
AB 660 IF CUR$="*" THEN GOSUB 1850:GOSUB 1530
DG 670 IF CUR$="£" THEN GOSUB 1850:GOSUB 1670
JS 680 IF CUR$="Q" THEN GOSUB 2130
MD 690 IF CUR$="{HOME}" OR CUR$="{CLR}" THEN GOSUB
        1360
PK 700 IF CUR$="1" THEN CHAR 1,3,10,"1":VO=1
FQ 710 IF CUR$="2" THEN CHAR 1,3,10,"2":VO=2
XC 720 IF CUR$="3" THEN CHAR 1,3,10,"3":VO=3
SQ 730 LIM=(RSPPOS(1,0)-X)/36
BA 740 IF ASC(CUR$)=29 AND LIM<6 THEN MOVSPR 1,+36
       ,+0
BQ 750 IF ASC(CUR$)=157 AND LIM>1{2 SPACES}THEN MO
       VSPR 1,-36,+0
AH 760 IF ASC(CUR$)=17 THEN INC=1:GOTO 810
DF 770 IF ASC(CUR$)=145 THEN INC=-1:GOTO 810
```

```
BK 780 IF CUR$=" " THEN GOSUB 1500
AH 790 GOTO 610
JS 800 REM * MOVE MARKERS *
XB 810 ON LIM+1 GOSUB 840,910,980,1060,1130,1200,1
       290
BG 820 GOTO 610
FA 830 REM * BASE FREQUENCY *
QQ 840 FY=RSPPOS(2,1)
MQ 850 FR=(FY-Y)*(-1000)
MB 860 IF FR>61500 AND INC=-1 THEN 610
DD 870 IF FR<=0 AND INC=1 THEN 610
DK 880 MOVSPR 2,+0,+INC
SX 890 RETURN
XM 900 REM * DURATION *
MB 910 DY=RSPPOS(3,1)
HX 920 DU=(DY-Y)*(-6)
GH 930 IF DU>366 AND INC=-1 THEN 610
FS 940 IF DU<=0 AND INC=1 THEN 610
HC 950 MOVSPR 3,+0,+INC
SD 960 RETURN
AG 970 REM * SWEEP DIRECTION *
QQ 980 IF INC=-1 AND TY>115 THEN MOVSPR 4,+0,-18
PK 990 IF INC=1 AND TY<150 THEN MOVSPR 4,+0,+18
SG 1000 TY=RSPPOS(4,1)
JK 1010 IF TY=114 THEN DI=0
RS 1020 IF TY=132 THEN DI=2
CX 1030 IF TY=150 THEN DI=1
DA 1040 RETURN
QM 1050 REM * MINIMUM SWEEP FREQ *
AF 1060 MY=RSPPOS(5,1)
GP 1070 MN=(MY-Y)*(-1000)
RS 1080 IF MN>61500 AND INC=-1 THEN 610
GK 1090 IF MN<=0 AND INC=1 THEN 610
MA 1100 MOVSPR 5,+0,+INC
CE 1110 RETURN
CC 1120 REM * STEP FREQUENCY *
EQ 1130 SY=RSPPOS(6,1)
GK 1140 SP=(SY-Y)*(-250)
CS 1150 IF SP>15375 AND INC=-1 THEN 610
SB 1160 IF SP<=0 AND INC=1 THEN 610
QM 1170 MOVSPR 6,+0,+INC
FM 1180 RETURN
JD 1190 REM * WAVE FORM *
BR 1200 IF INC=-1 AND WY>102 THEN MOVSPR 7,+0,-20
FE 1210 IF INC=1 AND WY<160 THEN MOVSPR 7,+0,+20
PE 1220 WY=RSPPOS(7,1)
BE 1230 IF WY=101 THEN WF=0
MJ 1240 IF WY=121 THEN WF=1
DR 1250 IF WY=141 THEN WF=2
```

```
DC 1260 IF WY=161 THEN WF=3
XB 1270 RETURN
XE 1280 REM * PULSE WIDTH *
CJ 1290 PY=RSPPOS(8,1)
GX 1300 PW=(PY-Y)*(-66)
QE 1310 IF PW>4091 AND INC=-1 THEN 610
DM 1320 IF PW<=0 AND INC=1 THEN 610
XH 1330 MOVSPR 8,+0,+INC
QF 1340 RETURN
MG 1350 REM * MAIN MENU *
DA 1360 PRINT "{CLR}{3 DOWN}{CYN}{10 SPACES}SOUND
        {SPACE}DESIGNER"
JH 1370 PRINT SPC(5)"{2 DOWN}PRESS:"
CH 1380 PRINT SPC(10)"{DOWN}+ TO VIEW BOARD"
EF 1390 PRINT SPC(10)"- TO VIEW TEXT
GJ 1400 PRINT SPC(10)"V TO VIEW VARIABLES
JS 1410 PRINT SPC(10)"↑ TO VIEW CATALOG
AE 1420 PRINT SPC(10)"* TO SAVE SOUND FILE
MM 1430 PRINT SPC(10)"£ TO LOAD SOUND FILE
QD 1440 PRINT SPC(10)"Q TO QUIT
EC 1450 PRINT SPC(9)"{RVS}{DOWN}CLR {OFF} TO VIEW
        {SPACE}THIS MENU"
SD 1460 PRINT SPC(9)"{RVS}HOME{OFF}"
GQ 1470 PRINT SPC(5)"{4 DOWN}PRESS {RVS}{YEL}SPACE
         BAR{CYN}{OFF} TO HEAR SOUND"
GX 1480 RETURN
BC 1490 REM * PLAY SOUND *
MG 1500 SOUND VO,FR,DU,DI,MN,SP,WF,PW
PX 1510 RETURN
HX 1520 REM * SAVE SOUND *
CH 1530 PRINT "{CLR}{DOWN}"SPC(12)"* SAVE SOUND *"
        :FS$=""
CQ 1540 GOSUB 1880
BQ 1550 APPEND#1,"CATALOG"
RS 1560 GOSUB 2080:IF DS THEN 2100
EX 1570 PRINT#1,FS$:GOSUB 2080:IF DS THEN 2100
CB 1580 DCLOSE#1
MB 1590 OPEN15,8,15,"S0:"+FS$:CLOSE15
QH 1600 DOPEN#2,(FS$),D0,U8,W
DP 1610 GOSUB 2080:IF DS THEN 2110
MQ 1620 PRINT#2,VO:PRINT#2,FR:PRINT#2,DU:PRINT#2,D
        I:PRINT#2,MN
BB 1630 PRINT#2,SP:PRINT#2,WF:PRINT#2,PW:GOSUB2080
        :IF DS THEN 2110
AF 1640 DCLOSE#2
SK 1650 RETURN
DJ 1660 REM * LOAD SOUND *
CQ 1670 PRINT "{CLR}"SPC(12)"{DOWN}* LOAD SOUND *"
XG 1680 GOSUB 1880
```

```
EE 1690 DOPEN#2,(FS$):GOSUB 2080:IF DS THEN 2110
QP 1700 INPUT#2,VO,FR,DU,DI,MN,SP,WF,PW
RK 1710 MOVSPR 2,X+2,Y-(FR/1000)+1
DP 1720 MOVSPR 3,X+38,Y-(DU/6)+1
XF 1730 IF DI=0 THEN TY=114
SP 1740 IF DI=2 THEN TY=132
XP 1750 IF DI=1 THEN TY=150
AA 1760 MOVSPR 4,X+74,TY
ER 1770 MOVSPR 5,X+110,Y-(MN/1000)+1
PH 1780 MOVSPR 6,X+146,Y-(SP/250)-1
JP 1790 WY=(WF*20)+101
HJ 1800 MOVSPR 7,X+182,WY
SX 1810 MOVSPR 8,X+218,Y-(PW/67)-1
PX 1820 DCLOSE#2:PRINT TAB(12)"{DOWN}{RVS}FILE IS
        {SPACE}LOADED{OFF}"
RC 1830 RETURN
HG 1840 REM * TEXT SCREEN *
GE 1850 GRAPHIC 0,0:FOR S=1 TO 8:SPRITE S,0:NEXT
GH 1860 RETURN
RB 1870 REM * INPUT SCREEN *
MK 1880 PRINT "{CYN}{3 DOWN}{12 SPACES}INPUT FILEN
        AME:"
JM 1890 INPUT "{DOWN}{13 RIGHT}";FS$
XS 1900 IF FS$="" THEN 590
BK 1910 RETURN
JP 1920 REM * DISK CATALOG *
GE 1930 PRINT "{CLR}{CYN}{4 DOWN}{11 SPACES}SOUND
        {SPACE}CATALOG"
FJ 1940 PRINT SPC(7)"{16 DOWN}PRESS {RVS}{YEL}SPAC
        E BAR{CYN}{OFF} TO PAUSE"
EM 1950 WINDOW 12,7,25,19
DJ 1960 DOPEN#1,"CATALOG":GOSUB 2080:IF DS THEN DC
        LOSE#1:PRINT"{2 HOME}":GOTO 590
KC 1970 INPUT#1,FILE$
SK 1980 RS=ST
JP 1990 PRINT FILE$
BK 2000 GET H$
EF 2010 IF H$<>" " THEN 2030
JB 2020 GET H$:IF H$<>" " THEN 2020
RQ 2030 IF RS=0 THEN 1970
JR 2040 DCLOSE#1
QJ 2050 PRINT "{2 HOME}"
CA 2060 RETURN
MA 2070 REM * DISK ERROR CHECK *
SH 2080 IF DS=62 THEN PRINT"{RVS}{DOWN}{YEL}FILE N
        OT FOUND ON THIS DISK{OFF}{CYN}":SLEEP 2:P
        RINT"{2 HOME}"
KF 2090 RETURN
BM 2100 DCLOSE#1:GOTO590
```

```
MJ 2110 DCLOSE#2:GOTO590
RH 2120 REM * QUIT SCREEN *
HB 2130 PRINT "{CLR}{5 DOWN}{4 SPACES}DO YOU WISH
        {SPACE}TO EXIT TO BASIC{3 SPACES}Y"
KE 2140 INPUT "{33 RIGHT}{UP}";AN$
PH 2150 IF ASC(AN$)=78 THEN 590
CC 2160 PRINT"{CLR}":END
FJ 2170 PRINT"{CLR}SOUND VARIABLES:":PRINT"{DOWN}S
        OUND"VO"{LEFT},"FR"{LEFT},"DU"{LEFT},"DI"
        {LEFT},"MN"{LEFT},"SP"{LEFT},"WF"{LEFT},"P
        W
KH 2180 GOTO 610
JK 2190 PRINT"{CLR}LOADING SPRITE DATA...":BANK0:F
        ORI=3584TO4095:READA:POKEI,A:X1=X1+A:NEXT
AQ 2200 IFX1<>11623THENPRINT"ERROR IN DATA STATEME
        NTS.":STOP
RP 2210 RETURN
JF 2220 DATA 255,192,0,127,128,0,63,0,0,30
EC 2230 DATA 0,0,0,0,0,0,0,0,0,0
HD 2240 DATA 0,0,0,0,0,0,0,0,0,0
MQ 2250 DATA 0,0,0,0,0,0,0,0,0,0
RR 2260 DATA 0,0,0,0,0,0,0,0,0,0
AR 2270 DATA 0,0,0,0,0,0,0,0,0,0
HS 2280 DATA 0,0,0,0,12,0,0,30,0,0
KH 2290 DATA 255,192,0,30,0,0,12,0,0,0
FS 2300 DATA 0,0,0,0,0,0,0,0,0,0
JX 2310 DATA 0,0,0,0,0,0,0,0,0,0
PJ 2320 DATA 0,0,0,0,0,0,0,0,0,0
SK 2330 DATA 0,0,0,0,0,0,0,0,0,0
CD 2340 DATA 0,0,0,0,0,0,0,0,12,0
JE 2350 DATA 0,30,0,0,255,192,0,30,0,0
EG 2360 DATA 12,0,0,0,0,0,0,0,0,0
FF 2370 DATA 0,0,0,0,0,0,0,0,0,0
CE 2380 DATA 0,0,0,0,0,0,0,0,0,0
XE 2390 DATA 0,0,0,0,0,0,0,0,0,0
AE 2400 DATA 0,0,0,0,0,0,0,0,0,0
DK 2410 DATA 0,0,255,192,0,255,192,0,255,192
XR 2420 DATA 0,255,192,0,255,192,0,255,192,0
PG 2430 DATA 255,192,0,255,192,0,255,192,0,0
EB 2440 DATA 0,0,0,0,0,0,0,0,0,0
BA 2450 DATA 0,0,0,0,0,0,0,0,0,0
SA 2460 DATA 0,0,0,0,0,0,0,0,0,0
XX 2470 DATA 0,0,0,0,0,0,12,0,0,30
FA 2480 DATA 0,0,255,192,0,30,0,0,12,0
FR 2490 DATA 0,0,0,0,0,0,0,0,0,0
GC 2500 DATA 0,0,0,0,0,0,0,0,0,0
DR 2510 DATA 0,0,0,0,0,0,0,0,0,0
AQ 2520 DATA 0,0,0,0,0,0,0,0,0,0
RQ 2530 DATA 0,0,0,0,0,0,0,0,0,0
```

```
GG 254Ø DATA 12,Ø,Ø,3Ø,Ø,Ø,255,192,Ø,3Ø
CF 255Ø DATA Ø,Ø,12,Ø,Ø,Ø,Ø,Ø,Ø,Ø
EK 256Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
BJ 257Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
SJ 258Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
PP 259Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
AX 26ØØ DATA Ø,Ø,Ø,Ø,255,192,Ø,255,192,Ø
CC 261Ø DATA 255,192,Ø,255,192,Ø,255,192,Ø,255
KE 262Ø DATA 19?2,Ø,255,192,Ø,255,192,Ø,255,192
CF 263Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
BE 264Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
QE 265Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
FS 266Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,12,Ø
RX 267Ø DATA Ø,3Ø,Ø,Ø,255,192,Ø,3Ø,Ø,Ø
PC 268Ø DATA 12,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
AA 269Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
DB 27ØØ DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
SB 271Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
RA 272Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
KG 273Ø DATA Ø,127
```

Mozart Magic

James Bagley

Based on a musical game devised by Mozart himself, this delightful program composes its own minuets in the style of the master.

"Mozart Magic" is a translation of a game by Wolfgang Amadeus Mozart. It composes a complete, original minuet at random. Mozart delighted in games of chance, so it was only natural that he should combine his two interests and produce an activity known as *Musikalisches Wuerfelspiel,* or musical craps. The idea was not original with Mozart, but his effort was the most successful.

Making Music

Type in and save the program; then run it. After playing an introduction and initializing, the program displays a menu. You can choose a different instrument for each voice, but most songs sound best if you choose the same instrument for all three voices. Some of the instruments such as the drum and xylophone may sound strange or faint. They're included for the sake of completeness so that you can hear what all the 128's instruments sound like.

The next menu allows you to change the tempo. Press F to increase the speed at which the minuet is played, press S to decrease the speed, and press E to exit the routine. The tempo always defaults to 8. The main menu reappears after the minuet is finished.

The program itself is structured to reflect the composer's original technique. Mozart set up two grids of 8 columns and 11 rows. The columns were numbered 1–8, and the rows were numbered 2–12. On the first throw of the dice, he scanned down the first column to the row numbered the same as the sum of the two dice. At this intersection would be a number. He then copied down a measure of music that corresponded to this number and repeated the process until he reached the eighth column of the first part.

In the eighth column of the grid, each number refers to a measure of music with two sets of notes. Because the music modulates to the dominant, the lower notes serve for the first ending and the upper notes for the second ending. Since these measures are all the same, M2$(1) is used in the program for the first ending and M2$(2) for the second ending of the first part of the minuet.

Mozart Magic

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
MF 5 PRINT CHR$(144):VOL 15
RM 10 SCNCLR:PRINT"{9 DOWN}{RVS}{14 RIGHT}MOZART M
      AGIC"
GH 20 TEMPO8:PLAY"O4QCICCCC.CSFQCRO3$BI$B$B$B$BSO4
      CO3$BA$BIARBQBIBBBBO4.CSDQESRE.FSDQCO3BO4C"
DB 30 DIMM$(7,11),M1$(8,11),M2$(2),R(7),R1(8)
RK 40 FORI=1TO7:FORJ=1TO11:READM$(I,J):NEXT:NEXT
EH 50 FORI=1TO8:FORJ=1TO11:READM1$(I,J):NEXT:NEXT
FR 60 M2$(1)="V2O4QDV3GV1O1IGO2SGFEDM":M2$(2)="V2O
      4QDV3GV1O1IGO2SBG#FEM"
GG 70 SCNCLR:FORV=1TO3
CB 80 PRINT"{HOME}{DOWN} CHOOSE AN INSTRUMENT FOR
      {SPACE}VOICE"V
PJ 90 PRINT"{DOWN} {RVS}0{OFF} PIANO
ME 100 PRINT"{DOWN} {RVS}1{OFF} ACCORDION
EF 110 PRINT"{DOWN} {RVS}2{OFF} CALLIOPE
KS 120 PRINT"{DOWN} {RVS}3{OFF} DRUM
DM 130 PRINT"{DOWN} {RVS}4{OFF} FLUTE
FR 140 PRINT"{DOWN} {RVS}5{OFF} GUITAR
EB 150 PRINT"{DOWN} {RVS}6{OFF} HARPSICHORD
DB 160 PRINT"{DOWN} {RVS}7{OFF} ORGAN
CD 170 PRINT"{DOWN} {RVS}8{OFF} TRUMPET
FE 180 PRINT"{DOWN} {RVS}9{OFF} XYLOPHONE
RD 190 GETKEYI$:IFI$<"0"ORI$>"9"THEN190
BE 200 INS=VAL(I$)
KM 210 IFV=1THENPLAY"V1
CX 220 IFV=2THENPLAY"V2
PR 230 IFV=3THENPLAY"V3
FS 240 IFINS=0THENPLAY"T0
KD 250 IFINS=1THENPLAY"T1
SA 260 IFINS=2THENPLAY"T2
DG 270 IFINS=3THENPLAY"T3
JP 280 IFINS=4THENPLAY"T4
RJ 290 IFINS=5THENPLAY"T5
FX 300 IFINS=6THENPLAY"T6
MQ 310 IFINS=7THENPLAY"T7
AG 320 IFINS=8THENPLAY"T8
```

```
PA 330 IFINS=9THENPLAY"T9
EJ 340 NEXT:SCNCLR
RQ 350 N=8:DO
AR 360 PRINT"{HOME}{DOWN} TEMPO{4 RIGHT}{3 SPACES}
       {4 LEFT}"N
DC 370 PRINT"{DOWN} {RVS}F{OFF}ASTER
FF 380 PRINT"{DOWN} {RVS}S{OFF}LOWER
AQ 390 PRINT"{DOWN} {RVS}E{OFF}XIT
JF 400 GETKEYT$
XS 410 IFT$="F"THENN=N+1:IFN=>255THENN=255
JQ 420 IFT$="S"THENN=N-1:IFN=<0THENN=1
DD 430 IFT$="E"THENEXIT
FF 440 LOOP:TEMPON
MF 450 FORI=1TO7:R(I)=INT(RND(1)*11+1):NEXT
JR 460 FORI=1TO8:R1(I)=INT(RND(1)*11+1):NEXT:SCNCL
       R
MP 470 FORK=1TO2:FORI=1TO7:PLAYM$(I,R(I)):NEXT:PLA
       YM2$(K):NEXT
KF 480 FORK=1TO2:FORI=1TO8:PLAYM1$(I,R1(I)):NEXT:N
       EXT
RF 490 GOTO70
AP 500 REM FIRST THROW
EX 510 DATA V1O2QCV3O4IECO3GM,V1O2QCV2EV3O3IGO4CEM
       ,V1O2QCV2EV3O4IGECM,V1O2QCV2EV3O4SCO3BO4CEO
       3GO4CM,V1O2QCV2EV3O5SCO4BO5CO4GECM,V1O2QCV3
       O4SEDEGO5CO4GM
BC 520 DATA V1O2QCV2EV3O4IGSFEDCM,V1O2QCV2EV3O4SEC
       GEO5CO4GM,V3O4ICV1O2SCV2EGMV3O3IGV1O2SCV2EG
       MV3O4IEV1O2SCV2EGM
XS 530 DATA V1O2QCV2EV3O4IGCEM,V1O2ICV2O3EV3O4CV1O
       2CV2O3EV3O4CV1O2CV2O3EV3O4CM
GX 540 REM SECOND THROW
MM 550 DATA V1O2QCV3O4IECO3GM,V1O2QCV2EV3O3IGO4CEM
       ,V1O2QCV2EV3O4IGECM,V1O2QEV2GV3O4SCO3GO4CEO
       3GO4CM
XP 560 DATA V1O2QCV2EV3O5SCO4BO5CO4GECM,V1O2QCV3O4
       SEDEGO5CO4GM,V1O2QCV2EV3O4IGSFEDCM,V1O2QCV2
       EV3O4SCO3GO4ECGEM
SK 570 DATA V1O2QCV2EV3O4ICO3GO4EM,V1O2QCV2EV3O4IG
       CMV1O2CV2GV3O4EM,V1O2ICV2O3EV3O4CV1O2CV2O3E
       V3O4CV1O2CV2O3EV3O4CM
BG 580 REM THIRD THROW
RH 590 DATA V1O1QBV2O2GV3O4SDEFDMV1O1IGV3O4CO3BM,V
       1O1QGV3O3IBO4DGM,V1O1QGV3O3IBO4SDO3BAGM,V1O
       2QGV2BV3O4IFDO3BM
SJ 600 DATA V1O1QBV2O2DV3O4SG#FGDO3BGM,V1O2QGV2BV3
       O4SFEFDCO3BM,V1O1QGV2O2GV3O3SBO4CDEMV1O1IBV
       2O2GV3O4SFDM
```

```
CC 61Ø DATA V1O2IGV2O3BV3O4DV1O2GV2O3BV3O4DV1O2GV2
       O3BV3O4DM,V1O1QGV3O3SBO4CDO3BAGM,V1O1QBV3O4
       IDO3BGM,V1O2QGV3O3SBABO4CDO3BM
QH 62Ø REM FOURTH THROW
JS 63Ø DATA V1O2QCV2EV3O4SCO3BO4CEO3IGM,V1O2QCV3O4
       SECO3BO4CO3IGM,V1O2QEV2GV3O4ICO3GEM,V1O2QEV
       2GV3O4ICEO3GM
PC 64Ø DATA V1O2QEV2GV3O4SCO3BO4CO3GECM,V1O2QCV2EV
       3O4ICSCDIEM,V1O2QCV2O4ICV3EV2SCV3EV2DV3FV2I
       EV3GM
XP 65Ø DATA V1O2QEV2GV3O4ICSECO3IGM,V1O2QEV2GV3O4S
       CO3GO4ECIGM,V1O2QEV2GV3O4ICSECIGM,V1O2QEV2G
       V3O4SCECO3GIEM
KE 66Ø REM FIFTH THROW
HS 67Ø DATA V1O2QCV3O4I#FSA#FD#FM,V1O2ICV2O3#FV3O4
       DV1O2CV2O4DV3#FV1O2CV2O4#FV3AM,V1O2QCV3O4SD
       O3AO4#FDA#FM
HB 68Ø DATA V1O2ICV2O3#FV3O4DV1O2CV2O3#FV3O4DV1O2C
       V2O3#FV3O4DM,V1O2QCV3O4IDO3SABAO4I#FM,V1O2Q
       CV3O4SD#CD#FA#FM
HM 69Ø DATA V1O2QCV2AV3O4I#FAMV1O2CV2AV3O4DM,V1O2I
       CV2#FV3O3AV1O2CV2#FV3O3SAO4DMV1O2ICV2AV3O4#
       FM
QX 7ØØ DATA V1O2ICV2O4DV3#FV1O2CV2O4DV3#FV1O2CV2O4
       DV3#FM,V1O2ICV2DV3O4#FV1O2CV2DV3O4S#FDMV1O2
       ICV2DV3O4AM,V1O2QCV2AV3O4S#FDO3AO4A#FDM
EM 71Ø REM SIXTH THROW
QH 72Ø DATA V1O2IBV2O2DV3O4SG#FMV1O1IBV2O2DV3O4SGB
       MV1O1IBV2O2DV3O4DM,V1O1QBV2O2DV3O4IGSBGDO3B
       M,V1O1QBV2O2DV3O4IGBDM
FR 73Ø DATA V1O1QBV2O2GV3O3IAS#FGBO4GM,V1O1QBV2O2D
       V3O4SG#FGDMV1O1IBV2O2GV3O3SBGM,V1O1QBV3O4IG
       SBGDGM,V1O1QBV2O2GV3O4IDSGDO3BO4DM,V1O1QBV2
       O2GV3O4IDSDGIBM
AA 74Ø DATA V1O1IBV2O2DV3O4SAGMV1O1IBV2O2DV3O4S#FG
       MV1O1IBV2O2GV3O4DM,V1O1QBV2O2DV3O4IGSGDIBM,
       V1O1QBV2O2DV3O4SGBGDO3IBM
EQ 75Ø REM SEVENTH THROW
XQ 76Ø DATA V1O2ICV3O4SECMV1O2IDV3O3SBAMV1O1IDV3O3
       SG#FM,V1O2ICV3O3SAO4EMV1O2IDV2O3SBV3O4DV2O3
       AV3O4CMV1O1IDV2O3SGV3BV2#FV3AM
BR 77Ø DATA V1O2ICV2O3SBV3O4DV2O3AV3O4CMV1O2IDV2O3
       AV3O4CV2O3GV3BMV1O1IDV2O3SGV3BV2#FV3AM,V1O2
       ICV3O4SEGMV1O2IDV3O4SDCMV1O1IDV3O3SBAM
AJ 78Ø DATA V1O2ICV3O3SAO4EMV1O2IDV3O4SDGMV1O1IDV3
       O4S#FAM,V1O2ICV3O4SEAMV1O2IDV3O4SGBMV1O1IDV
       3O4S#FAM,V1O2ICV3O4SCEMV1O2IDV3O4SGDMV1O1ID
       V3O3SAO4#FM
```

```
RP 79Ø DATA V1O2ICV3O4SEGMV1O2IDV3O4SDGMV1O1IDV3O3
       SBO4#FM,V1O2ICV3O4SECMV1O2IDV3O3SBGMV1O1IDV
       3O3SA#FM,V1O2ICV3O4SEO5CMV1O2IDV3O4SBGMV1O1
       IDV3O4SA#FM
DQ 8ØØ DATA V1O2ICV3O3AV1O2DV3O4SDCMV1O1IDV3O3SBAM
HX 81Ø REM PART TWO FIRST THROW
XG 82Ø DATA V1O2QDV3O4I#FSA#FMV1O2ICV3O4SD#FM,V1O2
       QDV2#FV3O4SDO3AO4D#FA#FM,V1O2IDV2AV3O4#FV1O
       2DV2#FV3O4AV1O2CV2DV3O4#FM
DG 83Ø DATA V1O2QCV2AV3O4S#FAO5DO4AMV1O2ICV2AV3O4#
       FAM,V1O2QDV3O3SD#FAO4DMV1O2ICV3O4S#FAM
PP 84Ø DATA V2O4IDV3#FV1O1SDO2DMV3O4Q#FV1O2S#CDCDM
       ,V1O2QDV2#FV3O4IA#FMV1O2CV2#FV3DM,V1O2QDV2#
       FV3O5IDO4SA#FMV1O2ICV2#FV3O4SDO3AM
GC 85Ø DATA V1O2QDV2#FV3O4SDO3AO4ID#FM,V1O2QCV2AV3
       O4S#FDO3IAMV1O2CV2AV3O4#FM,V1O2QDV2#FV3O3IA
       O4DMV1O2CV2AV3O4#FM
PD 86Ø REM PART TWO SECOND THROW
AG 87Ø DATA V1O1QBV2O2GV3O4IGSBGIDM,V3O4IGV1O1SBO2
       DMV3O3IGV1O2SGDMV3O3IGV1O1SBGM,V1O1QBV3O4SG
       BGBIDM
CC 88Ø DATA V1O1QBV2O2DV3O4SAGBGMV1O1IBV2O2DV3O4SD
       GM,V1O1QBV2O2DV3O4IGSDO3BMV1O1IBV2O2DV3O3GM
JK 89Ø DATA V1O1QBV2O2DV3O4SGBO5DO4BMV1O1IBV2O2DV3
       O4GM,V1O1QBV2O2DV3O4SGBGDO3BGM,V1O1QBV2O2DV
       3O4SGDGBMV1O1IBV2O2DV3O4SGDM
DP 9ØØ DATA V1O1QBV2O2DV3O4SGBIGMV1O1IBV2O2GV3O4DM
       ,V3O4IGV1O1SGBMV3O4QDV1O2IGO1BM,V1O1QBV3O4I
       GSBO5DO4IDM
AF 91Ø REM PART TWO THIRD THROW
XJ 92Ø DATA V2O4ICV3EV1O2SCEMV2O4ICV3EV1O2SGEMV2O4
       ICV3EV1O3SCO2CM,V1O2QEV3O4SCO3GO4CEMV1O2EV3
       O4GV1O2CV2O4CV3EM,V1O2QCV2GV3O4IESGEMV1O2IC
       V2DV3O4CM
XE 93Ø DATA V1O2QCV2GV3O4SECEGO5CO4GM,V1O2QCV2GV3O
       4SEGO5CO4GMV1O2ICV2GV3O4SECM,V2O4ICV3EV1O2S
       CO1BMV3O4QEV1O2SCDE#FM
PM 94Ø DATA V3O4IEV1O2SCV2EGMV3O4ICV1O2SCV2EGMV3O3
       IBV1O2SCV2EGM,V1O2QCV2GV3O4IESCEMV1O2CV2EV3
       O4GO5CM
RF 95Ø DATA V1O2QCV2GV3O4SECIEMV1O2CV2EV3O4GM,V1O2
       QCV2GV3O4SECO3IGMV1O2CV2GV3O4EM,V1O2QCV2GV3
       O4IEGMV1O2CV2EV3O5CM
FR 96Ø REM PART TWO FOURTH THROW
EP 97Ø DATA V1O2QGV2O4ICV3EV2O3BV3O4DMV1O1GM,V1O1Q
       GV2O2GV3O4SDO3BIGMV1O2GM,V1O2IGV2O4CV3EMV1O
       1GV2O3SBV3O4DV2O3GV3BV2IGM
```

```
JE 980  DATA V1O2QGV3O4SECDO3BIGM,V1O2IGV3O4SGEMV1O
        1IGV3O4SDO3BIGM,V1O2QGV3O3SBO4DGDMV1O1IGV3O
        3BM,V1O2IGV3O4SECMV1O1IGV3O3SBO4DIGM
RF 990  DATA V1O2QGV2BV3O4SDBGDO3IBM,V1O1QBV2O2GV3O
        4SDO3BIGMV1O1BV2O2DV3O4GM,V3O4IDV1O2SG#FMV3
        O3QBV1O2SGDO1BGM
GA 1ØØØ  DATA V1O2QGV2O3IBV3O4DSGBMV1O2IGV3O4DM
MK 1Ø1Ø  REM PART TWO FIFTH THROW
MJ 1Ø2Ø  DATA V3O4IEV1O2SCV2EGMV3O4ICV1O2SCV2EGMV3O
         3IGV1O2SCV2EGM,V3O3IGV1O2SCV2EGMV3O4ICV1O2
         SCV2EGMV3O4IEV1O2SCV2EGM
AK 1Ø3Ø  DATA V3O4IGV1O2SCV2EGMV3O4EV1O2SCV2EGMV3O4
         ICV1O2SCV2EGM,V1O2QCV2EV3O4SCO3BO4CO4EMV1O
         2IEV2GV3O3SGO4CM
PX 1Ø4Ø  DATA V1O2QCV2EV3O5SCO4BO5CO4GMV1O2ICV2GV3O
         4SECM,V1O2QCV2GV3O4SEDEGMV1O2ICV2EV3O5SCO4
         GM,V1O2QCV2EV3O4IGSFEDCM
SQ 1Ø5Ø  DATA V1O2QCV2EV3O4SCO3GO4ECGEM,V3O4ICV1O2S
         CV2EGMV3O3IGV1O2SCV2EGMV3O4IEV1O2SCV2EGM
XQ 1Ø6Ø  DATA V3O4IGV1O2SCV2EGMV3O4ICV1O2SCV2EGMV3O
         4IEV1O2SCV2EGM,V1O2ICV2O3EV3O4CV1O2CV3O3EV
         3O4CV1O2CV2O3EV3O4CM
GG 1Ø7Ø  REM PART TWO SIXTH THROW
RX 1Ø8Ø  DATA V3O4IEV1O2SCV2EGMV3O4ICV1O2SCV2EGMV3O
         3IGV1O2SCV2EGM,V1O2QCV2EV3O3IBO4CMV1O2CV2G
         V3O4EM,V3O4IGV1O2SCV2EGMV3O4IEV1O2SCV2EGMV
         3O4ICV1O2SCV2EGM
BQ 1Ø9Ø  DATA V1O2QCV2EV3O4SCO3BO4EMV1O2ICV2EV3O3SG
         O4CM,V1O2QCV2EV3O5SCO4BO5CO4GECM,V1O2QCV2G
         V3O4SEDECMV1O2ICV2EV3O5SCO4GM
QH 11ØØ  DATA V1O2QCV2EV3O4IGSFEMV1O2IEV2GV3O4SDCM,
         V1O2QCV2EV3O4SCO3GO4ECGEM,V3O4ICV1O2SCV2EG
         MV3O3IGV1O2SCV2EGMV3O4IEV1O2SCV2EGM
RX 111Ø  DATA V3O4IGV1O2SCV2EGMV3O4ICV1O2SCV2EGMV3O
         4IEV1O2SCV2EGM,V1O2ICV2O3EV3O4CV1O2CV2O3EV
         3O4CV1O2CV2O3EV3O4CM
BM 112Ø  REM PART TWO SEVENTH THROW
XK 113Ø  DATA V1O2QFV2AV3O4SDFDFMV1O2IGV2O3DV3SBO4D
         M,V1O2QFV3O4SDFAFMV1O2IGV3O4SDO3BM,V1O2QDV
         3O4SDFO3AO4DMV1O2IGV3O3SBO4DM
BB 114Ø  DATA V1O2QFV3O4SD#CDFMV1O2IGV3O3SGBM,V1O2I
         FV3O4FV1O2DV3O4DV1O2GV3O4GM,V1O2SFV3O4FV1O
         2EV3O4EV1O2DV3O4DV1O2EV3O4EV1O2FV3O4FV1O2G
         V3O2GM
BQ 115Ø  DATA V1O2SFV3O4FV1O2EV3O4EV1O2IDV3O4DV1O2G
         V3O4GM,V1O2QFV3O4SFEDCMV1O2IGV3O3SBO4DM,V1
         O2QFV3O4SFDO3IAMV1O2GV3O3BM
PF 116Ø  DATA V1O2QFV3O4SFAO3IAMV1O2GV3O3SBO4DM,V1O
         2QFV3O3IAO4SFDMV1O2IGV3O3SABM
```

```
XB 1170 REM SECOND PART EIGHTH THROW
CG 1180 DATA V3O4QCV1O2ICO1GCM,V3O4QCV1O2ICO1GCM,V
        3O4QCV1O2ICO1GCM,V3O4QCV1O2ICO1GCM,V3O4QCV
        1O2ICO1GCM,V3O4QCV1O2ICO1GCM
CG 1190 DATA V3O4QCV1O2ICO1GCM,V3O4QCV1O2ICO1GCM,V
        3O4QCV1O2ICO1GCM,V1O2QCV3O4ICO3CV1O1CM,V3O
        4QCV1O2ICO1GCM
```

Artimation

Jerry Crisci

The 128's high-resolution graphics commands can work wonders on your screen. Here are three short programs that create some interesting art in motion.

One of the many impressive features of the Commodore 64 is its ability to create beautiful high-resolution graphics. Unfortunately, the 64's BASIC 2.0 doesn't allow programmers to access high-resolution mode very easily; you have to perform hundreds of PEEKs and POKEs to draw a picture. When the 128 was released, several useful high-resolution graphics commands were included with BASIC 7.0.

Artimation is a term I coined to describe the process of creating kinetic computer art by "animating" circles, lines, and boxes on the screen, and allowing them to leave their image behind. And it's done with just a few lines in BASIC.

In traditional animation, an image is drawn on the screen, then erased, then drawn again in a different location. When this process is repeated very quickly, the image appears to move smoothly across the screen. If the same process is done without erasing the image after it is drawn, the image will leave its trail on the screen. If you also manipulate the object as it's moved (changing its size or rotating it), the resulting process usually creates an impressive piece of computer art.

Three short programs are included here. Program 1, "Cosmic Objects," creates ten three-dimensional objects on the screen in less than a minute. The images are created by using the CIRCLE statement. Each random-sized image is placed on a random part of the screen, creating the effect of objects floating in space. After the program draws ten objects, it stops. Pressing RUN/STOP–RESTORE (or blindly typing GRAPHIC0) brings you back to the text screen, where you can run the program again to create another picture.

Unlike Programs 1 and 3, "The Pit" (Program 2) creates the same picture every time it is run. The DRAW statement is used to draw a series of lines radiating from the center of the screen. As a result of their varying proximity to each other (as the lines travel to the edge of the screen), "interference patterns" develop, creating an ornate design that appears to be disappearing into a black abyss in the center of the screen. You can create variations on the design by changing the STEP values in lines 30–60 (changing the 2 to a 4 or 5 creates a slightly different texture).

"String Art" (Program 3) is the most exciting program to watch. It runs in an infinite loop—creating a pattern, displaying it for three seconds, then creating a new pattern in a different color. The lines on the screen are made with the BOX statement. Each box has no width, so the two sides overlap to form a line. The angle of the box changes as it is drawn, creating a pattern which gives the illusion of a line rotating in three dimensions. RUN/STOP–RESTORE halts the program and returns you to the text screen.

Notes on the Programs

After you press RUN/STOP–RESTORE, the last high-resolution screen created will still be in memory. Just type GRAPHIC 1 and press RETURN to view it. Since the F1 key is preset to print GRAPHIC, you can also use it for a shortcut, to switch back and forth between GRAPHIC0 (text) and GRAPHIC1 (hi-res). You can save the screen to disk with

BSAVE *"filename"*,B0,P7168 TO P16383

To reload a previously saved screen, type GRAPHIC 1:GRAPHIC 0 and press RETURN to make sure that the hi-res screen memory area has been allocated. Then enter

BLOAD *"filename"*

Typing GRAPHIC 1 and pressing RETURN will now display the newly loaded screen.

If you have the program *Doodle*, published by Crystal Rose Software, you can print out your artimation on your printer. Just follow the above procedure for saving the screen to disk, but use a filename which starts with the letters *DD*. For example,

**BSAVE "DD*filename*",B0,P7168 TO
P16383**

You can now load your picture into *Doodle* (remember to go into 64 mode first), and modify or print it.

Program 1. Cosmic Objects

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in these programs.

```
KK 10 REM COSMIC OBJECTS
FD 20 C=2:COLOR0,1:GRAPHIC1,1:COLOR1,5:COLOR4,1
HD 30 FORK=1TO10:CX=INT(RND(0)*320):CY=INT(RND(0)*
      200)
MF 40 R1=INT(RND(0)*40)+20:R2=INT(RND(0)*40)+20
XH 50 IFC=2THENC=45:GOTO80
PM 60 IFC=45THENC=90:GOTO80
EP 70 IFC=90THENC=2
HQ 80 FORI=R1TO0STEP-(R1/5):CIRCLE1,CX,CY,I,R2,,,,
      C:NEXT
RG 90 FORI=R2TO0STEP-(R2/5):CIRCLE1,CX,CY,R1,I,,,,
      C:NEXTI,K
```

Program 2. The Pit

```
HK 10 REM THE PIT
CX 20 COLOR0,2:COLOR4,2:GRAPHIC1,1:COLOR1,1:DRAW1,
      160,100
PB 30 FORI=1TO320STEP2:DRAW1,160,100TOI,0TO160,100
      :NEXT
FD 40 FORI=1TO200STEP2:DRAW1,160,100TO0,ITO160,100
      :NEXT
RF 50 FORI=1TO320STEP2:DRAW1,160,100TOI,200TO160,1
      00:NEXT
CM 60 FORI=200TO0STEP-2:DRAW1,160,100TO320,ITO160,
      100:NEXT
```

Program 3. String Art

```
DH 10 REM STRING ART
CX 20 COLOR0,1:C=2:COLOR4,1:GRAPHIC1,1:K=50:L=70:P
      =5
QM 30 COLOR1,5:S=3
DM 40 A=A+5:K=K+P:IFA>=2↑16THENA=0:GOTO40
XR 50 IFK>320THENP=-6:GOSUB80:SLEEP2:COLOR1,C:C=C+
      1:GRAPHIC1,1:IFC>15THENC=2
DB 60 BOX1,ABS(K),ABS(K),ABS(K),L,A:IFK<0THENP=5:G
      OSUB80
BF 70 GOTO40
QC 80 K=INT(RND(0)*320):L=INT(RND(1)*200):RETURN
```

Mastering 128 Sound and Music

D. C. Holmes

You don't have to be a musician to create professional-sounding music on the 128. Five short, useful programs demonstrate how just a few BASIC statements can make your 128 a powerful and versatile musical instrument.

The Commodore 64 offers extensive music-generating capabilities to the patient and interested programmer. The built-in Sound Interface Device (SID chip) has the potential for producing sophisticated electronic music, with the computer manipulating the various parameters of three independent voices simultaneously. Programming the SID chip is not that easy, however. A tedious process of POKEing values into memory locations is necessary to play even a simple musical composition. I suspect that many a masterpiece has been abandoned at the computer keyboard, owing to the frustration of dealing with PEEKs and POKEs.

