60 THE COMMODORE 64

HOME COMPUTING BEYOND THE HOBBYIST
COMMODORE BUSINESS MACHINES
PETSCII
THE VIC-II CHIP
THE KERNAL


```
(210)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

Figure 60.1
The Commodore 64 computer was released in 1982 as a followup to the Commodore VIC-20. As the name signals, it had sixty-four kilobytes of memory. Photo by Mark Richards. Courtesy of Mark Richards.

The Commodore 64 (see figure 60.1) has been hailed by *Guinness World Records* as the best-selling single model of computer ever. People associated with Commodore have estimated, officially and unofficially, that 22 million or 17 million units were sold. A detailed study of Commodore 64 serial numbers has provided a better estimate, that 12.5 million Commodore 64s were sold (Steil 2011), which is still enough to earn the computer this distinction.

Although production ended in 1994, this computer system remains functioning and part of the popular consciousness in many ways. VICE and many other emulators allow users to start up software editions of the Commodore 64 and to run software for that system on modern computers, which is the way most people now encounter Commodore 64 software. In 2004 Jeri Ellsworth's C64 Direct-to-TV—a single-chip implementation of the Commodore 64, packed into a joystick along with thirty games—brought at least part of the Commodore experience to new users. And, in 2011, a company necro-branded with the name Commodore USA announced that they would be making new all-in-one PCs in a case (and with a keyboard) that is visually almost identical to that of the original Commodore 64 (Olivarez-Giles 2011).

The original Commodore 64 computer has particular features—the PETSCII character set (figure 60.2), built-in BASIC, and the specific appearance of the screen—that determine how 10 PRINT runs. At the same time, it was one computer among many during the early 1980s that brought forth this significant era of personal computing and, perhaps more novel, home computing.

HOME COMPUTING BEYOND THE HOBBYIST

In the early 1980s, computers moved beyond the exclusive domain of hackers and hobbyists and into the home, a transition led by Apple, Radio Shack, and Commodore. In October 1984, 8.2 percent of all U.S. households reported owning a home computer. Of those households, 70 percent had acquired their computer quite recently, in either 1983 or 1984 (U.S. Bureau of the Census 1988, 2). By 1989—the outer boundary of the Commodore 64's mainstream popularity—computer ownership had skyrocketed to 15 percent. Households with school-aged children were nearly

Figure 60.2
The graphics characters for each key of the Commodore 64 keyboard are printed on the side. Here, the two characters used in `10 PRINT` are visible on the sides of the N and M keys. Photo by Mark Richards. Courtesy of Mark Richards.

twice as likely to own a computer (at 25.7 percent), while 45.6 percent of households earning more than $75,000 annually ($138,000 in 2012 dollars) owned computers (Kominski 1991, 1–3).

Yet even as microcomputers became *personal* computers, the prospect of computer ownership was closely tied to income (U.S. Bureau of the Census 1988, 2). This trend was exacerbated when race was factored in. Black and Hispanic families were far less likely to have a computer at home in the 1980s, and by 1997, this gap had translated into a digital divide online, in which Whites were twice as likely as Blacks and Hispanics to use the Internet (Kominski and Newburger 1999, 12).

Gender appears to have been less of a factor in computer use than race or socioeconomic status was. In 1984, boys (31.9 percent) were slightly more likely to use a computer than girls (28.4 percent), even at school, but by 1989 that small gap had closed (46.5 percent and 45.5 percent)

Figure 60.3

This 1983 advertisement for the Commodore 64 sold the system as a powerful computer within the financial reach of middle-class families.

```
{214}  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

(Kominski and Newburger 1999, table 3). The gap between adult females and males follows a similar trend: A small divide becomes smaller in the 1980s (Ibid., table 5). However, women (29 percent in 1984) were more likely to use a computer at work than men (21.2 percent in 1984), often because more women worked in data entry or administrative support positions (Ibid., table 6). A more statistically significant discrepancy appears in computer ownership by household income. Again, looking at 1984 and 1989, compare the rise in home computer ownership from 5.3 to 8 percent in households earning $15,000–$20,000 with 22.4 to 31.6 percent in households earning $50,000–$75,000 (Ibid., table 2). By 1989 the disparity appears magnified with 43.8 percent of families owning computers in the $75,000-plus range and only 3.7 percent in the $5,000–$9,000 range owning computers (table 2).

These socioeconomic, racial, and gender disparities are part of the context of `10 PRINT`, as much as the history of textured patterns or BASIC is. They can be seen playing out in one of the iconic Commodore 64 magazine advertisements of the era (figure 60.3).