Enter the 128. With the new sound statements available in BASIC 7.0, programming the SID chip is much less confusing and tedious. Technical understanding of memory maps and programming details is no longer a prerequisite for composers and arrangers. In fact, once you become familiar with the PLAY statement, the painstaking chore of translating a manuscript into computer language is virtually eliminated. You can type a score directly into the 128—straight from the sheet music.

Default Envelopes

Let's start our exploration of the musical capabilities of the 128 by examining the default parameters for the ENVELOPE statement. Then we'll take a detailed look at the TEMPO statement. Here are the ten default (built-in) envelopes defined in BASIC 7.0.

Envelope	Name
0	Piano
1	Accordion
2	Calliope
3	Drum
4	Flute
5	Guitar
6	Harpsichord
7	Organ
8	Trumpet
9	Xylophone

You can hear how these work in the first program, a short excerpt from Mozart's opera *The Magic Flute.* The computer has been programmed to play the piece over and over again, each time using a different default envelope. When you run this program, you'll be able to compare the sounds of the different envelopes. It doesn't appear that Commodore necessarily intended for these preset envelopes to imitate the instruments for which they're named. More likely, the intention was to provide a sampling of the various possibilities, and the envelopes were named to facilitate identification. To suggest that envelope 0 really sounds like a piano or that envelope 8 sounds just like a trumpet stretches the imagination a bit, but some of the envelopes do come reasonably close to the sound of the instrument they are named after.

In particular, I think envelope 2 sounds a lot like a steam calliope, so I selected that envelope for the second program, "American Patrol" by F. W. Meacham.

Take a look at line 30 of this program:

```
30 PLAY "V1T2V2T2V3T2"
```

Here, we've specified that all three of the SID chip's voices (V1, V2, and V3) will play in envelope 2 (T2). Since we've not used the ENVELOPE statement to redefine envelope 2, the 128 uses default envelope 2 (calliope).

When you've typed in this program and saved it, try specifying a different envelope in line 30. To use the organ envelope (default envelope 7), you would type

```
30 PLAY "V1T7V2T7V3T7"
```

Or you could even get fancy and play it with a mixed combo:

```
30 PLAY "V1T8V2T4V3T5"
```

This statement specifies envelope 8 for voice 1, envelope 4 for voice 2, and envelope 5 for voice 3. If you don't use line 30 at all, the 128 plays all three voices in the default envelope 0.

The TEMPO Statement

Composers usually specify a tempo for their works. It may be a general marking such as largo, andante, or allegro, or it may be very specific such as ♩ = 126, which indicates the exact tempo (126 beats per minute in this case). The TEMPO statement allows you to control the tempo of your musical program. The format for this statement is

TEMPO *n*

where *n* is a variable between 1 and 255. The equivalent musical notation is ♩ = 12.49 * *n*, because the statement defines the duration of a quarter note at 4.805/*n* seconds, or 12.49 * *n* quarter notes per minute. Here are some examples of TEMPO statements and the corresponding musical notations:

TEMPO	
4	♩ = 50
5	♩ = 62
6	♩ = 75
7	♩ = 87
8	♩ = 100
9	♩ = 112
10	♩ = 125
15	♩ = 187
20	♩ = 250

If you don't specify a tempo, the 128 automatically sets the value of *n* to 8.

In Program 1, "The Magic Flute," the TEMPO statement is used in line 10:

10 TEMPO 13

This defines a tempo of ♩ = 162. In Program 2, American Patrol, it's found in line 20:

20 TEMPO 20

This defines a tempo of ♩ = 250, which might seem to be a very rapid tempo, but since this piece is arranged in 2:2, or *cut time,* there are actually only 125 beats per minute.

After you've typed in and saved these two programs (found at the end of this article), try changing the value of *n* in the TEMPO statements to another number between 1 and 255. Then run the programs again and listen to the effect this change has on the tempo.

The PLAY Statement

The sound and music statements of BASIC 7.0 have streamlined the 128's music-programming process. The key to this instant virtuosity is the PLAY statement. Let's see how we can translate sheet music into BASIC program lines by using the PLAY statement.

The format for the PLAY statement is

PLAY "V*n*, T*n*, O*n*, U*n*, X*n*, *elements, notes*"

The uppercase letters (V, T, O, U, X) are characters that you actually type in. For each of the lowercase letters, substitute an appropriate number or character from Table 1. If you omit one of the control characters, the 128 will use the default value.

Table 1. PLAY Statement Parameters

V*n*	Voice	*n* = 1–3 (default 1)
T*n*	Tonal envelope	*n* = 0–9 (default 0)
O*n*	Octave	*n* = 0–6 (default 4)
U*n*	Volume	*n* = 0–9 (default 9)
X*n*	Filter	*n* = 1 for on, 0 for off (default 0)
Elements	#—sharp (♯)	
	$—flat (♭)	
	W—whole note (𝅝)	
	H—half note (𝅗𝅥)	
	Q—quarter note (♩)	
	I—eighth note (♪)	
	S—sixteenth note (𝅘𝅥𝅯)	
	.—dotted note	
	R—rest	
	M—wait for end of measure voice	
Notes	C, D, E, F, G, A, B	

For example, to play the first five notes of a C major scale with voice 2 and volume 6, you would use

```
PLAY "V2 U6 CDEFG"
```

Or you could use a string variable:

```
A$ = "V2 U6 CDEFG": PLAY A$
```

(The spaces are optional, but they make the string a little more readable.)

The SID chip is capable of producing three independent voices (sounds) simultaneously. When the control character V*n* appears in a character string used in a PLAY statement, it specifies which one of the three voices is to be programmed by the characters that follow. The characters apply to that voice until another V*n* control character is incurred in the string. If no voice is specified, the default V1 (voice 1) is assumed.

Tonal Envelope

The tonal quality of each of the voices used can be selected from one of the ten envelopes discussed earlier. You can use one of the preset envelopes, or you can create your own customized tonal envelope by using the ENVELOPE statement. The control character T*n* specifies the envelope for the voice whose control character most recently preceded it. The SID will continue to be tuned to that envelope for this voice until this V*n* immediately precedes another T*n*.

For example, take a look at line 20 of "Minuet," Program 3:

```
20 PLAY "V1 T6 V2 T0"
```

The SID is set to play voice 1 (V1) in envelope 6 (T6)—a harpsichord sound—and voice 2 (V2) in envelope 0 (T0)—a piano-like tone.

If no envelope is specified for a voice, that voice will use the default envelope 0.

The semantics of the T*n* control character are somewhat different from the other control characters in the PLAY string. Whereas T*n* always refers to only the V*n* which most recently preceded it, the other control characters (O*n*, U*n*, X*n*) refer to the notes that follow, regardless of which voice is programmed to play them.

Octave

Notes may be programmed in a six-octave range, corresponding roughly to the middle 72 keys of the piano. The control character O*n* in a PLAY statement character string dictates the octave range for all notes that follow, until another O*n* control character is encountered. If no octave is specified, the default O4 is assumed.

Volume

You can control the dynamic level (volume) by using the character U*n* in a PLAY string. The parameter *n* may range from 0 (no volume) to 9 (maximum volume), and it applies to all notes that follow in all voices until another U*n* character appears. You can't set individual volumes for the three voices, although changing the sustain values (with ENVELOPE) can make some sounds louder or softer than others. If U*n* does not appear in a program, the default value U8 is used. Volume may also be specified by using the VOL statement, which has the format

VOL *d*

where *d* is a value from 0 (off) to 15 (maximum volume).

Notice that the range of volume settings (normally 0–15 when the volume is controlled with the VOL statement or with POKEs) is compressed in the U control character to 0–9. Apparently the programmers who wrote the PLAY routine didn't want to have to deal with two-digit parameter settings (all the other PLAY control characters take only single-digit parameters). The U0 control character corresponds to VOL 0, while U9 corresponds to VOL 15. Other volume settings are distributed, roughly, evenly between. For example, U4 corresponds to medium volume (the equivalent of VOL 7). The U*n* control character in a PLAY string allows more precise volume control than does the VOL statement. There are, however, situations when it is more desirable to use the VOL statement and have volume control outside a character string.

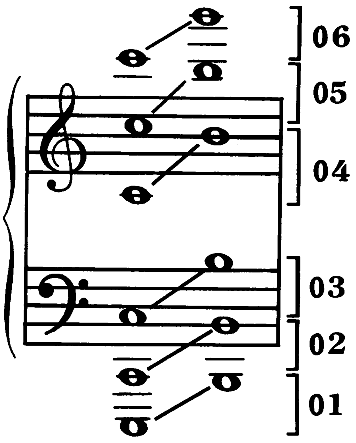
Filter

The X*n* control character allows additional creative control over the tonal quality of the 128's sound: X1 turns on the filter for a given voice, and X0 turns it off. Within a composition, you may use the filter on one or more voices. The SID chip has only one filter, however, and it applies to all filtered voices at any one time.

Notes and Elements

To sound a note, place the letter of the note you want to play within the PLAY character string. Sharps and flats may be played by placing the sharp (#) or flat ($) element prior to the letter of the note. Octaves start at C and end at B, with middle C being O4 C.

Entering Notes



To specify the length of time the note is to be held, precede it with one of the duration elements (S, I, Q, H, W). When a dot (.) precedes a duration element, the duration value of that element is increased by half. For example, in common time (4:4), the elements have these values:

S	Sixteenth note	1/4 beat
I	Eighth note	1/2 beat
.I	Dotted eighth note	3/4 beat
Q	Quarter note	1 beat
.Q	Dotted quarter note	1-1/2 beats
H	Half note	2 beats
.H	Dotted half note	3 beats
W	Whole note	4 beats
.W	Dotted whole note	6 beats

You can include a rest in the PLAY character string by following a duration element with the R element; for example, QR programs a quarter rest, SR a sixteenth rest, and so on.

The M element in a PLAY character string instructs the computer to wait for all voices currently playing to end the current measure.

Making Music

With all this in mind, you're ready to begin writing music to play on the Commodore 128. Let's start with the first measure in voice 1 of Bach's "G Major Minuet" (Program 3). The first note is a quarter note D in octave 5 (O5QD). This is followed by eighth notes G, A, and B in octave 4 (O4IGIAIB) and eighth note C in octave 5 (O5IC). To play only the first measure of voice 1, type

```
PLAY "V1 O5 QD O4 IG IA IB O5 IC"
```

Synchronizing voice 2 with voice 1 is a little trickier, though. Voice 2 begins with a half note G in octave 3, followed by a quarter note A in the same octave (O3HGQA). We want to program these notes in such a way that the half note G begins at the same moment as the quarter note D in voice 1, and the quarter note A is synchronized with the eighth note B in voice 1.

Coordinating Voices

To understand how voice coordination works, consider the logical way the computer reads and plays the notes in a PLAY character string. When the 128 reads a note, it follows two rules in determining when to start playing that note:

1. If the voice specified for this note is currently playing another note, the new note will begin after the old note has been played for the full duration specified. If this voice is not currently playing, the note begins immediately. (Remember that the voice is specified by the last V*n* character to precede a note in a PLAY string.)
2. The computer will not proceed to the next note in a string until the note just read has begun to be sounded (regardless of whether the notes are specified for the same, or different, voices).

So, we list the notes in the following order to play both voices in sync:

V2O3HG V1O5QD O4IG IA IB V2O3QA V1O5IC

This measure is programmed by the character string A$ in line 30 of the Minuet program. I've found that the practice of assigning a name to a string in one line, and then PLAYing that string in another line, facilitates organizing and debugging musical programs.

The notes of the second measure are listed in the same manner to synchronize their playing:

V1O5QD V2O3.HB V1O4IG IR IG IR

This string is named B$, the third measure C$, and so on.

Note that this melody is written in the key of G major, and that there is one sharp (F#) in the key signature. The 128 doesn't know what key it's playing in, so any sharp or flat notes must be preceded by a # (sharp) or $ (flat) in the PLAY character string, as demonstrated in C$ below.

One more important note about synchronization: *The Commodore 128 System Guide* offers this advice (page 156) on synchronizing notes of different durations: "As a rule, always start with the note with the longer duration." This is a rule I disagree with. If we followed this rule, we would come up with the following PLAY strings for the first six measures of Minuet:

```
A$="V2O3HG V1O5QD O4IG IA V2O3QA V1O4IB O5IC"
B$="V2O3.HB V1O5QD O4IG IR IG IR"
C$="V2O4.HC V1O5QE IC ID IE I#F"
D$="V2O3.HB V1O5QG O4IG IR IG IR"
E$="V2O3.HA V1O5QC ID IC O4IB IA"
F$="V2O3.HG V1O4QB O5IC O4IB IA IG"
```

If you type these in and play them, you'll find that the voices gradually lose their synchronization. By the end of the fifth measure, voice 2 is an eighth note ahead of voice 1. The explanation is fairly simple: There is a very small but definite period of time required for the computer to read and process each note. While a whole note should play for exactly the same duration as two half notes, the two half notes will take slightly longer to play on the 128 than the whole note. The reason is that there are two notes to read and process instead of one. Four quarter notes, then, take longer to play than 2 half notes; 8 eighth notes longer yet; and 16 sixteenth notes even longer.

To overcome this inherent problem with the PLAY statement, I have my own rule for synchronization. To synchronize two or three notes to start playing at the same time, list the notes in the following order in the PLAY character string:

1. List first the note played by the last voice to stop playing before the point of synchronization. If more than one voice is currently playing, this voice will be the one that was last named in a PLAY character string.
2. List second the note played by the next-to-the-last voice to stop playing before the point of synchronization.

You can see how I've followed this rule in lines 30–190 in Minuet. Strings B$, C$, D$, E$, and F$ all begin with voice 1, since voice 1 was playing the last note specified in each preceding string.

Once you understand these few concepts, it's fairly straightforward to list the character strings for the rest of the piece. Although Minuet uses only two voices, the principles of listing and synchronizing are the same for musical arrangements that use all three voices.

The ENVELOPE and FILTER Statements

The frequency of a sound wave is the property that determines its pitch. Our auditory sense perceives a high-frequency sound as a "high note" and a low-frequency sound as a "low note." The ear is sensitive to other characteristics as well, and it can distinguish between different instruments playing the same pitch. All of us, for example, can tell the difference between middle C struck on the piano and middle C bowed on the violin.

The peculiar properties of a sound that enable us to make this differentiation include its *timbre* (waveform) and amplitude qualities (ADSR—Attack, Decay, Sustain, and Release). Through manipulation of 11 variables related to waveform and ADSR, a seemingly infinite number (actually about 70 trillion) of individual voice registrations are possible on the 128. Professional synthesizer players refer to each of these combinations as a *patch*. Control of these variables in BASIC 7.0 is achieved by using the ENVELOPE and FILTER statements.

The timbre of a sound is determined by its waveform, which is set by the ENVELOPE statement (explained below). The 128 can produce four types of waveforms (triangle, sawtooth, square, and noise). Waveforms can be modified with the FILTER statement.

The FILTER statement defines the characteristics of the SID chip's filter, allowing certain audio frequencies while suppressing others—similiar to the tone control on a radio or record player which can be adjusted to boost the bass or treble frequencies. The format for the FILTER statement is

FILTER ***f,lp,bp,hp,re***

where:

f	= frequency (0–2047)
lp	= low-pass (on = 1, off = 0)
bp	= band-pass (on = 1, off = 0)
hp	= high-pass (on = 1, off = 0)
re	= resonance (0–15)

There are three basic filter types: low-pass, high-pass, and band-pass. The cutoff frequency of the filter determines which type of filter is in use. A low-pass filter allows only sound *below* the cutoff frequency to pass. A high-pass filter passes only sounds *above* the cutoff frequency, while a band-pass filter allows a range of frequencies centered on the given frequency to pass unattenuated. The filter types can be used in combination, yielding some unusual effects.

Defining the filter parameters is not enough, however. The filter must also be turned on by using X1 in the PLAY statement. To turn off the filter, use X0.

```
FILTER 1047,1,0,1,13
PLAY "X1 O4 C D E F G F E D C"
```

The sound can be further altered by using the ENVELOPE statement alone or in conjunction with the FILTER statement. The format for the ENVELOPE statement is

ENVELOPE ***n,a,d,s,r,wf,pw***

where:

n	= envelope	(0–9)
a	= attack	(0–15)
d	= decay	(0–15)
s	= sustain	(0–15)
r	= release	(0–15)

wf = waveform (0 = triangle, 1 = sawtooth, 2 = square, 3 = noise, 4 = ring modulation)
pw = pulse width for square waveform (0–4095)

The amplitude variations throughout the duration of a sound are described by the ADSR values. *Attack* is the rate at which the volume initially rises to its peak. The volume then decreases to a plateau *(sustain).* The rate at which this decrease occurs is the *decay* rate. The amplitude decreases again from the sustain level down to zero volume. The rate of this final decay is the *release* rate.

These parameters are initialized to the following values in the 128's ten default envelopes:

	n	*a*	*d*	*s*	*r*	*wf*	*pw*	**Envelope Name**
ENVELOPE	0,	0,	9,	0,	0,	2,	1536	Piano
ENVELOPE	1,	12,	0,	12,	0,	1		Accordion
ENVELOPE	2,	0,	0,	15,	0,	0		Calliope
ENVELOPE	3,	0,	5,	5,	0,	3		Drum
ENVELOPE	4,	9,	4,	4,	0,	0		Flute
ENVELOPE	5,	0,	9,	2,	1,	1		Guitar
ENVELOPE	6,	0,	9,	0,	0,	2,	512	Harpsichord
ENVELOPE	7,	0,	9,	9,	0,	2,	2048	Organ
ENVELOPE	8,	8,	9,	4,	1,	2,	512	Trumpet
ENVELOPE	9,	0,	9,	0,	0,	0		Xylophone

If you're unfamiliar with the sounds of the default envelopes, you may want to run The Magic Flute (Program 1). It gives a sampling of the 128's preset envelopes. Once you can distinguish the differences between these sounds, the effects of changing the various parameters are more easily understood.

Parameter Values

Here is an explanation of the ENVELOPE values and their effects:

a (attack, 0–15) is the rate at which a note reaches its peak volume. Note that the value of *a* is initialized to zero in the piano, calliope, drum, guitar, harpsichord, organ, and xylophone. The attack is instantaneous for these sounds, creating a percussive effect. For the preset accordion, flute, and trumpet envelopes, the attack is more gradual; the notes sneak in gently. As the value of *a* increases, the attack becomes "softer."

d (decay, 0–15) describes the rate at which the volume of a sound decreases from its peak level to its sustain level. Note that the value of *d* is initialized to zero in the accordion and calliope. Notes played in these envelopes maintain peak volume throughout the plateau, or sustain, phase. The rest of the preset envelopes decay at various rates to their sustain level.

s (sustain, 0–15) is the volume level a note holds from the end of the decay phase until the beginning of the release phase. In the cases where the value of *s* is 0 (piano, harpsichord, and xylophone), the volume decays completely to a zero volume level. Notes played in these envelopes have a staccato quality, and a whole note sounds the same as a quarter note followed by a dotted half rest. When *s* is 15, as in the preset calliope envelope, the sustain volume is the same as the peak volume in the attack phase; there is no decay phase. Whenever *s* is 15, *d* must be 0, or unexpected cyclical decay-sustain effects will result.

r (release rate, 0–15) is the last phase in the duration of a note, and it follows the sustain phase. The value of *r* determines the rate at which the volume drops from the sustain level to zero volume. Note that *r* is preset to either 0 or 1 in all of the default envelopes. This means that the release occurs rather rapidly, and there is a good reason always to set *r* to a low value in musical programs. If a note is followed by another specified for the same voice, the SID chip will turn off the first note and switch to the next at the specified moment. If, however, a note is not followed by another, the SID turns off the last note according to the value of *r* for its specific envelope. For example, when *r* is set at 15, the final whole note in a piece may linger for a surprisingly long time.

wf (waveform, 0–4) determines the timbre of the musical sounds produced by the 128. The SID chip is capable of generating sounds of four waveforms: triangle (*wf* = 0), sawtooth (*wf* = 1), square (*wf* = 2), and noise (*wf* = 3). The triangle waveform sounds warm and muted (calliope, flute, xylophone). The sawtooth waveform sounds bright (accordion, guitar) and is generally louder than the triangle. The sound of the square waveform varies according to the pulse width (*pw*) specified. The noise waveform generates a nonpitched "white noise" sound, which can be used to mimic drums, cymbals, and other percussion instruments. If *wf* is set to 4, a triangle wave is activated which is *ring-modulated* between voices. This is an interesting effect, but for now it will produce unwanted results if it's used in place of the other values of *wf* in our programs.

pw (pulse width, 0–4095) is specified only when the square waveform (*wf* = 2) is used. This parameter designates the duration of each pulse, and a variety of harmonic effects can be produced (*pw* = 512 for harpsichord and trumpet; *pw* = 1536 for piano; *pw* = 2048 for organ).

Customizing Envelopes

Program 4, "Custom Envelope," gives you an opportunity to adjust the ENVELOPE parameters and hear the results. Enter values for *a, d, s, r, wf* (and *pw* if *wf* = 2), and select a default (preset) envelope for comparison. The 128 plays an ascending C scale using the preset envelope, followed by a descending scale using your custom envelope. Then you can choose to play it again, select a different preset envelope for comparison, or change the parameters of your custom envelope. This should help to acquaint you with the ADSR and waveform elements of sound quality.

A Piano Is a Piano Is a ...

Before the piano was invented, there was the harpsichord. It's a wonderful sounding instrument, but it has just one sound. Hit a key hard or touch it lightly—the sound is the same. Release a key quickly or hold it down—no difference. It has the same ADSR every time. That's why there was so much excitement when the piano was introduced. It's touch-sensitive, and all aspects of ADSR are within the control of a skilled player. The name *pianoforte* was coined for this instrument because you could play dynamic levels ranging from soft (piano) to loud (forte) and anything in-between. The name, of course, has been shortened to *piano* over the years.

The 128's preset piano envelope (T0) plays with a percussive attack and a complete decay with no sustain. This is appropriate for staccato phrases, but legato and sustained parts aren't rendered true to the manuscript. In fact, the whole notes decay as rapidly as the quarter notes.

Program 5, "Vision Fugitive XVI," is a Prokofiev composition for piano which features both staccato and legato passages. The custom envelope is specified in line 20:

```
20 ENVELOPE 0,0,9,2,1,2,1536
```

This is essentially the preset piano envelope, with the sustain level raised to 2 and the release rate changed to 1. These changes allow you to hold the long notes for their full values. Try entering one of your custom envelopes in line 20 and see how this sounds on your "instrument."

The Prokofiev piece has an eerie feel, although you'll find that your 128 doesn't really sound much like a piano. You can make some unusual and entertaining sounds on the 128, but you'll never mistake it for a piano. If it's any consolation, remember that Horowitz can't use his Steinway to run a spreadsheet or vaporize aliens.

Program 1. The Magic Flute

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
MR 10 VOL 8:TEMPO 13:PLAY"X0U9":A$="V1O5IGIFQEV2O4
      QGV3QCQCV2QGV1O5QEQRQE"
FR 20 B$="V1O5QFV2O4QGV3O3QBQBV2O4QGV1O5QFQRIFIE"
AA 30 C$="V1O5QDV2O4QGV3O3QBQBV2O4QGV1O5QDQRQD"
HH 40 D$="V1O5HEV2O4QGV3QCQCV2QGV1QRO5IGIF":E$="V1
      O5HEV2O4HGV3HCHCV2HGV1O5QEQE"
JJ 50 F$="V1O5.QFV2O4WAV3WFV1O5IGQAQF":G$="V1O5QEV
      2O4HGV1O5QEQDV2O4HGV1O5QD"
HF 60 H$="V1O5HCV2O4QCQEQCV1QR"
QR 70 FOR T=0 TO 9:PRINT USING"{CLR}{7 DOWN}{TAB}
      {LEFT}THIS IS ENVELOPE NUMBER #";T
HQ 80 READ EN$(T):PRINT USING"{7 DOWN}{TAB}
      {5 SPACES}#############";EN$(T);:T$=STR$(T)
JK 90 PLAY"V1T":PLAY T$:PLAY"V2T":PLAY T$:PLAY"V3T
      ":PLAY T$
GK 100 PLAY A$:PLAY B$:PLAY C$:PLAY D$:PLAY E$:PLA
       Y F$:PLAY G$:PLAY H$:NEXT:END
BK 110 DATA "{3 SPACES}PIANO"," ACCORDION","
       {2 SPACES}CALLIOPE","{4 SPACES}DRUM","
       {3 SPACES}FLUTE","{3 SPACES}GUITAR","HARPSI
       CHORD","{3 SPACES}ORGAN","{2 SPACES}TRUMPET
       "," XYLOPHONE"
```

Program 2. American Patrol

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
JD 10 PRINT"{CLR}{5 DOWN}{TAB}{4 SPACES}AMERICAN P
      ATROL":PRINT"{5 DOWN}{2 TAB}{3 SPACES}BY":PR
      INT"{5 DOWN}{TAB}{5 SPACES}F. W. MEACHAM"
AJ 20 TEMPO 20:VOL 8:PLAY"X0U9":PU$="V1O4IASBO5S#C
      "
```

```
RE 3Ø  PLAY"V1T2V2T2V3T2":REM ***
XJ 4Ø  A$="V1O5.IDV3O3QDO4S#FV2SAV1O5.IDV2SRO4SAV3S
       #FSR"
KX 5Ø  B$="V3O3QAV1O5IDI#CIDV2O4QAV3Q#FV1O5IE"
DJ 6Ø  C$="V1O5.I#FV3O3QDO4S#FV2SAV1O5.I#FV2SRO4SAV
       3S#FSR"
GA 7Ø  D$="V3O3QAV1O5I#FIFI#FV3O4Q#FV2QAV1O5IG"
HC 8Ø  E$="V1O5.IAV3O3QDO4S#FV2SAV1O5.IAV2SRO4SAV3S
       #FSR"
QH 9Ø  F$="V3O3QAV1O5IAI#GIAV3O4Q#FV2QAV1O6ID"
CM 1ØØ  G$="V1O5HAV3O3QDQ#FV2O4SASRSASR":H$="V2QRV3
        O3QGQ#FV2O4QAV1O5.I#F"
KG 11Ø  I$="V3O3QAV1O5.IGV3O4Q#CV2QGV1O5IGI#F":J$="
        V1O5.IEV3O3QEO4Q#CV2QGV1O5.IG"
DP 12Ø  K$="V3O3QDV1O5.I#FV3O4Q#FV2QAV1O5I#FIE":L$=
        "V1O5.IDV3O3QAO4Q#FV2QAV1O5.I#F"
XD 13Ø  M$="V3O4IEV1O5.IEV3O4I#FIEV2Q#GV1.IBV3ID":N
        $="V3O4Q#CV1O5.I#CV3O3QBV2O4Q#GV1O5.ID"
DD 14Ø  O$="V3O3IAV1O5HEV3O3IBIAV2O4QGV3O3IG":P$="V
        3O3Q#FQEV2O4QGV1O4IASBO5S#C"
QQ 15Ø  W$="V1O5HAV3O3.IDSRO3QDV2O4QA":X$="V3O3QEQ#
        FV2O4QAV1O5.I#F"
QH 16Ø  Y$="V3O3QGV1O5.IBSR.IAV3O4SDV2SBSRSBV3SDSR"
        :Z$="V3O3QDV1O5.IGV3O4QGV2QBV1O5.I#F"
SK 17Ø  AA$="V3O3QEV1O5.IESR.IDV3O4S#GV2SBSRSBV3S#G
        SR":AB$="V3O3Q#GV1O5.I#CSR.IDV3O4QEV2QB"
RM 18Ø  AC$="V3O3QAV1O5QEI#FV3O4Q#CV2QEV1O5IG":AD$=
        "V1O5.I#FV3O2QAO4Q#CV2QGV1O5.IE"
PX 19Ø  AE$="V3O3QDV1O5.IDV3O4QDV2Q#FV1IAIB":AF$="V
        1O5I#CV3O3QAV1O5IDIEV3O4QDV2Q#FV1O5I#F"
JJ 2ØØ  AG$="V1O5.IGV3O3QAO4S#CV2SEV1O5.IGV2SRO4SEV
        3S#CSR"
BB 21Ø  AH$="V3O3QEV1O5.IGV3O4Q#CV2QEV1O5.IG"
AR 22Ø  AI$="V3O3QAV1O5IGI#FHGV3O4S#CV2SESRSEV3S#CS
        R":AJ$="V3O3QEO4Q#CV2QAV1O5Q#G"
XM 23Ø  AK$="V1O5.IAV3O3QDO4SDV2S#FV1O5.IAV2SRO4S#F
        V3SDSR"
DA 24Ø  AL$="V3O3QAV1O5IAIGI#FV3O4QDV2QAV1O5IG":AM$
        ="V1O5HAV3O3QDO4SDV2S#FSRS#FV3SDSR":AN$="V3
        O3Q#FO4QDV2QAV1O5Q#F"
KR 25Ø  AV$="V3O3Q#FO4QDV2QAV1O5QA":AW$="V1O5.IBV3O
        3QGO4SDV2SGV1O5.IBV2SRO4SGV3SDSR"
SR 26Ø  AX$="V3O2QBV1O6.QDV3O4QDV2QGV1O5IB"
PE 27Ø  AY$="V1O5.IAV3O3QDO4SDV2S#FV1O5.IAV2SRO4S#F
        V3SDSR":AZ$="V3O3QAV1O5IAIG.I#FV3O4QDV2QA"
BK 28Ø  BA$="V3O3Q#FV1O5.IGV3O4SEV2S#AV1O5.IGV2SRO4
        S#AV3SESR"
JB 29Ø  BB$="V3O3Q#CV1O5.I#CV3O4Q#FV2Q#AV1O5.IG":BC
        $="V3O2QBV1O5.I#FV3O4SDV2SBV1O5.I#FV2SRO4SB
        V3SDSR"
```

```
HR 300 BD$="V3O3Q#FV1O5I#FIE.IDV3O4Q#FV2QBV1SR":BH
       $="V3O3Q#FV1O5IAIG.I#FV3O4QDV2QA"
MF 310 BI$="V3O3QAV1O5.IGV3O4S#CV2SEV1O5.IGV2SRO4S
       EV3S#CSR"
AC 320 BJ$="V3O3QEV1O5.I#CV3O4Q#CV2QAV1O5.IE":BK$=
       "V3O3QDV1O5.IDV3O4S#FV2SAV1O5.IDV2SRO4SAV3S
       #FSR"
CE 330 BL$="V3O3QAV2O4Q#FV1O5.IDSRS#FSRS#FSR":BM$=
       "V1O5W#FV3QRO3I#FV2O4Q#A":BN$="V2O4QBV3O3I#
       GIRI#AV2O5Q#C"
PG 340 BO$="V2O5QDV3O3IBV1QRO5IDI#C":BP$="V1O5IDIE
       I#FIG":BQ$="V1O5WAV3QRO3IAV2O5Q#C"
JG 350 BR$="V2O5QDV3O3IBIRO4I#CV2O5QE":BS$="V2O5Q#
       FV3O4IDV1QRO5I#FI#E":BT$="V1O5I#FI#GIAIB"
MQ 360 BU$="V1O6W#CV3QRO4I#CV2O5Q#E":BV$="V2O5Q#FV
       3O4I#DIRI#EV2O5Q#G":BW$="V2O5QAV3O4I#FV1QRO
       5IAI#G"
KJ 370 BX$="V1O5IAIBO6I#CID":BY$="V1O6WEV3QRO4S#GV
       2O6SDSRSDV3O4S#GSR":BZ$="V3O4I#GV2O6IDIRIDV
       3O4I#GIR"
AK 380 CA$="V3O4IAV2O6Q#CV1QEQR":CB$="V1O2QAV3HAV2
       HA":CC$="V3O3IAV1O5IDV2O4I#FIRI#FV1O5IDV3O3
       IAIR":CD$="V3O3IAV2O4I#FV1O5ID"
BK 390 FOR R=1TO2:PLAY PU$:PLAY A$:PLAY B$:PLAY C$
       :PLAY D$:PLAY E$:PLAY F$:PLAY G$:PLAY H$
BE 400 PLAY I$:PLAY J$:PLAY K$:PLAY L$:PLAY M$:PLA
       Y N$:PLAY O$:PLAY P$:Q$=A$:R$=B$:S$=C$:T$=D
       $
BE 410 PLAY Q$:PLAY R$:PLAY S$:PLAY T$:U$=E$:V$=F$
       :PLAY U$:PLAY V$:PLAY W$:PLAY X$:PLAY Y$:PL
       AY Z$:PLAY AA$:PLAY AB$
PP 420 PLAY AC$:PLAY AD$:PLAY AE$:PLAY AF$:PLAY AG
       $:PLAY AH$:PLAY AI$:PLAY AJ$:PLAY AK$:PLAY
       {SPACE}AL$:PLAY AM$:PLAY AN$
PE 430 AO$=AG$:AP$=AH$:AQ$=AI$:AR$=AJ$:PLAY AO$:PL
       AY AP$:PLAY AQ$:PLAY AR$:AS$=AK$:AT$=AL$:AU
       $=AM$
CF 440 PLAY AS$:PLAY AT$:PLAY AU$:PLAY AV$:PLAY AW
       $:PLAY AX$:PLAY AY$:PLAY AZ$:PLAY BA$:PLAY
       {SPACE}BB$:PLAY BC$:PLAY BD$
DC 450 BE$=AW$:BF$=AX$:BG$=AY$:PLAY BE$:PLAY BF$:P
       LAY BG$:PLAY BH$:PLAY BI$:PLAY BJ$:IF R=2 T
       HEN480
GG 460 PLAY BK$:PLAY BL$:PLAY BM$:PLAY BN$:PLAY BO
       $:PLAY BP$:PLAY BQ$:PLAY BR$:PLAY BS$:PLAY
       {SPACE}BT$
HJ 470 PLAY BU$:PLAY BV$:PLAY BW$:PLAY BX$:PLAY BY
       $:PLAY BZ$:PLAY CA$:VOL 15:PLAY CB$:VOL 8
HB 480 NEXT R:PLAY CC$:PLAY CD$:END
```

Program 3. Minuet

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
CM 10  PRINT"{CLR}{5 DOWN}{TAB}{4 SPACES}MINUET (G
       {SPACE}MAJOR)":PRINT{9 SPACES}"{5 DOWN}
       {2 TAB}{3 SPACES}BY":PRINT"{5 DOWN}{TAB} JOH
       ANN SEBASTIAN BACH"
XP 20  TEMPO 10:VOL 6:PLAY"X0U9":PLAY"V1T6V2T0"
GD 30  A$="V2O3HGV1O5QDO4IGIAIBV2O3QAV1O5IC":B$="V1
       O5QDV2O3.HBV1O4IGIRIGIR"
EP 40  C$="V1O5QEV2O4.HCV1O5ICIDIEI#F":D$="V1O5QGV2
       O3.HBV1O4IGIRIGIR"
DG 50  FOR R=1TO2:PLAY A$:PLAY B$:PLAY C$:PLAY D$:E
       $="V1O5QCV2O3.HAV1O5IDICO4IBIA"
MA 60  F$="V1O4QBV2O3.HGV1O5ICO4IBIAIG":G$="V1O4Q#F
       V2QDO3QBV1O4IGIAIBV2O3QGV1O4IG"
KE 70  H$="V1O4QBV2QDO3IDV1O4HAV2O4ICO3IBIA":PLAY E
       $:PLAY F$:PLAY G$:PLAY H$
XA 80  I$="V2O3HBV1O5QDO4IGIAIBV2O3QAV1O5IC":J$="V1
       O5QDV2O3QGQBV1O4IGIRIGV2O3QG"
DS 90  K$="V2O4.HCV1O5QEICIDIEI#F":L$="V1O5QGV2O3QB
       O4ICV1O4IGIRV2O3IBIAV1O4IGIRV2O3IG":PLAY I$:
       PLAY J$:PLAY K$:PLAY L$
QC 100 M$="V2O3HAV1O5QCIDICO4IBV2O3Q#FV1O4IA":N$="
       V1O4QBV2O3HGV1O5ICO4IBIAV2O3QBV1O4IG"
MR 110 O$="V1O4QAV2QCQDV1IBIAIGV2O3QDV1O4I#F":P$="
       V1O4.HGV2O3HGO2QG":PLAY M$:PLAY N$:PLAY O$:
       PLAY P$:NEXT R
BF 120 FOR R=1TO2:Q$="V2O3.HGV1O5QBIGIAIBIG":R$="V
       1O5QAV2O3.H#FV1O5IDIEI#FID"
CX 130 S$="V1O5QGV2O3QEQGV1O5IEI#FIGV2O3QEV1O5ID":
       T$="V1O5Q#CV2O3HAV1O4IBO5I#CO4QAV2O2QA"
BP 140 VOL 8:PLAY Q$:PLAY R$:PLAY S$:PLAY T$:U$="V
       2O3.HAV1O4IAIBO5I#CIDIEI#F":V$="V1O5QGV2O3I
       BIRO4QDV1O5Q#FQEV2O4Q#C"
KH 150 W$="V2O4QDV1O5Q#FO4IAV2O3I#FIRIAV1O5I#CIR":
       X$="V1O5.HDV2O4QDO3QDO4QC":PLAY U$:VOL 10:P
       LAY V$:PLAY W$:PLAY X$
SQ 160 Y$="V2O3QBV1O5QDO4IGV2QDV1I#FQGV2O3QB":Z$="
       V2O4QCV1O5QEO4IGV2QEV1I#FIGV2QC"
BX 170 AA$="V2O3QBV1O5QDQCV2O3QAQGV1O4QB":AB$="V1O
       4IAV2HDV1IGI#FIGQA":VOL 6:PLAY Y$:PLAY Z$:P
       LAY AA$:PLAY AB$
HE 180 AC$="V1O4IDV2O3HDV1O4IEI#FIGIAV2O3Q#FV1O4IB
       ":AD$="V1O5QCV2O3QEQGV1O4QBQAV2O3Q#F"
CK 190 AE$="V2O3QGV1O4IBO5IDO4IGV2O2IBIRO3IDV1O4I#
       FIR":AF$="V1O4.HGV2O3IGIRIDIRO2IGIR"
RJ 200 PLAY AC$:VOL 8:PLAY AD$:PLAY AE$:PLAY AF$:N
       EXT R:END
```

Program 4. Custom Envelope

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
BK 5 GOSUB 1ØØØ
HA 1Ø TEMPO 8:VOL 8:PLAY"XØU9"
CK 15 FOR PRESET=Ø TO 9:READ N(PRESET),A(PRESET),D
      (PRESET),S(PRESET),R(PRESET),W(PRESET)
GA 17 IF PRESET=Ø OR PRESET=6 OR PRESET=7 OR PRESE
      T=8 THEN READ P(PRESET)
CG 19 NEXT
RS 2Ø INPUT"{CLR}{DOWN}DEFAULT ENVELOPE TO COMPARE
       (Ø-9)";E:IF E<Ø OR E>9 THEN2Ø
EX 3Ø PRINT"{2 DOWN}{RVS}FOR CUSTOM ENVELOPE{OFF}
      {DOWN}":INPUT"ATTACK RATE (Ø-15)";A:IF A<Ø O
      R A>15 THEN3Ø
EH 4Ø INPUT "DECAY RATE (Ø-15)";D:IF D<Ø OR D>15 T
      HEN4Ø
RD 5Ø INPUT"SUSTAIN LEVEL (Ø-15)";S:IF S<Ø OR S>15
       THEN5Ø
CE 6Ø INPUT"RELEASE RATE (Ø-15)";R:IF R<Ø OR R>15
      {SPACE}THEN6Ø
CF 7Ø INPUT"WAVEFORM (Ø-4)";W:IF W<Ø OR W>4 THEN7Ø
FK 8Ø IF W<>2 THEN1ØØ
KH 9Ø INPUT"PULSE WIDTH (Ø-4Ø95)";P:IF P<Ø OR P>4Ø
      95 THEN9Ø
KQ 1ØØ PRINT"{DOWN}PRESS P TO PLAY AND COMPARE YOU
       R CUSTOM":PRINT USING"ENVELOPE TO DEFAULT E
       NVELOPE #";E
SP 11Ø PRINT"{DOWN}PRESS C TO CHANGE YOUR CUSTOM E
       NVELOPE":PRINT"{DOWN}PRESS D TO CHANGE THE
       {SPACE}DEFAULT ENVELOPE":PRINT"FOR COMPARIS
       ON"
PG 115 PRINT:PRINT"PRESS Q TO QUIT"
CJ 12Ø GET X$:IF X$="P" THEN18Ø
HH 13Ø IF X$="C" THEN PRINT"{CLR}":GOTO3Ø
KF 14Ø IF X$="D" THEN16Ø
EM 145 IF X$="Q" THEN PRINT"{CLR}SID CLEARED.":GOS
       UB 1ØØØ:END
SP 15Ø GOTO12Ø
PK 16Ø INPUT"{CLR}{DOWN}DEFAULT ENVELOPE TO COMPAR
       E (Ø-9)";E:IF E<Ø OR E>9 THEN16Ø
EP 17Ø GOTO1ØØ
DQ 18Ø GOSUB 1ØØØ:A$="V1O4QCIDIEIFIGIAIBO5WCQR":B$
       ="V2O5QCO4IBIAIGIFIEIDWCQR":ENVELOPE N(E),A
       (E),D(E),S(E),R(E),W(E),P(E)
DC 19Ø PRINT USING"{DOWN}{OFF}{6 SPACES}THIS IS DE
       FAULT ENVELOPE #";E
```

```
MF 200 A$="V1T"+STR$(E)+A$:PLAY A$:GOSUB 1000:PRIN
       T"{UP}{6 SPACES}{RVS}THIS IS YOUR CUSTOM EN
       VELOPE":ENVELOPE E,A,D,S,R,W,P
RJ 210 B$="V2T"+STR$(E)+B$:PLAY B$:PRINT"{UP}
       {34 SPACES}":GOSUB 1000:GOTO120
JM 1000 CLEAR=54272:FOR SID=CLEAR TO CLEAR+24:POKE
        SID,0:NEXT:RETURN
CK 3000 DATA 0,0,9,0,0,2,1536
SK 3002 DATA 1,12,0,12,0,1
MX 3004 DATA 2,0,0,15,0,0
SP 3006 DATA 3,0,5,5,0,3
BJ 3008 DATA 4,9,4,4,0,0
DP 3010 DATA 5,0,9,2,1,1
HP 3012 DATA 6,0,9,0,0,2,512
SS 3014 DATA 7,0,9,9,0,2,2048
DJ 3016 DATA 8,8,9,4,1,2,512
RP 3018 DATA 9,0,9,0,0,0
```

Program 5. Vision Fugitive XVI

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
MB 10 PRINT"{CLR}{5 DOWN}{TAB}{2 SPACES}VISION FUG
      ITIVE XVI":PRINT"{5 DOWN}{2 TAB}{3 SPACES}BY
      ":PRINT"{5 DOWN}{TAB}{3 SPACES}SERGEI PROKOF
      IEV"
JS 20 TEMPO 6:ENVELOPE 0,0,9,2,1,2,1536:PLAY"X0U9"
      :PLAY"V1T0V2T0V3T0"
CC 30 A$="V1O5HEV3QRO4HEV1O5Q#DV2HFV1QDV3O4HE"
BS 40 B$="V1O5Q#CV2HEV1QCV3O4HEV1O4QBV2O5H#DV1O5IC
      V3O4HEV1IB"
BS 50 C$="V1O4Q#FV2O5HDV1O4Q#GV3HEV1QAV2O5H#CV1O4I
      #AV3HEV1IB"
EF 60 D$="V1O5QCV2HCV1I#CV3O4HEV1O5ID.Q#DV2O4HBV3H
      EV1O5SES#G":VOL 9:PLAY A$:PLAY B$:VOL 7:PLAY
       C$:VOL 9:PLAY D$
QK 70 E$="V1O5HEV2O4H#AV3HEV2O5HFV1Q#DQDV3O4HE":F$
      ="V1O5Q#CV2HEV1QCV3O4HEV1QBV2O5H#DV1ICV3O4HE
      V1IB"
QQ 80 G$="V1O4Q#FV2O5HDV1O4Q#GV3HEV1QAV2O5H#CV1O4I
      #AV3HEV1IB"
FQ 90 H$="V1O5HCV2HCV3O4HEV2HBV1O5HEV3O4QE"
KX 100 VOL 11:PLAY E$:PLAY F$:VOL 9:PLAY G$:VOL 7:
       TEMPO 5:PLAY H$:TEMPO 6
HS 110 I$="V3O3SEV2O4IGV1IBIRV3O3SBSRO4SFV2QDV1QAV
       3SRO3SBSRO2SAV1O4IEIRV3O3SFSRSBV1O4QDV3SRO3
       SFSR"
```

```
CS 12Ø J$=I$:K$="V3O3SEV1O4IEIRV3O3SBSRO4SFV1QDV3S
       RO3SBSRO2SAV2O4IGV1IBIRV3O3SFSRSBV2O4QDV1QA
       V3SRO3SFSR"
QC 13Ø L$="V3O3SEV1O4IEIRV3O3SBSRO4SFV1QDV3SRO3SBS
       RO2SAV1O4HEV3SRO3SFSRSBSR.IFV2IR"
CB 14Ø VOL 5:PLAY I$:PLAY J$:PLAY K$:PLAY L$
AX 15Ø M$="V2IRV1O4ICIRV3O3SASRO4SFV1QDV3SRO3SASRS
       CV2O4IAV1O5ICIRV3O3SGSRO4SCV2QEV1QBV3SRO3SG
       SR"
EA 16Ø N$="V3O3SFV1O4IDIRV3O3SASRO4SGV1QEV3SRO3SAS
       RSCV2O4IBV1O5IDIRV3O3SASRO4SEV2QFV1O5QCV3SR
       O3SASR"
QQ 17Ø O$="V3O3HBV1O4IGIRV2SESRSDV1QFV2SRICIRV1IBV
       3O3QAV2O4SCSRSDV3QFV1QAV2SRIE"
MM 18Ø P$="V2IRV1O4IGV3O3HBV2O4SESRSDV1QFV2SRICIRV
       1IBV3O3QAV2O4SCSRSDV3QFV1QAV2SRIE"
CB 19Ø PLAY M$:VOL 7:PLAY N$:PLAY O$:PLAY P$:Q$="V
       2IRV1O4HGV3O3WBV2O4SCSRSDSRIEIRV1HFV2SDSRSC
       SRID"
GM 2ØØ R$="V2O4HDV1HGV3O3SBSRSASRSGSRSFSRHEV2O4HEH
       R"
SR 21Ø S$=A$:T$=B$:PLAY Q$:VOL 5:TEMPO 5:PLAY R$:T
       EMPO 6:VOL 9:PLAY S$:PLAY T$:U$=C$
BJ 22Ø V$="V1O5HCV2WCV3O4WEV1O6.HC":W$="V2QRO3QAV3
       QDV1O4QAQGV3O3QCV2QGQFV3O2QBV1O4QF"
BH 23Ø X$="V1O5HEV3O4QEHEV1O5Q#DV2HFV1QDV3O4HE":PL
       AY U$:PLAY V$:VOL 7:PLAY W$:PLAY X$
QG 24Ø Y$=T$:Z$=U$:AA$=V$:AB$=W$:PLAY Y$:PLAY Z$:P
       LAY AA$:TEMPO 5:PLAY AB$:AC$=X$:AD$=Y$
RS 25Ø AE$="V1O4Q#FV2O5HDV1O4Q#GV3HEV1QAV2O5H#CV1O
       4I#AV3QEV1IB"
CE 26Ø AF$="V1O5HCV3QCO4QEQCV1O6HEV3O3QE"
ES 27Ø VOL 5:PLAY AC$:PLAY AD$:PLAY AE$:PLAY AF$
JK 28Ø AG$="V3O3WEV1O6WE":AH$="V1O6.WEV2O5.WEV3O3.
       WE":PLAY AG$:VOL 3:PLAY AH$:END
```

Chapter 4

CP/M

Programming the Z80

Morris Simon

If you've already explored BASIC and 6502 machine language, the Commodore 128 offers a brand-new and fascinating world of programming—the Z80 chip.

Are you taking full advantage of the powerful Z80 processor in your Commodore 128? I don't mean just using CP/M mode to run the thousands of free programs in public domain libraries, although that's a good start. I mean writing your own applications in Z80 assembly language.

Why? For the same reasons that many professional programmers prefer machine language (or assembly language) over high-level languages like BASIC: speed, efficiency, maximum control, and just plain fun.

With a little reading and a little money, you can convert your 128 into an excellent Z80 CP/M development system. All you need to start is an assembler to assemble your source code into machine language, a loader to put your program where it'll be running in your RAM, and a debugger—since machine language programs don't always run the first time. Note that I haven't mentioned an editor: Most word processors will do just fine. Except for the Commodore 128, the rest of the system can cost less than $100.

You have to start learning one of two kinds of assembly language: either Intel's 8080 or Zilog's Z80 instruction set. Which one? By the time you finish this article, you'll be able to decide for yourself.

The Transition from 6502 Machine Language

If you've ever written a machine language (ML) program for the 64 or 128, learning Z80 machine language shouldn't be very difficult. The same ML operations found in the 6502 family of chips are available in the Z80 (plus a few more), but the Z80 uses different mnemonics. If you know what the 6502's LDA #5 means, you'll probably recognize the Z80 instruction LD A,5. The two instructions are spelled a little differently, but they do the same thing; they put the number 5 into the accumulator.

You should be aware of a few differences before starting, especially if you're experienced in 6502 ML. First, the Z80 has more registers than the 6502's A, X, and Y. The Intel instructions support seven registers: A, B, C, D, E, H, and L. These are sometimes paired up for use as pointers to memory: BC, DE, and HL (the memory location pointed to by HL is called the M register, even though it's in RAM and isn't a true register). The Zilog commands include other registers, IX and IY, for example. All of these additional registers serve a purpose: Indirect-Y addressing, which is widely used in 6502 programs, is not directly supported on the Z80. Instead of putting an address in a zero-page address and loading the accumulator through the pointer there, as you would in a 6502 program, you put the address into a register pair (HL, for example). The register pair serves the same purpose on the Z80 as zero-page indirect addressing does on the 6502.

There's a minor difference in hexadecimal notation as well. When you're writing Z80 ML, it's more common to add the letter *H* after a hex number instead of putting a dollar sign in front of it. You'd substitute 400H for $0400, for example.

In Commodore 6502 ML, the Kernal routines are important; they provide common entry points, regardless of which computer you own, for essential routines that open files, get a character, print to the screen, close files, and so on. CP/M uses the same idea in its Basic Disk Operating System (BDOS) and Basic Input/Output System (BIOS) routines. The BIOS call for getting a character from the keyboard works the same on a 128, Kaypro, Osborne, or other CP/M computer. The machines are different on the hardware level, but from a software standpoint, the operating system looks the same. If you plan to do a Z80 program, it pays to learn how the BDOS and BIOS routines work.

Intel Mnemonics

The Intel instruction set was designed for the Z80's precursor, the 8080. In the early days of microcomputers, most programmers wrote in 8080 assembly language. Digital Research's CP/M and most of the earlier CP/M software are written in 8080 code.

Intel (8080) mnemonics can be confusing when you're first learning them. For example, MOV means load, as in

```
MOV A,C  ;LOAD REGISTER A WITH BYTE IN REGISTER C
```

(In machine language, anything after a semicolon is a comment to be read and will not be processed by an assembler.)

In the example above, it might appear that the byte in register C has been shifted to register A, but actually it's only been *copied* to register A (the accumulator). Other instruction sets, including the Zilog, use LD (for LoaD) for this operation. Note that MOV and LD are just different ways of saying the same thing; when the program is assembled, the object code—the numbers in memory—will be the same.

Zilog Mnemonics

The Zilog instruction set reflects the greater power of the Z80 CPU, which has many more 16-bit registers and register pairs, and facilities for direct input/output operations. The major improvements over the 8080 instruction set include the following:

- Direct loading of all 16-bit registers and pairs
- Instructions for two new 16-bit index registers (IX and IY)
- Instructions to switch sets of registers
- Direct input and output addressing
- Block shift and block transfer instructions
- Enhanced jump instructions
- And a more comprehensible system of mnemonics

Obviously, Zilog's instruction set was designed to take advantage of all the Z80 improvements over its cousin, the pioneer 8080. In fact, if you write programs using Z80 mnemonics, you're accessing about as much power as you can from any eight-bit processor. So why would anyone go to the trouble of learning 8080 code when the Z80 instruction set can be mastered just as easily? Let's look into that.

The choice between Intel and Zilog instruction sets depends on the assembler you're using to convert your source code into machine language. Digital Research (the people who invented CP/M) decided to stick with the Intel 8080 instruction set for all their software, including their workhorse assemblers, which Commodore packages for the 64 (ASM) and the 128 (MAC and RMAC).

The Z80 commands are a *superset* of 8080 commands, in the same way that Commodore 128 BASIC (version 7.0) is a superset of the Commodore 64's BASIC 2.0. The Z80 can do everything the 8080 can do, plus it has many additional commands and more registers.

Even though the 128's CP/M assembler supports the Intel mnemonics, there's an easy way to use the Zilog instruction set if you want to.

Macros

The Zilog set includes a very useful single instruction, DJNZ *address*, which is used to control loops. (DJNZ is Decrement and Jump if Not Zero.) To use it, simply load register B with the number of times you want a loop to occur, follow the load instruction with your loop subroutine, and end it with DJNZ and the address of the beginning of the loop. The program automatically decreases the value in register B and repeats the loop until B equals zero.

If you want to emulate DJNZ with Intel instructions, you have to list both components of DJNZ separately, first by DEC B, which decreases the counter by one, and then by JNZ *address*, which performs the routine as long as B does not equal zero. You might then declare the combination "DEC B" + "JNZ *loop address*" as an external macro.

Not all Z80 instructions can be emulated by Intel mnemonics, however, even with complex Intel macros. That's why you'll find a file called Z80.LIB on one of the CP/M source disks from Commodore (these are the disks you receive when you mail the card in the CP/M section of the *System Guide;* there's also a very detailed 500+ page CP/M manual included with the disks).

To use Z80.LIB, put this line at the beginning of your source file:

```
MACLIB Z80
```

This tells the assembler you want to include the macro library file called Z80. With this line at the beginning of your program, you can use any or all of the Z80 command set. In a nutshell, it provides a lookup table that matches the Z80 mnemonics with the appropriate opcodes. It doesn't just emulate DJNZ by splitting it into DEC and JNZ; for example, it actually assembles the DJNZ instruction into the corresponding 8080-style ML instruction.

CP/M and Assembly Language

CP/M is an operating system rather than a language. The relationship between CP/M and machine language is very simple. From your program, you make *calls*—or *jumps*—to constant locations in CP/M memory anytime you need the operating system to do something. CP/M has further instructions stored at those locations and will return control to your program when the job (whatever it is) is done.

For example, location 0000H contains a jump instruction to the warm-boot routine in CP/M's BIOS. Therefore, anytime you wish to exit a program you can include JP 0000H (or JP 0 since decimal 0 = 0H), and the program will exit to a warm boot at that point. This is the way to finish a program and send the user back to the CP/M A> prompt.

Most CP/M calls are made to location 0005H, where another jump instruction passes control of the program to CP/M's BDOS. Before you CALL BDOS, your program needs to give CP/M a little more information, particularly the number of the *service call* you wish to make. You put this number in register C. For example, CP/M service call 9 is a print string function which prints to the screen an ASCII string ending in a dollar sign. BDOS must see the number 9 in register C *and* the string's starting address in register pair DE. At another place in your program, you'll define the string to be printed with a DB (Data Byte) instruction (and end it with a $). For example,

```
BDOS EQU  0005H   ;THIS EQUATE WILL OCCUR EARLY
                  IN PROGRAM
     ...          ;OTHER CODE
     LD   C,9     ;CP/M FUNCTION 9 NOW IN REG C
     LD   DE,MSG  ;POINT REGS DE TO ADDRESS OF
                  MESSAGE
     CALL BDOS    ;"BDOS" WAS DEFINED EARLIER AS
                  "0005H"
     ...          ;OTHER CODE
MSG  DB "This is the message to be printed.$"
```

Notice how the entries BDOS and MSG are defined in the leftmost label column. A good assembly program will clearly separate labels, instructions, operands, and comments:

```
LABEL    INSTR    OPERAND    ;COMMENT
```

It's common practice among Z80 programmers to place comments on nearly every line of source code.

In the 128 version of CP/M (CP/M 3.0, also called CP/M-Plus), there are more than 60 specialized CP/M service calls you can use in your assembly language programs. To get the most out of your machine, you should begin practicing each of the service calls in sample programs until you understand what they do.

Z80 Programming Techniques

In the examples that follow, I've used true Zilog mnemonics simply because they take less space and are easier to understand.

Loading and Storing Registers

The Z80 allows you to load practically any register from memory, from another register or by immediate loading of a value. The basic form of a typical load instruction is

LD *destination, source*

The *destination* may be either a register or a memory location, and *source* may be a register, memory location, or a value. For example,

```
LD A,C         ;LOAD ACCUMULATOR FROM C
```

Values can be loaded directly into registers:

```
LD C,6         ;PUT VALUE 6 DECIMAL IN REGISTER C
```

or indirectly from addresses in RAM:

```
LD A,(5973H)   ;LOAD A WITH BYTE FROM ADDRESS 5973
               HEX
```

or indirectly from addresses in registers:

```
LD A,(BC)      ;LOAD A WITH BYTE FROM ADDRESS IN BC
```

Note that a value inside parentheses is an address, while one without parentheses is a numeral. Remember that if you want a hex number (or address) you must add the letter *H;* Z80 assemblers default to decimal numbers.

Storage of data is done by the reverse procedure:

LD (*address*), *register*

The stack pointer (SP) is a 16-bit value in the Z80, which means you can point it to any location in memory. This is quite useful in situations where you need to clear out a section of memory. Just set the stack pointer and PUSH a series of zeros onto the stack. If you wanted to save the stack pointer first, you'd use this instruction:

```
LD (0A71H),SP ;LOAD STACK POINTER (A 16-BIT ADDRESS)
              ; INTO THE TWO-BYTE LOCATION START-
              ING AT
              ; A71 HEX; THE LOW BYTE WILL BE STORED
              ; FIRST, THEN HIGH BYTE AT A72 HEX
```

A common application of the Z80 loading operations is to initialize RAM locations with certain values. For example, let's store a byte of data (BYTE) at location 500H:

```
LD A,BYTE    ;PUT DATA BYTE INTO ACCUMULATOR
LD (0500H),A ;COPY BYTE IN REGISTER A TO 500 HEX
```

The same principle can be used to initialize word-length values by sending them through the HL register pair, which functions almost as a 16-bit accumulator:

```
LD HL,WORD ;PUT DATA WORD INTO HL PAIR
LD (0500H),HL ;COPY L INTO 500H & H INTO 501H
```

Arithmetic and Logic

The Z80 allows greater flexibility in arithmetic and logical operations than most other eight-bit CPUs. For a simple example, let's add a constant, NUMBER, to whatever variable happens to be in a given location, (VAR), and then change the variable:

```
LD A,(VAR)     ;COPY VARIABLE INTO ACCUMULATOR
ADD A,NUMBER   ;DO THE ADDITION
LD (VAR),A     ;AND CHANGE THE VARIABLE AT LOCA-
               TION VAR
```

Further examples of such routines would take up too much space, but there are good mathematical macros in most Z80 libraries for anything you want to do. The Z80's double (16-bit) registers make double-precision calculations a breeze.

Bit Manipulation

Most eight-bit computers have only one way to manipulate individual bits—by using logical instructions such as AND and OR. The Z80 lets you use those, if you wish, but it also provides three special bit-manipulation instructions: SET, RES (for reset), and BIT (to test a bit). Each of these codes works the same way:

```
SET n,r   ;SET BIT n OF REGISTER r
RES n,r   ;CLEAR (RESET) BIT n OF REGISTER r
BIT n,r   ;TEST BIT n OF REGISTER r—SET THE ZERO
          ; FLAG IF n=0 OR CLEAR IT IF n=1
```

Shift Operations

For those of you who like binary operations, the Z80 allows you to perform shift and rotate functions on any register or memory location.

Branching Instructions

The Z80's amazing repertoire of relative and absolute jumps allows conditional and unconditional branching by testing single bits, individual flags, and value comparisons. You can test a value either in a register or in memory for sign, carry, zero, or parity/overflow, and then branch in ways that resemble GOTO, IF, and THEN instructions in BASIC. Some common examples:

```
CP    LIMIT    ;IS ACCUMULATOR GREATER THAN
               LIMIT?
JR    NC,NEXT  ;NO CARRY FLAG SO GOTO NEXT
AND   A        ;SET ALL FLAGS TO TEST ACCUMULATOR
JP    P,NEXT   ;IF ACCUMULATOR IS POSITIVE GOTO
               NEXT
CP    KEY      ;ELSE SEE IF A=KEY
JR    Z,LAST   ;YES, A=KEY, SO GOTO LAST
NEXT
          . . .
LAST
          . . .
```

Here's something to watch out for if you're a 6502 programmer. With the 6502, you set the carry (SEC) before a subtract-with-carry (SBC) operation. If the carry is still set after the subtraction, it means the first number was larger than the second. The Z80 is just the opposite: You *clear* the carry before adding or subtracting. This also affects comparisons. In the example above, the CP instruction compares the accumulator to the number called LIMIT. If there's no carry, the accumulator is larger than LIMIT. (On the 6502, the carry will be set if LIMIT is smaller.)

Loops

Looping with subroutines is facilitated in the Z80 by the DJNZ instruction mentioned earlier. It's very easy to combine registers B and C to use a 16-bit loop for longer executions, or to design nested loops within loops using DJNZ.

Handling Arrays, Tables, and Indexing

The two extra index registers, plus easier access to all register pairs, allow you to handle both indexed and unindexed arrays neatly and quickly. One common style of array handling involves pointing HL to a memory location, and then loading byte after byte from that location into DE for further processing. If you wish, register B can be used as a counter:

```
LD      HL,(START ) ;POINT HL TO START OF ARRAY
LD      B,10        ;SET B AS A COUNTER FOR TEN
                    REPETITIONS
MORE    LD E,(HL)   ;LOW BYTE GOES INTO E
INC     HL          ;INCREMENT HL BY ONE, POINT TO
                    NEXT BYTE
LD      D,(HL)      ;HIGH BYTE INTO DE
INC     HL          ;POINT TO NEXT ELEMENT
CALL    LOUT        ;THIS MIGHT BE A ROUTINE TO OUT-
                    PUT DE
DJNZ    MORE        ;RETURN FROM ROUTINE, DECREASE
                    COUNTER,
                    ;AND DO IT AGAIN UNTIL B=0
```

With indexing, a register is loaded with an offset value, which is then added to the base of the array each time the loop is repeated. This next example loads the accumulator with every eighth element in an array until 20 entries have been checked:

```
        LD    HL,(START) ;POINT HL TO START OF ARRAY
        LD    B,20       ;COUNTER SET FOR 20 ENTRIES
        LD    DE,8       ;OFFSET = 8 DECIMAL
MORE    ADD   HL,DE      ;OFFSET ADDED TO BASE,
                         STORED IN HL
        LD    A,(HL)     ;GET 8TH ELEMENT INTO
                         ACCUMULATOR
        CALL  LOUT       ;AND DO SOMETHING TO IT
        DJNZ  MORE       ;RETURN AND DO IT ALL AGAIN
                         UNTIL B=0
```

Block Move (LDIR) and Compare (CPIR)

These categories of instructions provide two of the more powerful enhancements of the Z80 over the 8080. Logically, both procedures are very simple and work the same way. You put the size of a block (in bytes) in register pair BC and then point to the beginning of the block with HL. If you're moving the block, you put the destination starting address in register pair DE:

```
LD    BC,128       ;SET COUNTER FOR 128 BYTES
LD    DE,NEWADD    ;DESTINATION TO START AT NEWADD
LD    HL,OLDADD    ;SOURCE STARTS AT OLDADD
LDIR               ;MOVE IT!
```

If all you want to do is scan a block of data for a particular byte, you can omit the destination address since the comparison will be done in the accumulator:

```
LD    BC,128     ;SET COUNTER FOR 128 BYTES
LD    HL,START   ;STARTING ADDRESS OF BLOCK IN HL
LD    A,1AH      ;PUT CONTROL-Z IN ACCUMULATOR
CPIR             ;COMPARE HL WITH A AND INCREMENT
                 HL
                 ;UNTIL EITHER ^Z IS FOUND OR BC=0
```

Enhanced Input and Output Instructions

With the Z80, you can input or output data from either your registers or RAM directly to and from your computer's various ports. And you can do this by blocks. The relevant instructions are IN, OUT, INI, IND, OUTI, OUTD, OTIR, and OTDR. They're easy to use, simply by specifying the port in question:

```
IN A,(28H)  ;GET A BYTE FROM PORT 28 HEX
OUT (C),B   ;OUTPUT BYTE IN B VIA PORT NUMBER IN C
```

The block transfers (INIR, INID, OTIR, OTDR) work very much like the block move and compare instructions described in the last section, reserving register C for the port address and HL for a memory address.

Interrupt Processing

Fast interrupts on the Z80 use the RST instruction plus the destination address. For example,

```
RST 38H ;TRANSFERS CONTROL TO ADDRESS 38 HEX
```

There will usually be a permanent jump instruction at the destination which then sends control to a special routine, such as a graphics driver. You can also store an elaborate subprogram in the alternate register set and shift to it with an RST jump.

Your Choice

Have you decided which instruction set and assembler you want to use in your 128? If you've already bought the optional Commodore development package, which includes MAC, RMAC (Relocatable code MACro assembler), the SID debugger, LINK loader, and LIB library manager, then you've got a choice. If you include the Z80.LIB macro file, you can use either (or both) instruction sets.

Two other fine assembler packages are Echelon's ZAS-ZLINK and Microsoft's M80-L80 systems. Both are compatible with the Intel-based ASM/MAC family, and ZAS can even assemble code for the Hitachi HD64180, a new eight-bit CPU that is upward-compatible with the Z80.

Regardless of your assembler selection, you should build up a collection of good Z80 subroutines. The best Z80 library I know of is SYSLIB3, a public domain version that may be on a CP/M bulletin board near you. Echelon distributes SYSLIB3 in a set with other libraries and manuals for Z80 development systems for under $100.

You really can't appreciate the speed and efficiency of your Z80 until you start speaking its own language. Until then, you'll just have to be content with running thousands of excellent programs written by other people. Learning Z80 assembly language will give you even greater computing power and will allow you to do exactly what you want to do with your machine. To make it easier, here's a list of books and software that will get you started.

Books

Cortesi, David E. *Dr. Dobb's Z80 Toolbox.* M&T Publishing, 1985.

Uses Intel instruction set; available on disks.

Leventhal, Lance A., and Winthrop Saville. *Z80 Assembly Language Subroutines.* Osborne/McGraw-Hill, 1983.

My favorite on Zilog mnemonics.

Miller, Alan. *8080/Z80 Assembly Language.* Wiley, 1981.
A good introduction to both instruction sets.

Waite, Mitchell, and Robert Lafore. *Soul of CP/M.* Sams, 1983.
A classic on the Intel instruction set.

Software

Commodore 128 development package (MAC, RMAC, SID, and so forth).
Commodore Business Machines, 1200 Wilson Dr., West Chester, PA 19380.

Echelon development system utilities (Z-Tools), for either Intel or Zilog programs.
Echelon, 885 N. San Antonio Rd., Los Altos, CA 94022.

Microsoft development system (M80, L80, and so on), for either Intel or Zilog programs.
Microsoft, 10700 Northrup Way, Bellevue, WA 98004

SYSLIB3, by Richard Conn (in my opinion, the best collection of Z80 routines).
Available in either Intel or Zilog mnemonics from Echelon or on some bulletin boards.

A Hands-On Introduction to 128 CP/M

Todd Heimarck

If you regard the 128's CP/M mode as a somewhat-forbidding new territory, this article is for you. It explains some common CP/M commands, with examples to try out, and concludes with a simple application you can use.

If you own a Commodore 128, there's a good chance that you previously owned a 64. You may regard 64 mode as "an old friend" and 128 mode as a welcome upgrade—with the commands you already know, more memory, and some amazing new keywords for making sound, music, graphics, and disk operations easier.

You might see CP/M as the foreign mode of this three-headed computer. There's a lot of software available, but how do you use it? How do you load and run programs? How do you format a disk and copy a program over? How does CP/M work?

Booting CP/M

Getting into CP/M mode is relatively easy. Turn on your disk drive and insert the CP/M disk that came with your 128. Then turn on the television or monitor. Finally, power on the 128, which should check the disk drive for a boot sector, and follow the instructions there. The boot sector on the CP/M disk causes the 128 to give control over to the Z80 chip and load CP/M. If you turn on the computer first, it will default to 128 mode. When 128 mode is active, you can type BOOT to move into CP/M mode—assuming the disk drive is turned on with the CP/M disk inside.

If booting doesn't seem to work, try turning the disk over.

The label that says *System Disk,* with a serial number, should be facing up.

Before reading any further, boot CP/M, either by typing BOOT in 128 mode or by turning on the computer last. The CP/M disk is formatted as a single-sided 1541 disk, so you should be able to use either a 1541 or a 1571 (or compatible third-party drive). Pay attention to the 40/80 DISPLAY switch. Working in 80 columns (with the button down and an 80-column monitor) is much easier, although 40 columns is marginally acceptable.

Your First Command: DIR

If everything works right, you should see the BOOTING CP/M message and some miscellaneous information about what's being loaded. When everything's ready and running, the A> prompt should appear. This means CP/M is waiting for a command and disk drive A is the default drive. If you try to run a program, it will load from drive A. If you own a second drive, which is device 9 in 64/128 mode, you can switch to it by typing B: (the prompt should change to B>). A third drive (device 10) would be drive C:, and so on. If you wanted to leave drive A: as the default, but temporarily use a program from B:, you would precede the program name with B: (B:HELP, for example), and if you wanted a command to act on the second drive, you would put the B: *after* the command (DIR B:, for example).

Type DIR, an abbreviation for DIRectory, and the screen will display the names of the files found on drive A. Equally acceptable is DIR A:. If you're accustomed to pressing F3 in 128 mode to see the directory, you'll be pleasantly surprised to find that F3 is preset to print DIR. It displays the directory of the current drive.

Press F3 or type DIR to see the directory. The 80-column screen shows everything. But if you're using a television or a 40-column display, you'll see only part of the directory. Hold down the CONTROL key and press the right-arrow key (above INST/DEL) to scroll the screen to the right. To move back, hold CONTROL and the left-arrow key. Even when you're working in 40 columns, the screen is 80 characters wide. You must scroll back and forth to see the whole thing.

The constant shifting back and forth can become annoying after a while, which is why the 80-column screen is preferable in CP/M.

Ask for HELP

Leave the main CP/M disk in the drive and type HELP DIR, and you'll be treated to an explanation of the DIR command. The detailed HELP files can explain a lot when you're new to CP/M. If you wish, enter .BUILT-IN or .WITHOPTIONS for more details about DIR (note the period in front of the subtopics). From the HELP> prompt, you can also type DIR BUILT-IN or DIR WITHOPTIONS. (Be sure to place a space between DIR and BUILT-IN; the space is a separator that divides the main topic and a subtopic.)

If you look at the directory, you'll see a file called HELP.COM, which is the HELP command (or HELP program). Typing HELP starts the program running. You don't have to type LOAD or RUN, just the name of a program that ends with the .COM extension. When you enter HELP DIR, you effectively tell CP/M to run the HELP program and act on the input DIR. Some programs take optional information like this; you might run across a sort program that requires the following syntax:

```
A:SORT B:NAMES.ASC A:INORDER.ASC;DIAMOND
```

This means use the SORT program from drive A:, make it read the file NAMES.ASC from B:, and send the alphabetized output to a file called INORDER.ASC on drive A:. The semicolon and a password are sometimes required to run a program (type HELP SET for more about passwords).

A complete list of HELP topics is at your fingertips. Just type ? or HELP at the HELP> prompt. Or type HELP HELP at the A> prompt. Many of these topics have subordinate subtopics and subtopics beneath subtopics. To print out the various help files, turn on your printer and press CONTROL-P. The files will not only print to the screen, but they'll also be sent to the printer. You can also use the DEVICE.COM program to set the console-out device (CONOUT:) to both screen and printer. Type HELP DEVICE to find out more about this program. It sometimes helps to add a space and [NOPAGE] after the topic name to prevent the PRESS RETURN TO CONTINUE prompt.

The Other Side of the Disk

Remove the CP/M disk, turn it over, and place it in your disk drive. There's information on both sides, and you'll have to flip the disk to read the other side. It's a good idea to press CONTROL-C before typing DIR. CONTROL-C "logs out" a disk; it tells the system that you're planning to switch disks. If you don't CONTROL-C first, CP/M sometimes thinks you're working with the same disk that was previously in the drive.

Type DIR (or press F3), and the directory of the other side will list on the screen. Now try the unshifted CRSR-down key (under the RETURN key), and DIR appears again. The CRSR-down key, which is not the same as the gray down-arrow key on the top row, allows you to repeat the last command—a sort of do-it-again key.

While the flip-side of the disk is in the drive, try HELP DIR again. The computer prints your command followed by a question mark, because it doesn't know how to HELP you.

Built-In Versus Transient: Commands and Programs

The first side of the CP/M disk contains a file called HELP.COM; the second side doesn't have this file. So the HELP command works when the first side is in the 1541 or 1571, but it's not a legitimate command when the disk is turned over.

HELP is a command (the .COM extension means COMmand), but it's on the disk; it's not part of the operating system.

In 64 or 128 mode, there's a definite distinction between *files* (programs or data on a disk) and *commands* (keywords that cause the computer to do something). A 64 or 128 file is on the disk, but a command is inside the computer. To run a program from 64/128 mode, you must first use the LOAD or DLOAD command (LOAD is built into the computer) to move the program from disk into memory. When the program has been transferred into the computer's RAM, you type RUN (another built-in command) to make the program start up. The BASIC program, in turn, contains additional commands that your 64 or 128 recognizes.

CP/M does things differently. Almost always, a command is also a program. In CP/M, you can run the HELP program, which could be called the HELP.COM command, or a BACKGAMMON program (the BACKGAM.COM command), or the

ALIEN INVADERS game (the ALINV.COM command), or the *WordStar* word processor program (the WS.COM command). A program is a disk-based command; in CP/M they're the same thing.

In order for you to use a CP/M command, it must be on the current disk and it must have a .COM extension. In 128 mode there are nearly 200 commands built in, but CP/M mode offers only 6: DIR, DIRSYS, ERASE, RENAME, TYPE, and USER. DIR prints the directory (remember to press CTRL-P if you want it sent to the printer). DIRSYS tells you if there are nonsystem files on the disk. ERASE scratches a file, and RENAME changes the name of a file. TYPE prints out the contents of a data file; you can try to TYPE a .COM file, but you won't see much that makes sense. USER changes the user area, which allows you to break a disk into up to 16 separate areas that act like subdirectories. These 6 built-in commands are exceptions to the rule that commands are disk-based. All commands other than these 6 are *transient.*

Transient commands are loaded from disk and executed, and then they disappear. Transient commands take up memory in the transient program area (TPA) while they're being executed, but when they're done, they're gone.

Here's something that might be a little confusing: DIR is a built-in command, but if you look at your CP/M disk on side 2, you'll see a DIR.COM file. This second DIR command is a transient program with more features than the built-in DIR. If you type DIR, CP/M uses the built-in command. But A:DIR makes CP/M go to disk A: for the command.

If you'd like to look at another disk, type A:DIR E:. The DIR command is read from disk A:, and then you're prompted to insert another disk (press RETURN when the second disk is in the drive).

Drive E: is a *virtual* drive, very useful when you own only one disk drive. If you have a command on one disk and want it to work on another disk, tell it to load from drive A:, but operate on E:, and CP/M will prompt you to switch disks at the appropriate time.

A Vulnerable Operating System

Like the disk-based commands, the entire CP/M operating system is on a disk. If you spill a cup of coffee on your system disk, you've lost CP/M. It's not part of your computer as 64 mode and 128 mode are. If you lose or ruin the disk, you no longer have CP/M.

Thus, it's very important to make a backup copy as soon as possible, and then store the original CP/M disk in a very safe place. Without the CP/M operating system, the Z80 chip might as well be without sight, hearing, or speech—unable to read the keyboard, unable to recognize any peripherals, unable to print to the screen. If the Z80 is the brain of the computer, CP/M is the eyes, ears, and mouth.

Before you do anything else, copy both sides of the CP/M disk to a new double-sided disk if you own a 1571, or to two single-sided disks if you have a 1541. Contrary to what the *System Guide* says, you can't use COPYSYS to back up CP/M. You must run a program called *PIP*.

First, you'll need the FORMAT.COM program. Use the DIR command to find the FORMAT command on one side or the other of your CP/M disk. If you don't want to look at the whole directory, type DIR FORMAT.* to look for any files called FORMAT.

When the proper disk is in the 1541 or 1571, type FORMAT (don't include the .COM extension; CP/M already knows that it's a command). If you have two drives (with the second set up as device 9), you can put the disk containing FORMAT.COM in drive A (device 8) and type A:FORMAT B:, which means use the FORMAT command from drive A: and make it work on drive B:. If the format program is in drive B: (device 9) and you want it to format the disk in drive A:, type B:FORMAT A:, which means "take FORMAT from drive B: and apply it to drive A:."

The FORMAT program can tell whether you've got a single- or double-sided drive. If you're using a 1541, you'll have two choices of disk format: *C128 single-sided* or *C64 single-sided.*

Use the gray cursor keys to select one or the other, and then press RETURN. The only reason you'd ever choose the C64 option is if you're planning to use the disk with a 64 and the discontinued CP/M 2.2 cartridge, or if you happen to know someone with the 64 CP/M cartridge and want to send him or her some files. Otherwise, you should always choose the first (C128) option; it gives you more disk space.

1571 users have one more option: *C128 double-sided.* Always choose this format for disks you'll be using yourself; it gives you double the disk space. Of course, if you plan to give a CP/M disk to someone who owns a 1541, you would choose one of the other options.

After formatting the disks, try to get a directory by pressing F3 (or typing DIR). You should see the message NO FILE, which means there's nothing on the disk (yet).

PIP Means Copy

Now that the disks are formatted, you can begin to make the backup copies. Use DIR to find a program called PIP.COM, and type PIP. You'll see an asterisk (*) prompt. To escape the program, just press RETURN. But to use PIP, insert the source disk that you're copying *from,* and type the following line without any spaces:

```
e:=a:*.*
```

Knowing that drive E: is a virtual drive, you might be able to figure out how this works. It tells PIP to copy to drive E: everything from drive A: (the destination drive is always listed first). The asterisks are wildcards meaning every filename and every extension—in other words, everything on the disk—will be copied to drive E:. You copy a single file like DIR by typing

```
PIP E:=A:DIR.COM
```

You'll be prompted to switch disks several times. Remember that the source disk is drive A: and the destination disk is drive E:. Press RETURN each time you swap disks. It will take some time if you're working with a single drive.

If you have two drives, you can PIP a lot faster by typing

```
PIP B:=A:*.*
```

(PIP to B: everything from A:). You won't have to change disks, which saves a lot of time.

After copying the first side of the CP/M disk, flip it over and copy everything on the second side. When you're done, store the master disk in a safe place.

PIP is more than just a copy program; it allows you to read from one device and write to another. You can PIP a disk file to the printer, PIP from a modem to the screen, or PIP from the keyboard to a disk file.

An Introduction to Submit Files

In 64 mode, a common series of commands for loading a machine language program would go something like this:

```
LOAD "UTILITY",8,1
NEW
SYS 49152
```

You must type the three commands on separate lines, and you generally have to wait for the computer to finish executing the previous command before you type the next one.

Now imagine using a word processor to type the three lines and then creating a disk data file that contains these commands. If you could somehow tell the computer to execute all three commands, you wouldn't have to type each line. You'd just type something like EXECUTE "BUNCHOFCOMMANDS", and the three lines would be read from disk and executed, one by one. Many computers have this ability to do batch processing, to perform a series of commands stored in a file.

And this is exactly what the CP/M SUBMIT program does. Before going any further, you might want to use the HELP program to read more about SUBMIT and the editor program called ED (type HELP SUBMIT or HELP ED).

Creating a PROFILE Program

We're now going to create a special kind of submit file named PROFILE, which runs immediately after CP/M is booted (if you've used an IBM, CP/M's PROFILE.SUB can be compared to an AUTOEXEC.BAT file on an IBM).

Format a disk and copy (PIP) the following files to it:

```
CPM+.SYS
CCP.COM
ED.COM
SUBMIT.COM
DIR.COM
```

At the A> prompt, type ED PROFILE.SUB, which means edit a file called PROFILE.SUB. The ED.COM program will load and then print NEW FILE, because there's currently no file called PROFILE.SUB on the disk. A new file will be created. Enter the following lines at the given prompts:

```
 :* i
1 a:dir{CONTROL-Z}
 :* e
```

The initial asterisk (*) prompt indicates that ED is ready for a command. Typing **i** means insert a line and, since it's a new file, the first line is number one, which is why a 1 appears on the next line. The text we're adding is **a:dir**, because we're going to make the PROFILE file automatically run the DIR.COM program. Don't press RETURN (if you do, it will go on to line 2, which would be okay if we wanted a second command in the file, but we don't). {CONTROL-Z} marks the end of the file. The asterisk acts as a reminder that we're back at the command level, where typing **e** means "exit and write the file to disk."

When you return to the A> prompt, type DIR to verify that a file called PROFILE.SUB has been created. If you'd like to read this file, enter the following line (remember, TYPE is one of the six built-in commands):

type profile.sub

Now you're ready to test it. Hold down the CONTROL key and press ENTER (on the numeric keypad). CONTROL–ENTER works a lot like RUN/STOP–RESTORE in 64 or 128 mode. It forces CP/M to reboot. After CP/M loads, it will find the PROFILE.SUB file (which requires SUBMIT.COM to work). The text in the file (A:DIR) is printed on the screen, and the DIR.COM program runs. An alphabetized directory of the disk is then printed on the screen.

To run the commands in PROFILE again, type SUBMIT PROFILE. This is just a simple example; you could add several more commands to the PROFILE.SUB file. Or you could make PROFILE automatically load and run a game or other program. If you decide to start time-stamping your files, you could create a PROFILE.SUB routine that asks you for the date and time when you first turn on CP/M. For more about timestamps, see the HELP files on INITDIR, SET, and DATE.

If you'd like to get rid of the ED file, type ERASE ED.COM. You can shorten ERASE to ERA.

Experimenting with CP/M

There's a lot more you can do with CP/M. Many languages are available, including BASIC, C, COBOL, Forth, Turbo Pascal, and many others. If you plan to write programs, you'll need a language (CP/M is an *operating system,* not a language). To write actual .COM files such as we've been using requires either a Z80 machine language assembler or a compiler package for the language you're using.

In addition, there are a lot of good public domain programs available, if you can find a local CP/M user group (usually Kaypro or Osborne) or a local bulletin board. For more about public domain programs, see the following article, "CP/M Public Domain Software." To use a modem to download programs, you'll need the December 6, 1985 (or later) version of CP/M, which is now being shipped with 128s. If your version date is earlier (June or August, 1985), you can download the upgrade program in 64 or 128 mode, transfer it from a Commodore disk to a CP/M disk, and then use this program to modify CPM+.SYS. (Instructions for upgrading to the new version of CP/M are available on CompuServe and QuantumLink.)

CP/M Public Domain Software

James Adams

If you're not currently using CP/M on your 128, there's a whole new world of software waiting for you—and a lot of it is available at little or no cost.

One evening last fall, I turned on my Commodore 128 and dialed the bulletin board sponsored by my user group. The public messages had been scrolling past, when suddenly there was one that piqued my curiosity: the latest of many messages concerning the availability of the mysterious CP/M.

TO: ALL
SUBJECT: CP/M on 128

HELP, I REALLY LIKE MY 128, BUT WHERE CAN I GET SOFTWARE FOR THE CP/M MODE? PLEASE LEAVE E-MAIL IF YOU CAN HELP!

One of the attractive features of the 128 is its compatibility with the 64 and the large base of existing 64 software. Another selling point is 128 mode with its large and expandable memory, the fast disk drive, and a new BASIC with a wealth of new commands. But probably only a minority of 128 buyers were charmed by the 128's compatibility with CP/M. Living on the trailing edge of technology (CP/M is anything but new) has its advantages, however. There are thousands of CP/M programs, ready to run and waiting for the 128 user.

Free Programs

For those readers who are first-time users of computers, *public domain software* means programs that you can acquire at no cost—you don't pay anything. Here's how it works: Whenever you write something original, a poem, a song, a story, or a computer program, you own the copyright to that original work. This is true whether or not you actually register the copyright with the Library of Congress. If you then sell your creation to a publishing company, either you or the publisher will then own the copyright (the right to make copies), and you're paid a royalty fee based on the number of sales of your work. After a certain period of time, the copyright expires and the artistic work passes into the public domain. All songs written before 1900, for example, are in the public domain; anyone can perform them without paying a fee to the composer.

So if you write a program, you own the copyright to it, unless you choose to voluntarily put it in the public domain, meaning that anybody can use it and distribute copies. User groups are often an excellent source of public domain software.

Sometimes you'll pay a few dollars for postage, the cost of the disk, or the time it takes to copy the program. And, of course, if you download from a bulletin board system that qualifies as a long-distance call, you'll pay the usual long-distance charges.

Commodore computer owners are used to having a large base of public domain software as well as reasonably priced commercial programs from which to choose. Many Commodore business application programs are under $50; only a few top out above $100. The cost of a good CP/M program may run as much as four times more than its top-of-the-line Commodore counterpart. Commodore owners may balk at the thought of paying these prices for their software. Fortunately, there's an alternative for the new CP/M user: Public domain and freeware or shareware programs are abundant.

Freeware and *shareware* are terms for copyrighted software that the author distributes free of charge. Frequently, a notice is included with the program or documentation that if you enjoy the program you might send a small donation to the author. Often, you'll receive more detailed instructions or a chance to find out about bugs or upgrades when you send the money. Freeware and shareware are copyrighted software for which you don't have to pay if you don't want to.

Kaypro and Osborne Programs

When the 128 was first announced, reviewers noted that the new 1571 disk drive would be able to read disks formatted for several different computers. These included the Kaypro and Osborne computers. There's a lot of public domain software available for both of these computers. Find software for them, and your 128 is in business. It should be noted that the 1571 disk drive is a *must* because of CP/M's unique DOS format; a 1541 won't read Kaypro or Osborne disks. An 80-column display monitor, like the Commodore 1902, is also a practical necessity. CP/M uses 80-column display; while the 128 has the ability to run your programs with a 40-column screen using the window feature and right or left scrolling, you won't want to do this for very long.

There are several ways to get inexpensive or free CP/M programs. Seek out computer stores in your area which carry Kaypro equipment. Kaypro is an active company, and there's a lot of support for their machines. Keep your eyes open for Kaypro literature. Books and magazines may provide valuable hints, tips, CP/M information, and software advertisements.

If you live in or near a larger city, you'll probably be close to a Kaypro user group. Many of these groups have a public domain library which should be a wonderful source of material. They may also have literature available or sponsor lectures on CP/M. Also, a FOG (First Osborne Group) chapter may be in your area. FOG is a user group which began with the Osborne 1 computer in 1981. The group has expanded and now boasts a very loyal following of over 15,000 members who use or are interested in CP/M. They have a large library, and disks are available by mail.

While looking through CP/M-specific magazines, you'll come across many advertisements for public domain software collections. Many of these programs will run as is on your 128. Some of the public domain material is excellent and often supported by well-written, yet inexpensive books.

If several disk formats are offered, choose Osborne double-density (Osborne DD). Your second choice should be Kaypro II or Kaypro IV. If it's necessary to "install" the software and you're given a choice of terminal types, it's usually safe to say you have either an ADM-3A or an ADM-31.

Other Sources of Public Domain Software

Here are a couple of sources of public domain software that you'll want to contact:

Peopletalk Associates has put together a series of public domain disks formatted for the Kaypro. Utilities, useful business programs, games, and lots of documentation files are included. The *Free Software Handbook* is a valuable companion to the disks, providing documentation and advice on how to use the programs.

Micro Cornucopia also has several public domain disks—some with the dedicated programmer in mind.

Highly Recommended

As you build your CP/M library, you might want to consider some public domain titles that I recommend:

VDO. An acronym for *Video Display Oriented* text editor, this is a very basic word processing program that takes up only 8K. It could be a good educational tool for getting the feel of CP/M commands. The commands are similar to those used in *WordStar* (a commercial word processing program for CP/M).

PC FILE. This is a freeware program. If you like it, you can send the author a fee. Distribution is encouraged. This program will handle many of your database needs.

Adventure. This classic down-in-the-cave adventure game will access 192K at one time or another. The 500+ point version will keep you going for years as you discover new twists. The author's humor is refreshing in frustrating situations. *Adventure* is available in the Peopletalk series.

NSWP. The acronym stands for *New Sweep.* It may turn out to be the most valuable utility you'll ever own for your CP/M system. With a single program of only 11K, it replaces other utilities that would normally take up 100K of space. *NSWP* lets you copy, rename, delete, view, print, alphabetize, count, and size up your files. Put a copy on every disk you own; you'll use it in every session at your computer. Several versions of this program are in the public domain; I prefer *NSWP.205. NSWP.207* offers the disk label, while *NSWP.208* offers the directory listing in alphabetical order.

D. This directory program is almost as fast as the DIR command. *D* will list the contents of your directory along with the number of bytes used for each file and a brief summary of your used and unused disk space. When you begin to work with CP/M's "user areas," *D* can also give you directories of the hidden areas on your disk with a single command.

Dirf. CP/M has an abundance of directory programs. This one will allow you to add a short description to each directory entry—a feature that may be indispensable if you find yourself creating a lot of text files with a program such as *VDO.*

NULU. Short for *New Library Utility,* this utility is very similar to *NSWP.* It's used with special files called *libraries.* Libraries may be new even if you're a veteran Commodore user, so you may want to explore this form of disk management.

Handy System 200. This series of files is excellent if you have a computer on your desk at work. It's a calendar, memo pad, appointment book, phone book, decision support system, and more.

A Cautionary Tale

There are a few things to watch for with CP/M software. Your 128, running in CP/M mode, may do strange things, depending on the setup of the computer your software was originally configured to run on. The first version of *VDO* that I tried was patched for a 1984 Kaypro. It barely ran on my 128. Luckily, *VDO* and many other CP/M programs can be patched or configured for different machines and printers. I was able to get a version which had been patched for a 1983 Kaypro. This version was much better, but still a bit erratic on the 128. Later, I tried many of the two dozen patches for other computers. Several of these worked very well. Always try to get the most generic version available. Try to avoid versions that use graphics or unique screen displays.

Utility programs seem to be the most reliable on the 128. *Adventure, NSWP, D, Dirf,* and *NULU* all run as is on my 128. *Handy System 200* (or *Handy Version #2*) still has some problems clearing the screen. To my surprise, *PC File,* originally for the Osborne 1, does not run on a friend's 1984 Kaypro, but runs perfectly on my 128.

Programs which require a bit of explanation usually come with a "doc" file which can be viewed easily with an *NSWP*-type program. In a pinch, you can use the built-in TYPE command. Many programs will contain options that will allow you to configure the program to your individual likes and dislikes. Be sure to copy all of the files associated with a program. You may need several files just to run a single program.

In addition to the *Free Software Handbook* from Peopletalk Associates, I should also mention one other helpful book: *Free Software,* by Tony Bove, Cheryl Rhodes, and Kelly Smith, available at many computer stores. It contains a lot of CP/M-specific information about public domain programs. The first half of the book discusses modems, communications programs, and downloading programs from various user groups. The rest of the book consists of valuable information about using the public domain programs you've obtained.

Where to Write

Here are a few contacts you may wish to make to start building your CP/M library (as with most publishing companies, you're more likely to get a reply if you include a self-addressed stamped envelope):

FOG
P.O. Box 3474
Daly City, CA 94015

Peter C. Hawxhurst
(author of *Handy System 200)*
705 Bayside Court
Wheeling, IL 60090

Micro Cornucopia
P.O. Box 233
Bend, OR 97709

Peopletalk Associates
P.O. Box 863652
Plano, TX 75086

Chapter 5

Programming

SPRDEF Enhancer

Terry Roper

"SPRDEF Enhancer" upgrades the 128's built-in sprite editor by adding 11 new commands. Included are vertical and horizontal scrolls and flips as well as facilities for changing background, sprite, and multicolors from within sprite-definition mode. A reverse function is also provided.

Written in machine language, "SPRDEF Enhancer" runs in BANK 15 RAM from $18C1 to $1BFF, inclusive. To get started, enter the program listing and save your work. Refer to Appendix C for instructions on entering machine language programs.

Thanks to the BLOAD command, it is not necessary to load your programs in any particular order. Either BLOAD "ENHANCER" and then load your own programs, or vice versa, or load no other programs. In any case, type SYS 6337 and press RETURN. When you are ready to begin editing, type SPRDEF and press RETURN. You will see the normal editor screen. Answer the SPRITE NUMBER? prompt. SPRDEF Enhancer is now activated.

Editing Sprites

At the upper right corner of the screen, you'll see a sprite enclosed in a box. The color within this box is the same as the background color. Beneath the box, the status area appears:

```
HIRES  B C
  FUNC 1 3   5 7
  KEY# 1 2   3 4
MULTI  B M1 C M2
```

Depending on whether the sprite is in high resolution or multicolor, the lines that contain the word *HIRES* or *MULTI* and its corresponding row of colors will be masked. Press and hold the M key for a few seconds. Observe how the status area changes. Press M until MULTI B M1 C M2 appears. Above this line you'll see a parallel layout of keys. The top row contains the unshifted function keys. The next row contains the number keys 1–4. Below the number keys is a row of colors. The letters on the bottom line have these meanings:

Code	**Affected**	**Register**
B	Background color	$D021
M1	Sprite multicolor 1	$D025
C	Unique sprite color	$D027 + sprite number (0–7)
M2	Sprite multicolor 2	$D026

To change a color, pick a code. Then press the function key indicated by the number in the corresponding column. Similarly, set pixels by pressing the appropriate number keys. Now practice a little in hi-res mode. Note that the number keys set points exactly the same way as the regular editor. They are displayed only as an aid. In fact, all SPRDEF functions work exactly as they did before with the exception of the top row of cursor keys. Use these keys to scroll the sprite.

The last three commands are

(main keyboard)

Left arrow	**Horizontal flip**
Up arrow	**Vertical flip**
Asterisk	**Reverse**

These functions are self-explanatory. You might notice, while scrolling horizontally in multicolor mode, that it is faster than you might expect. This is because pairs of bits, not single bits, are scrolled. For the same reason, it is important to perform operations on sprites in the same mode in which they were drawn. Otherwise, you can really botch up a pattern.

All SPRDEF Enhancer commands are disengaged at prompts which require number input. When you exit the editor, press RUN/STOP–RESTORE to reactivate the function and top cursor keys.

SPRDEF Enhancer

See instructions in article, and read Appendix C, "MLX," before typing in the following program listings.

```
18C1:A2 CC AØ 18 8E 2A Ø3 8C BB
18C9:2B Ø3 6Ø 68 48 C9 5F FØ FB
18D1:Ø3 4C EB EE AD 4C 2D DØ CC
18D9:2C 78 A2 57 AØ 1B 8E 3C D2
18E1:Ø3 8C 3D Ø3 AD E7 1F 85 5F
18E9:68 29 FØ 85 66 A5 68 ØA B3
18F1:ØA ØA ØA 45 66 85 67 2Ø 77
18F9:3Ø 1B 2Ø A1 1A A9 ØØ 8D 2C
19Ø1:ØØ FF 58 18 6Ø 2Ø EB EE ØA
19Ø9:85 65 A2 ØA DD DD 1B FØ D9
```

```
1911:Ø5 CA 1Ø F8 3Ø 2F A5 4B DE
1919:A4 4C 85 63 84 64 A9 19 B9
1921:48 A9 33 48 BD EA 1B 8D 2A
1929:E8 1B BD F5 1B 8D E9 1B AB
1931:6C E8 1B A6 63 A4 64 86 9E
1939:4B 84 4C 2Ø D1 75 2Ø 4A AC
1941:76 A9 ØØ 85 65 2Ø A1 1A 7A
1949:A5 65 18 6Ø A2 14 AØ Ø2 59
1951:B1 4B 99 FA ØØ 88 1Ø F8 4D
1959:A5 FC 4A 66 FA 66 FB 66 1D
1961:FC 2C FA 12 1Ø Ø9 A5 FC 8A
1969:4A 66 FA 66 FB 66 FC C8 5C
1971:B9 FA ØØ 91 4B CØ Ø2 DØ 8A
1979:F6 E6 4B E6 4B E6 4B CA 1Ø
1981:1Ø CC 6Ø A2 14 AØ Ø2 B1 FD
1989:4B 99 FA ØØ 88 1Ø F8 A5 43
1991:FA ØA 26 FC 26 FB 26 FA CØ
1999:2C FA 12 1Ø Ø9 A5 FA ØA C2
19A1:26 FC 26 FB 26 FA C8 B9 13
19A9:FA ØØ 91 4B CØ Ø2 DØ F6 E6
19B1:E6 4B E6 4B E6 4B CA 1Ø C5
19B9:CC 6Ø AØ 3E B1 4B 99 BE ØF
19C1:ØØ 88 CØ 3B DØ F6 B1 4B F2
19C9:C8 C8 C8 91 4B 88 88 88 DA
19D1:88 1Ø F3 C8 B9 FA ØØ 91 A2
19D9:4B CØ Ø2 DØ F6 6Ø AØ ØØ A9
19E1:B1 4B 99 FA ØØ C8 CØ Ø3 4A
19E9:DØ F6 B1 4B 88 88 88 91 36
19F1:4B C8 C8 C8 C8 CØ 3F DØ 3A
19F9:F1 88 B9 BE ØØ 91 4B CØ Ø8
1AØ1:3C DØ F6 6Ø A9 14 85 4E 63
1AØ9:AØ Ø2 B1 4B 48 88 1Ø FA 78
1A11:AØ Ø2 68 2C FA 12 3Ø ØA 7Ø
1A19:A2 Ø7 4A 26 4D CA 1Ø FA BC
1A21:3Ø ØD A2 Ø3 4A Ø8 4A 26 62
1A29:4D 28 26 4D CA 1Ø F5 A5 CF
1A31:4D 91 4B 88 1Ø DC E6 4B 6F
1A39:E6 4B E6 4B C6 4E 1Ø C8 9D
1A41:6Ø 18 A5 4B 69 3C 85 4D A9
1A49:A5 4C 69 ØØ 85 4E A2 Ø9 44
1A51:AØ ØØ B1 4B 48 B1 4D 91 F5
1A59:4B 68 91 4D C8 CØ Ø3 DØ 74
1A61:F1 E6 4B E6 4B E6 4B C6 73
1A69:4D C6 4D C6 4D CA 1Ø EØ A2
1A71:6Ø AØ 3E A9 FF 51 4B 91 CD
1A79:4B 88 1Ø F7 6Ø EE 21 DØ C8
1A81:6Ø 2C FA 12 3Ø 12 AE FC 95
1A89:12 18 BD 27 DØ 69 Ø1 9D C2
1A91:27 DØ 6Ø 2C FA 12 3Ø EE CB
```

```
1A99:EE 25 DØ 6Ø EE 26 DØ 6Ø CØ
1AA1:2C FA 12 Ø8 Ø8 Ø8 AD 21 4A
1AA9:DØ 29 ØF Ø5 66 CD 19 1C 7B
1AB1:FØ 21 A2 19 AØ 1C 86 4D 5C
1AB9:84 4E A2 ØA AØ ØE 91 4D 66
1AC1:88 1Ø FB 48 18 A5 4D 69 9D
1AC9:28 85 4D 9Ø Ø2 E6 4E 68 D6
1AD1:CA 1Ø E9 AA A4 68 28 1Ø 7E
1AD9:Ø3 A6 68 A8 8E 27 1E 8C AA
1AE1:9F 1E AE FC 12 BD 27 DØ B9
1AE9:29 ØF Ø5 66 AA A4 68 28 5E
1AF1:1Ø Ø3 A6 68 A8 8E 29 1E 3A
1AF9:8C A4 1E A6 68 A4 68 28 9A
1BØ1:1Ø 1Ø AD 25 DØ 29 ØF Ø5 99
1BØ9:66 AA AD 26 DØ 29 ØF Ø5 83
1B11:66 A8 8E A1 1E 8C A6 1E 1F
1B19:A2 ØE A5 68 2C FA 12 1Ø EØ
1B21:Ø2 45 67 9D F9 1D 45 67 A6
1B29:9D C1 1E CA 1Ø EC 6Ø A2 A6
1B31:4C AØ 1B 86 4D 84 4E A2 3D
1B39:ØA AØ 19 B1 4D 2Ø DB 68 E5
1B41:C8 CØ 28 DØ F6 18 A5 4D CE
1B49:69 ØF 85 4D 9Ø Ø2 E6 4E 26
1B51:E8 EØ 12 DØ E4 6Ø CØ 53 Ø1
1B59:9Ø Ø7 CØ 57 BØ Ø3 B9 8A B6
1B61:1B 4C B7 C6 A2 A2 A2 A2 23
1B69:A2 A2 A2 A2 A2 A2 A2 A2 9F
1B71:A2 A2 A2 B8 B8 B8 B8 B8 54
1B79:B8 B8 B8 B8 B8 B8 B8 B8 AF
1B81:B8 B8 48 49 52 45 53 2Ø 4E
1B89:42 2Ø 43 2Ø 2Ø 2Ø 2Ø 2Ø 35
1B91:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø C7
1B99:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø CF
1BA1:2Ø 46 55 4E 43 2Ø 31 2Ø 26
1BA9:33 2Ø 2Ø 35 2Ø 37 2Ø 2Ø 17
1BB1:4B 45 59 23 2Ø 31 2Ø 32 74
1BB9:2Ø 2Ø 33 2Ø 34 2Ø 2Ø 2Ø F2
1BC1:2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø F7
1BC9:2Ø 2Ø 2Ø 2Ø 2Ø 4D 55 4C 4B
1BD1:54 49 2Ø 42 2Ø 4D 31 2Ø 65
1BD9:43 2Ø 4D 32 89 8A 8B 8C AØ
1BE1:85 86 87 88 5F 5E 2A ØØ BE
1BE9:ØØ DF BB 84 4D 7E 82 94 D5
1BF1:9D Ø5 42 72 19 19 19 19 2Ø
1BF9:1A 1A 1A 1A 1A 1A 1A ØØ 16
```

Disk Commands on the 128

Todd Heimarck

Whether you have a 1541 disk drive or a new 1571, this article will show you how to use the 128's powerful disk commands. A number of useful hints and shortcuts are included.

The Commodore 128's BASIC 7.0 is a vast improvement over previous Commodore BASICs. The computer has its share of flashy new commands, the ones that give you POKEless sprites, easy-to-program music and sound effects, and high-resolution graphics. The glamour of these powerful keywords can easily bewitch a new 128 owner.

Disk commands, on the other hand, are just disk commands. They're mundane. But if you learn about the new ways of loading, saving, and handling files, you'll save a lot of time, time that could be spent programming—or playing with sprites, music, and hi-res graphics.

We'll concentrate on using the 128 disk commands, most of which work equally well on the 1541 disk drive or the new 1571. But we'll also touch briefly on a few of the new 1571 DOS commands.

Loading BASIC Programs

As in BASIC 2.0, the LOAD command defaults to tape, so you must include the device number when loading from disk. But LOAD should never be necessary when DLOAD and RUN are available.

DLOAD is a new command; the *D* stands for disk, and it defaults to drive 0, device 8. If you own a dual drive, you can add a comma and either D0 or D1 to pick a drive for loading. Unfortunately, 128 owners may never see the 1572 dual drive. As of this writing, Commodore has apparently decided not to manufacture it. You can still add single drives to your system, though. To access a second or third drive, follow DLOAD with

A Dozen Ways to Load

If you want to load a BASIC program, you have four choices:

1. LOAD"*filename*",8
2. DLOAD"*filename*"
3. RUN"*filename*"
4. Press SHIFT–RUN/STOP

For machine language or binary files:

5. LOAD"*filename*",8,1
6. BLOAD"*filename*"
7. BLOAD"*filename*", B*bank*, P*address*
8. BOOT"*filename*"

From within the machine language monitor:

9. L"*filename*",8
10. L"*filename*",8,*address*

Finally, there are two ways to start up autoboot programs:

11. BOOT
12. Turn on or reset the computer with an autoboot disk in the drive

a comma and U9, U10, and so forth. You can select the current device number of the 1571 by flipping switches on the back. To change to device 9, for example, make sure it's turned off and flip down the switch nearest the cords. This is much simpler than what's required to modify a 1541—opening it up and cutting a solder trace.

The next command on the list, RUN, has been modified. By itself, it still runs a program, but if you add a program name, the program loads and runs. As with DLOAD and most other disk commands, you can specify a drive number with D or a device number with U after the program name.

In 64 mode, pressing SHIFT–RUN/STOP still loads and runs the first program from tape. But in 128 mode, this combination loads and runs the first program on disk.

BLOADing Binary Files

A binary file is most often a machine language program, although there are several other possibilities: sprite shapes, redefined characters, function-key definitions, hi-res pictures, to name just a few. With binary files it's usually important that they load into a specific area of memory.

If you're familiar with the VIC or 64, you'll recognize LOAD*"filename"*,8,1. It loads a file back into the part of memory from which it was saved.

BLOAD does the same thing, but you don't have to include the 8 and the 1. BLOAD can also send a file to a different section of memory if you append a *B* (Bank number) and a *P* (Position). With an unexpanded 128, the only two choices for the bank are 0 and 1. BASIC programs are stored in bank 0, variables in bank 1. The position can be any memory location in the range 0–65535.

BOOT*"filename"* loads a machine language program and executes a SYS to the starting address. It's the machine language equivalent of RUN*"filename"* for BASIC programs.

You can also load from within the machine language monitor with the L command. After the filename, you must include a comma and an 8 (for device 8, the disk drive). If you wish to relocate the program to a different section of memory, you can include the new address as well.

Autoboot Sectors

When you first turn on a 128, the computer checks to see whether a disk drive is attached and turned on. If so, it tries to read track 1, sector 0, into memory (the 256 bytes of the boot sector are read into locations $0B00–$0BFF). If the letters *CBM* are found at the beginning of that disk sector, the autoboot sequence begins. You can see how this works by following this power-on sequence:

1. Turn on your TV/monitor and disk drive, but not the 128.
2. Insert the CP/M disk that came with the 128 into the 1541/1571.
3. Turn on the 128.

The CP/M disk has an autoboot sector; it's designed to load and run CP/M automatically. An alternative to resetting the computer is to enter BOOT without a filename.

Autoboot sectors aren't limited to CP/M. It's possible to create disks that automatically load and run a BASIC or an ML program. To create such a disk, load and run the AUTOBOOT MAKER program on the disk that comes with the 1571.

The first three bytes of track 1, sector 0 (the ASCII values of the characters *C, B,* and *M*), are followed by the low byte of the load address, the high byte, the bank number for the load, and the number of sequential disk sectors to be loaded. These four bytes aren't important when you're autobooting BASIC programs, so they should usually be zeros. Starting at the eighth byte, you put the disk name (for the BOOTING message) and end with a zero. Next is the name of the program you want to load, again terminated by a zero. Finally, there's machine language which will be called after the load.

Chained Programs

Commodore computers have always had problems with chaining, the process of loading and running one program from within another. The difficulties stemmed from the way variables were stored in memory in previous Commodores: The beginning of variable storage immediately followed the end of the BASIC program.

Chaining is a snap on the 128. Since the program is kept separate from variables, you don't need to worry about program length. To load and run another program, just follow these rules:

1. If you want to keep the variables from the first program, use DLOAD. The second program loads and runs. All variable values are retained.
2. If you want to clear the variables, use RUN"*filename*", where *filename* is the name of the second program.
3. To load a binary file, use either BLOAD"*filename*" or BOOT"*filename*".

A Shortcut

There's a quick and convenient way to DLOAD or RUN a program if you save it a certain way. Include this line at the beginning of the program you're working on:

```
1 REM           DSAVE "01PROGRAM-
 NAME {SHIFT-SPACE}:
```

The {SHIFT–SPACE} means to hold down SHIFT and press the space bar. Play with the spacing of the line so that pressing TAB once puts the cursor in front of DSAVE and pressing it twice lands the cursor on the 1 in front of the program name. When you want to do a safety save of an incomplete program, LIST 1 and TAB twice. Change version number 01 to 02, and press RETURN. Now cursor up to the beginning of the line and TAB once. Tap the ESC key (next to TAB) and then press P. This will erase everything from the cursor to the beginning of the line. (ESC-Q will erase everything to the end of a line—you can remember these two ESC commands if you "mind your p's and q's." Press RETURN, and your program is saved to disk with the new version number.

Later, when you come back to work on the program, press F3 to see the directory (if it goes by too fast, the Commodore key slows it down; the NO SCROLL key temporarily pauses it). When you see the latest version, press RUN/STOP. Cursor up to the program name and type DLOAD or RUN. Better yet, press F2 (DLOAD) or F6 (RUN). The SHIFT–SPACE in line 1 puts a quotation mark between the program name and the colon. Without the colon, DLOAD or RUN will interpret PRG as part of the command.

Another advantage to including the DSAVE on line 1 is that, when you send a program listing to your printer, the version number is right there at the top of the page.

Saving

Here are a few ways to save programs:

1. SAVE"*filename*",8
2. DSAVE"*filename*"
3. BSAVE"*filename*", B*bank*, P*start* TO P*end*
4. From the ML monitor: S"*filename*",8, *start*, *end*+1

The first two, SAVE and DSAVE, are just ordinary ways to save ordinary BASIC programs. BSAVE and the monitor save are a little more interesting. They save a section of memory as a binary file. Note that when you're in the monitor, you have to add 1 to the ending address of the memory being saved.

You might think these two methods would be most useful for saving ML programs. They are good for that, of course, but there are also several areas of memory you may want to BSAVE for use in a BASIC program:

$0E00–$0FFF	**Sprite definitions**
$1000–$10FF	**Ten function-key definitions**
$1C00–$3FFF	**Hi-res screen**

The addresses are listed in hexadecimal. To convert to decimal, use the DEC function, PRINT DEC("0E00"), for example.

If you create several sprites with SPRDEF for a game, you can BSAVE the sprite area to disk. In the game you would then BLOAD them back into memory. This works a lot faster than POKEing them into memory or reading a sequential file, especially if you're using a 1571.

In case you're wondering about the reference above to ten function keys, yes, there are ten redefinable keys. There are the eight you can define with the KEY statement (labeled F1–F8), but also SHIFT–RUN/STOP and HELP. If you go into the monitor and do a memory display of 1000–10FF, you can see the ten key definitions. The first ten bytes in this area list the length of each function key. The rest are the actual characters that print when you press one of them. The number 13 is ASCII for a carriage return, the equivalent of pressing the RETURN key. After redefining the keys, you can BSAVE their new values. To retrieve the previous key definitions, use BLOAD.

Handling Sequential Files

DOPEN and DCLOSE are new ways of establishing and breaking connections with a sequential file. There's not much to say about them; if you already know how to OPEN and CLOSE sequential files, you'll catch on quickly. The difference in syntax is illustrated below:

OPEN 3,8,4,"*filename*,S,W"
DOPEN#3,"*filename*",W

Note that DOPEN doesn't need as much information as OPEN. OPEN is a general-purpose command. It can set up a logical file to a disk file, a tape file, a printer, a modem, and so on. DOPEN, on the other hand, is for disk files only. So OPEN needs the device number and disk channel (,8,4), but DOPEN doesn't. The *S* after the filename indicates that a sequential file will be opened. Since DOPEN defaults to sequential files, it too is unnecessary. Also, note that the *W* (Write) is outside the quotation marks in the second example.

APPEND is a variation on DOPEN. It opens an already existing disk file for a write operation. Any information written to the sequential file is added to the end. Data at the beginning of the file is safe and unchanged.

There aren't any new ways of reading or writing files. You still PRINT# to send data and either INPUT# or GET# to read a file.

Relative Files Are Much Easier

Being able to randomly access records in a file can sometimes greatly speed up a program. With sequential files you may sometimes have to read through 50 records just to get to the fifty-first. A relative file allows you to obtain the information you need almost immediately.

In BASIC 2.0, creating and maintaining a relative file requires sending a number of CHR$ codes. If you write programs for relative files in 64 mode, you'll have to learn the complexities of relative files. But not on the 128. In just a few lines, you can open and write to a relative file. Let's say you want 100 records with 20 characters in each record. Your program to set up a file will look something like this:

```
10 DOPEN#3,"XYZFILE",L20
20 RECORD#3,100
30 PRINT#3,"LAST RECORD"
40 RECORD#3,100
50 CLOSE3
```

That's all there is to it. When DOPEN is followed by an *L* and a number, it opens a relative file. The length of each record is set by L20. Records can be from 1 to 254 bytes long. Because the record length is stored in the directory, you need to use the L parameter only when the file is first created.

RECORD# positions the pointer to the desired record (up to 65535 can be accessed, depending on the record length). You must include the logical file number and the record number. A third number can be added if you want to start reading or writing partway into the record. If this number is omitted, you'll begin at the first byte of the record.

In line 30, we PRINT# to record number 100. Printing to a previously nonexistent record forces the disk drive to create that record and all previous records. Line 40 positions the pointer again, to avoid a rare bug that sometimes corrupts files. And then file 3 is closed. (See "Relative Files: Speed and Economy" in the June 1985 issue of *COMPUTE!'s Gazette* for more about the 64's relative-file bug.)

Once the file is created, you can easily access records with DOPEN and RECORD#. You PRINT# to write and either GET# or INPUT# to read records.

Utilities

There are more new commands that help when you're programming. The F3 key is defined to print DIRECTORY. So, with the press of a single key, you can see what's on a disk.

Two very useful reserved variables are DS (short for Disk Status) and DS$. DS returns the disk error number, while DS$ prints out the error message. If the red light on a 1541 starts flashing (the green light on a 1571), just enter PRINT DS$ and you can see what went wrong. Consult your disk drive manual for a complete list of error messages.

Within a program, DS is usually more helpful than DS$. After a disk operation, add a line

```
IF DS>19 THEN 500
```

where line 500 is the beginning of an error-handling routine.

The variable DS will normally hold a zero if no errors occur. But if DS is equal to 20 or more, something has gone wrong. There are a few exceptions, though: Error 01 is not an error; it's triggered after a SCRATCH operation. The error message will be FILES SCRATCHED, followed by a comma and the number of files that have been scratched. Error 50, RECORD NOT PRESENT, is no matter for concern if you've just created or expanded a relative file. If you write to a previously nonexistent record in a relative file, it's added to the disk. The record was not present before the operation and thus causes the error 50. Finally, when you first turn on or reset a disk drive, you'll receive an error 73, which is simply an announcement of which version of DOS is inside the drive.

Several other new commands make file management easier. RENAME and SCRATCH are fairly straightforward. SCRATCH is followed by a filename inside quotation marks. Pattern matching—using wildcards like question marks or asterisks—is available for those times when you want to scratch several files with similar names. To change the name of a file, RENAME *"oldname"* TO *"newname"*. This syntax is certainly easier to remember than

OPEN 15,8,15, "R0:*newname=oldname*"

which is the required syntax on the VIC or 64.

COLLECT validates the disk. It's used mostly for cleaning up the block allocation map (BAM) to get rid of improperly closed files. These were formerly called poison files, but the 1571 disk drive manual refers to them as splat files. They're marked by an asterisk in the directory (*PRG, for example).

DCLEAR initializes the disk; it's the same as sending "I0" to channel 15.

CONCAT combines the contents of two sequential files. You can use it on program files, but the result won't be a merged program because the two zeros that mark the end of a BASIC program get in the way.

Two disk commands designed primarily for dual drives are COPY and BACKUP. The first copies a file from one drive to another. But you must use a dual drive—COPY won't work with two single drives. It can also make a copy to the original disk (if you want to rearrange a directory, for example). BACKUP copies a whole disk. It too requires a dual drive.

A Few Quirks

The 128 has a few annoying features—not bugs, just bothersome quirks. The most serious of these is that SHIFT–RUN/STOP loads and runs the first file on disk. A nice feature if that's what you want, but sooner or later, while programming, you'll accidentally press SHIFT-LOCK and RUN/STOP or the Commodore key and RUN/STOP. When the disk drive starts spinning, you have only a few seconds to unlock the SHIFT-LOCK key and press RUN/STOP to prevent the first program from loading. If you fail to stop it, the program loads and runs, and you've lost any part of your other program that was not saved. To avoid this situation, you may want to put a sequential file at the beginning of a disk. If you accidentally type SHIFT–RUN/STOP, the computer will try to load the sequential file, but it won't work. The program you're working on will be safe if this precaution is taken.

You can also accidentally save a program. The default values for F7 and F3 are LIST and DIRECTORY—very helpful when you want to take a look at what's on a disk or what's in a program. But in between these two keys is F5, which is defined as DSAVE. If you reach up to list a program and accidentally press F5 and F7, the computer will print DSAVE"LIST and begin saving your program under that name.

VERIFY and DVERIFY don't always work as you would expect. Each line of a BASIC program contains a memory pointer to the beginning of the next line. When you allocate a graphics area, the BASIC program is moved up by 9K, and all the line links change. Line links that don't match will lead to a false VERIFY ERROR. You can test this by entering a one-line program and saving it to disk. Now type

```
GRAPHIC1: GRAPHIC0
```

to allocate a graphics area. List the program and use DVERIFY to check your save. You should see an error message.

Something to remember when you're using disk commands is that variables must be enclosed in parentheses. The following two examples show the right and wrong ways to use variables:

```
RENAME H$ TO "FINALFINAL": REM
  WRONG
RENAME (H$) TO "MOSTRECENT":
  REM CORRECT
```

Another minor annoyance is that large relative files have a limit of 720 disk sectors. Relative files contain up to six side sectors, which can keep track of 120 sectors each. A formatted 1541 disk has 664 blocks free, so you'll run out of disk space before you reach the 720-sector limit. But the 1571 formats both sides of a disk, for 1328 blocks free. It's unfortunate that you can't use all of that space for a relative file. You're limited to about 180K per file. The 720-sector limit gets in the way.

How Fast Is the 1571?

If you already own a 1541 drive, you can use it with a 128. You don't need to buy a 1571, unless speed is important. Here's how the two drives compare:

	1541	**1571**
9K LOAD (hi-res screen)	27 seconds	4 seconds
Disk format	89 seconds (one side)	43 seconds (two sides)
Quick format (no ID)	2.5 seconds (one side)	3.4 seconds (two sides)

Going into 80 columns and using the FAST command to double the speed of the microprocessor saves only a few tenths of a second on disk operations. So the speed of the computer is not a factor. The bottleneck is the speed at which the data travels through the serial cable.

Note that formatting, which is handled completely within the disk drive, is twice as fast for twice the disk capacity. This suggests that writing operations are quicker on the 1571.

Even when you send the command to make the 1571 act like a 1541, it's faster. A 1541 takes nearly a minute and a half to format a disk. The 1571 in 1541 mode takes only one minute and 12 seconds.

The "act like a 1541" command is

```
OPEN 15,8,15: PRINT#15, "U0>M0"
```

To reset to 1571 mode, use

```
PRINT#15, "U0>M1"
```

(Both of these commands can be used in 64 mode as well.)

While the 1571 is emulating the 1541, you can choose which read/write head is used with PRINT#15, "U0>H0" or PRINT#15, "U0>H1". By switching heads, you can format both sides of a disk as if they were separate disks. This isn't an especially useful feature, but it suggests a solution for the 720-sector limit on relative files. You could format both sides of the disk with separate names and IDs and then create a relative file on each side. Within the program, you'd need to figure out which side of the disk contains the information and switch back and forth. You might lose some time in the head switching, but you'd be able to expand a relative file to about 330K.

Another plus for the 1571 is its ability to read a variety of CP/M formats. If you plan to do much with CP/M, the 1571 provides more flexibility. Even if you don't, it's faster and can handle more data than the 1541.

Commodore 128 Hi-Res Text Manipulation

James R. Schwartz

Drawing high-resolution screens on the 128 is reasonably easy because of BASIC 7.0's powerful graphics statements. This tutorial explores several interesting techniques for adding text to your hi-res artwork.

If you load the same high-resolution picture into a Commodore 64 and a 128, you'll find that the two screens look exactly the same. This shouldn't be surprising; the two computers use the same video chip. But the 128 (in 128 mode) has many drawing statements that aren't available on the 64. The 128 can DRAW, CIRCLE, BOX, PAINT, and so on, whereas the 64 is limited to PEEKs and POKEs.

These graphics statements give you enormous power over the hi-res screen. Exactly 64,000 picture elements (pixels) fit on the screen, arranged in a grid that's 320 pixels wide and 200 deep. In the hi-res mode you have complete control over each pixel.

You can use this power to manipulate text as well as graphic images. The CHAR statement, for example, displays text on the screen. In hi-res mode, you can use CHAR to add labels to your art.

But CHAR is limited to even eight-bit boundaries. You can use CHAR to place a character at position 0, 8, 16, 24, and so on, but not on the pixels in between. This means you can choose from 40 positions across and 25 down, the same dimensions found on the regular text screen.

Better Than CHAR

Using the fairly simple techniques mentioned here, you'll be able to fine-tune the placement of text on the screen. You can even create your own special characters. And you'll be able to write vertically on the screen, a useful feature for charts and graphs.

The first step is to create two arrays, H$ and V$, for the horizontal and vertical character sets, respectively. Type in this short program which draws the two character sets on the screen. (Use "The Automatic Proofreader," found in Appendix B, to type in these programs.)

```
HR 100 GRAPHIC2,1,4:FAST:FORI=35TO93:X=I-35-ABS(40
       *(I>74)):Y=ABS(I>74):CHAR1,X,Y,CHR$(I)
SG 110 FORH=0TO7:FORV=0TO7:LOCATEH+X*8,V+Y*8:IFRDO
       T(2)=1THENDRAW1,V+X*8,16-H+Y*8+8
FX 120 NEXT:NEXT:NEXT:SLOW:WINDOW0,4,39,24,1
```

Even in FAST mode, it will still take quite a while to run—about 45 seconds. Fortunately, however, if you save the result as a binary file on a disk, future applications will take only a few seconds. To do this, enter the following line in direct mode when the READY prompt appears:

**BSAVE"HIRES CHAR",B0,P8192 TO
 P9471**

Now type NEW to erase the original program and type in part 2:

```
SB 100 GRAPHIC1,1
FH 110 FAST:BLOAD"HIRES CHAR",D0,U8,B0,P8192:DIMH$
       (93),V$(93):X=0:Y=0:FORI=35TO93:SSHAPEH$(I)
       ,X,Y,X+7,Y+7:SSHAPEV$(I),X,Y+17,X+7,Y+23:X=
       X+8:IFX=320THENX=0:Y=Y+8
BM 120 NEXT:SLOW
```

Here we're using the SSHAPE statement to save rectangular areas of the screen into string variables. Once you've SSHAPEd a portion of the hi-res screen, you can use GSHAPE to place it anywhere on the screen. These two statements are like creating a rubber stamp which can copy a shape to different portions of the screen. In this case, the areas saved are the size of the individual characters. For ease of operation, the number in the array corresponds to the ASCII value of the character in that array position. For example, H$(65) contains the horizontal *A*; V$(90) contains the vertical Z, and so forth.

Writing on the Hi-Res Screen

We're now ready to manipulate text. To place a horizontal string on the screen, use the following subroutine. You should enter this subroutine with S$ defined as the string to be printed, and X and Y as the *x* and *y* coordinates of the upper left-hand corner of the first character in that string. The *x* coordinate can be 0–319; the *y* coordinate can be 0–199.

```
SQ 1000 FORH=1TOLEN(S$):IFASC(MID$(S$,H,1))=32THEN
        1010:ELSEGSHAPEH$(ASC(MID$(S$,H,1))),X,Y-8
JX 1010 X=X+8:NEXT:RETURN
```

To *perfectly* center a line on the screen, add the following subroutine. This time, you have to supply only S$ and Y, since X is calculated for you.