Given how costly home computing was, Commodore shrewdly positioned its computers as economical yet more powerful than its competitors'. This 1983 advertisement declares, "You can't buy a better computer at twice the price" as it shames Apple, Radio Shack, and IBM for pricing their personal computers at a range only "wealthy," "whiz-kid," or "privileged" persons could possibly afford. The difference between Commodore and these other PCs is not measured solely in dollar amounts. The three costlier computers are crowded into a black and white background, almost hidden from view by the large "FOR NOBODY." The Commodore 64 occupies the bottom half of the page, bathed in warm colors. A father and mother watch their child explore the galaxy on the computer, suggesting that the Commodore is a portal to a larger universe—a universe of knowledge and opportunity. The family indicates a carefully targeted market. Parents were twice as likely to purchase a computer. It is telling, too, that this family is white and middle-class and that their child appears to be a boy. Though the statistics suggest more gender balance in access to computers, the advertisement reinforces a narrative of home computers as the realm of boys. Doug Thomas identifies the broader "hacker demographic" as predominantly "white, suburban boys" (2002, x), and contemporary programming culture, from gender imbalances in undergraduate studies

to professional spaces, suggests of the force of that legacy. As would be typical of advertising of that era, "everybody" actually turned out to be an extremely specific demographic.

While the market for home computers was smaller than advertisers acknowledged, the computers themselves spanned a range of styles and forms that went far beyond the Apple-Commodore-TRS-80 trifecta. In addition to the more well-known brands, there were also Sinclair ZX Spectrums, BBC Micros, and computers from Amstrad and Acorn, all of which originated in the United Kingdom. The Texas Instruments TI-99/4A and the Coleco Adam were available, too. Among so many choices, advertisers had to build the personality of not only the brands but the individual machines as well. In the world of computing since the inundation of PC clones, it is difficult to imagine the aura produced around individual machines. Yet today's programmers can still recall their first Apple IIc, VIC-20, or TRS-80. Apple alone now clings to the marketing of "different" machines, though even their computers have Intel inside and the company tends to market product lines rather than individual model numbers. It was a very different landscape that saw the advent of a personal computer that wore its sixty-four kilobytes of memory as a badge of honor. To buy a Commodore 64 was to buy capacity itself.

This diversity meant that different manufacturers could try different types of hardware design and burn different operating systems in ROM. It fostered certain types of corporate exploration of the home computer market, while also limiting the way that software could be shared—even if that software was in the lingua franca of BASIC, given the variety of BASIC dialects. The experience of home computing was in many ways stratified by platform. The Apple Store was not the first example of a platform-specific retail establishment to sell computers. Many vendors would at least specialize in a particular company's computers; in some cases, stores were exclusively Apple, Atari, or Commodore outfits, just as Radio Shack was exclusively a seller of TRS computers (see figure 60.4).

Computer owners also created and joined user groups that were specific to platforms and that met in person. As discussed in the chapter on BASIC, they also subscribed to and read magazines that were for computers of a certain type. When Bulletin Board Systems (BBSs) came onto the scene, some hosted the users of many different types of computer and others, particularly those devoted to making software available for download,

Figure 60.4
Local students at Bob West Computers in Brevard, NC, take turns with a Commodore computer. Courtesy of Bob West.

focused on a single platform.

This did not mean that every computer user was paired with a single platform. Some households had more than one computer—perhaps to keep the work computer from being occupied by a younger member of the family, the parents would decide to provide another computer more geared to games and education. Even those without a computer at home might have access to several at retail stores (which often allowed children to enjoy extended sessions with the computers available for sale), at school, and at friends' houses. Given this environment for computing, even those who were mainly Apple II users or who tooled on their Coleco Adam systems at home might have had an opportunity to play around a bit with a Commodore 64. With limited time and particularly in the context of a school or retail store, where the available software might be limited or nonexistent, it would not have been a bad idea for a visitor to the Commodore 64 to learn about and modify one-liners such as `10 PRINT`.

COMMODORE BUSINESS MACHINES

The history of the Commodore 64 begins with the Canadian company Commodore, founded in 1958 in Toronto by Jack Tramiel. Tramiel was born Idek Tramielski, was a Polish concentration camp survivor, and changed his name after World War II, when he emigrated to the United States. After serving in Korea, Tramiel worked as a typewriter repair technician, eventually opening a repair company with a business partner. Commodore was the successor company that they formed. This new company did not repair typewriters; it manufactured them (Bagnall 2010, xiii). Once again, the history of `10 PRINT` is intertwined with earlier technologies. Personal computers were hardly a natural progression or simple next step from typewriters, but their prominent keyboards, their use as office equipment, and their use for typewriter-like word processing tasks all demonstrate they had affinities with earlier devices.