```
QK 2000 X=159-LEN(S$)*4:GOTO1000
```

Here's an example showing the advantages of hi-res text manipulation. Add these lines to the program you have so far:

```
SE 500 GRAPHIC1,1:DRAW1,119,39TO199,39TO199,71TO11
       9,71TO119,39
GS 510 DRAW1,119,87TO199,87TO199,119TO119,119TO119
       ,87
XE 520 A$="SAMPLE":B$="CENTERING"
PQ 530 CHAR1,20-LEN(A$)/2,6,A$:CHAR1,20-LEN(B$)/2,
       7,B$
GH 540 S$=A$:Y=103:GOSUB2000:S$=B$:Y=111:GOSUB2000
       :SLEEP10:GRAPHIC0:END
```

When you type RUN, you'll see the difference between the standard CHAR statement (top box) and hi-res text manipulation (bottom box).

Even more fascinating is the ability to print vertically on the screen. Add this subroutine:

```
BR 3000 FORV=1TOLEN(S$):IFASC(MID$(S$,V,1))=32THEN
        3010:ELSEGSHAPEV$(ASC(MID$(S$,V,1))),X-8,Y
        -8
CS 3010 Y=Y-8:NEXT:RETURN
```

In direct mode, type DELETE 500–540; then enter these lines for a new demonstration:

```
FX 500 GRAPHIC0:INPUT"{CLR}STRING TO BE PRINTED";S
       $:INPUT"X COORDINATE, Y COORDINATE";X,Y
FS 510 IFX<7ORX>319ORY<LEN(S$)*8ORY>199THENPRINT"O
       UT OF RANGE":SLEEP2:GOTO500
CD 520 GRAPHIC1,1:GOSUB3000:SLEEP10:GRAPHIC0:END
```

Run the program. You'll be prompted to enter a string and the appropriate coordinates. Note that line 510 makes sure that the *x* and *y* coordinates you enter will allow the string to fit on the screen; if not, you're returned to the prompt.

Custom Characters in Hi-Res

Now you can create your own special characters. Suppose, for example, that you need to create a cent sign (¢). The first step is to find a character you will not need and replace it with your new character. For this example, we will replace the dollar sign, CHR$(36), with the cent sign.

Draw an 8 × 8 grid (Figure 1). Label the vertical rows as shown and draw your new character in the grid. Now, add up the binary values for each horizontal row (Figure 2). Add these numbers as DATA statements in your program, with the following lines:

```
AA 200 FAST:GRAPHIC1,1:RESTORE230:FORI=8192TO8199:
       READAD:POKEI,AD:NEXT
BS 210 FORH=0TO7:FORV=0TO7:LOCATEH,V:IFRDOT(2)=1TH
       ENDRAW1,V+10,ABS(H-7)
DF 220 NEXT:NEXT:SSHAPEH$(36),0,0,7,7:SSHAPEV$(36)
       ,10,0,18,7:SLOW
BJ 230 DATA16,16,60,102,96,102,60,16
```

The program contains the DATA statements that make the cent sign. Also, note that the 36 in line 220 refers to the replacement of CHR$(36), the dollar sign that's replacing the cent sign.

Figure 1. Grid

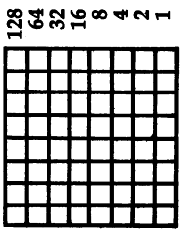
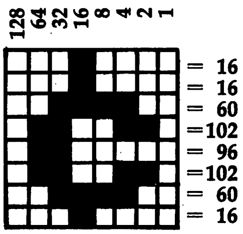


Figure 2. Grid with Binary Values



In order to output a cent sign to the hi-res screen, simply enter any of the subroutines with a dollar sign in the place where you want a cent sign to appear. Delete lines 500–520, and enter the following for a new demonstration:

```
JF 500 GRAPHIC1,1:S$="99$":X=151:Y=100:GOSUB1000:S
       LEEP10:GRAPHIC0:END
```

Type RUN to see the results.

You now have the basic tools to control text on the hi-res screen exactly as you see fit. You're no longer bound by the restraints of the CHAR command.

ESCaping with the 128

Jim Vaughan

The ESCape key makes the Commodore 128 an even more powerful machine. With it, you can access 27 new screen-editing features. This article discusses each of the ESC-key sequences and tells how to use them in your own programs. Two helpful demonstration programs are also included.

The Commodore 128 offers a variety of editing and screen control features which are accessed by using the ESC (ESCape) key, located on the far left of the upper row of gray keys. These features—new to Commodore machines—make Commodore's popular full-screen editing even better.

Each of the 27 screen-editing features is accessed by first pressing the ESC key, then releasing it and pressing another key—either a letter from *A* through Z or the @ *(at)* sign. Unlike when you're using control characters such as CONTROL-9 ({RVS}), where you hold down one key (CONTROL, Commodore, or SHIFT) while pressing another, you *should not* hold down the ESC key. Press the ESC key once, and then press the other key. The two keystrokes together form an *escape sequence.*

You can also use the escape sequence functions within programs by printing CHR$(27)—the character code for ESC—followed by the appropriate letter for the sequence you wish to use. It's purely coincidental, but there are 27 new editing sequences for ESC, which is CHR$(27). The sequences function identically with shifted and unshifted letters (ESC SHIFT-A does the same thing as ESC A), so the same techniques work if SHIFT LOCK or CAPS LOCK is pressed. The exception is ESC @, since ESC SHIFT-@ does nothing.

The new set of commands can be broken down into three categories: editing enhancements, screen control enhancements, and miscellaneous sequences.

Editing Enhancements

The new editing commands allow quicker movement around the screen and easier entry or deletion of part or all of a program line. These sequences are summarized in Table 1.

Table 1. Editing Enhancements

ESC A	Enable auto-insert mode
ESC C	Cancel auto-insert mode
ESC D	Delete the current logical line
ESC I	Insert a blank line
ESC J	Move to the beginning of the current line
ESC K	Move to the end of the current line
ESC O	Cancel quote mode (ESC ESC also works)
ESC P	Erase from beginning of line to cursor
ESC Q	Erase from cursor to end of line
ESC @	Erase from cursor to end of screen

ESC A, the first sequence, puts you into auto-insert mode. This allows you to enter text or program lines at the current position of the cursor without the rest of the line being overwritten. Auto-insert mode is most useful when you need to go to the middle of a program line and add statements.

ESC C cancels the insert mode. (The Commodore 128 *System Guide* that comes with the computer erroneously states that this sequence cancels quote mode.) Once you're finished inserting text, it's advisable to cancel the mode with ESC C, since printing is noticeably slower when the computer is in auto-insert mode. Note that auto-insert mode is different from the other kind of insert mode that appears when you hold down SHIFT and press the INST/DEL key. With ESC A, you don't have to open up the space first. Furthermore, auto-insert mode doesn't act like quote mode. Control characters such as RVS ON won't print in reverse video as they do in quote mode or insert mode.

ESC D allows you to delete the program line the cursor is currently on. (To delete just part of a line, see ESC P and ESC Q below.) Be warned, however, that if the program line extends over more than one physical screen line, this will delete the entire logical line—not just the single screen row the cursor is on.

ESC I allows you to insert a blank line at the current cursor position. This sequence will move down one row all text that is on or below the line containing the cursor. Anything on the bottom row will be scrolled off the screen, and the cursor will remain on the new blank line. If you insert a line in the middle of a multirow program line, the text on the rows following the new line will still be considered part of the program line. Before you issue this sequence, remember to put the cursor on the screen row where you want the new line to be added.

ESC J and **ESC K** allow quick movement on the program line the cursor is on. ESC J moves the cursor to the beginning of the program line, and ESC K moves the cursor to the position just beyond the last nonspace character in the program line. (Again, these commands work with logical program lines, which may consist of more than one physical screen row.) I find these sequences the most useful when editing, since I'm always adding statements onto the end of program lines. Using ESC K saves me from having to hold down the cursor-right key and wait until the cursor reaches the end of the line.

ESC O cancels quote mode. Probably every Commodore programmer has tried editing a portion of a program line between quotation marks and ended up with a collection of reversed characters. The operating system treats editing characters like {DOWN} and {RVS} differently when they are typed within quotation marks. Instead of acting immediately, they appear as reverse characters. This is fine if you're trying to create a string to print cursor-right characters, but annoying if you want to use the cursor-right key to move to the end of the string.

By using ESC O, you can cancel quote mode to use the normal editing keys without getting the reverse characters. Insert mode (which is almost identical to quote mode) is in effect when you insert spaces with the INST/DEL key. For example, if you insert five spaces, those five character positions behave as if they were within quotation marks. ESC O cancels this effect as well. It's not documented in the manual, but pressing ESC twice in a row (ESC ESC) is the same as ESC O, a handy shortcut. (ESC O is erroneously defined in the *System Guide* as canceling auto-insert mode. The manual has reversed ESC C and ESC O functions.)

ESC P will erase everything from the beginning of a program line to the position of the cursor. This means that if you put the cursor in the middle of a program line and then press ESC P, the first half of your line (including the line number) will be erased.

ESC Q is the complement of ESC P, meaning that it will erase everything from the cursor to the end of the line. Erased positions are filled with spaces; the cursor and remaining text do not move. As before, it's important to remember that these sequences affect logical lines, not just screen rows. If you press ESC P when the cursor is in the middle of the third row of a program line that spans four screen rows, two and a half rows will be erased—not just the row on which the cursor resides.

ESC @ is an enhanced version of ESC Q. Instead of erasing to the end of the current line, it erases everything from the cursor position to the bottom right corner of the current output window. This can be useful when you're loading a program from the directory. Press F3 to list the disk directory, move the cursor to the program you want to load, and type DLOAD (or RUN) to the left of the filename. Now press TAB two or three times, until the cursor is past the name of the program. Finally, type ESC @ to clear the bottom portion of the screen, and press RETURN to load the program.

Screen Enhancements

The following group of sequences manipulate the entire screen area as opposed to just editing lines. These sequences are summarized in Table 2. It's not really proper to refer to the screen when you're talking about the 128's display, since all screen operations actually depend on the height and width of the currently defined output window. It's easy to forget this; the output window is most often set to cover the entire screen, but that doesn't have to be the case. The 128's windowing capabilities aren't as powerful as those of a Macintosh or Amiga (you have only one window to work with), but they do allow you great flexibility in designing screen displays.

ESC B and **ESC T** can be used to change the boundaries of the output window. ESC T defines the current cursor position as the top left corner of the window—the home position of the output display, where the cursor will go when the CLR/HOME key is pressed. ESC B defines the current cursor

Table 2. Screen Control Enhancements

ESC B	Set bottom right corner of output window
ESC L	Allow scrolling
ESC M	Disable scrolling
ESC T	Set top left corner of output window
ESC V	Scroll screen up one line
ESC W	Scroll screen down one line
ESC X	Switch between 40- and 80-column modes

position as the bottom right corner of the window. Just move the cursor to the desired position and use the sequence to set the new boundary. You probably won't use these escape sequences in a program (although you could) because the WINDOW command is more convenient to use.

There are many creative ways to employ windows when you're programming. For example, you may decide you want to look at a disk directory: Just create a window and press F3. Whatever is outside the window will remain unchanged. You could also compare two sections of a program by listing the first part, setting up a window, and listing the second part. (See "Windows on the 128," *COMPUTE!'s Gazette,* April 1986, for more about windows.)

The window isn't cleared when it is resized, but all lines are unlinked (the text in each row is "disconnected" from that in any previous or following rows). The sequences are easy to remember: ESC T for Top and ESC B for Bottom. Pressing CLR/HOME twice in a row (HOME HOME) will reset the window to full screen size.

ESC M and **ESC L** control how text is handled when there's no more room at the bottom of the window. Normally, all lines of text in the window scroll upward to add the text to a new line at the bottom of the window (the previous top line is scrolled off the screen). ESC M turns off the scrolling. When scrolling is disabled and the screen is full, the next PRINT statement will cause the text to be displayed at the top of the screen, *unless the PRINT statement is followed by a semicolon.* When you LIST a program or directory which is longer than the current height of the window, printing automatically jumps back to the top of the window after a line is displayed at the bottom. However, things get a bit strange when the

PRINT statement ends with a semicolon. Press ESC M; then try the following line:

```
FOR I=1 TO 5000: PRINT
  CHR$(INT(RND(1)*64+32));: NEXT
```

When the bottom line of the window fills up, the printing will continue in the position at the lower right corner of the screen, with each successive PRINT overwriting the previous character in that position. This mode is useful when you're trying to display some graphics effects that you don't want to scroll off the screen. However, you must plan your PRINT statements carefully when scrolling is disabled. Use ESC L to turn the normal scrolling feature back on.

ESC V and **ESC W** move all lines of text in the window up (ESC V) or down (ESC W) one row, adding a new blank line at the bottom or top of the screen. Any text that is scrolled off the top or bottom of the window will be lost—it's not recovered when the window is scrolled back in the opposite direction.

Scrolling down produces an interesting effect which could be exploited in an auto-racing game. You'd put the cursor on the top line, print the two edges of the road, scroll it down, and repeat. It would be relatively easy to add a sprite for the automobile, a joystick movement routine, and a routine that subtracts points when the car hits the side of the road (using the COLLISION statement).

The 128 provides both 40- and 80-column displays, but only one can be active at any given time. That is, only one display can have a "live" cursor, and all printing will be sent to that display.

ESC X allows you to switch between the two display modes. This sequence is a toggle: Whenever you use ESC X, the display that is currently active becomes inactive, and the one that was previously inactive becomes the active display. This sequence doesn't actually turn either video chip on or off; whatever is currently displayed on the screen that becomes inactive will remain intact until the screen becomes active again (or until the RESET button or RUN/STOP–RESTORE combination is pressed). If you are using the same monitor for both displays, you still must switch the monitor to the desired display. The 40/80 DISPLAY key has no effect on ESC X (other than to determine which display will be active after RESET or RUN/STOP–RESTORE).

Miscellaneous Sequences

This last set of sequences is called the miscellaneous group simply because none of them falls into either of the other groups. These sequences are summarized in Table 3.

Table 3. Miscellaneous Sequences

ESC E	Set cursor to nonblinking mode
ESC F	Set cursor to flashing mode
ESC G	Turn on (enable) bell tone
ESC H	Turn off (disable) bell tone
ESC N	Return to normal 80-column display
ESC R	Reverse 80-column character and background colors
ESC S	Change 80-column cursor to block
ESC U	Change 80-column cursor to underline
ESC Y	Restore default tab stops
ESC Z	Clear all tab stops

The 128 allows you to customize the cursor to your own tastes. Some people find the blinking cursor irritating, so ESC E stops cursor flashing. The cursor remains visible—it's now a nonblinking block. ESC F will make it flash again. Those who use the 80-column screen have an additional choice. ESC U changes the 80-column display's cursor from a character-sized block into a character-width underline (the 40-column cursor is unaffected). ESC E and ESC F, respectively, can still be used to make the underline cursor nonblinking or flashing. ESC S changes the cursor back to a block.

The 128 has a feature that is common on most other computers and terminals, but which had been missing on the VIC-20 and Commodore 64: a BELL character. Printing CHR$(7)—or holding down the CONTROL key and pressing G—causes a bell-like tone if your monitor is equipped with a speaker. This is a handy way to generate an attention-getting sound, but some programs may make excessive use of the feature. Maybe you just find the noise annoying. Whatever your reason for using it, ESC H disables the tone. Use ESC G to turn it back on.

The background color of the display is normally determined by the value in a video chip register, and the color of each character is normally determined independently by the value in a color memory location corresponding to the character's position on the display. On the 80-column display, this can be reversed. After you issue the ESC R sequence, all characters take what was previously the background color, and the background for each character is determined by the color value in the corresponding color memory position. (The 40-column display is unaffected.) Use ESC N to return the display to normal. ESC R is useful when you have the 80-column output displayed on a monochrome monitor and want dark characters on a light background.

The two final sequences deal with tab-stop settings. Whenever you press the TAB key or print the tab character—CHR$(9)—the cursor moves to the next column for which a tab stop has been set or, if there are no more tab stops, to the right boundary of the output window. This is useful for aligning columns of text and numbers. When the 128 is turned on or reset, it establishes a tab stop every eight spaces.

ESC Z erases all tab stops so the cursor always goes to the right margin of the window when you press TAB. You can change the tab-stop setting of the column where the cursor currently resides by pressing SHIFT-TAB (or CONTROL–TAB). This is a toggle: If no tab stop has been previously set at the column, one will be set there when you press SHIFT-TAB or CONTROL–TAB. However, if a tab stop has already been set for the column, it will be cleared.

You can get the same effect within a program by printing CHR$(24). ESC Y resets the default tab stops (one every eight spaces). Changing tab settings, including using ESC Y and ESC Z, affects only the active display. Each display has its own tab-stop table, so it's perfectly possible to have default tab settings for the 80-column display while having tab stops every four spaces, or no tab stops at all, for the 40-column display.

Using ESC Sequences from Within a Program

As noted earlier, even though these new ESC sequences are useful for program editing and for onscreen fun, they can also be used within a BASIC or machine language program to achieve the same results. Some of the sequences may not have an obvious effect while the program is running. For example, since the cursor is turned off when a program is running, changing the cursor mode won't have any immediately visible effect except in INPUT statements. Program 1 is a short demonstration of how a few of these ESC codes can be used in a BASIC program. Type in the program carefully, since it contains a lot of cursor control codes inside quotation marks. (Using "The Automatic Proofreader," Appendix B, will help you type it in correctly.)

To use these sequences in a machine language program, print the ESC code ($1B) using the BSOUT routine ($FFD2); then print the ASCII code for the desired function. For example, the following code will scroll the display up one line:

```
LDA #$1B
JSR  $FFD2
LDA #$56
JSR  $FFD2
```

A handy way to use these sequences is to include them in programmable function-key definitions. You can use the line

```
KEY 1, CHR$(27)+"I"
```

and a blank line of text will be inserted whenever you press F1.

If you want to have the editing definitions loaded automatically whenever the computer is turned on or reset, you can write a key definition and make it an autobooting disk file. Program 2 was written for just that purpose. It assigns an editing function to each of the eight function keys and prints a menu at the bottom of the display for easy reference. The output window is then reduced by two lines so that the menu won't be overwritten. To make the program autobooting, use the "Autoboot Maker" program on the 1571 Test/Demo disk that came with your drive, or use a program like "128 Autoboot" (*COMPUTE!'s Gazette,* March 1986).

The key definitions included in Program 2 may not be the ones that you want to use most often, but you can easily change any of the definitions to ones that suit you. The definitions currently included in Program 2 are as follows:

F1	**Enable auto-insert mode**
F2	**Cancel auto-insert mode**
F3	**Insert a blank line**
F4	**Delete a program line**
F5	**Move cursor to start of line**
F6	**Move cursor to end of line**
F7	**Delete to the start of line**
F8	**Delete to the end of line**

Once you start to learn the ESC sequences, you'll find yourself using them more and more frequently.

Program 1. ESC Sequence Demo

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
CA 1Ø REM ****************************************
       *********************
FF 2Ø REM *{59 SPACES}*
PD 3Ø REM *{22 SPACES}PROGRAM 1{28 SPACES}*
XG 4Ø REM *{59 SPACES}*
SQ 5Ø REM *{2 SPACES}DEMO PROGRAM FOR USING ESCAPE
        CODES IN A BASIC PROGRAM{3 SPACES}*
KH 6Ø REM *{59 SPACES}*
QR 7Ø REM ****************************************
       *********************
PF 8Ø REM * CHECK TO SEE IF WE ARE IN 8Ø COLUMNS *
RH 9Ø REM ****************************************
GF 1ØØ PRINTCHR$(14);
PA 11Ø MD=RGR(X):IF MD<>5 THEN PRINT"{CLR}{DOWN}
        {RVS}{2 SPACES}PLEASE USE 8Ø COLUMNS FOR TH
        IS DEMO.{2 SPACES}{OFF}":END
AF 12Ø REM ********************************
GP 13Ø REM * PRINT HEADER AND MAKE WINDOW *
JD 14Ø REM ********************************
EJ 15Ø PRINT"{HOME}{RVS}{13 SPACES}USING ESCAPE CO
        DES IN A BASIC PROGRAM - EXAMPLES
        {34 SPACES}":WINDOWØ,1,79,24,1
XF 16Ø PRINT"{CLR}";
MF 17Ø REM *******************************
GS 18Ø REM * ESC D - DELETE LINE EXAMPLE *
CD 19Ø REM *******************************
FD 2ØØ PRINT"{3 DOWN}{3 RIGHT}THIS LINE WON'T BE E
        RASED..."
KD 21Ø PRINT"{3 RIGHT}THIS LINE WILL BE ERASED USI
        NG ESC D .....{RVS}HIT ANY KEY{OFF}"
QE 22Ø PRINT"{3 RIGHT}THIS LINE WON'T BE ERASED...
        BUT WILL MOVE UP ONE LINE..";
CC 23Ø GETKEYA$:PRINT"{UP}"CHR$(27)"D"
PK 24Ø REM ***************************************
        **
KX 25Ø REM * ESC Q - DELETE TO END-OF-LINE EXAMPLE
         *
DJ 26Ø REM ***************************************
        **
MG 27Ø PRINT"{2 DOWN}{3 RIGHT}THIS LINE WILL BE HA
        LF ERASED USING ESC Q...HIT ANY KEY
        {23 LEFT}";
```

```
MD 280 GETKEYA$:PRINTCHR$(27)"Q"
BJ 290 REM ****************************************
       ****
AQ 300 REM * ESC P - DELETE TO START-OF-LINE EXAMP
       LE *
SK 310 REM ****************************************
       ****
EP 320 PRINT"{3 RIGHT}THIS LINE WILL BE HALF ERASE
       D USING ESC P...HIT ANY KEY{24 LEFT}";
JA 330 GETKEYA$:PRINTCHR$(27)"P"
HX 340 REM *********************************
DA 350 REM * ESC I - INSERT A LINE EXAMPLE *
XA 360 REM *********************************
KM 370 PRINT"{3 DOWN}{RVS}YOU COULD DEFINE THE FUN
       CTION KEYS AS WORD-PROCESSING TOOLS"
MF 380 KEY 1,CHR$(27)+"I" :PRINT"{RVS}WITH: KEY 1,
        CHR$(27)+'I'{OFF}"
XF 390 PRINT"{DOWN}{3 RIGHT}THIS LINE WILL STAY HE
       RE..."
RR 400 PRINT"{3 RIGHT}THIS LINE WILL MOVE DOWN ONE
        USING ESC I...{RVS} HIT F1 {OFF}";
XD 410 GETKEYA$:GETKEYB$:C$=A$+B$:PRINTC$;
CX 420 PRINT"{RVS}...AND I WILL INSERT THIS LINE..
       .":PRINT"{2 DOWN}";
KF 430 REM ******************************************
       **
AH 440 REM * ESC A - AUTOMATIC INSERT MODE EXAMPLE
        *
BE 450 REM ******************************************
       **
QK 460 PRINT"{DOWN}WE'LL INSERT TEXT INTO THE MIDD
       LE OF THIS LINE USING ESC A ...HIT ANY KEY.
       .";
DX 470 GETKEY A$
RA 480 PRINT"{22 LEFT}";CHR$(27)"A";"{RVS}THIS IS
       {SPACE}INSERT MODE "
SK 490 REM ******************************************
       *****
HQ 500 REM * REMEMBER TO CANCEL INSERT MODE FOR SP
       EED *
FJ 510 REM ******************************************
       *****
EM 520 PRINTCHR$(27)"C";
XS 530 REM ******************************************
       ***
GS 540 REM * ESC @ - ERASE TO END-OF-SCREEN EXAMPL
       E *
HJ 550 REM ******************************************
       ***
```

```
PH 560 PRINT"{2 DOWN}NOW WE'LL USE ESC @ TO ERASE
       {SPACE}HALF OF THE SCREEN...HIT ANY KEY";
FF 570 GETKEYA$
BQ 580 PRINT"{HOME}{14 DOWN}";CHR$(27)"@";
BP 590 REM ****************************************
       ****
XC 600 REM * ESC T & ESC B - SET UP A WINDOW EXAMP
       LE *
QE 610 REM ****************************************
       ****
QE 620 PRINTCHR$(27)"T";"{30 RIGHT}{15 DOWN}";CHR$
       (27)"B"
XM 630 PRINT"{CLR}THIS IS A WINDOW":PRINT"USING ES
       C T AND ESC B":SLEEP 3:CATALOG
HF 640 PRINT"{DOWN}YOU CLEAR THE WINDOW":PRINT"BY
       {SPACE}PRINTING 'HOME'":PRINT"TWICE."
FB 650 PRINT"{DOWN}HIT ANY KEY TO END":GETKEYA$
KD 660 PRINT"{2 HOME}";:PRINT"{CLR}{3 DOWN}
       {5 RIGHT}{RVS}ALL DONE !!"
QA 670 END
RA 680 PRINT"HIT ANY KEY..":GETKEYA$:RETURN
```

Program 2. ESC Sequence Function Keys

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
HS 10 REM ****************************************
      ******
FF 20 REM *{44 SPACES}*
JC 30 REM *{10 SPACES}PROGRAM 2 - FUNCTION KEYS
      {9 SPACES}*
XG 40 REM *{44 SPACES}*
DP 50 REM ****************************************
      ******
KF 60 REM
XH 70 REM DEMO 2 - FUNCTION KEY EDITING ASSIGNMENT
      S
HH 80 REM
MD 90 REM DEFINE KEYS:
BX 100 REM
MX 110 KEY 1, CHR$(27)+"A" : REM INSERT MODE
JD 120 KEY 2, CHR$(27)+"C" : REM CANCEL INSERT MOD
       E
ED 130 KEY 3, CHR$(27)+"I" : REM INSERT A LINE
JD 140 KEY 4, CHR$(27)+"D" :REM DELETE A LINE
PD 150 KEY 5, CHR$(27)+"J" :REM MOVE TO START OF L
       INE
```

```
BQ 160 KEY 6, CHR$(27)+"K" :REM MOVE TO END OF LIN
       E
RX 170 KEY 7, CHR$(27)+"P" :REM ERASE TO START OF
       {SPACE}LINE
GF 180 KEY 8, CHR$(27)+"Q" :REM ERASE TO END OF LI
       NE
GG 190 REM
AM 200 REM{2 SPACES}SET UP MENU AT BOTTOM OF SCREE
       N
DG 210 REM
XQ 220 PRINT"{HOME}{23 DOWN}";CHR$(27)"M";
MR 230 PRINT"{RVS}{3 SPACES}F1-INSERT{5 SPACES}F3-
       INSERT LINE{5 SPACES}F5-MOVE TO START OF LI
       NE{3 SPACES}F7-ERASE TO START"
JX 240 PRINT"{RVS}{3 SPACES}F2-NO INSERT{2 SPACES}
       F4-DELETE LINE{5 SPACES}F6-MOVE TO END OF L
       INE{5 SPACES}F8-ERASE TO END{2 SPACES}"
DK 250 REM
HG 260 REM PROTECT THE MENU WITH A WINDOW
PM 270 REM
QP 280 WINDOW 0,0,79,22,1
AR 290 PRINT"{CLR}";CHR$(27)"L";
```

128 Machine Language

Jim Butterfield

In this article we'll examine some basic architectural features of the 128, including memory banking, and look at a program that passes information between BASIC and machine language. From there we'll explore the built-in machine language monitor and look at ways to call and link ML programs from BASIC.

The Commodore 128 is truly three computers in one—a Commodore 128 when in 128 mode, a Commodore 64 when in 64 mode, and a Z80-based CP/M computer when in CP/M mode. We'll focus here on programming the computer in machine language (ML) in 128 mode.

When in this mode, the 128's 8502 microprocessor can execute the same instructions as the Commodore 64's 6510 microprocessor. Many of the programming techniques used on the 64 work exactly the same on the 128. This article is directed especially at programmers who need to make the transition from 64 machine language to 128 ML programming. Of course, if you're familiar with 6502/6510 programming, but the 128 is your first Commodore computer, you can still benefit from the information presented here.

Ground Rules

Here are two simple ground rules to keep you out of trouble on the 128:

First, it's important to stay in bank 15 when you're writing programs with the computer's built-in machine language monitor (we'll explain what a bank is in a moment). This rule is necessary because of the 128's memory architecture, which can be confusing to a beginner. If you choose a bank number lower than 12, you may end up in a machine configuration which has no read only memory (ROM), making it impossible for your program to call any of the computer's built-in ROM routines.

Second, stay away from areas of random access memory (RAM) which are usually safe on the 64. On the 64, for instance, the cassette buffer located at 828–1019 ($033C–$03FB) is a good place to put short ML programs, and the free RAM block at 49152–53247 ($C000–$CFFF) is ideal for longer programs. Both areas are unusable on the 128, as you'll quickly learn if you try to put ML code there. The lower area contains critical system vectors and subroutines; if you change their contents, the system will crash. The higher area is covered by Kernal ROM; you can't easily put an ML program there and still have access to ROM routines.

Instead, the 128 has safe areas in locations 2816–3071 ($0B00–$0BFF) and 4864–7167 ($1300–$1BFF). The first area is the 128's cassette buffer, and the second area is currently unused by the system.

Why Bank 15?

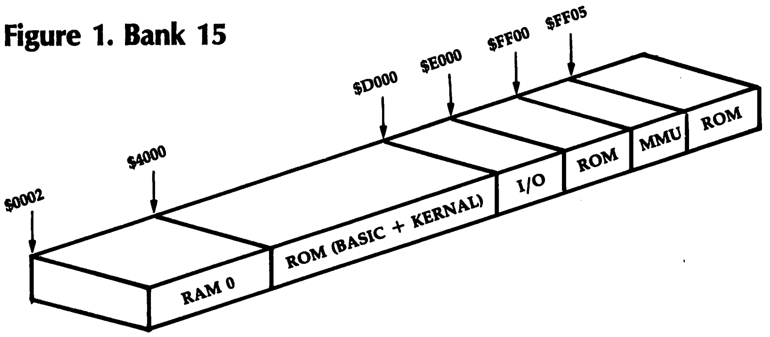
The 128 is capable of seeing its memory as 16 different banks, numbered from 0 through 15. The term *banks* is somewhat misleading, since a bank does not represent a separate 64K block of memory. Instead, each bank represents a different configuration or arrangement of the various available RAM and ROM elements. The bank number determines what the 128 sees within various areas. In some banks, the 128 sees nothing but RAM; in others, it sees a combination of RAM and ROM. Still other configurations include RAM, ROM, input/output (I/O) addresses, and so on.

In fact, there are 256 possible memory configurations. Most of these, however, are of little or no use. For example, though you can configure the computer to see only half of its BASIC ROM and none of its Kernal ROM, it's hard to imagine any use for such an arrangement. Commodore has chosen 16 configurations that seem most useful, named these different configurations *banks*, and identified them with numbers from 0 through 15.

Figure 1 illustrates the configuration for bank 15. In locations $0002–$3FFF there is RAM. The 128 in the computer's name means that the computer has a total of 128K of RAM, which is arranged in two 64K blocks called *RAM 0* and *RAM 1*. Don't confuse these blocks with banks—some RAM from one or both of these blocks appears in every bank, but the amount varies.

The RAM in bank 15 is from RAM 0, the block that holds BASIC program text along with various buffers, vectors, and system variables and subroutines. For the moment, it's important to notice that a BASIC program's working values—variables, arrays, and strings—are *not* contained in the same bank as the program text itself.

Figure 1. Bank 15



As you can see in Figure 1, most addresses above 16384 ($4000) are seen as ROM. The BASIC interpreter alone occupies a hefty 28K, all the way up to 45055 ($AFFF). Above that, we have the machine language monitor and operating system (Kernal) interspersed with some I/O addresses and a tiny area earmarked for the memory management unit (MMU).

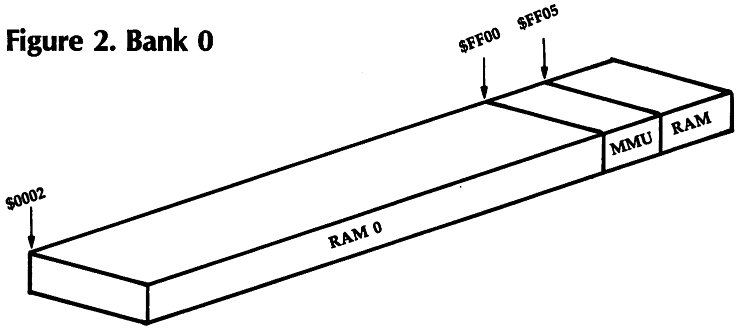
In the I/O section, locations 53248–57343 ($D000–$DFFF), all the chips from the Commodore 64 appear in the same addresses. Thus, your favorite 64 POKEs to make sound effects, and so forth, work exactly the same in 128 mode. There are numerous extra I/O locations to do new jobs, such as controlling the 80-column video chip and reading the extra keys on the 128's keyboard.

At this point, we won't worry about the machinations of the MMU; it's enough to learn that bank 15 provides access to all the I/O chips as well as to the Kernal ROM.

When you put a machine language program in RAM 0, you might be tempted to issue a BANK 0 statement from BASIC before you start the program with SYS. After all, bank 0 gives you access to all the memory in RAM 0. Don't do this: It's better to stay in bank 15.

Figure 2 shows the bank 0 configuration. Putting the computer in this configuration will certainly allow it to see your ML program in RAM 0. But the computer can't see its I/O chips or Kernal ROM. The computer has lots of memory, but no way to communicate with the outside world.

Figure 2. Bank 0



What's the lesson? Stay in bank 15. You are limited to 16K of RAM, but that's plenty for most applications.

If you don't specify a bank, the computer defaults to bank 15. However, it's prudent to execute a BANK 15 statement just before any SYS from BASIC. This insures that your program will work even if some other program has left the machine configured for a different bank. As a courtesy to other programmers (and users in general), programs that use other configurations should end by returning the machine to the default bank.

Memory Use in RAM 0

Figure 3 illustrates typical memory usage in the first 16K of RAM 0. Note that there are several unused memory areas available for program storage. Unless you're using a graphics mode, BASIC program space begins at 7168 ($1C00). (While programming in ML, you might want to avoid using an otherwise handy program known as the "DOS Shell"; it moves the start of BASIC up to $5B01 and occupies memory above $1A00—memory you may want to use for your own purposes.)

Figure 3. RAM 0 Memory Usage

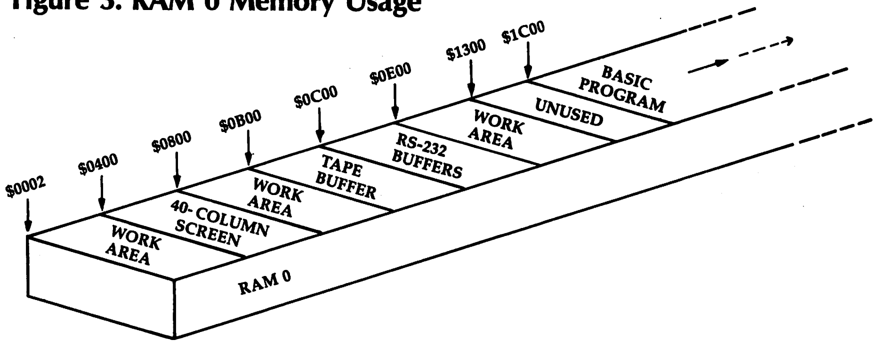


Figure 3 also reveals other unused or little-used memory zones. If you don't need to use a tape drive, the cassette buffer at 2816–3071 ($0B00–$0BFF) is free. If you aren't using telecommunications, the RS-232 buffers in locations 3072–3583 ($0C00–$0DFF) are also available. And there's a large block of empty memory marked *reserved for applications software* that stretches from 4864 to 7167 ($1300–$1BFF), providing over 2K of contiguous free space.

Friendlier BASIC

BASIC 7.0, the vastly improved BASIC in 128 mode, has several features that simplify the process of combining BASIC and ML. We won't explain all of them in detail, but here is a brief survey. (Your *System Guide* contains additional information.)

In addition to calling an ML routine, the SYS statement can also pass values from BASIC to ML. The values must be in the range 0–255 and are placed in the microprocessor's registers just before the ML routine takes over. Simply tack them onto the end of the SYS statement, separated by commas. Conversely, the RREG command lets you read the processor's registers from BASIC after an ML routine has finished.

The BLOAD command can bring in any ML module (or a graphics screen, and so forth) with no fuss or bother. The file loads into the same memory area from which it was saved, and BASIC continues with the next command. This is much simpler than the gyrations required in earlier versions of Commodore BASIC.

BASIC 7.0 also makes it easy to convert numbers between decimal and hexadecimal. The DEC function converts a hexadecimal string into a decimal number. The HEX$ function converts a decimal number into a hexadecimal string.

A Rudimentary Example

The following program isn't particularly useful, but it may interest you in the 128's new features. It counts the number of 1 bits in any eight-bit number and prints them out in a table. You may not be excited to learn that the number 14 (binary 00001110) contains three 1 bits, while the number 16 (binary 00010000) contains only one, but the program does demonstrate how to pass information from BASIC to machine language and back again.

We'll explain the purpose of each program line as we go. Here's the first one:

```
100 BANK 15
```

This statement puts the computer into bank 15, the safest configuration. Since the ML part of our program won't use any Kernal routines or I/O chips, you could use bank 0. But there's no advantage in doing so, and another time you might not be so lucky. Remember, it's always wise to set the bank explicitly rather than assume everyone's computer will be in bank 15.

```
110 DATA 162,0,74,144,1,232,168,208,249,96
```

This is the short ML program, stored in the form of DATA statements. It takes a value from the accumulator (A register), counts the 1 bits in the value, and places the result in the X register.

```
120 FOR J=2816 TO 2825
```

The actual ML code goes in locations 2816–2825 ($0B00–$0B09), the bottom of the cassette buffer.

```
130 READ X:T=T+X
140 POKE J,X
150 NEXT J
```

Before the ML can be used, it has to be READ from the DATA line and POKEd into memory. A simple additive checksum detects most typing errors.

```
160 IF T<>1334 THEN STOP
```

If the program stops at line 160, you've made a typing error, most likely in the DATA statements. If not, the ML code is safely planted in memory, and you can proceed to the job of bit counting.

```
200 FOR J=0 TO 20
```

We're going to count the 1 bits in numbers from 0 to 20. You can examine higher numbers if you like, but don't try anything over 255.

```
210 SYS 2816,J
```

This statement calls the ML program at its starting address of 2816 and passes the value of the variable J to the processor's A register. When the machine language program begins to run, the A register will contain that value. You could also have passed values to the X and Y registers, but this program doesn't require them.

```
220 RREG S,T
```

When we reach line 220, the ML program has returned control to BASIC. We'd like to know what values are in the processor's registers, especially the X register, which contains the bit count. The RREG command reads the registers and places their values into BASIC variables. The A register goes into variable S and the X register goes into T. Now T contains the bit count.

```
230 PRINT J,T
240 NEXT J
```

That's all it takes. We print the value of J and the bit count T, and then go back to do it again.

Now let's turn our attention to the 128's excellent built-in machine language monitor.

A Monitor at Your Fingertips

Some of the earlier Commodore products had no built-in machine language monitor. To work on machine language on the VIC-20 or Commodore 64, for example, you had to load a machine language monitor from tape or disk, or rely on a plug-in cartridge. Other products had simple monitors: Many PET/CBM models had monitors which could display and change memory, save or load programs, and not much else. The built-in monitor on the Commodore 128 has many attractive features; the best way to learn them is to try them.

Type MONITOR and press RETURN. You'll see the familiar register display, with values under the titles: PC (program counter), SR (status register), AC (accumulator, or A register), XR (X register), YR (Y register), and SP (stack pointer). They're all similar to what you may have met on other machines, except the value under PC looks a little odd. It has five digits instead of four. The extra digit at the beginning is the *bank number,* and since it's an *F,* we're in bank 15.

We've already noted that *bank* isn't quite the right term. We should more properly say *configuration* 15, since each configuration consists of a mixture of memory elements. Refer back to Figures 1 and 2 to see the configurations for banks 15 (the default) and 0. You'll notice that for addresses below $4000, both bank 0 and bank 15 use exactly the same memory. Thus, the contents of address $F1000 is exactly the same as the contents of address $01000. In fact, it's the same memory. We'll see for ourselves in a few moments.

Number Conversion

You may be comfortable with hexadecimal numbers. You may even be able to do hex-to-decimal conversions in your head and amaze your friends. I can't, however, and I like the number-conversion features that are built into the monitor. We've talked about hexadecimal address $4000 already. Let's find its value in decimal.

Type in the value $4000 on a line by itself and press RETURN. You'll see a display of this number as it appears in various number bases. First, the hexadecimal number. The dollar sign means hex, of course, so the monitor simply echoes what you typed in: $4000. The next line starts with a plus sign (+). To the 128's monitor, the plus sign means decimal. So you can see that $4000 equals decimal 16384. The following line starts with an ampersand (&), which means *octal,* a notation that's rarely if ever used with Commodore machines. (Octal numbers are base 8, so &40000 is equal to four times eight raised to the fourth power.) Finally, the number that starts with a percent sign (%) is the *binary* representation of $4000. Since the computer's internal code is always binary—not decimal or hexadecimal—it's sometimes useful to be able to look at a number this way.

You may also convert a decimal number to the other bases by typing it in, leading off with a plus sign. If you like, try entering +16384 and watch the computer figure out that it's the same as $4000. And if you ever need to do so, you can convert from octal or binary the same way.

Conversions are convenient, but the monitor includes another bonus: *Any number may be entered in any base, any time.* If you put in a number without a prefix, the monitor will assume you mean it to be hexadecimal. But you can slip in a decimal number anywhere by prefixing it with the plus sign. We'll be doing this; you'll see how handy it is.

Looking at Memory

You may display memory with the command M. If you follow M with two addresses, the monitor displays all the values between them. Thus, to display the contents of addresses $1000–$1029, just type

```
M 1000 1029
```

and press RETURN.

You'll get more than you bargained for. Depending on whether you are on a 40-column or 80-column screen, the monitor will display 8 or 16 memory locations at a time. Each group of locations is on a single line, with the address of the first item on the line showing at the left. We asked for 42 locations, but we got 48, since the computer always finishes the line it's working on.

On the right, we see the ASCII character equivalent of the contents of the memory locations. Some locations don't happen to have an alphanumeric equivalent, in which case a period is printed. If you display the addresses suggested above, you'll see some readable text in this area. The zone of memory we're looking at holds the function key definitions.

Just to confirm something that was said before, try using M to display memory locations F1000–F1029. That's bank 15 instead of bank 0, but you'll see that it is in fact the same memory. And you might like to try M +4096 +4137, which uses decimal addresses for the same locations.

If you follow an M command with only one address, you'll get a fixed number of memory locations. This can save you typing, and here's a tip for browsing through large amounts of memory: If you type M alone with no addresses, you'll get a continuation of the last memory display.

Making Changes Directly

The simplest way to change memory is to display the area you're interested in, then move the cursor back and type over the values on the screen. When you press RETURN, the monitor enters all the values for that line. It's a bit like screen editing in BASIC.

Try it. If you have displayed memory as suggested above, you may see the word *GRAPHIC* on the right-hand side of the memory display. Let's change the *G* stored in memory to a *T* so that it says *TRAPHIC*. The code for a *G* is $47; it's found in the left-hand part of that line. Move the cursor over the 47 and type 54, which is the code for *T*. Now press RETURN, and the memory change is made.

Remember that you can't change the right-hand ASCII side of the display. And, by the way, this is *not* the recommended way to change the function-key definitions. It's easier (and better) to use BASIC's KEY statement.

You can't change locations in read only memory (ROM). Entering

```
M F4200 F4200
```

will show you part of the BASIC ROM. Move the cursor back, type over a value, and press RETURN. You'll see from the display that the original values have been restored and ROM has not changed. Here's a note for technical types: The values from the line have "poked through" into the RAM memory which lies beneath ROM, but the monitor shows only the ROM.

The first character on the memory display line is the greater-than sign (>). This is in fact a synonym for the change memory command. On rare occasions, you might like to use this command directly.

Here's a typical case where the greater-than sign might be typed: You want to change a single location in an I/O chip. Using the "display and type over" method, you'd change 8 or 16 locations at a time. Usually, that's okay, but I/O chips are delicate, and you don't want to change other registers accidentally. As a simple example, you might like to change the 40-column border color to red, but you don't want to change anything else. You may type

```
>FD020 2
```

(remember that the I/O chips are in bank 15), and the border will change. The monitor will display a full line of memory locations, but you've changed only one. By the way, did you notice that the address you changed does not now contain the value 2 you put in? Funny things, I/O chips. If you're interested, you might type $D020 to ask the computer what decimal address in bank 15 you have changed. You might recognize the answer, +53280.

Writing a Simple ML Program

Let's write a short program to print a line of asterisks. We'll use the built-in *assembler*. Here goes:

```
A 1500 LDX #0
```

The A means assemble. The address at which we will put this instruction is 1500; it's in hexadecimal (put a dollar sign in front if you like). The instruction itself is LDX #0, load counter X with a value (the # character means a value, not an address) of zero. Press RETURN, and you'll see that the line has changed to

```
A 01500 A2 00       LDX #$00
```

The machine code in addresses 1500 and 1501 (bank 0, but in this area that's the same as bank 15) is hex A2 00. These two bytes have been placed in memory, and the monitor is ready for your next line of code; in fact, it has typed part of it for you. Complete the next line so that it reads

```
A 01502 LDA #$2A
```

This instruction, when the program runs, will load the ASCII code for an asterisk (hex 2A) into the A register; that's the register we use for printing. Continue with

```
A 01504 JSR $FFD2
A 01507 INX
A 01508 CPX #+20
```

The first instruction in this group prints a character, calling the Kernal ROM routine usually known as BSOUT (also known in the Commodore 64 as CHROUT). The next adds one to the X register, which we're using as a counter. The last instruction says, "Compare the counter with decimal 20." Note the plus sign for decimal. When you press RETURN, the line changes to

```
A 01508 E0 14       CPX #$14
```

The value 20 has been changed to hexadecimal. Don't be surprised; it's still the same number. Continue entering with

```
A 0150A BNE $1504
A 0150C LDA #$0D
A 0150E JMP $FFD2
```

The instruction BNE $1504 sends the program back to print again if you haven't reached 20 characters. The sequence LDA #$0D:JMP $FFD2 prints a carriage return and terminates the program (we know that the ROM routine at $FFD2 ends with RTS, so we can save a little code by using that RTS to return, rather than ending with the more conventional JSR $FFD2:RTS). After typing the last line, the computer prompts you with A 01511. Simply press RETURN to end the assembly.

If you like, you can proofread your program by entering the command

D 1500 150C

The D command is for disassemble, which performs an activity more or less the reverse of an assembly.

Starting Up

You can go to this program with a G (go) command, which doesn't permit a return. Better, you can call it with a J (jump subroutine) command. But first, *you must think about what bank you are in.*

If you enter the command J 1500, you'll have a disaster on your hands. Why? Because you're entering bank 0, which contains no Kernal ROM and no I/O chips. Remember, the program uses the Kernal ROM routine BSOUT to print each character. If you JSR to this routine when the Kernal ROM is absent, you'll never print those asterisks, and your program will almost certainly fail. If you really want to call this program from the machine language monitor, invoke bank 15 with J F1500.

It's also quite simple to call the routine from BASIC. First, find the starting address. Type $1500 and read the answer, decimal +5376.

Back to BASIC

Return to BASIC by giving the X (exit) command. You'll see the familiar READY response of BASIC. Now type NEW (don't worry, your machine language program won't be harmed) and enter the following program:

```
100 BANK 15
110 SYS 5376
120 PRINT "THIS WORKS"
130 SYS 5376
140 PRINT "WITHOUT PROBLEMS"
150 SYS 5376
```

Run the program, and you should see a row of asterisks. If you've done these exercises, you should have a feeling for the 128's machine language monitor. It's convenient and flexible.

Now let's look at linking BASIC and machine language programs together.

Calling and Linking

The usual way to activate a machine language program from BASIC is with a SYS statement. Typically, you load and run a BASIC program, and that program loads the machine language program as needed. Sometimes the BASIC program and its accompanying ML code are combined in a single file. When you load such a program, the ML comes into memory along with the BASIC program text, so all you need is the SYS. In other cases, the BASIC program loads the ML file in a separate operation, a process known as *overlaying*.

Overlaying is a flexible technique. A BASIC program can load more than one machine language program; it can also load data, graphics screens, or other material. When programming an overlay, you must take care so that a program doesn't self-destruct by loading something into memory which the program itself occupies.

Where memory is limited, overlays can greatly expand the capabilities of a computer. The program can load a machine language program into memory and use it; then the program can load a different program to the same part of memory, and so on. In theory, there's no limit to how big a program might be when it's brought into memory as a series of overlays. The CP/M system, which can also be used by the 128, works largely by means of overlays (in fact, when it boots in CP/M mode, the computer loads the entire CP/M operating system from disk).

Overlay Example

Let's write a simple machine language program and load it into memory. The program will, on request, print a given character a certain number of times, followed by a carriage return. We'll use it to draw a simple bar graph. Type MONITOR and press RETURN; then enter the following lines:

```
A 1400 JSR  $FFD2
A 1403 DEX
A 1404 BNE $1400
A 1406 LDA #$0D
A 1408 JMP  $FFD2
```

As you enter each line, the computer rewrites the line and prompts you with the address for the next line. A question mark means that you need to retype the line. After you enter the last line, the computer displays this line:

```
A 0140B
```

To end the assembly, press RETURN on this line without typing anything else. The line at 1400 calls the print routine, which prints whatever character is in the A register. The value in that register will be set by the BASIC calling routine. The line at 1403 subtracts one from the counter value in the X register; this value is set from BASIC as well. Lines 1404–1408 say, "If the count has not hit zero, go back; otherwise, load and print a RETURN character and return to BASIC."

After you enter the program, save it to disk with the following command:

```
S "0:+ML",8,1400,140B
```

This command saves the program under the filename +ML. There's nothing magical about the plus sign at the beginning of the filename. I prefer to put a special character at the start of the name of any file that is not intended to be loaded with a BASIC LOAD or DLOAD. This serves as a visual reminder of the file's special purpose when you are scanning a disk directory. Any legal Commodore filename can be used when you're saving files from the ML monitor. However, the BASIC program listed below expects to find a file named +ML, so you should include the plus sign for this example.

After you press RETURN, you'll see the disk light come on and hear the disk motor run. Now, for a handy feature of the machine language monitor: We'll ask the disk whether or not everything went well. Type the single character @ and press RETURN. You'll get a report from the disk. There will be a number (the error type, normally 0); a message (normally OK); and then two more numbers, which indicate the disk track and sector where the error occurred (in cases where that information is relevant). If you get the OK message, your program has been saved and you're ready to proceed.

The disk commands of the machine language monitor are very useful. They are similar to those of the disk wedge programs used in other Commodore computers. For example, type

@,$0

and press RETURN. You'll get the directory of your disk.

Now let's destroy the program we have just written and saved to disk. That way, we can confirm that our BASIC program will load it correctly from disk. We'll use the F (Fill) command to store zeros in memory locations 1400–1480:

F 1400 1480 0

The BASIC Portion

Our machine language program is gone. To exit to BASIC, type X and press RETURN. Now let's write the main program. Type NEW; then enter this program:

```
100 BANK 15
110 BLOAD "+ML"
120 IF DS<>0 THEN PRINT DS$:STOP
130 V=10
140 FOR J=1986 TO 1996
150 PRINT J;:SYS 5120,42,V
160 V=V*1.1
170 NEXT J
```

We specify bank 15 so that Kernal ROM will be visible when the machine language routine is executed. The BLOAD command brings in the program. Since we don't specify a bank, the program goes to bank 15 (which, for the addresses concerned, is the same as bank 0). Because we don't specify a starting address, the program loads at the address from which it was saved.

After the load, the program checks the disk status to make sure everything went well. The disk status reserved variable, DS, must be zero; if not, we print the status message (DS$) and stop. We don't want to SYS to a program that might not be there.

The main program plots a value that grows at 10 percent per year over 11 years. It prints each year (J) and calls the machine language routine. The operation of SYS has been enhanced in the 128's BASIC 7.0. Additional values can be added after the address; these are stored in the various microprocessor registers when the routine is executed. The SYS in line 150 places the value 42 (the character code for an asterisk) into the accumulator and the value of the variable V (which starts at 10 and grows a little for each line) into the X register. If you like, you can change the program to print a character other than the asterisk. Simply replace the number 42 with the character code for the desired symbol. Similarly, you can play around with the values of V. Remember, however, that you can only pass values less than 256 in this manner.

If you use overlay techniques, you may load your machine language program to any free memory area. Stay below location $4000 (decimal 16384), however, unless you're familiar with the fine points of the 128's banking architecture. Don't interfere with areas containing working values. Use the spare locations indicated earlier in Figure 3.

Liberating Memory

If you need a good deal of space and want to use the overlay method, there's a trick that will liberate an extra 9K block of memory up to $4000. You can easily switch BASIC so that it starts at address $4000, leaving free space in the former BASIC program area from $1C00 to $3FFF. Here's how to do it. At the start of your BASIC program, add the following line:

```
GRAPHIC 1:GRAPHIC 0
```

Here's how the trick works. When the GRAPHIC 1 statement is executed, BASIC is moved up to make room for a high-resolution graphics screen. BASIC now starts at location $4001. GRAPHIC 0 returns the display to the normal text screen, but the high-resolution screen area remains allocated, and BASIC does not move back down. The result is lots of empty memory for you to use (this method assumes that you don't need high-resolution graphics, of course).

If you use this technique, you might like to deallocate the graphics area and restore your BASIC program's original position (starting at $1C01) when the program is finished. The command to do this is GRAPHIC CLR.

Joining to BASIC

If you don't like the extra disk activity that overlays require, you might prefer a technique that is popular on many other Commodore computers: tacking a machine language program onto the end of a BASIC program. The advantage of this technique is that a single load operation brings in both the BASIC program and the machine language program. This technique works equally well with disk or tape. But there are a few points to remember.

When using this technique on other Commodore computers, you must take care not to change the BASIC program once it is in place. It's obvious when you think about it: If you add to the BASIC program, the machine language portion moves higher in memory in order to make room for the new program line(s). As a general rule, you must write the BASIC program first and refrain from changing it once it's finished.

The 128 adds another difficulty to this technique. You can't tack something onto a BASIC program if you don't know where the BASIC program is located. To explain, BASIC usually starts at $1C01, but if someone has been using graphics, the start of BASIC might be at $4001. It's no use writing a program to sit behind BASIC—at, say, location $1F80—and then discover that it sometimes loads to $4280. Chances are that it won't work in the new location, especially since it's above the dreaded $4000 barrier.

There are several ways around this problem. One is to check the start of BASIC and refuse to call the ML code if it's wrong. Another is to begin every program with GRAPHIC CLR in an attempt to move the program down to the desired area. Be careful with GRAPHIC CLR, however; it has a pitfall which we'll get to in a moment.

Sample Combining Program

Here's a small program that combines BASIC and machine language in one package. Let's write the BASIC part first:

```
100 GRAPHIC CLR
110 BANK 15
120 PRINT "SPEED TYPING"
130 PRINT "TRY TO TYPE A SENTENCE"
140 PRINT "END WITH RETURN"
150 SYS XXXX
160 PRINT "FAST, HUH?"
```

Do *not* run this program yet; the machine language is not in place. Now type GRAPHIC CLR to make sure the program is situated in the right part of memory. Enter the machine language monitor with MONITOR, and then type this command:

```
M 2D 2D
```

The first two bytes displayed on the screen should be 01 1C. This operation confirms that BASIC does indeed start at address $1C01. Now enter this command to see where the program ends:

```
M 1210 1211
```

Depending on how you typed in the BASIC program (whether you included extra spaces, for example), you'll see a first byte with a value of about $8D and a second byte of $1C. Assuming this is the place where the program ends, you can tack on machine language anywhere after about $1C8D. To give ourselves some slack, let's pick $1CC0 as our machine language starting point. Now that you've chosen this address, type $1CC0 and press RETURN. The monitor prints +7360, indicating that the decimal value of $1CC0 is 7360. Now exit to BASIC and change line 150 as shown here:

```
150 SYS 7360
```

Now reenter the monitor and enter the following machine language program:

```
A 1CC0 JSR  $FFE4
A 1CC3 CMP  #$0D
A 1CC5 BEQ  $1CD8
```

As we write this program, we'll guess at the exit address, since we haven't got there yet. We can always come back to correct this address if it's not correct.

```
A 1CC7 BCC  $1CC0
A 1CC9 LDX  +30
```

Note that the monitor changes the decimal value 30 to $1E when you press RETURN.

```
A 1CCB JSR  $FFD2
A 1CCE DEX
A 1CCF BNE  $1CCB
A 1CD1 LDA  #$0D
A 1CD3 JSR  $FFD2
A 1CD6 BNE  $1CC0
A 1CD8 RTS
```

On the last line, we see that the exit address is $1CD8. If you had guessed wrongly on line 1CC5, this would be the time to go back and correct it. Now, here's the payoff. Display the end-of-BASIC pointer with the command M 1210 1211. You'll see the same addresses as before. Move the cursor back and change the display to read

```
>01210 D9 1C .. .. ..
```

After you press RETURN, it's safe for you to save the entire package. When you do so, the BASIC and machine language files will be saved as one block. When you reload the file, both programs will come in together. But there's a pitfall which is related to the GRAPHIC CLR statement we used in the BASIC program. When you execute GRAPHIC CLR, you may reset the contents of locations $1210–$1211 back to their original values. If you use GRAPHIC CLR in a program as we've done here, be sure to save the program *before* you run it. To save the program, return to BASIC and save the program with the usual DSAVE command. Run the program and try typing a sentence; you'll be amazed to discover what a speedy typist you've become.

Chapter 6

Utilities

TurboDisk 128

Don Lewis

Are you using a 1541 disk drive with your Commodore 128? Here's a powerful utility that can reduce the time you spend waiting for programs to load by 300 percent or more.

If you've upgraded to a Commodore 128 from a VIC-20 or 64 and are still using a 1541 disk drive, you're probably envious of those fellow 128 owners whose 1571 drives can load programs in the blink of an eye. Perhaps you've used "TurboDisk 64" in 64 mode and wished for an equivalent speedup for 128 mode. Here's the answer: "TurboDisk 128," a new and improved version specifically for the 128/1541 combination.

TurboDisk 128 works only in 128 mode. And the program works only with a 1541; it isn't useful in conjunction with a 1571. If you own a 1571 disk drive, you don't need a turbo program while you're in 128 mode: The 128 and 1571 can use the fast serial-transfer hardware built into both disk drive and computer, which is as much as eight times faster than a standard 1541—about twice as fast as TurboDisk.

But even if TurboDisk 128 doesn't permanently cure your desire for a 1571, it will make your life with the 1541 more bearable. In fact, once you start using TurboDisk, you'll wonder how you got along without it. TurboDisk turbocharges the loading process by a factor of three times or more. In fact, the longer the program, the more improvement you'll see. Like TurboDisk 64, the 128 version requires no modifications to your disk drive or computer. It loads programs saved in the usual manner; no special Turbosave is required. It works with most BASIC and machine language programs. It does not compromise reliability. And you can switch it on or off at any time by typing a single command.

Typing It In

Since TurboDisk is written entirely in machine language, it must be entered with the 128 version of our "MLX" machine language entry program, which is found in Appendix C. Be sure that you read and understand the instructions for using MLX before you begin entering the data for TurboDisk. When you first run MLX, you'll be asked for starting and ending addresses. The correct values are

Starting address: 1300
Ending address: 16CF

Now you can begin entering the data for TurboDisk. When you've finished entering the numbers, be sure to use the MLX Save option to make at least one copy of the TurboDisk data. You'll probably find TurboDisk so useful that you'll want a copy on every disk you use. You can use the MLX Save option repeatedly to make copies on different disks. If you want to put a copy of TurboDisk onto a new disk at some later date, you can use any copy program. Or load an existing copy of TurboDisk, place the disk on which you wish to store the new copy of TurboDisk into the drive, and use a command of the form

BSAVE *"filename"*,B0,P4864 TO P5839

To load TurboDisk, use a command of the form

BLOAD "TURBODISK 128":SYS DEC("1300")

(replacing TURBODISK 128 with whatever filename you used when you stored the TurboDisk data). The message C128 TURBODISK ENABLED signals that you're ready for high-speed loading.

Turbo LOADs

Once TurboDisk is activated, no special commands are necessary. Just type LOAD *"filename"*,8 or DLOAD *"filename"* or BLOAD*"filename"* as usual. You'll be amazed at the difference.

One thing you'll notice immediately is that the red light on the disk drive doesn't come on at all during a Turboload. Don't panic; this is normal. It's also normal for the 40-column screen to blank while TurboDisk works. When the program is loaded, the screen reappears unaltered.

You may occasionally find it necessary to deactivate TurboDisk and use a normal LOAD instead. For example, 1541 disk drives are prone to head-alignment problems, so if you have a disk formatted on a drive other than your own, you may find that your drive has difficulty loading programs from it. You can switch off TurboDisk at any time without erasing it from memory by entering SYS DEC("1303") or the equivalent SYS 4867. You should see the message C128 TURBODISK DISABLED. To be safe, it would be wise to include a BANK 15 before the SYS to insure that the system is in its normal BASIC configuration. To reactivate TurboDisk, enter SYS DEC("1300") or the equivalent SYS 4864 (again, it would be wise to precede this with a BANK 15). You should see the message C128 TURBODISK ENABLED to indicate that turboloading is now available.

You'll also find it necessary to use the SYS to reactivate TurboDisk after pressing RUN/STOP–RESTORE, which effectively disconnects TurboDisk.

TurboDisk resides in the currently unused area of free memory starting at address 4864–5839 (hex $1300–$16CF), so it's completely safe from BASIC. However, this memory area is rapidly becoming popular with 128 machine language programmers, and you may find other programs that use these locations. Such programs cannot be used with TurboDisk because loading them will overwrite the TurboDisk program. TurboDisk also uses the block of memory at 3072–3327 ($0C00–$0CFF) as a buffer for the data read from disk. This area is the RS-232 input buffer, but since the 128 can't turboload and receive RS-232 input simultaneously, this dual usage should cause no conflict. However, you should be aware that some programmers use the RS-232 buffers for machine language routines. Such routines cannot be used with TurboDisk.

TurboDisk speeds up LOAD, DLOAD, BLOAD, and the monitor's L command, but it can't speed up SAVE or VERIFY. It also doesn't affect the speed of disk file handling with PRINT#, GET#, and so forth. It's not compatible with certain features of some programs and may not work with some commercial software.

How It Works

The machine language for TurboDisk is unusual in that only half of it works within your computer—the rest is actually executed within the 1541 drive itself. Commodore disk drives are *intelligent* units, containing their own microprocessors, RAM, and ROM. This means that they can be programmed for special effects, like turboloading.

During the brief delay you notice between the time you enter the load command and the time the drive starts spinning, 464 bytes of machine language are transferred from the computer to the drive's RAM. In the 128, this data is stored in locations 5376–5839 ($1500–$16CF). This required transfer before each Turboload adds a certain amount of overhead time, which explains why TurboDisk gives less improvement in speed for short programs.

The 128-resident portion of TurboDisk operates by changing the ILOAD vector at locations 816–817 ($330–$331) to point to itself, bypassing the normal LOAD routines in ROM. TurboDisk first checks to see whether a disk directory or a verify operation has been requested. In either of these cases, control is returned to the ROM routines for normal processing. If a program load has been requested, the routine adds the filename to the code for the disk drive portion and then transfers that data to the drive's memory.

The portion of TurboDisk in the disk drive uses routines in the drive's ROM to locate the desired program and read it from the disk, sector by sector. To improve speed, routines like the one that turns on the red light are omitted, and only the essential ones are used. The 256 bytes of data from each disk sector are sent to a 256-byte buffer within the computer. As mentioned above, this buffer is at locations 3072–3327 ($0C00–$0CFF). TurboDisk sends data over both the DATA and CLK lines on the serial port, instead of just the DATA line as in normal serial data transfers. Thus, TurboDisk temporarily converts your serial bus into a two-bit parallel bus. When the entire 256 bytes from a disk sector have been transferred into the computer's buffer, data from the buffer is added to the program in memory while the drive is reading the next sector from the disk.

The Longer, the Faster

Despite a few limitations, TurboDisk is one of the most valuable general-purpose utilities a disk user can own. To discover exactly how fast it is, we ran some tests. The results in the table demonstrate that TurboDisk yields the most improvement with medium to long programs. Results with different disk drives may vary.

TurboDisk Results

Program	Blocks	Normal Load (seconds)	Turboload (seconds)	Factor
1	7	7	3	2.33
2	16	13	4	3.25
3	28	20	6	3.33
4	55	40	10	4.00
5	138	94	25	3.76

Note to Readers Outside North America

High-speed TurboDisk data transfers rely on precise timing, so the program may fail to operate on systems that use the European PAL video system instead of the North American NTSC system. The reason is rather technical: 128s with PAL video use a slightly different microprocessor clock frequency. An Australian reader submitted a modification to compensate for this in the Commodore 64 version. Since we do not have access to a 128 with PAL video for testing, we cannot guarantee that the same modification will work in the 128 version, but if TurboDisk 128 will not operate properly with your PAL video system you can try changing the following line:

```
1468:A2 Ø4 CA DØ FD EA A2 Ø4 2E
```

TurboDisk 128

See instructions in article, and read Appendix C, "MLX," before typing in the following program listing.

Starting address: 1300
Ending address: 16CF

```
13ØØ:4C 1B 13 A9 6C 8D 3Ø Ø3 ØD
13Ø8:A9 F2 8D 31 Ø3 AØ ØØ B9 D8
131Ø:29 13 FØ Ø6 2Ø ØC CØ C8 89
1318:DØ F5 6Ø A9 5C 8D 3Ø Ø3 47
132Ø:A9 13 8D 31 Ø3 AØ 1A DØ 44
1328:E6 ØD 43 31 32 38 2Ø 54 87
133Ø:55 52 42 4F 44 49 53 4B ØC
1338:2Ø 44 49 53 41 42 4C 45 CE
134Ø:44 ØD ØØ ØD 43 31 32 38 18
1348:2Ø 54 55 52 42 4F 44 49 84
135Ø:53 4B 2Ø 45 4E 41 42 4C 93
1358:45 44 ØD ØØ 85 93 A5 93 2D
136Ø:FØ Ø5 A5 93 4C 6C F2 A2 CA
```

```
1368:10 A9 A0 9D C0 16 CA 10 F2
1370:FA A0 00 20 AE F7 C9 24 4B
1378:F0 E8 AD 30 D0 85 FA A9 46
1380:0B 8D 11 D0 A9 FD 8D 30 4F
1388:D0 A0 01 20 AE F7 C9 3A 84
1390:F0 04 A0 00 F0 01 C8 A2 04
1398:FF E8 20 AE F7 9D C0 16 B5
13A0:C8 C4 B7 90 F4 20 8E 14 B5
13A8:A5 BA 20 B1 FF A9 6F 20 15
13B0:93 FF A9 55 20 A8 FF A9 78
13B8:43 20 A8 FF 20 AE FF 78 D1
13C0:20 4D 14 2C 00 0C 30 5D 7D
13C8:A4 C3 A6 C4 A5 B9 F0 06 4F
13D0:AC 02 0C AE 03 0C 84 AE 3A
13D8:86 AF A2 04 AD 00 0C F0 39
13E0:15 20 2F 14 20 4D 14 AD CC
13E8:00 0C 30 3C F0 06 20 2D E8
13F0:14 4C E4 13 A2 02 86 FB 28
13F8:A0 00 BD 00 0C 20 BF F7 7F
1400:C8 E6 FB A6 FB EC 01 0C D1
1408:90 F0 BD 00 0C 20 BF F7 C4
1410:C8 20 40 14 18 A6 FA 8E CD
1418:30 D0 A2 1B 8E 11 D0 A6 93
1420:AE A4 AF 58 60 A9 04 2C 22
1428:A9 00 38 B0 E8 A2 02 86 93
1430:FB A0 00 BD 00 0C 20 BF 8A
1438:F7 C8 E6 FB A6 FB D0 F3 E5
1440:18 98 65 AE 85 AE A5 AF 14
1448:69 00 85 AF 60 A9 FC 8D 02
1450:30 D0 A0 00 AD 00 DD 30 32
1458:FB A9 17 8D 00 DD AD 00 77
1460:DD 10 FB A9 07 8D 00 DD E1
1468:A2 04 CA EA D0 FC A2 04 AE
1470:AD 00 DD 0A 08 0A 26 95 16
1478:28 26 95 CA D0 F2 A5 95 D0
1480:49 FF 99 00 0C C8 D0 CC 72
1488:A9 FD 8D 30 D0 60 A9 10 25
1490:85 FF A9 00 85 FB 85 FD D5
1498:A9 15 85 FC A9 05 85 FE C6
14A0:A5 BA 20 B1 FF A9 6F 20 0F
14A8:93 FF A5 FD A4 FE 8D F0 5C
14B0:14 8C F1 14 A0 00 B9 ED EB
14B8:14 20 A8 FF C8 C0 06 D0 2E
14C0:F5 A0 00 B1 FB 20 A8 FF D8
14C8:C8 C0 20 90 F6 A5 FB 69 42
14D0:1F 85 FB A5 FC 69 00 85 D6
14D8:FC A5 FD 69 20 85 FD A5 F7
14E0:FE 69 00 85 FE 20 AE FF 11
14E8:C6 FF D0 B4 60 4D 2D 57 C3
```

```
14FØ:ØØ ØØ 2Ø FF FF FF FF FF 1D
14F8:FF FF FF FF FF FF FF FF 21
15ØØ:2Ø 42 DØ 78 A9 15 8D Ø7 3Ø
15Ø8:1C A9 12 AØ Ø1 8D ØØ Ø3 38
151Ø:8C Ø1 Ø3 2Ø CD Ø5 A9 Ø3 FB
1518:85 3C A2 ØØ 86 4B FØ 2B D6
152Ø:AØ ØØ B1 3B 29 BF C9 82 E2
1528:DØ 19 C8 C8 C8 B9 BD Ø6 55
153Ø:C9 2A FØ 42 C9 3F FØ Ø4 3D
1538:D1 3B DØ Ø7 C8 CØ 12 FØ Ø3
154Ø:35 DØ EA E6 4B A6 4B EØ 71
1548:Ø8 FØ Ø7 BD 6E Ø5 85 3B 3D
155Ø:DØ CE AD ØØ Ø3 FØ Ø6 AC EØ
1558:Ø1 Ø3 4C 13 Ø5 A9 FF 8D DA
156Ø:ØØ Ø3 2Ø 96 Ø5 A9 3A 8D 89
1568:Ø7 1C 58 4C 45 D9 Ø2 22 A4
157Ø:42 62 82 A2 C2 E2 E6 3B 79
1578:AØ ØØ B1 3B 8D ØØ Ø3 C8 18
158Ø:B1 3B 8D Ø1 Ø3 2Ø CD Ø5 4D
1588:2Ø 96 Ø5 AD ØØ Ø3 DØ F5 87
159Ø:A9 3A 8D Ø7 1C 6Ø AØ ØØ E3
1598:B9 ØØ Ø3 85 85 A9 Ø2 8D BC
15AØ:ØØ 18 AD ØØ 18 29 Ø4 FØ E4
15A8:F9 A9 ØØ 8D ØØ 18 A2 Ø4 BC
15BØ:A9 ØØ Ø6 85 2A ØA Ø6 85 D3
15B8:2A ØA 8D ØØ 18 CA DØ FØ AA
15CØ:48 68 48 68 A9 ØØ 8D ØØ 21
15C8:18 C8 DØ CC 6Ø A9 Ø3 85 4D
15DØ:86 AC Ø1 Ø3 84 Ø7 AD ØØ 55
15D8:Ø3 C5 Ø6 Ø8 85 Ø6 28 FØ BC
15EØ:1Ø A9 BØ 85 ØØ 58 24 ØØ 95
15E8:3Ø FC 78 A5 ØØ C9 Ø1 DØ CD
15FØ:54 A9 EE 8D ØC 1C A9 Ø6 9Ø
15F8:85 32 A9 ØØ 85 33 85 3Ø DB
16ØØ:A9 Ø3 85 31 2Ø 5C Ø6 5Ø 54
16Ø8:FE B8 AD Ø1 1C 99 ØØ Ø3 F1
161Ø:C8 DØ F4 AØ BA 5Ø FE B8 4B
1618:AD Ø1 1C 99 ØØ Ø1 C8 DØ DE
162Ø:F4 2Ø EØ F8 A5 38 C5 47 5B
1628:FØ Ø4 A9 22 DØ 11 2Ø E9 1A
163Ø:F5 C5 3A FØ Ø4 A9 23 DØ FC
1638:Ø6 A9 EC 8D ØC 1C 6Ø C6 AØ
164Ø:86 DØ AE FØ Ø3 18 69 18 2C
1648:85 44 A9 FF 8D ØØ Ø3 2Ø 1Ø
165Ø:96 Ø5 A9 3A 8D Ø7 1C A5 48
1658:44 4C C8 C1 2Ø 62 Ø6 4C D1
166Ø:9E Ø6 A5 12 85 16 A5 13 16
1668:85 17 A5 Ø6 85 18 A5 Ø7 11
167Ø:85 19 A9 ØØ 45 16 45 17 FE
```

```
1678:45 18 45 19 85 1A 2Ø 34 9Ø
168Ø:F9 A2 5A 2Ø 9E Ø6 5Ø FE 4C
1688:B8 AD Ø1 1C D9 24 ØØ DØ 8E
169Ø:Ø6 C8 CØ Ø8 DØ FØ 6Ø CA 6Ø
1698:DØ E9 A9 2Ø DØ A1 A9 DØ 1Ø
16AØ:8D Ø5 18 A9 21 2C Ø5 18 4E
16A8:1Ø 9E 2C ØØ 1C 3Ø F6 AD 47
16BØ:Ø1 1C B8 AØ ØØ 6Ø FF FF Ø7
16B8:FF FF FF FF FF FF FF FF E4
16CØ:AØ AØ AØ AØ AØ AØ AØ AØ EC
16C8:AØ AØ AØ AØ AØ AØ AØ AØ F4
```

Keymaster

Bob Kodadek

This powerful utility becomes an extension of the 128's operating system, providing full use of the extended keyboard in the 64 mode.

When the Commodore 128 was designed, a major concern was that it should be hardware- and software-compatible with the Commodore 64. Early reports stated that a complete 64 would be built inside the 128. Actually, the 64 mode is a 128 running in an emulation state—the 128 thinks it's a 64. This is better than having another 64 around. In fact, it can open up some doors for use.

Inside the 128

There are two processors inside the 128, the new 8502 and the Z80. The 8502 chip is the main processor in the 64 and 128 modes and it is used as a coprocessor for input/output (I/O) in the CP/M mode. In order to keep the 128 compatible with the vast amount of 64 software, only the standard 66 keys were made available in the 64 mode. Since the extended keyboard is a most desirable feature on the 128, it would have been pleasing to have use of the full keyboard in the 64 mode. The VIC-II video chip, however, has two new registers, 47 and 48. Register 47 appears at location 53295 ($D02F) and is used by the 128 to read the numeric keypad, the top row of keys which include the cursor keys, HELP, TAB, and other special keys. With special coding, this register can be read in the 64 mode and utilized in the same manner.

Extending the Keyboard

The accompanying program, "Keymaster," is an all machine language program, with the ML in the DATA statements. It becomes an extension of the operating system, providing full use of the extended keyboard in the 64 mode, while still maintaining full compatibility. The repeating numeric keypad and the cursor keys located on the top row of keys, NO

SCROLL, and TAB keys are programmed to perform exactly as in 128 mode. The CAPS LOCK key is used as an aid in typing DATA statements. When this key is depressed, the period located in the keypad becomes a comma, allowing one-handed decimal data entry.

With the exception of the 40/80 DISPLAY key, each key is read and assigned a value from 65 to 88 and a CHR$ code. When a key is pressed, the CHR$ code is put into the keyboard buffer, and the keypress value is stored into two accessible RAM locations as the current keypress and the last keypress. This allows each key to be read and programmed from within a program by the use of simple PEEK or GET instructions.

This tiny, yet powerful, utility program has its own interrupt-driven scan-key routine, a wedge to operate the NO SCROLL and TAB key functions, and a nonmaskable interrupt (NMI) handler. The NMI handler prevents the program from being disabled when the RUN/STOP and RESTORE keys are pressed. The keyboard will stay active until you turn off the computer or perform a system reset.

Using the Program

Type in and save the BASIC loader program (Program 2). When you run it, the top-of-memory pointer will be moved down, all variables will be cleared, and the machine code will be POKEd into a newly protected area of BASIC RAM at 40624–40959 ($9EB0–$9FFF). I chose to place the code in BASIC memory so that many other machine language programs that you may have will be able to run with Keymaster. Programs such as *Merlin 64* and *Micromon* use the area 49152–53247 ($C000–$CFFF) and can be used with the new keyboard.

Keymaster actually becomes part of the operating system. Though it alters the interrupt request (IRQ) vector to point to itself, you may still use your own interrupts as long as you remember to save the address in 788–789 ($314–$315) and jump to it rather than the normal IRQ located at $EA31. In an emergency, pressing RUN/STOP–RESTORE will function as usual, except the Keymaster's vectors will be reset instead of the default vectors. The best way to disable the new keyboard is to perform a system reset. Do this by holding down the Commodore key and pressing the RESET button, or by doing a SYS

64738. Either of these resets will return the machine to the 64 mode with no loss of memory.

After a reset, the BASIC pointers will be restored to their default values, allowing BASIC to store variables on top of the keyboard code. In this condition, any attempt to load or save a program will ruin the code because the filename will be stored there. If you want to restart Keymaster after a reset, you'll need to do the following in direct mode:

```
POKE 55,175:POKE 56,158
CLR :SYS 40624
```

If you don't need the current BASIC program in memory, the easiest method is to load and run the BASIC loader again.

All of the new keys can be read and used in your own programs. Each key has been assigned a CHR$ code that is

The New Keys

Key Pressed	Value	CHR$ Code
None	64	—
HELP	65	21
8	66	56
5	67	53
TAB	68	22
2	69	50
4	70	52
7	71	55
1	72	49
ESC	73	27
+	74	43
−	75	45
LINE FEED	76	10
ENTER	77	13
6	78	54
9	79	57
3	80	51
ALT	81	23
0	82	48
.	83	46
Crsr Up	84	145
Crsr Dn	85	17
Crsr Lt	86	157
Crsr Rt	87	29
NO SCROLL	88	24

placed in the keyboard buffer when pressed. Using the GET command will return the code for each new key. Similar to location 197, new location 40956 returns the current new key being pressed. If no key is pressed, a 64 will be returned. New keys will return the values 65–88. Location 40955 holds the value of the last key that was pressed during the interrupt, and it is used to initiate the pause before a key begins repeating.

Refer to the table for a list of all the new keys and the corresponding keypress values and CHR$ codes.

To get a better idea of how all this works, type in and run the short BASIC demo program, Program 1, when Keymaster is active.

Another important location, 40800, holds the number of spaces for the TAB function. The default is 5, but you may place any number from 0 through 9 here.

The NO SCROLL function works a little better than the one in 128 mode. The routine is wedged into the IBSOUT vector at 806–807 ($326–$327). It looks for a carriage return before allowing you to stop a listing. Therefore, it won't stop in the middle of a line. It will also work with a Commodore printer to freeze printing.

Program 1. Keymaster Demo

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
FC 10 PRINT"{DOWN}{2 SPACES}PRESS ANY KEY":PRINT
BX 20 GETA$:IFA$=""THEN20
DA 30 REG=PEEK(197):EXT=PEEK(40956)
QB 50 PRINT" REGULAR KEYPRESS:"REG
ER 60 PRINT"EXTENDED KEYPRESS:"EXT
HD 70 PRINT"{7 SPACES}CHR$ CODE ="ASC(A$)
KS 80 PRINT"{7 SPACES}CHARACTER = "A$
KX 90 POKE 198,0:IFA$=CHR$(27)THEN END
SJ 100 GOTO10
```

Program 2. Keymaster

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
FQ 10 POKE 55,175:POKE 56,158:CLR
HS 20 PRINT"READING...":ML=40624
SH 30 FOR I=0 TO 334:READ BYTE:POKE ML+I,BYTE
EQ 40 CK=CK+BYTE:NEXT I
```

```
SX 5Ø IF CK <> 43161 THEN PRINT"DATA ERROR!":END
SP 6Ø SYS ML:NEW
BD 1ØØ DATA 169,127,141,13,22Ø,32,19Ø,158
CS 1Ø1 DATA 169,129,141,13,22Ø,96,162,8
GF 1Ø2 DATA 16Ø,159,142,2Ø,3,14Ø,21,3
BG 1Ø3 DATA 169,226,16Ø,158,141,24,3,14Ø
QJ 1Ø4 DATA 25,3,162,175,16Ø,159,142,38
GM 1Ø5 DATA 3,14Ø,39,3,169,Ø,141,254
KF 1Ø6 DATA 159,96,72,138,72,152,72,169
AG 1Ø7 DATA 127,141,13,221,172,13,221,48
CR 1Ø8 DATA 2Ø,32,188,246,32,225,255,2Ø8
KH 1Ø9 DATA 12,32,19Ø,158,32,163,253,32
SD 11Ø DATA 24,229,1Ø8,2,16Ø,76,114,254
KP 111 DATA 162,Ø,142,Ø,22Ø,172,1,22Ø
XE 112 DATA 192,255,2Ø8,8Ø,169,64,141,252
FJ 113 DATA 159,162,46,165,1,41,64,2Ø8
JP 114 DATA 2,162,44,142,244,159,14Ø,Ø
GX 115 DATA 22Ø,2ØØ,14Ø,47,2Ø8,174,1,22Ø
XM 116 DATA 224,255,24Ø,37,2ØØ,169,254,141
MR 117 DATA 47,2Ø8,162,8,72,173,1,22Ø
EE 118 DATA 2Ø5,1,22Ø,2Ø8,248,74,176,3
CP 119 DATA 14Ø,25Ø,159,2ØØ,192,25,176,33
QR 12Ø DATA 2Ø2,2Ø8,242,56,1Ø4,42,76,55
BQ 121 DATA 159,173,252,159,141,251,159,16Ø
AC 122 DATA 5,14Ø,253,159,169,255,141,47
CQ 123 DATA 2Ø8,169,127,141,Ø,22Ø,76,49
PM 124 DATA 234,1Ø4,172,25Ø,159,185,225,159
PR 125 DATA 17Ø,152,24,1Ø9,252,159,141,252
HA 126 DATA 159,2Ø5,251,159,24Ø,13,16Ø,16
QS 127 DATA 14Ø,14Ø,2,141,251,159,32,52
FG 128 DATA 235,2Ø8,3,32,13,235,173,252
MB 129 DATA 159,2Ø1,68,2Ø8,15,174,253,159
RD 13Ø DATA 24Ø,1Ø,162,32,32,52,235,2Ø6
CD 131 DATA 253,159,16,241,76,1ØØ,159,2Ø1
XP 132 DATA 13,2Ø8,43,72,138,72,32,225
AQ 133 DATA 255,24Ø,27,173,252,159,2Ø1,88
EK 134 DATA 2Ø8,15,173,254,159,73,255,141
KK 135 DATA 254,159,173,252,159,2Ø1,64,2Ø8
GK 136 DATA 249,173,254,159,2Ø8,224,169,Ø
RD 137 DATA 141,254,159,1Ø4,17Ø,1Ø4,76,2Ø2
EH 138 DATA 241,Ø,21,56,53,22,5Ø,52
GP 139 DATA 55,49,27,43,45,1Ø,13,54
HA 14Ø DATA 57,51,23,48,46,145,17,157
RP 141 DATA 29,24,Ø,Ø,Ø,5,Ø
```

KeyDef

A. F. Shephard

With this short utility, you can redefine any—or all—of the keys to print whatever character you choose.

The Commodore 128 is a highly versatile and powerful machine. It offers windows, high-resolution graphics, 40- or 80-column displays, redefinable function keys, and more. But it suffers from a few shortcomings. For example, the numeric keypad is almost useless for DATA statements because there is no comma. You could, of course, use the KEY statement to redefine one of the function keys as a comma, but wouldn't it make more sense to change the period to a comma? Unfortunately, KEY acts only on the function keys. But "KeyDef" lets you redefine any key on the keyboard and gives you access to six completely different keyboards.

Scanning for a Code

To understand how this is possible, you need to understand how the 128 determines which character to display when a key is pressed. Here's a short synopsis: The computer interrupts its normal operations 60 times per second to perform housekeeping chores. One of these chores is checking to see whether a key is pressed. This process, called the *keyscan,* involves checking each key in sequence to see whether it is pressed. The 88 keys are arranged electrically in a matrix as 8 rows of 11 columns. Each key has a *keycode* that reflects the key's position within the matrix. Only one keycode is returned per keyscan; if more than one key is pressed, the value returned will be that for the key with the higher keycode. Keycodes range from 0 to 87; if no key is pressed, the keyscan routine returns the value 88. You can find the keycode for most keys by running this one-line program:

```
10 PRINT PEEK(213): GOTO 10
```

The accompanying table is a complete list of keycodes.

Commodore 128 Keycodes

Key	Keycode	Key	Keycode
←	57	RETURN	1
1	56	Z	12
2	59	X	23
3	8	C	20
4	11	V	31
5	16	B	28
6	19	N	39
7	24	M	36
8	27	,	47
9	32	.	44
0	35	/	55
+	40	CRSR up/down	7
−	43	CRSR left/right	2
£	48	Space	60
CLR/HOME	51	F1	4
INST/DEL	0	F3	5
Q	62	F5	6
W	9	F7	3
E	14	ESC	72
R	17	TAB	67
T	22	HELP	64
Y	25	LINE FEED	75
U	30	NO SCROLL	87
I	33	Cursor ↑	83
O	38	Cursor ↓	84
P	41	Cursor ←	85
@	46	Cursor →	86
*	49		
↑	54	**Numeric Keypad**	
RUN/STOP	63	0	81
A	10	1	71
S	13	2	68
D	18	3	79
F	21	4	69
G	26	5	66
H	29	6	77
J	34	7	70
K	37	8	65
L	42	9	78
:	45	+	73
;	50	−	74
=	53	.	82
		ENTER	76

Translation Tables

If you're familiar with the 128's character (ASCII) codes, it will be obvious that the keycodes don't have any direct relationship to the character codes for the corresponding letters and numbers on the faces of the keys, so you may be wondering how the 128 translates the keycode into a character code. Note that none of the shift keys (SHIFT, Commodore, CONTROL, ALT, or CAPS LOCK) appears in the keycode table. Instead, the keyscan routine checks these keys to select one of six translation tables. The keycode is then used as an index to the table to find the equivalent character code for the key being pressed. For example, suppose SHIFT-N is pressed; the keyscan routine will select the SHIFT translation table and read the thirty-ninth value in the table as the value for SHIFT-N (the keycode for N is 39).

To determine the addresses of the translation tables, the 128 maintains a set of six *pointers* in memory (a pointer consists of two consecutive memory locations that together hold an address in the standard low-byte/high-byte format). The pointers to the keyboard translation tables are found on page 3 of the 128's memory:

830–831 ($033E–033F)	**Standard (unshifted)**
832–833 ($0340–0341)	**SHIFT**
834–835 ($0342–0343)	**Commodore**
836–837 ($0344–0345)	**CONTROL**
838–839 ($0346–0347)	**ALT**
840–841 ($0348–0349)	**CAPS LOCK**

On power-up or reset, these locations are initialized to point to tables in ROM. The pointers can be changed to point to customized tables in RAM, however, and that's just what KeyDef does. It copies the usual values from the ROM tables down to a free area of RAM memory and then changes the pointers to point to the new tables in RAM. By POKEing new ASCII codes into the relocated tables, you can redefine the keyboard.

Each of the translation tables contains 89 bytes; the first 88 bytes in the table each correspond to one keycode, while the last byte is the value returned if no key is pressed. There are actually only five different tables in ROM. The ALT key pointer starts out with exactly the same value as the unshifted key pointer, which explains why ALT doesn't seem to do anything. However, KeyDef sets up a separate table for ALT, so it can now have definitions completely independent of the unshifted key definitions.

Using the Program

KeyDef is written entirely in BASIC so there are no special instructions for typing it in. Just remember to save a copy before running it.

After a brief pause, during which the key definitions are copied from the translation tables in ROM down to a free area of RAM memory, KeyDef prompts you to enter the key you wish to redefine. Press the appropriate key. Next, you're asked for the table in which you wish to redefine the key (unshifted, SHIFT, Commodore, CONTROL, ALT, or CAPS LOCK). You are then told the current ASCII (character) code value of the key in the selected table. Finally, you're prompted to enter the new character code for the key.

If you're unfamiliar with Commodore's character codes, look in Appendix E of the *System Guide*. Unfortunately, this table was copied verbatim from the 64 *User's Guide*, and some of the character codes between 0 and 32 are incorrect. For a more accurate list of these codes, see Appendix I. CTRL-G (code 7), for example, produces a bell tone. It's listed in Appendix I, but not in Appendix E.

There are some tricky keys that we need to mention. One is NO SCROLL. You must press it twice for KeyDef to recognize it when it asks "Which Key?" After that, it functions as you want it to. Also, the current function-key definitions are stored and erased at the beginning of KeyDef, then restored at the end. Thus, to keep your function-key definitions intact, do not quit KeyDef with RUN/STOP–RESTORE. HELP and SHIFT–RUN/STOP are normally character codes 132 and 131, respectively. These can also be moved around the keyboard. RUN/STOP is an odd case. Press SHIFT with it if you want to redefine it.

Notice that the numeric keypad numbers are different from the numbers on the top of the keyboard, even though normally they act the same. The four cursor keys at the top of the keyboard also have different keycodes from the pair at the bottom, so the two sets can be independently redefined. A word of caution: Be careful about redefining the numbers on the keyboard, especially 1–6. These numbers are used as menu items in the program and if they're redefined, you'll have problems returning them to their normal status without rebooting the system.

If you need to recopy the standard keyboard definitions to set things back to normal after KeyDef is first run, exit the program and type RUN 20. Should you wish to save a set of definitions, the procedure is easy. Just enter the following commands:

BSAVE *"filename1"*,B0,P5888 TO P6422
BSAVE *"filename2"*,B0,P830 TO P842

These redefinitions can be reloaded with these commands:

BLOAD *"filename1"*
BLOAD *"filename2"*

The KeyDef program doesn't need to be present in memory once you've used it to make the new definitions. The tables that are BLOADed are sufficient to redefine the keyboard. The computer automatically handles the keyscanning and character translation. However, you must take some care when changing the pointers to the translation tables. Before you change the pointers, the addresses to which you redirect the pointers must contain valid translation tables (which is why you must BLOAD the tables before you BLOAD the vector values). Otherwise, you won't be able to type on the keyboard because the computer will not be able to interpret your keypresses. In fact, you can't even use RUN/STOP–RESTORE if the RUN/STOP key isn't defined; your only recourse will be the RESET switch.

The area of memory set aside for the new translation tables is $1700–$1915 (5888–6421)—a total of 534 bytes for the six 89-byte tables. According to the memory map in the *System Guide*, this is part of the area reserved for function-key software. Basically, it doesn't get used by BASIC, so it's a perfect place for data like this. (Other machine language routines may use this area, however, so you should be on the lookout for conflicts. "TurboDisk 128," at the beginning of this chapter, also resides in part of this reserved area, but it uses addresses lower than KeyDef, so the two programs can safely be used together.)

It should be noted that KeyDef changes only the keyboard behavior; the way the computer runs is unaltered. Since the purpose of this program is not to create custom characters, the letters on the screen keep their usual shapes. Also, if you change the letter *T* to *Z*, you've banished *T* from the keyboard, but not from the computer. It can still print a *T* if you type ? CHR$(84). The ASCII codes for the characters remain the same.

A Few Suggestions

The first thing you might want to do with KeyDef is to correct a minor, but annoying, bug. The CAPS LOCK key on the top row works like the SHIFT LOCK key for all alphabetic keys except for the letter *Q*, which still prints its unshifted form. This is sometimes called the *caps-lock-q bug*, and the reason it occurs is very simple—whoever at Commodore prepared the CAPS LOCK key translation table put the wrong character code value in the keycode position for *Q*. KeyDef allows you to fix this bug. Just replace the value for *Q* in the CAPS LOCK table with the proper value (209).

You may find the ESCape code sequences very useful when you're programming (see Appendix I in the *System Guide* for a complete list). ESC-A turns on automatic insert mode, ESC-I inserts a blank line, ESC-Q erases to the end of a line, and so on. But you have to reach up to the top left corner to reach the ESC key, which is somewhat awkward. If you defined the *0* on the numeric keypad as ESC—CHR$(27)—and the other keys on the keypad as *A*, *I*, *Q*, and so on, the escape sequences will be easier to type.

If you use a modem in telecommunications, you probably know that certain control characters are common (CTRL-C, CTRL-P, CTRL-S, and CTRL-Q are a few of the important ones). You need two hands to type these keys, which is somewhat inconvenient. If you predefine some keys to print the control characters, it will simplify things a bit.

Another good use, mentioned above, is redefining the period on the numeric keypad to be a comma. This is helpful when you have to type in a lot of DATA statements. To make typing in DATA statements even easier, turn on automatic line numbering with the AUTO command and use the KEY statement to change one of the function keys to print DATA.

One final suggestion: The Apple IIc has a small toggle switch above the keyboard for changing the usual keyboard layout, which is most often called the QWERTY keyboard, to a Dvorak keyboard. (The name *QWERTY* comes from the layout of the first six letter keys on the "home" key row.) The woman who holds the title as the world's fastest typist prefers the Dvorak keyboard because it enables faster typing. If you'd like to experiment, you could redefine the CAPS LOCK translation table in the Dvorak layout and press CAPS LOCK to switch between the two.

KeyDef

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
HC 1Ø  IF PEEK(7166)=175 THEN 5Ø:ELSE POKE 7166,175
AF 2Ø  FAST:BANK15:FORI=Ø TO 5:K=I:IFI=4THENK=Ø:ELS
       EIFI=5THENK=4
QJ 3Ø  FORJ=Ø TO 88:POKE 5888+I*89+J,PEEK(64128+K*8
       9+J):NEXTJ,I
QF 4Ø  BANKØ:SLOW
EK 5Ø  FORI=ØTO9:K(I)=PEEK(4Ø96+I):POKE4Ø96+I,Ø:NEX
       T:TRAP 18Ø:DIMK$(87):FORI=ØTO87:READK$(I):NE
       XT
SF 6Ø  DATA INST/DEL,RETURN,CR RT/DN,F7,F1,F3,F5,CR
        DN/UP,3,W,A,4,Z,S,E,,5,R,D,6,C,F,T,X,7,Y,G,
       8,B,H,U,V,9,I,J,Ø,M,K,£,N,+,P,L,-,.,":",@,/
       ,O,*,;
EQ 7Ø  DATA CLR/HOME,,=,↑,/,1,←,,2,SPACE,,Q,RUN/STO
       P,HELP,8,5,TAB,2,4,7,1,ESC,+,-,LF,ENTER,6,9,
       3,,Ø,.,CR UP,CR DN,CR LT,CR RT
JF 75  DATA NO SCROLL
FS 8Ø  FORI=ØTO5:A=5888+I*89:POKE83Ø+I*2,AAND255:PO
       KE831+I*2,A/256:NEXT
FG 9Ø  IFRGR(Ø)=Ø THEN COLORØ,1:COLOR4,1:COLOR5,13:
       ELSE COLOR 6,1:COLOR 5,4:FAST
CE 1ØØ SCNCLR:PRINTCHR$(11);CHR$(14);"KEYDEF
SK 11Ø PRINT"{DOWN}PRESS KEY TO BE REDEFINED:";:DO
       :A=PEEK(213):LOOP UNTIL A<88:PRINTK$(A):POK
       E2Ø8,Ø
DS 12Ø PRINT"1> UNSHIFTED":PRINT"2> SHIFT":PRINT"3
       > COMMODORE":PRINT"4> CONTROL":PRINT"5> ALT
       ":PRINT"6> CAPS{SHIFT-SPACE}LOCK"
AX 13Ø PRINT"WHICH TABLE";:GETKEYT$:T=VAL(T$)-1:IF
       (T<Ø)OR(T>5)THENPRINT:GOTO12Ø
GH 14Ø PRINT"{DOWN}OLD ASCII CODE="PEEK(5888+T*89+
       A)
GB 15Ø INPUT"NEW ASCII CODE";C:IF C>255 THEN 15Ø:E
       LSE POKE5888+T*89+A,C
GE 16Ø INPUT"DO ANOTHER KEY{2 SPACES}Y{3 LEFT}";A$
       :IFA$="Y"THEN11Ø
FQ 17Ø FORI=ØTO9:POKE4Ø96+I,K(I):NEXT:END
FK 18Ø IF ERR=3Ø THEN RESUME:ELSE PRINT:PRINT"?"ER
       R$(ER)" ERROR IN"EL
```

Auto Run

Kevin Mykytyn

Here's a short utility for the 64 and 128 that makes loading and running programs a snap. List the disk directory, move to the program you want to load, and press RETURN. After the program is loaded, just press RETURN again to run it.

You'll find "Auto Run" quite simple to use. Once it has been installed, you can load any program by listing the directory, cursoring to the program, and pressing RETURN. If you've chosen a BASIC program, RUN appears on the screen after the program is loaded. If you've selected a machine language program, it loads, and then SYS and the appropriate starting address are printed on the screen. Either way, you just press RETURN to start up the program.

The Auto Run routine is written entirely in machine language, but a BASIC loader program is used to place the machine language into memory. The program is written to load at 3072. This area of memory is popular for machine language routines. If you own another utility program that needs the same section of memory, you can move Auto Run to a new location by changing the value of SA in line 10. The program requires 256 bytes of memory and must begin on an even-page boundary (a memory location evenly divisible by 256).

Good places to relocate it are the RS-232 buffers (at 3072 or 3328) or in memory between 4864 and 7167.

Running It

Once you've decided on a good place in memory and have set the value of SA accordingly, run the program. There will be a short delay while the machine language is POKEd into memory. The 40-column screen will go blank for a moment while the program puts the 128 into FAST mode. (This speeds up the POKEing to memory.)

You'll also be asked whether you want to load only BASIC programs or whether you might be loading BASIC and machine language programs. BASIC programs start at different addresses on the 128, depending on whether or not a hi-res graphics area has been allocated. Since the program uses the starting address of the program to determine whether it is BASIC or machine language, Auto Run can get confused. Therefore, anytime you won't need to load any machine language programs, choose the first option (BASIC only).

The program ends by printing the activation and deactivation addresses on the screen. Write these down for future reference. SYS to the address displayed, and Auto Run is activated.

Now display the directory on the screen: Type DIRECTORY, CATALOG, or just press the F3 function key.

Next, move the cursor up to the program you want to run and press RETURN. The program is loaded, the screen is cleared, and the word *RUN* or *SYS* followed by an address is printed at the top of the screen. It's important to check whether the SYS address displayed at the top of the screen is the actual starting address for the program loaded. Although most likely this is the address to SYS to, that's not always the case.

If any error occurs during the load, the appropriate error message is printed at the top of the screen. Once the program is loaded, you can simply press RETURN to run it or type LIST to view the program.

Auto Run should be compatible with most programming utilities. If you seem to have a memory conflict, just relocate Auto Run to another portion of memory.

Auto Run

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
DS 10 FAST:SA=3072:HB=INT(SA/256):LB=SA-256*HB
QA 20 C=0:Q=SA:FORA=1TO246:READB:C=C+B:IFB=500THEN
      POKEQ,HB:GOTO50
SJ 30 IFB<0THENPOKEQ,ABS(B):Q=Q+1:POKEQ,HB:GOTO50
RS 40 POKEQ,B
GG 50 Q=Q+1:NEXT:SLOW:IFC<>30994THENPRINT"{CLR}ERR
      OR IN DATA STATEMENTS":STOP
HB 60 PRINT"{CLR}{2 DOWN} 1. BASIC":PRINT"{DOWN} 2
      . BASIC AND ML":GETKEY A$:IF A$="1" THEN POK
      E SA+132,0:POKESA+157,76:POKESA+158,169:POKE
      SA+159,HB:GOTO 80
PX 70 IF A$="2" THEN 80:ELSE GOTO 60
XF 80 PRINT"{CLR}{2 DOWN} SYS"SA"TO ACTIVATE":PRIN
      T"{DOWN} SYS"SA+17"TO DEACTIVATE"
RA 180 DATA 162,1,189,4,3,157,-30
```

```
MX 190 DATA 202,16,247,162,32,160,500,208
DK 200 DATA 6,174,-30,172,-31,142
QF 210 DATA 4,3,140,5,3,96,0,0
JQ 220 DATA 162,255,232,189,0,2,201,48
PS 230 DATA 144,4,201,58,144,244,201,32
MK 240 DATA 240,240,201,34,240,3,108,-30
GM 250 DATA 169,0,141,0,255,232,134
SD 260 DATA 252,232,240,242,189,0,2,201
CR 270 DATA 34,208,246,138,56,229,252,166
BH 280 DATA 252,160,2,32,189,255,169,1
MA 290 DATA 162,8,160,0,32,186,255,169
AJ 300 DATA 0,170,32,104,255,32,192,255
RS 310 DATA 162,1,32,198,255,32,207,255
GH 320 DATA 133,253,32,207,255,133,254,169
BG 330 DATA 1,32,195,255,32,204,255,169
JH 340 DATA 1,162,8,160,1,32,186,255
RQ 350 DATA 169,0,166,45,164,46,32,213
KH 360 DATA 255,8,169,147,32,210,255,40
RF 370 DATA 176,55,32,79,79,165,253,197
XM 380 DATA 45,208,16,165,254,197,46,208
QM 390 DATA 10,32,125,255,82,85,78,0
BE 400 DATA 24,144,15,32,125,255,83,89
HK 410 DATA 83,32,0,166,253,165,254,32
CX 420 DATA 50,142,32,125,255,13,145,0
MG 430 DATA 32,204,255,32,250,81,108,2
EP 440 DATA 3,169,15,162,8,168,32,186
KS 450 DATA 255,169,0,32,189,255,32,192
HJ 460 DATA 255,162,15,32,198,255,32,207
GX 470 DATA 255,72,32,210,255,104,201,13
AB 480 DATA 208,244,169,15,32,195,255,24
AC 490 DATA 144,206
```

64 Mode Speed-Up

Gary Lamon

Once you get used to the 128's fast mode, 64 mode seems especially slow. This short program offers a way to speed things up significantly. For the 128 in 64 mode only.

The more you use a computer, the more you wonder if it couldn't be just a bit faster, especially when it's in the middle of a time-consuming task like alphabetizing a list of 800 names. If you own a Commodore 128, you can use the FAST command to double the speed of programs running in 80 columns. Although it also works in 40 columns, the screen goes blank. When you type GO 64, you give up access to the FAST command, but you don't have to give up fast mode. There are several interesting ways to squeeze more speed out of the Commodore 128's 64 mode. First, let's look at some background information.

Set the Pace

Every computer has an internal clock which paces the processor. The faster the clock's speed, the more instructions the computer can execute in a given time. A Commodore 64 contains a 6510 microprocessor with a clock speed of about 1 MHz (megahertz), one million cycles per second. On the other hand, the Commodore 128 uses an 8502 microprocessor that's compatible with the 6510, but can run at a speed of either 1 or 2 MHz. When you're using the 64 mode on your 128, the system automatically sets the speed of the 8502 so that the machine performs exactly like a Commodore 64.

It seems a waste that 128 users cannot make use of this additional speed when running their old 64 programs in 64 mode. But there is a way. You *can* double the computer's speed in 64 mode with a few simple POKEs:

POKE 53296,1	Double speed
POKE 53296,0	Normal speed
POKE 53296,3	Double speed and screen off

If you try the first or third of these POKEs in 64 mode, you'll indeed find that your programs run at twice the normal speed; but there's a problem. The screen fills with a flashing checkerboard pattern (if you use the first POKE) or goes completely blank (if you use the third). The regular screen is still there, but it cannot be read. The problem is that the 40-column video chip (the VIC-II) cannot keep up with the 8502 when the 8502 is running at 2 MHz. The third POKE works well on a 64 program that does, say, a great deal of number crunching. With this kind of program, it's probably not important to have video for that part of the program's execution.

There's another way to achieve a significant speed increase while retaining an almost normal picture. The program "64 Mode Speed-Up" provides approximately a 20 percent speed increase and leaves the screen readable. After typing in the program, save a copy. While in 64 mode, load and run the program, and then type NEW. Your machine will be 20 percent faster. To return to normal speed, type SYS 49236 or press RUN/STOP–RESTORE. To get back to fast speed, type SYS 49152. If you'd like to check this, write a short BASIC program with a large loop (such as: 10 FOR I = 1 TO 30000:NEXT), and time it to measure the speed increase. *(Note: You should return to regular speed before all disk or tape operations.)*

How It Works

You may have noticed a flash at the top of the screen while fast speed was in effect. This is normal. But what causes this flash and how does the program work?

The program works by using a machine language *raster interrupt* routine in locations 49152–49258. You can think of the raster as a sort of paintbrush that paints the picture on the video screen. The raster paints one line at a time across the screen, starting at the top left, and then moves down one line at a time. The program takes advantage of the fact that you can see only raster lines 51–251. The computer is interrupted when the raster is at line 251 (the last visible line) and told to speed up to 2 MHz. This speed is maintained until the raster reaches line 51 (the first line you can see) and is then reduced to 1 MHz. While the screen is "painted," the computer is running at regular speed. The result is a computer that runs faster, and you don't have to sacrifice the screen.

But what causes the flash at the top of the screen? Occasionally, the computer is performing a task and does not want to be interrupted quite yet, so a few raster lines are done at the 2 MHz speed. (Remember what happened when you typed POKE 53296,1?)

Two memory locations within the interrupt program can be used to speed up the system even more:

POKE 49257	Top raster line
POKE 49258	Bottom raster line

As an example, try this with the fast mode operating (after SYS 49152):

POKE 49257,150

You'll find that the flashing garbage will expand to fill the upper half of the screen, but the lower half will remain normal. The computer will now run about 1.6 times faster than a normal 64. You can expand or contract the screen any way you like with the two POKEs listed above. The more "garbage" visible, the faster the computer. One good way to visibly check the speed of the computer is to load a BASIC program and LIST it at the fast speed and at regular speed. The listing will scroll by considerably faster with the interrupt operating.

64 Mode Speed-Up

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
XB 10 PRINT"{CLR}{WHT}SPEED UP - 64 MODE ONLY"
KQ 20 FORI=49152TO49258:READX:C=C+X:POKEI,X:NEXT
BQ 30 IFC<>12470THENPRINT"DATA ERROR":END
ER 40 SYS 49152
XA 50 DATA 120,173,105,192,141,18,208,173
GX 60 DATA 17,208,41,127,141,17,208,169
RK 70 DATA 129,141,26,208,169,192,160,32
QM 80 DATA 141,21,3,140,20,3,88,96
EP 90 DATA 173,25,208,141,25,208,41,1
SQ 100 DATA 208,3,76,49,234,173,18,208
JA 110 DATA 205,106,192,176,14,172,106,192
MX 120 DATA 140,18,208,169,0,141,48,208
XX 130 DATA 76,78,192,172,105,192,140,18
HP 140 DATA 208,169,1,141,48,208,104,168
QD 150 DATA 104,170,104,64,120,169,234,141
MB 160 DATA 21,3,169,49,141,20,3,169
EP 170 DATA 0,141,48,208,141,26,208,88
FS 180 DATA 96,50,250
```

Boot 64

Mike Tranchemontagne

Most Commodore 128 owners know that their computer can automatically load and run any 128 program from disk. This easy-to-use program adds the same convenience for Commodore 64 programs as well, allowing the 128 to load and run any 64 program automatically when you boot the system. A disk drive is required.

The Commodore 128 has many outstanding features, not the least of which is its ability to run thousands of excellent Commodore 64 programs and games. The 128 can automatically load and run any program written for 128 mode or CP/M mode. Although there are programs for the 64 that automatically run after loading from disk, it's still necessary to type in a command like LOAD "PROGRAM",8,1 to activate the disk drive in 64 mode. "Boot 64" automates this process so that you can load and run any Commodore 64 program simply by putting the disk in the drive and turning on the computer. This feature is ideal for younger members of the family or for infrequent computer users. Even experienced programmers will appreciate the extra convenience it affords.

Creating an Autoboot Disk

Type in Programs 1, 2, and 3, and save copies of all three programs. In order for the boot sector created by Program 2 to work properly, you must use the filename 128BOOT64 when saving Program 1. To create an autobooting disk for 64 mode, follow these three steps:

1. Select the disk which will contain the 64 program you want to load and run automatically. Load Program 2, insert the disk in the drive, and run the program. When Program 2 is finished, the disk will contain a 128 boot sector that will cause the computer to load and run a program named 128BOOT64. (You do not need to save Program 2 on the target disk.)
2. Load Program 1 and save it on the disk. Remember, you *must* save this program with the filename 128BOOT64.

3. Load the 64 program that you want to load and run automatically; then save it on the disk using the filename BOOT64. You *must* save the program with this filename.

Once you've performed all three steps, place the disk in the drive and reboot by turning the power off and on or by pressing the RESET switch. If the computer does not load and run the desired program, check Programs 1 and 2 for typing errors and repeat the process. Keep in mind that the process won't work unless you use the filenames noted above.

Autobooting Machine Language Programs

With this technique, you can load and run any Commodore 64 BASIC program. The same is true of any machine language (ML) program that runs like BASIC. For instance, *SpeedScript,* COMPUTE!'s word processor, ordinarily starts with LOAD"SPEEDSCRIPT",8 and RUN. To autoboot and run *SpeedScript,* simply save *SpeedScript* to disk with the filename BOOT64 as described in Step 3.

You can also autoboot and start a machine language program that normally loads with ,8,1 and starts with SYS instead of RUN. Program 3 is a very short BASIC loader which loads an ML program into memory, then activates it with SYS. As listed, the program loads and starts DOS 5.1, the "DOS Wedge" program supplied on the 1541/1571 Test/Demo disk. To load a different ML program, replace the name DOS 5.1 in line 20 with the filename of your program, and replace the address 52224 in line 30 with the correct SYS address for the program. When that's done, perform steps 1 and 2 as described earlier; then save Program 3 on the disk with the filename BOOT64. Of course, you must also copy the ML program to the same disk, using the filename you specified in line 20 of Program 3.

How Autobooting Works

When you turn on the 128 (or reboot by pressing the RESET button), the computer automatically performs several checks to determine which mode it will operate in. If an autostart cartridge is plugged into the cartridge port, the cartridge takes control. If the Commodore key is pressed, the computer enters 64 mode. If the RUN/STOP key is pressed, the 128 enters the built-in machine language monitor.

If none of these conditions applies, the 128 looks on sector 0, track 1, of the current disk (known as the *boot sector*) to see whether it contains a boot header. If no boot header is found, the computer simply starts BASIC, which produces the familiar READY prompt. However, if the boot header information is present, the 128 automatically loads and runs the program indicated in the boot sector. This process works equally well with a 1571 or 1541 disk drive.

In 128 mode, the 128 can switch to 64 mode by performing the command GO64. However, there is no provision for loading and running a program after you enter 64 mode. To achieve the same effect, this program creates a boot sector that tells the computer to load and run the program 128BOOT64. That program, in turn, stores a short machine language program and cartridge-identifier bytes in the special memory area where Commodore 64 autostarting cartridges normally reside. The ML program causes the computer (now in 64 mode) to perform a normal reset. When the reset occurs, the computer detects the cartridge-identifier bytes, concludes that a cartridge is present, and runs the ML routine found at the cartridge start address. This program, in turn, uses the dynamic-keyboard technique to load and run a program named BOOT64 from disk. The process may seem complicated, but it all happens very quickly, and you need not understand the details in order to take advantage of it.

Program 1. 128BOOT64

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in these programs.

```
EP 10 A=32768: PRINT "(SWITCH TO 40 COLUMN DISPLAY
      )"
XK 20 READ D$: IF D$="-1" THEN GO64
HR 30 POKE A,DEC(D$):A=A+1: GOTO 20
PH 40 DATA 09,80,5E,FE,C3,C2,CD,38,30
HM 50 DATA 8E,16,D0,20,A3,FD,20,50,FD
QX 60 DATA 20,15,FD,20,5B,FF,58
QH 70 DATA 20,53,E4,20,BF,E3,20,22,E4
CQ 80 DATA A2,FB,9A
PH 90 DATA A2,00,BD,41,80,F0,06
AK 100 DATA 20,D2,FF,E8,D0,F5
HA 110 DATA A9,0D,8D,77,02,8D,78,02
FG 120 DATA A9,02,85,C6
JA 130 DATA 4C,74,A4
```

```
BR 140 DATA ØD,4C,4F,41,44,22,42,4F,4F,54,36,34,22
       ,2C,38
BQ 150 DATA ØD,ØD,ØD,ØD,ØD,52,55,4E,91,91,91,91,91
       ,91,91,Ø,-1
```

Program 2. Boot Sector Maker

```
RJ 1Ø REM PROGRAM 2, CREATE BOOT SECTOR FOR 128BOO
      T64
JF 2Ø DCLEAR: OPEN 15,8,15: OPEN 2,8,2,"#": PRINT#
       15,"B-P:2,Ø"
RR 3Ø READ D$: D=DEC(D$): IF D>255 THEN 5Ø
EE 4Ø PRINT# 2,CHR$(D);: GOTO 3Ø
RJ 5Ø PRINT# 15,"U2;2,Ø,1,Ø"
SP 6Ø PRINT DS$: CLOSE 2: CLOSE 15
XG 7Ø DATA 43,42,4D,ØØ,ØØ,ØØ,ØØ,31,32,38,42,4F,4F,
      54,36,34,ØØ,ØØ,A2,18
RM 8Ø DATA AØ,ØB,4C,A5,AF,52,55,4E,22,31,32,38,42,
      4F,4F,54,36,34,ØØ,1ØØ
```

Program 3. ML Loader

```
PM 1Ø REM C64 ML PROG LOADER EXAMPLE
KM 2Ø IF A=Ø THEN A=1: LOAD "DOS 5.1",8,1
QE 3Ø SA=52224: REM START ADDRESS
KH 4Ø SYS SA
```

Personalizing the 128

Steve Stanko

Have you ever wished you could change the screen colors on your 128? Of course, there are BASIC commands to do this, but wouldn't it be nice to have the machine default to your colors each time you power up? How about having cursor mode, tab settings, and key repeats set up automatically? Want a favorite utility to load and run each time you turn on the computer? These programs can give your computer a whole new personality. For use with a 1541 or 1571 disk drive.

How can you change the default settings of the 128? The routine that specifies screen colors, tab settings, and the like, when the computer is powered up or reset is in ROM—memory that can't be modified. However, you may have noticed that whenever the 128 is turned on or reset while the disk drive is on, it tries to read from a disk. If there is a disk in the drive, the 128 checks the sectors of the disk reserved for boot information. The boot information can be a short machine language routine, a command to load another program, or both. The key to customizing the 128 is to create an autobooting disk with a program that sets things up the way you like.

Program 1, "Autoboot Generator," creates the autobooting disk. Simply type in the program and save a copy of it on a disk you don't plan to use for autobooting. (Deleting the REMs in Program 1 won't hurt the program and will save a lot of space and typing time.) After you've saved a copy, load the program and type RUN. You'll be asked to insert the disk on which you want the autoboot information. The boot code must be written to a newly formatted disk; otherwise, the boot sector may use a place on the disk that other programs already occupy.

If you haven't formatted the disk, the program offers to do that for you. (Before you format the disk, *be sure that it doesn't contain any programs you want to save.*) If you choose this option, be prepared for a slight delay while the disk is formatted. The boot information will then be written to the proper sectors. If a successful transfer occurs, a message will appear telling you so. You now have an autobooting disk.

But this alone is not enough to make the computer default to your own personal preferences. What the autoboot information does is attempt to load and run a BASIC program named "Preferences" from the same disk. This program should set the values for the features you want to change. Program 2 is a short example of the sort of preference-selection program you might write. To use Program 2, type in a copy and save it to the autoboot disk you prepared with Program 1. For the technique to work correctly, you must save Program 2 with the filename PREFERENCES.

Have It Your Way

Program 2 selects a 40-column text screen. If you prefer to start out with the 80-column display active, simply change the GRAPHIC 0 statement in line 10 to GRAPHIC 5. In fact, you can select any one of six different screen modes by changing the statement to GRAPHIC *mode*, where *mode* has one of the values shown in Table 1.

Table 1. Screen Modes

Mode	Screen
0	40-column text
1	Bitmapped graphics (40-column)
2	Split bitmapped graphics and text (40-column)
3	Multicolor bitmapped graphics (40-column)
4	Split multicolor bitmapped graphics and text (40-column)
5	80-column text

If you want the selected screen to be cleared, change the number following the SCNCLR statement in line 10 to the same value as the screen mode.

Colors

The COLOR statement can be used to specify new screen background, border, and character colors. Line 20 of Program 2 selects a light-gray background and a medium-gray border. The default character color will be dark gray. (That combination of colors shows up most clearly on my monochrome monitor.) To choose other colors, simply modify the COLOR statements. The statements have the format

COLOR *color source,color number*

Valid values for *color source* are given in Table 2. Table 3 shows valid *color number* values for the 40-column modes. The numbers are the same for the 80-column screen, but the colors are slightly different (see Table 4).

Table 2. Color Sources

Number	Source
0	40-column background
1	40-column foreground
2	Multicolor 1
3	Multicolor 2
4	40-column border
5	Character color (40- and 80-column)
6	80-column background

Table 3. 40-Column Colors

Number	Color	Number	Color
1	Black	9	Orange
2	White	10	Brown
3	Red	11	Light red
4	Cyan	12	Dark gray
5	Purple	13	Medium gray
6	Green	14	Light green
7	Blue	15	Light blue
8	Yellow	16	Light gray

Table 4. 80-Column Colors

Number	Color	Number	Color
1	Black	9	Dark purple
2	White	10	Dark yellow
3	Dark red	11	Light red
4	Light cyan	12	Dark cyan
5	Light purple	13	Medium gray
6	Dark green	14	Light green
7	Dark blue	15	Light blue
8	Light yellow	16	Light gray

Further Enhancements

Your Preferences program can change more than just screen colors. There are many other user-selectable options such as tab stops, repeating keys, and cursor modes. Many of these are controlled by printing ESC (escape) sequences. An escape sequence is composed of an ESC character followed by an alphabetic character (or @). Table 5 lists the 128's escape sequences. Note that when you use escape sequences, you should first press the ESC key, then release it and press the other key. Unlike when you're using control characters where you hold down one key (CONTROL, Commodore, or SHIFT) while pressing another, you *should not* hold down the ESC key. (For more about ESC sequences, see "ESCaping with the 128," elsewhere in this book.)

Table 5. Escape Sequences

Function	ESC Sequence
Clear to end of screen	ESC @
Enable auto-insert mode	ESC A
Set bottom right corner of window	ESC B
Disable auto-insert mode	ESC C
Delete current line	ESC D
Switch to solid (nonblinking) cursor	ESC E
Switch to flashing cursor	ESC F
Enable bell tone when CHR$(7) is printed	ESC G
Disable bell tone	ESC H
Insert a blank line	ESC I
Move cursor to start of current line	ESC J
Move cursor to end of current line	ESC K
Enable screen scrolling	ESC L
Disable screen scrolling	ESC M
Set 80-column screen to normal video	ESC N
Cancel quote, insert, and reverse modes	ESC O
Erase to start of current line	ESC P
Erase to end of current line	ESC Q
Set 80-column screen to reverse video	ESC R
Switch to block cursor (80-column)	ESC S
Set top left corner of window	ESC T
Switch to underline cursor (80-column)	ESC U
Scroll screen up one line	ESC V
Scroll screen down one line	ESC W
Switch between 40- and 80-column displays	ESC X
Set default tab stops (every 8 columns)	ESC Y
Clear all tab stops	ESC Z

In immediate mode, simply press and release the ESC key; then press the letter key for the function you want. To use these sequences in a BASIC program such as Preferences, print an ESC character, CHR$(27), followed by the letter for the desired function. Line 30 of Program 2 uses this method to change the cursor to a solid, nonblinking block. You can add any other ESC sequences you want to your own Preferences program to control the various options.

However, there is no ESC sequence for many of the features you might want to customize. For instance, there is no ESC sequence to specify which keys repeat. In this case, the operation is usually controlled by the value in a *flag* location in RAM. For example, location 2594 controls key repeating. The default value is 128, which allows all keys but the special function keys to repeat if they're held down. POKEing the value 64 into this location will prevent any keys from repeating. As a former Commodore 64 owner, my preference is to have only the space bar, INST/DEL key, and cursor keys repeat. Storing a value of zero in location 2594 allows this. Line 40 in the example serves this purpose. There are many other features you can control by POKEing values into other locations. Refer to a detailed 128 memory map for more information.

Tab Stops

Another important option is tab settings. Although default stops are set at every eighth column, you can change this to any pattern you desire—even an unusual sequence of stops such as at columns 5, 7, 19, 26, 27, and 39. You can set or clear tab stops indirectly by printing CHR$(24), but it's often easier to change the stops directly. Tab-stop settings are controlled by the values in the tab-stop bitmap, locations 852–861. Each bit in the map corresponds to a column on the screen. You change the tab settings by POKEing new values into the bitmap. To determine the numbers you want, let's look at the default 40-column settings:

Location	**Value**
852	128
853	128
854	128
855	128
856	128

The bitmap makes more sense if you look as these values in binary format:

1000000010000000100000001000000010000000

Bits set to 1 represent tab stops. In Program 2, tab stops are set every ten spaces, which yields the following pattern:

0000000001000000000100000000010000000001

Or, if you group it into bytes:

00000000 01000000 00010000 00000100 00000001

When you translate the bit patterns back into decimal, the values to be placed in the tab-stop bitmap locations are

0, 64, 16, 4, 1

Lines 50–60 of Program 2 set these tab stops. Tabs for an 80-column screen are just the same, only you need five more bytes to map out the full 80-columns.

Your Preferences program can load other utilities or a favorite program that you use every time you turn on the computer. To load and execute another BASIC program, all you need to do is add a line with the statement RUN *"filename"*; or, if you want to retain the variables created in the Preferences program, add DLOAD *"filename"*. For machine language programs, use BLOAD *"filename"* followed by the SYS command to start the machine language, or use BOOT *"filename"* to have the program start automatically after loading.

You're no longer stuck with the decisions the 128's designers made. By autobooting this program, you can turn the 128 into a truly personal computer.

Custom Autoboots

Program 1 creates an autoboot disk which attempts to load a BASIC program named Preferences. Then it executes the program by placing the characters RUN and a RETURN—CHR$(13)—into the keyboard buffer by using a short machine language program following the filename data on an autobooting disk. It's quite simple to modify Program 1 to create an autoboot disk for any other BASIC program you want. For a disk to be autobooted by the 128, sector 0 of track 1 of the disk must be set up as follows:

Bytes	Description
0–2	The characters *CBM,* the autoboot flag
3–4	Load address for program to autoboot, in low-byte/high-byte order (both bytes 0 for a BASIC program)
5	Bank number into which program should be loaded (0 for a BASIC program)
6	Number of additional sectors to load (0 for a BASIC program)

In Program 1, these values are contained in the DATA items of lines 1000–1010.

Then, starting at byte 7 of the sector, come the character codes for the name to be displayed after the BOOTING message, followed by a zero as delimiter. Program 1 uses PREFERENCES, since that's the name of the BASIC program being booted. However, there's no real requirement for using the actual filename; these characters are used only for the message, so you could substitute any other message you like by changing the DATA in line 1020 of Program 1. Just remember that the character string must end with a zero (see line 1030).

Following the zero byte come the character codes for the filename to load, also followed by a zero byte. In this case, the character codes are important: They must correspond exactly to the name of the program you wish to boot from the disk. Program 1 uses the name PREFERENCES. To change this to another program name, modify the DATA in line 1040. Again, remember that the filename characters must be followed by a zero (see line 1050).

A short machine language routine can follow the filename; it will be executed after the specified program has been loaded. Since BASIC programs like Preferences will not run automatically after they're loaded, Program 1 adds a routine that puts four characters into the keyboard buffer: R, U, N, and RETURN. After this, a value of 4 is put into the location which holds the number of characters in the keyboard buffer. This causes the computer to act as if you had typed RUN and pressed RETURN. In Program 1, the routine is contained in the DATA statements in lines 1060–1100.

To write this boot information to the disk, Program 1 frees sector 0 of track 1 so that data can be written to that block (line 220). It then opens a buffer for the sector data (line 230) and resets the buffer pointer (line 240). Lines 250–270 transfer the boot information from the DATA statements (lines 1000–1100) to the buffer. When there is no more data, the error trapping sends the program to line 280. If an OUT OF DATA error has occurred, the program continues; otherwise, the program prints the type of error and halts. Line 290 pads the rest of the boot sector with zeros. The disk command U2 (line 300) writes the sector buffer to the disk. Line 310 closes the disk buffer, and line 320 marks sector 0 of track 1 to make sure that the autoboot data isn't accidentally overwritten by a program.

Program 1. Autoboot Generator

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
AJ 100 TRAP 280
JB 110 COLOR 4,12:COLOR 0,12:PRINT"{CLR}{DOWN}
       {YEL}PLEASE INSERT DISKETTE FOR AUTOBOOT"
EC 120 PRINT"PREFERENCES TO BE WRITTEN ON. IT MUST
       "
AE 130 PRINT"BE FORMATTED TO WORK. IF IT IS NOT,"
KM 140 PRINT"ANSWER YES TO THE PROMPT BELOW."
PK 150 INPUT"{DOWN}FORMAT DISK";AN$
PC 160 IF LEFT$(AN$,1)<>"Y" THEN 210
MG 170 PRINT"{DOWN}DISK NAME (UP TO 16 CHARACTERS)
       ":INPUT DN$
BM 180 INPUT"{DOWN}DISK ID (TWO CHARACTERS)";DI$
KB 190 PRINT"{DOWN}INSERT DISK AND PRESS ANY KEY":
       GETKEY DL$
BM 200 OPEN 15,8,15,"N0:"+DN$+","+DI$:GOTO 220
RG 210 OPEN 15,8,15
QR 220 PRINT#15,"B-F:0,1,0":REM UN-WRITE-PROTECT T
       RACK 1 SECTOR 0
BG 230 OPEN 5,8,5,"#":REM OPEN A BUFFER
JS 240 PRINT#15,"B-P:5,0":CN=0:REM SET BLOCK POINT
       ER TO START OF DATA
FE 250 READ D:CN=CN+1
CR 260 PRINT#5,CHR$(D);:REM WRITE DATA TO BUFFER
RF 270 GOTO 250
GE 280 IF ER=13 THEN RESUME 290:ELSE PRINT ERR$(ER
       ):STOP
JR 290 FOR C=1 TO (255-CN):PRINT#5,CHR$(0);:NEXT C
       :REM SET REST OF BLOCK TO 0
DP 300 PRINT#15,"U2:5,0,1,0":REM WRITE AUTOBOOT DA
       TA TO DISK
MA 310 CLOSE 5
SB 320 PRINT#15,"B-A:0,1,0":REM WRITE-PROTECT TRAC
       K 1 SECTOR 0
```

```
CA 330 CLOSE 15
XA 340 PRINT"{CLR}{DOWN}AUTOBOOT PREFERENCES CODE
       {SPACE}TRANSFERRED.."
JJ 350 PRINT"IN ORDER FOR THE DISK TO BOOT YOUR"
JS 360 PRINT"PREFERENCES, YOU MUST HAVE A COPY OF"
BR 370 PRINT"THE PROGRAM 'PREFERENCES' ON THIS DIS
       K."
HP 380 PRINT"{DOWN} -REFER TO THE ARTICLE FOR DETA
       ILS-"
HS 390 A$=DS$:END
GM 1000 DATA 67,66,77:REM 'CBM'
SF 1010 DATA 0,0,0,0:REM NULLS
DR 1020 DATA 80,82,69,70,69,82,69,78,67,69,83:REM
        {SPACE}MESSAGE FOR BOOT 'PREFERENCES'
JH 1030 DATA 0:REM DELIMITER
DB 1040 DATA 80,82,69,70,69,82,69,78,67,69,83:REM
        {SPACE}BASIC PROGRAM 'PREFERENCES'
GA 1050 DATA 0:REM DELIMITER
FX 1060 DATA 169,82,141,74,3:REM 'R'
KC 1070 DATA 169,85,141,75,3:REM 'U'
QR 1080 DATA 169,78,141,76,3:REM 'N'
AD 1090 DATA 169,13,141,77,3:REM (RETURN)
DK 1100 DATA 169,4,133,208,96:REM SET KEYBOARD BUF
        FER TO 4 CHARS AND EXIT
```

Program 2. Preferences

For mistake-proof program entry, use "The Automatic Proofreader," Appendix B, to type in this program.

```
KQ 10 GRAPHIC 0:SCNCLR 0
MM 20 COLOR 0,16:COLOR 5,12:COLOR 4,13
PS 30 PRINT CHR$(27);"E":REM DISABLE BLINK
DJ 40 POKE 2594,0:REM DISABLE REPEAT
FK 50 POKE 852,0:POKE 853,64:POKE 854,16
KR 60 POKE 855,4:POKE 856,1:REM TAB BITS
QA 70 NEW:CLR
```

Appendix A

How to Type In Programs

To make it as easy as possible for you to enter the programs in this book, we have included two program-entry aids written in BASIC: "The Automatic Proofreader" and "MLX," a machine language entry program. You'll find them in Appendices B and C. COMPUTE! Publications has established the following listing conventions to assist you in understanding how to enter these programs.

Generally, BASIC program listings like the one for MLX contain words within braces that spell out any special characters. For example, {DOWN} means press the cursor-down key; {5 SPACES} means press the space bar five times.

In our listings, an underlined key indicates that the key should be shifted (press the key while holding down the SHIFT key). For example, <u>S</u> means to press the S key while holding down the SHIFT key. This will appear on your screen as a heart symbol. If you find an underlined key enclosed in braces, for example, {10 <u>N</u>}, you should press the key as many times as indicated. In this case, you would enter ten shifted *N*'s.

If a key is enclosed within special brackets, [<>], 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, press the key as many times as indicated: [<9@>] means that you should type Commodore-@ nine times.

Refer to Figure A-1 on the next page when entering cursor and color control keys.

Figure A-1. Keyboard Conventions

When You Read:	Press:		See:
{CLR}	SHIFT	CLR/HOME	▣
{HOME}		CLR/HOME	▣
{UP}	SHIFT	↑ CRSR ↓	▣
{DOWN}		↑ CRSR ↓	▣
{LEFT}	SHIFT	← CRSR →	▣
{RIGHT}		← CRSR →	▣
{RVS}	CTRL	9	▣
{OFF}	CTRL	0	▣
{BLK}	CTRL	1	▣
{WHT}	CTRL	2	▣
{RED}	CTRL	3	▣
{CYN}	CTRL	4	▣
{PUR}	CTRL	5	▣
{GRN}	CTRL	6	▣
{BLU}	CTRL	7	▣
{YEL}	CTRL	8	▣

When You Read:	Press:		See:
{1}	COMMODORE	1	▣
{2}	COMMODORE	2	▣
{3}	COMMODORE	3	▣
{4}	COMMODORE	4	▣
{5}	COMMODORE	5	▣
{6}	COMMODORE	6	▣
{7}	COMMODORE	7	▣
{8}	COMMODORE	8	▣
{ F1 }		f1	▣
{ F2 }	SHIFT	f1	▣
{ F3 }		f3	▣
{ F4 }	SHIFT	f3	▣
{ F5 }		f5	▣
{ F6 }	SHIFT	f5	▣
{ F7 }		f7	▣
{ F8 }	SHIFT	f7	▣
←		←	▣
$\underline{\uparrow}$	SHIFT	↑	▣

Appendix B

The Automatic Proofreader

Philip I. Nelson

"The Automatic Proofreader" helps you type in program listings without typing mistakes. It's a short error-checking program that conceals itself in memory and adheres to your Commodore's operating system. Each time you press RETURN to enter a program line, the Proofreader displays a two-letter checksum in reverse video at the top of your screen. If the checksum on your screen doesn't match the one in the printed listing, you've typed the line incorrectly. It's that simple. You don't have to use the Proofreader to enter printed listings, but doing so greatly reduces your chances of making a typo.

Getting Started

First, type in the Automatic Proofreader program *exactly* as it appears in the listing. Since the Proofreader can't check itself, type carefully to avoid mistakes. Don't omit any lines, even if they contain unfamiliar commands, or you think they don't apply to your computer. As soon as you've finished typing the Proofreader, save at least two copies on disk or tape before running it the first time. This is very important because the Proofreader erases the BASIC portion of itself when you run it, leaving only the machine language (ML) portion in memory.

When that's done, type RUN and press RETURN. After announcing which computer it's running on, the Proofreader installs the ML routine in memory, displays the message PROOFREADER ACTIVE, erases the BASIC portion of itself, and ends. If you type LIST and press RETURN, you'll see that no BASIC program remains in memory. The computer is ready for you to type in a new BASIC program.

Entering Programs

Once the Proofreader is active, you can begin typing in a BASIC program as usual. Every time you finish typing a line and press RETURN, the Proofreader displays a two-letter checksum (reverse-video letters) in the upper left corner of the screen. Compare this checksum with the two-letter checksum printed to the left of the corresponding line in the program listing. If the letters match, it's almost certain the line was typed correctly. If the letters don't match, check for your mistake and correct the line.

The Proofreader ignores spaces that aren't enclosed in quotation marks, so you can omit spaces (or add extra ones) between keywords and still see a matching checksum. For example, these two lines generate the same checksum:

```
10 PRINT"THIS IS BASIC"
10 PRINT           "THIS IS BASIC"
```

However, since spaces inside quotation marks are almost always significant, the Proofreader pays attention to them. For instance, these two lines generate different checksums:

```
10 PRINT"THIS IS BASIC"
10 PRINT"THIS ISBA       SIC"
```

Out of Order

A common typing mistake is transposition—typing two successive characters in the wrong order, like PIRNT instead of PRINT or 64378 instead of 64738. A checksum program that adds up the values of all the characters in a line can't possibly detect transposition errors (it can only tell whether the right characters are present, regardless of what order they're in). Because the Proofreader computes the checksum with a more sophisticated formula, it is also sensitive to the *position* of each character within the line and thus catches transposition errors.

The Proofreader does *not* accept keyword abbreviations (for example, ? instead of PRINT). If you prefer to use abbreviations, you can still check a line with the Proofreader: Simply LIST the line after you type it, move the cursor back onto the line, and press RETURN. LISTing the line substitutes the full keyword for the abbreviation and allows the Proofreader to work properly. The same technique works for rechecking a program you've already typed in: Reload the program, LIST several lines on the screen, and press RETURN at the end of each line.

If you're using the Proofreader on the Commodore 128 in 128 mode, *do not perform any GRAPHIC statements while the Proofreader is active.* When you perform a statement like GRAPHIC 1, the computer moves everything at the start of BASIC program space—including the Proofreader—to another memory area, causing the Proofreader to crash. The same thing happens if you run any program that contains a GRAPHIC statement. The Proofreader deallocates any graphics areas before installing itself in memory, but you are responsible for seeing that the computer remains in this configuration.

Though the Proofreader doesn't interfere with other BASIC operations, it's always a good idea to disable it before running any other program. Some programs may need the space occupied by the Proofreader's ML routine or may create other memory conflicts. However, the Proofreader is purposely made difficult to dislodge: It's not affected by tape or disk operations, or by pressing RUN/STOP–RESTORE. The simplest way to disable it is to turn the computer off, then on again. A gentler method is to SYS to the computer's built-in reset routine: Use SYS 65341. This reset routine erases the BASIC program currently in memory, so be sure to save the program you're typing before entering the SYS command.

The Automatic Proofreader

```
10 VEC=PEEK(772)+256*PEEK(773):LO=43:HI=44
20 PRINT "AUTOMATIC PROOFREADER FOR ";:IF VEC=4236
   4 THEN PRINT "C-64"
30 IF VEC=50556 THEN PRINT "VIC-20"
40 IF VEC=35158 THEN GRAPHIC CLR:PRINT "PLUS/4 & 1
   6"
50 IF VEC=17165 THEN LO=45:HI=46:GRAPHIC CLR:PRINT
   "128"
60 SA=(PEEK(LO)+256*PEEK(HI))+6:ADR=SA
70 FOR J=0 TO 166:READ BYT:POKE ADR,BYT:ADR=ADR+1:
   CHK=CHK+BYT:NEXT
80 IF CHK<>20570 THEN PRINT "*ERROR* CHECK TYPING
   {SPACE}IN DATA STATEMENTS":END
90 FOR J=1 TO 5:READ RF,LF,HF:RS=SA+RF:HB=INT(RS/2
   56):LB=RS-(256*HB)
100 CHK=CHK+RF+LF+HF:POKE SA+LF,LB:POKE SA+HF,HB:N
   EXT
110 IF CHK<>22054 THEN PRINT "*ERROR* RELOAD PROGR
   AM AND CHECK FINAL LINE":END
120 POKE SA+149,PEEK(772):POKE SA+150,PEEK(773)
```

```
13Ø IF VEC=17165 THEN POKE SA+14,22:POKE SA+18,23:
   POKESA+29,224:POKESA+139,224
14Ø PRINT CHR$(147);CHR$(17);"PROOFREADER ACTIVE":
   SYS SA
15Ø POKE HI,PEEK(HI)+1:POKE (PEEK(LO)+256*PEEK(HI)
   )-1,Ø:NEW
16Ø DATA 12Ø,169,73,141,4,3,169,3,141,5,3
17Ø DATA 88,96,165,2Ø,133,167,165,21,133,168,169
18Ø DATA Ø,141,Ø,255,162,31,181,199,157,227,3
19Ø DATA 2Ø2,16,248,169,19,32,21Ø,255,169,18,32
2ØØ DATA 21Ø,255,16Ø,Ø,132,18Ø,132,176,136,23Ø,18Ø
21Ø DATA 2ØØ,185,Ø,2,24Ø,46,2Ø1,34,2Ø8,8,72
22Ø DATA 165,176,73,255,133,176,1Ø4,72,2Ø1,32,2Ø8
23Ø DATA 7,165,176,2Ø8,3,1Ø4,2Ø8,226,1Ø4,166,18Ø
24Ø DATA 24,165,167,121,Ø,2,133,167,165,168,1Ø5
25Ø DATA Ø,133,168,2Ø2,2Ø8,239,24Ø,2Ø2,165,167,69
26Ø DATA 168,72,41,15,168,185,211,3,32,21Ø,255
27Ø DATA 1Ø4,74,74,74,74,168,185,211,3,32,21Ø
28Ø DATA 255,162,31,189,227,3,149,199,2Ø2,16,248
29Ø DATA 169,146,32,21Ø,255,76,86,137,65,66,67
3ØØ DATA 68,69,7Ø,71,72,74,75,77,8Ø,81,82,83,88
31Ø DATA 13,2,7,167,31,32,151,116,117,151,128,129,
   167,136,137
```

Appendix C

MLX Machine Language Entry Program

128 Version by Ottis R. Cowper

"MLX" is a new way to enter long machine language (ML) programs without a lot of fuss. MLX lets you enter the numbers from a special list that looks similar to BASIC DATA statements. It checks your typing on a line-by-line basis. It won't let you enter invalid characters or let you continue if there's a mistake in a line. It won't even let you enter a line or digit out of sequence.

Using MLX

Type in and save some copies of MLX (you'll want to use it to enter future ML programs from COMPUTE! Publications). When you're ready to enter an ML program, load and run MLX. It will ask you for a starting address and an ending address. You'll find these addresses in the article accompanying the MLX-format program listing that you're typing.

If you're unfamiliar with machine language, the addresses (and all other values you enter in MLX) may appear strange. Instead of the usual decimal numbers you're accustomed to, these numbers are in *hexadecimal*—a base-16 numbering system commonly used by ML programmers. Hexadecimal (hex for short) includes the numbers 0–9 and the letters *A–F.* But don't worry; even if you know nothing about ML or hex, you should have no trouble using MLX.

After you enter the starting and ending addresses, MLX will offer you the option of clearing the workspace. Choose this option if you're starting to enter a new listing. If you're continuing a listing that's partially typed from a previous session, don't choose this option.

It's not necessary to know more about this option to use MLX, but here's an explanation if you're interested. When you first run MLX, the workspace area contains random values.

Clearing the workspace fills it with zeros. This makes it easier to find where you left off if you enter the listing in multiple sittings. However, clearing the workspace is useful only before you first begin entering a listing; there's no need to clear it before you reload to continue entering a partially typed listing.

When you save your work with MLX, the entire contents of the data buffer is stored. If you clear the workspace before starting, the incomplete portion of the listing will be filled with zeros when saved and thus refilled with zeros when reloaded. If you don't clear the workspace when you first start, the incomplete portion of the listing will be filled with random data. Whether or not you clear the workspace before you reload, this random data will refill the unfinished part of the listing when you load your previous work. The rule, then, is to use the clear-workspace feature before you begin entering data from a listing and not to bother with it afterward.

At this point, MLX presents a menu of commands:

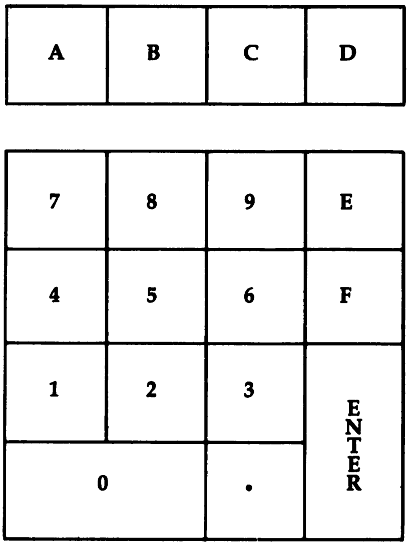
Enter data
Display data
Load file
Save file
Catalog disk
Quit

Entering a Listing

To begin entering data, press E. You'll be asked for the address at which you wish to begin entering data. (If you press E by mistake, you can return to the command menu by pressing RETURN.) When you begin typing a listing, you should enter the starting address here. If you're typing in a long listing in multiple sittings, you should enter the address where you left off typing at the end of the previous session. In any case, make sure the address you enter corresponds to the address of a line in the MLX listing. Otherwise, you'll be unable to enter the data correctly.

After you enter the address, you'll see that address appear as a prompt with a nonblinking cursor. Now you're ready to enter data. Type in all nine numbers on the line, beginning with the first two-digit number after the colon (:). Each line represents eight data bytes and a checksum. Although an MLX-format listing appears similar to the "hex dump" machine language listings you may be accustomed to, the extra checksum number on the end allows MLX to check your typing. (You *can* enter the data from an MLX listing using the built-in monitor if the rightmost column of data is omitted, but we recommend against it. It's much easier to let MLX do the proofreading and error checking for you.)

Only the numbers 0–9 and the letters *A–F* can be typed in. If you press any other key (with some exceptions noted below), you'll hear a warning buzz. To simplify typing, MLX redefines the function keys and the + and − keys on the numeric keypad so that you can enter data one-handed. The figure shows the keypad configuration supported by MLX:



MLX checks for transposed characters. If you're supposed to type in *A0* and instead enter *0A*, MLX will catch your mistake. To correct typing mistakes before you finish a line, use the INST/DEL key to delete the character to the left of the cursor. (The cursor-left key also deletes.) If you mess up a line really badly, press CLR/HOME to start the line over.

The RETURN key is also active, but only before any data is typed on a line. Pressing RETURN at this point returns you to the command menu. After you type a character of data, MLX disables RETURN until the cursor returns to the start of a line. Remember, you can press CLR/HOME to get to a line number prompt quickly.

Beep or Buzz

When you enter a line, MLX recalculates the checksum from the eight bytes and the address and compares this value to the number from the ninth column. If the values match, you'll hear a pleasant beep to indicate that the line has been entered correctly. The data is then added to the workspace area, and the prompt for the next line of data appears. But if MLX detects a typing error, you'll hear a low buzz and see an error message. MLX will then redisplay the line for editing.

To make corrections in a line that MLX has redisplayed for editing, compare the line on the screen with the one printed in the listing. Then move the cursor to the mistake and type the correct key. The cursor-left and -right keys provide the normal cursor controls. (The INST/DEL key now works as an alternative cursor-left key.) You cannot move left beyond the first character in the line. If you try to move beyond the rightmost character, you'll reenter the line. During editing, RETURN is active; pressing it tells MLX to recheck the line. You can press the CLR/HOME key to clear the entire line if you want to start from scratch, or if you want to get to a line number prompt to use RETURN to get back to the menu.

After you have entered the last number on the last line of the listing, MLX automatically moves to the Save option.

Other MLX Functions

The second menu choice, Display Data, examines memory and shows the contents in the same format as the program listing (including the checksum). When you press D, MLX asks you for a starting address. Be sure that the starting address you give corresponds to a line number in the listing. Otherwise, the display will be meaningless. MLX displays program lines until it reaches the end of the program, at which point the menu is redisplayed. You can pause the scrolling display by pressing the space bar. (MLX finishes printing the current line before halting.) To resume scrolling, press the space bar again. To break out of the display and return to the menu before the ending address is reached, press RETURN.

Two more menu selections let you save programs and load them back into the computer. These options are Save File and Load File; their operations are quite straightforward.

When you press S or L, MLX asks you for the filename. (Again, pressing RETURN at this prompt without entering anything returns you to the command menu.) Next, MLX asks you to press either D or T to select disk or tape.

You'll notice the disk drive starting and stopping several times during a load or save. Don't panic; this is normal behavior. MLX opens and reads from or writes to the file instead of using the usual LOAD and SAVE commands. Remember that MLX saves the entire workspace area from the starting address to the ending address, so the save or load may take longer than you might expect if you've entered only a small amount of data from a long listing. When you're saving a partially completed listing, be sure to note the address where you stopped typing so that you'll know where to resume entry when you reload.

Error Alert

MLX reports any errors detected during the save or load and displays the standard error messages. (Tape users should bear in mind that the Commodore 128 is never able to detect errors when saving to tape.) MLX also has three special load error messages:

• INCORRECT STARTING ADDRESS. The file you're trying to load does not have the starting address you specified when you ran MLX. In this case, no data will be loaded.

• LOAD ENDED AT *address*. The file you're trying to load ends before the ending address you specified when you started MLX. The data from the file has been loaded, but it ends at the address specified in the error message.

• TRUNCATED AT ENDING ADDRESS. The file you're trying to load extends beyond the ending address you specified when you started MLX. The data from the file has been loaded, but only up to the specified ending address.

If you see one of these messages and feel certain that you've loaded the right file, exit and rerun MLX, being careful to enter the correct starting and ending addresses.

If you want to check which programs are on a disk, select the C option from the command menu to display a directory. You can use the 128's NO SCROLL key to pause the display. Afterward, press any key to return to the menu.

The Quit menu option has the obvious effect: It stops MLX and enters BASIC. The RUN/STOP key is disabled, so the Q option lets you exit the program without turning off the computer. (Of course, RUN/STOP–RESTORE also will get you out.) You'll be asked for verification; press Y to exit to BASIC or any other key to return to the menu. After quitting, you can type RUN again and reenter MLX without losing your data, as long as you don't use the clear-workspace option.

The Finished Product

When you've finished typing all the data for an ML program and saved your work, you're ready to see the results. The instructions for loading and using the finished product vary from program to program. Some ML programs are designed to be loaded and run like BASIC programs, so all you need to type is LOAD *"filename"*,8 for disk or LOAD *"filename"* for tape, and then RUN. (These programs usually have 1C01 as their MLX starting address.) Others must be reloaded to specific addresses with a command such as LOAD *"filename"*,8,1 for disk or LOAD *"filename"*,1,1 for tape, and then started with a SYS to a particular memory address. In either case, always refer to the article that accompanies the ML listing for information on loading and running each program.

An Ounce of Prevention

By the time you finish typing in the data for a long ML program, you may have several hours invested in the project. Don't take chances; use our "Automatic Proofreader" (Appendix B) to type MLX, and then test your copy *thoroughly* before using it to enter any significant amount of data. Make sure all the menu options work as they should. Enter fragments of the program starting at several different addresses; then use the Display Data option to verify that the data has been entered correctly. And be sure to test the Save and Load options several times to insure that you can recall your work from disk or tape. Don't let a simple typing error in MLX cost you several nights of hard work.

MLX

```
AE 100 TRAP 960:POKE 4627,128:DIM NL$,A(7)
XP 110 Z2=2:Z4=254:Z5=255:Z6=256:Z7=127:BS=256*PEE
       K(4627):EA=65280
FB 120 BE$=CHR$(7):RT$=CHR$(13):DL$=CHR$(20):SP$=C
       HR$(32):LF$=CHR$(157)
KE 130 DEF FNHB(A)=INT(A/256):DEF FNLB(A)=A-FNHB(A
       )*256:DEF FNAD(A)=PEEK(A)+256*PEEK(A+1)
JB 140 KEY 1,"A":KEY 3,"B":KEY 5,"C":KEY 7,"D":VOL
        15:IF RGR(0)=5 THEN FAST
FJ 150 PRINT"{CLR}"CHR$(142);CHR$(8):COLOR 0,15:CO
       LOR 4,15:COLOR 6,15
GQ 160 PRINT TAB(12)"{RED}{RVS}{2 SPACES}{9 @}
       {2 SPACES}"RT$;TAB(12)"{RVS}{2 SPACES}{OFF}
       {BLU} 128 MLX {RED}{RVS}{2 SPACES}"RT$;TAB(
       12)"{RVS}{13 SPACES}{BLU}"
FE 170 PRINT"{2 DOWN}{3 SPACES}COMPUTE!'S MACHINE
       {SPACE}LANGUAGE EDITOR{2 DOWN}"
DK 180 PRINT"{BLK}STARTING ADDRESS{4}";:GOSUB 260:
       IF AD THEN SA=AD:ELSE 180
FH 190 PRINT"{BLK}{2 SPACES}ENDING ADDRESS{4}";:GO
       SUB 260:IF AD THEN EA=AD:ELSE 190
MF 200 PRINT"{DOWN}{BLK}CLEAR WORKSPACE [Y/N]?{4}"
       :GETKEY A$:IF A$<>"Y" THEN 220
QH 210 PRINT"{DOWN}{BLU}WORKING...";:BANK 0:FOR A=
       BS TO BS+(EA-SA)+7:POKE A,0:NEXT A:PRINT"DO
       NE"
DC 220 PRINT TAB(10)"{DOWN}{BLK}{RVS} MLX COMMAND
       {SPACE}MENU {4}{DOWN}":PRINT TAB(13)"{RVS}E
       {OFF}NTER DATA"RT$;TAB(13)"{RVS}D{OFF}ISPLA
       Y DATA"RT$;TAB(13)"{RVS}L{OFF}OAD FILE"
HB 230 PRINT TAB(13)"{RVS}S{OFF}AVE FILE"RT$;TAB(1
       3)"{RVS}C{OFF}ATALOG DISK"RT$;TAB(13)"{RVS}
       Q{OFF}UIT{DOWN}{BLK}"
AP 240 GETKEY A$:A=INSTR("EDLSCQ",A$):ON A GOTO 34
       0,550,640,650,930,940:GOSUB 950:GOTO 240
SX 250 PRINT"STARTING AT";:GOSUB 260:IF(AD<>0)OR(A
       $=NL$)THEN RETURN:ELSE 250
BG 260 A$=NL$:INPUT A$:IF LEN(A$)=4 THEN AD=DEC(A$
       )
PP 270 IF AD=0 THEN BEGIN:IF A$<>NL$ THEN 300:ELSE
        RETURN:BEND
MA 280 IF AD<SA OR AD>EA THEN 300
PM 290 IF AD>511 AND AD<65280 THEN PRINT BE$;:RETU
       RN
SQ 300 GOSUB 950:PRINT"{RVS} INVALID ADDRESS
       {DOWN}{BLK}":AD=0:RETURN
RD 310 CK=FNHB(AD):CK=AD-Z4*CK+Z5*(CK>Z7):GOTO 330
```

```
DD 320 CK=CK*Z2+Z5*(CK>Z7)+A
AH 330 CK=CK+Z5*(CK>Z5):RETURN
QD 340 PRINT BE$;"{RVS} ENTER DATA ":GOSUB 250:IF
       {SPACE}A$=NL$ THEN 220
JA 350 BANK 0:PRINT:F=0:OPEN 3,3
BR 360 GOSUB 310:PRINT HEX$(AD)+":";:IF F THEN PRI
       NT L$:PRINT"{UP}{5 RIGHT}";
QA 370 FOR I=0 TO 24 STEP 3:B$=SP$:FOR J=1 TO 2:IF
        F THEN B$=MID$(L$,I+J,1)
PS 380 PRINT"{RVS}"B$+LF$;:IF I<24 THEN PRINT"
       {OFF}";
RC 390 GETKEY A$:IF (A$>"/" AND A$<":") OR(A$>"@"
       {SPACE}AND A$<"G") THEN 470
AC 400 IF A$="+" THEN A$="E":GOTO 470
QB 410 IF A$="-" THEN A$="F":GOTO 470
FB 420 IF A$=RT$ AND ((I=0) AND (J=1) OR F) THEN P
       RINT B$;:J=2:NEXT:I=24:GOTO 480
RD 430 IF A$="{HOME}" THEN PRINT B$:J=2:NEXT:I=24:
       NEXT:F=0:GOTO 360
XB 440 IF (A$="{RIGHT}") AND F THEN PRINT B$+LF$;:
       GOTO 470
JP 450 IF A$<>LF$ AND A$<>DL$ OR ((I=0) AND (J=1))
        THEN GOSUB 950:GOTO 390
PS 460 A$=LF$+SP$+LF$:PRINT B$+LF$;:J=2-J:IF J THE
       N PRINT LF$;:I=I-3
GB 470 PRINT A$;:NEXT J:PRINT SP$;
HA 480 NEXT I:PRINT:PRINT"{UP}{5 RIGHT}";:L$="
       {27 SPACES}"
DP 490 FOR I=1 TO 25 STEP 3:GET#3,A$,B$:IF A$=SP$
       {SPACE}THEN I=25:NEXT:CLOSE 3:GOTO 220
BA 500 A$=A$+B$:A=DEC(A$):MID$(L$,I,2)=A$:IF I<25
       {SPACE}THEN GOSUB 320:A(I/3)=A:GET#3,A$
AR 510 NEXT I:IF A<>CK THEN GOSUB 950:PRINT:PRINT"
       {RVS} ERROR: REENTER LINE ":F=1:GOTO 360
DX 520 PRINT BE$:B=BS+AD-SA:FOR I=0 TO 7:POKE B+I,
       A(I):NEXT I
XB 530 F=0:AD=AD+8:IF AD<=EA THEN 360
CA 540 CLOSE 3:PRINT"{DOWN}{BLU}** END OF ENTRY **
       {BLK}{2 DOWN}":GOTO 650
MC 550 PRINT BE$;"{CLR}{DOWN}{RVS} DISPLAY DATA ":
       GOSUB 250:IF A$=NL$ THEN 220
JF 560 BANK 0:PRINT"{DOWN}{BLU}PRESS: {RVS}SPACE
       {OFF} TO PAUSE, {RVS}RETURN{OFF} TO BREAK
       E4{DOWN}"
XA 570 PRINT HEX$(AD)+":";:GOSUB 310:B=BS+AD-SA
DJ 580 FOR I=B TO B+7:A=PEEK(I):PRINT RIGHT$(HEX$(
       A),2);SP$;:GOSUB 320:NEXT I
XB 590 PRINT"{RVS}";RIGHT$(HEX$(CK),2)
```

```
GR 600 F=1:AD=AD+8:IF AD>EA THEN PRINT"{BLU}** END
        OF DATA **":GOTO 220
EB 610 GET A$:IF A$=RT$ THEN PRINT BE$:GOTO 220
QK 620 IF A$=SP$ THEN F=F+1:PRINT BE$;
XS 630 ON F GOTO 570,610,570
RF 640 PRINT BE$"{DOWN}{RVS} LOAD DATA ":OP=1:GOTO
        660
BP 650 PRINT BE$"{DOWN}{RVS} SAVE FILE ":OP=0
DM 660 F=0:F$=NL$:INPUT"FILENAME{4}";F$:IF F$=NL$
       {SPACE}THEN 2?20
RF 670 PRINT"{DOWN}{BLK}{RVS}T{OFF}APE OR {RVS}D
       {OFF}ISK: {4}";
SQ 680 GETKEY A$:IF A$="T" THEN 850:ELSE IF A$<>"D
       " THEN 680
SP 690 PRINT"DISK{DOWN}":IF OP THEN 760
EH 700 DOPEN#1,(F$+",P"),W:IF DS THEN A$=D$:GOTO 7
       40
JH 710 BANK 0:POKE BS-2,FNLB(SA):POKE BS-1,FNHB(SA
       ):PRINT"SAVING ";F$:PRINT
MC 720 FOR A=BS-2 TO BS+EA-SA:PRINT#1,CHR$(PEEK(A)
       );:IF ST THEN A$="DISK WRITE ERROR":GOTO 75
       0
GC 730 NEXT A:CLOSE 1:PRINT"{BLU}** SAVE COMPLETED
        WITHOUT ERRORS **":GOTO 220
RA 740 IF DS=63 THEN BEGIN:CLOSE 1:INPUT"{BLK}REPL
       ACE EXISTING FILE [Y/N]{4}";A$:IF A$="Y" TH
       EN SCRATCH(F$):PRINT:GOTO 700:ELSE PRINT"
       {BLK}":GOTO 660:BEND
GA 750 CLOSE 1:GOSUB 950:PRINT"{BLK}{RVS} ERROR DU
       RING SAVE: {4}":PRINT A$:GOTO 220
FD 760 DOPEN#1,(F$+",P"):IF DS THEN A$=DS$:F=4:CLO
       SE 1:GOTO 790
PX 770 GET#1,A$,B$:CLOSE 1:AD=ASC(A$)+256*ASC(B$):
       IF AD<>SA THEN F=1:GOTO 790
KB 780 PRINT"LOADING ";F$:PRINT:BLOAD(F$),B0,P(BS)
       :AD=SA+FNAD(174)-BS-1:F=-2*(AD<EA)-3*(AD>EA
       )
RQ 790 IF F THEN 800:ELSE PRINT"{BLU}** LOAD COMPL
       ETED WITHOUT ERRORS **":GOTO 220
ER 800 GOSUB 950:PRINT"{BLK}{RVS} ERROR DURING LOA
       D: {4}":ON F GOSUB 810,820,830,840:GOTO220
QJ 810 PRINT"INCORRECT STARTING ADDRESS (";HEX$(AD
       );")":RETURN
DP 820 PRINT"LOAD ENDED AT ";HEX$(AD):RETURN
EB 830 PRINT"TRUNCATED AT ENDING ADDRESS ("HEX$(EA
       )")":RETURN
FP 840 PRINT"DISK ERROR ";A$:RETURN
KS 850 PRINT"TAPE":AD=POINTER(F$):BANK 1:A=PEEK(AD
       ):AL=PEEK(AD+1):AH=PEEK(AD+2)
```

```
XX 860 BANK 15:SYS DEC("FF68"),Ø,1:SYS DEC("FFBA")
       ,1,1,Ø:SYS DEC("FFBD"),A,AL,AH:SYS DEC("FF9
       Ø"),128:IF OP THEN 89Ø
FG 870 PRINT:A=SA:B=EA+1:GOSUB 92Ø:SYS DEC("E919")
       ,3:PRINT"SAVING ";F$
AB 880 A=BS:B=BS+(EA-SA)+1:GOSUB 92Ø:SYS DEC("EA18
       "):PRINT"{DOWN}{BLU}** TAPE SAVE COMPLETED
       {SPACE}**":GOTO 22Ø
CP 890 SYS DEC("E99A"):PRINT:IF PEEK(2816)=5 THEN
       {SPACE}GOSUB 95Ø:PRINT"{DOWN}{BLK}{RVS} FIL
       E NOT FOUND ":GOTO 22Ø
GQ 900 PRINT"LOADING ...{DOWN}":AD=FNAD(2817):IF A
       D<>SA THEN F=1:GOTO 8ØØ:ELSE AD=FNAD(2819)-
       1:F=-2*(AD<EA)-3*(AD>EA)
JD 910 A=BS:B=BS+(EA-SA)+1:GOSUB 92Ø:SYS DEC("E9FB
       "):IF ST>Ø THEN 8ØØ:ELSE 79Ø
XB 920 POKE193,FNLB(A):POKE194,FNHB(A):POKE 174,FN
       LB(B):POKE 175,FNHB(B):RETURN
CP 930 CATALOG:PRINT"{DOWN}{BLU}** PRESS ANY KEY F
       OR MENU **":GETKEY A$:GOTO 22Ø
MM 940 PRINT BE$"{RVS} QUIT E4]";RT$;"ARE YOU SURE
        [Y/N]?":GETKEY A$:IF A$<>"Y" THEN 22Ø:ELSE
        PRINT"{CLR}":BANK 15:END
JE 950 SOUND 1,5ØØ,1Ø:RETURN
AF 960 IF ER=14 AND EL=26Ø THEN RESUME 3ØØ
MK 970 IF ER=14 AND EL=5ØØ THEN RESUME NEXT
KJ 980 IF ER=4 AND EL=78Ø THEN F=4:A$=DS$:RESUME 8
       ØØ
DQ 990 IF ER=3Ø THEN RESUME:ELSE PRINT ERR$(ER);"
       {SPACE}ERROR IN LINE";EL
```

Index

Unlock the Hidden Power of your Commodore Computer with

COMPUTE!'s GAZETTE

More fun...More challenge... More all new programs each and every month.

Subscribe to *COMPUTE!'s Gazette* through this special money-saving introductory offer—and start unleashing the full power of your Commodore computer.

Month after month, look to *COMPUTE!'s Gazette* to deliver the latest inside word on everything from short programming tips to the best new software. Our expert analysis and insights mean you have more fun...get more enjoyment...more of what you bought your computer for.

As a subscriber, you also receive up to 20 all-new, action-packed programs each month. Every big issue of *COMPUTE!'s Gazette* comes complete with a steady supply of the most useful, the most entertaining, the highest quality programs like Number Quest, Address File, Treasure Hunt, Castle Dungeon, Vocab Builder, SpeedScript, and hundreds of other educational, home finance, and game programs.

So subscribe today—and unleash the hidden power of your Commodore computer. Return the card below—or call 1-800-247-5470 (in Iowa, 1-800-532-1272). Do it now.

COMPUTE!'s GAZETTE

$3.00 July 1986 Issue 37, Vol. 4, No. 7

FOR COMMODORE PERSONAL COMPUTER USERS

Saloon Shootout

Hi-Res Dump

Quick printouts of your screen artwork with this program for the 128, 64, and Plus/4.

Power BASIC: 64 RAM Disk

Budget Planner

Manage your money easy way with this mini-spreadsheet to handle weekly and monthly income for an entire year. For the Commodore 128, 64, and Plus/4.

Home

typical American home may soon. (Tomorrow's electronic "smart" home is just around the corner.)

SAVE 50%

Subscription Savings Card

☐ **YES!**

I know a great deal when I see one. Sign me up for 12 big issues of *COMPUTE!'s Gazette* for just $18. I save 50% off the newsstand price.

☐ Payment enclosed ☐ Bill me ☐ Charge my VISA/MasterCard

Credit Card #________________ Exp. Date________

Signature________________________

Name________________________

Address________________________

City________________ State______ Zip______

Outside U.S.A. please add $6 (U.S.) per year for postage.

Say YES now to COMPUTE!'s GAZETTE

You can search far and wide and you simply *won't* find a better magazine...a better source of insightful, stimulating, usable information for your Commodore computer than *COMPUTE!'s Gazette.*

COMPUTE's Gazette works harder...digs deeper...researches further—all to help guarantee that you get the absolute most from your Commodore.

Subscribe today and each and every month you'll receive up to 20 all-new action-packed programs. That's up to 200 programs each year—game programs...education programs...home applications programs...personal and budgeting programs...sorting and filing programs.

Add it up for yourself. Where else can you get exciting programs each month...expert advice...insightful analysis...up-to-the-minute software reviews and so much more—all for just $18.

So why wait. Subscribe now to *COMPUTE!'s Gazette*—and get the most from your Commodore computer. Return card below—or call 1-800-247-5470 (in Iowa, 1-800-532-1272).

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7551 DES MOINES, IA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE!'s Gazette

P.O. Box 10775
Des Moines, IA 50347-0775

128 Power

Commodore's 128 has been a winner from the start, and its popularity continues. *COMPUTE!'s Second Book of Commodore 128* offers a collection of over 25 articles, some never before published, to inform you, utilities to ease your way, applications to work for you, and games to challenge and amuse you.

If you're new to computing, you'll find understandable articles that will lead you, step by step, into discovering—and using—the power of the 128. If you already know programming, you'll find new ideas that will increase your knowledge of the 128, as well as many practical applications. Both nonprogrammers and longtime computer buffs will find a library of useful programs that are ready to type in and run.

Here is a sample of what's inside:

- "Pig$ for Buck$," a game that children and adults alike will enjoy
- "Mozart Magic," based on a musical game devised by the composer, creates its own minuets in the master's style
- An explanation of 128 disk commands
- A patch program that allows 80-column monitors to be used with the original *SpeedScript* word processor
- "Puzzle Grid, " a 25-piece puzzle that's a real baffler
- "TurboDisk 128," a powerful utility that can reduce by 300 percent or more the time you spend waiting for programs to load
- "KeyDef," a handy utility that lets you redefine keys to print any characters you want
- The world of CP/M—how to access free and low-cost software
- Plus many more games, applications, and utilities

Whatever your interests, *COMPUTE!'s Second Book of Commodore 128* gives clear explanations of new techniques and provides programs that have been tested and are ready to run. It's a collection you'll use and enjoy—again and again.

The programs in this book are all available on a companion disk. See the coupon in the back for details.

ISBN 0-87455-077-7

COMPUTE!'s Second Book of Commodore 128

COMPUTE! Books