In the mid-1960s Commodore shifted its focus from manufacturing typewriters to making calculators, a move driven by strictly financial considerations. In hindsight, however, it seems to evoke the same tension between text and numbers, between poetics and algorithms, that underwrites the aesthetic and procedural dimensions of `10 PRINT`. Caught in a

```
{218}  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

price war with Texas Instruments and Japanese manufacturers in the 1970s, Tramiel sought the cheapest calculator components he could find, eventually buying parts from MOS Technology, a semiconductor company where many former Motorola engineers worked. While MOS Technology earned its revenue from selling calculator chips (mostly to Commodore, its largest customer), the company was also developing a microprocessor, the 6502.

This chip, the 6502, is now legendary for its role in 1980s computing and videogaming. The 6502 became the central processing unit (CPU) for the original Apple I, the Apple II, the Atari 400 and 800, the Nintendo Entertainment System (NES), and of course, modified with an I/O port, the Commodore 64. In a lower-cost package, the chip also powered the Atari 2600. Yet MOS Technology never intended the chip to be used in computers or videogame systems. The 6502 was designed as a single chip replacement for the two- or three-chip processors found in cash registers, appliances, and industrial machines. "If we were going to do a computer," Chuck Peddle, the lead engineer on the project confessed, "we would have done something else" (Bagnall 2010, 14).

With an eye on vertical integration and the 6502 microprocessor, Jack Tramiel bought MOS Technology in September 1976, but not in the most straightforward fashion. Tramiel, widely considered a ruthless businessman, withheld payments to MOS—whether because Commodore was cash-strapped or there was a problem with an order of chips, or both, is a matter of speculation. Nevertheless, it meant that MOS was in turn facing a cash shortfall. The problem was compounded by a lawsuit from Motorola over possible intellectual property infringement (Bagnall 2010, 56). Tramiel was able to buy MOS Technology at a bargain price—about $750,000—which meant that Commodore gained its own chip design and production facility.

The PET

Tramiel was still intent on dominating the calculator business, however, and it took Chuck Peddle and Commodore's vice-president of engineering, Andre Sousan, to persuade him that a personal computer would in fact be the next generation calculator, leapfrogging over Hewlett-Packard's successful programmable HP-65 calculator (Bagnall 2010, 62). Thus was born the project that would become the eight-bit Commodore PET (figure 60.5), the first computer under $1,000 ($3,733 in 2012 dollars) to include a monitor.

Figure 60.5
The Commodore PET computer was released in 1977. It featured four kilobytes of memory and a tape drive for storing and loading programs. Photo by Mark Richards. Courtesy of Mark Richards.

```
(220)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

The PET was particularly successful in Europe, where Commodore already had a strong presence from its calculator business. With nearly 70 percent of its sales in Europe through the 1970s, it is no surprise that Commodore would include a pound sterling symbol on the keyboards of the VIC-20 and Commodore 64. The PET's name is a sign of the times; Sousan came up with this name to capitalize on the pet rock craze of the late 1970s, and only afterward did Peddle suggest "Personal Electronic Transactor" as a "backronym" that would explain the PET's name logically (Freiberger 1982, 13).

As discussed in the BASIC chapter, the PET was the first of Commodore's computers to include BASIC in ROM, making the PET ready for programming the moment the computer had booted up. Another legacy of the PET that made its mark on the Commodore 64 and on `10 PRINT` is its extended graphical character set, informally dubbed PETSCII. (The name "extends" ASCII, the standard character set for computers.) PETSCII was largely designed for the PET by engineer Bill Seiler and Leonard Tramiel (Jack Tramiel's son), who worked at the time as Commodore's in-house tester and debugger (Bagnall 2010, 92–93). The chief rationale for PETSCII, which included the 128 characters of ASCII plus 128 additional graphic characters, was to provide a simple way to produce graphical characters such as playing card symbols. It is commonplace to observe that innovations in computer graphics drive much innovation in computers—chip speed, bus speed, memory sizes, and so on—and here is a less obvious example. While the graphical character set of PETSCII, which features the four suits of cards, shaded patterns, and various brackets and lines, could hardly be said to be an innovation, it made possible early computer games in BASIC without the need to program sprites or other animated figures. And PETSCII made `10 PRINT` possible as well, providing programmers with the two diagonal characters found in the maze way back in 1977.

The VIC-20

While business and education were the primary markets for the PET computers, its follow-up the VIC-20 was aimed squarely at the home computer market. Released in 1980, the outside of the VIC-20 was exactly the same physical form that the Commodore 64 would later have. (The VIC-20's plastic was lighter in color, more of an off-white instead of the Commodore

64's taupe.) Like both the PET and the Commodore 64, the VIC-20 was powered by the 6502 chip and included Microsoft's version of BASIC. The VIC-20, however, was sold with only five KB of RAM, a tiny slice of the Commodore 64's sixty-four KB. The system also had a color display that was twenty-two characters wide, powered by the forty-pin VIC chip (Video Interface Controller). The VIC 6560 chip had been designed by MOS Technology engineer Al Charpentier to be sold to other manufacturers like Apple and Atari, but none were interested (Bagnall 2010, 178). Ultimately it found its way into the VIC-20. Its shortcomings inspired the creation of a more powerful graphics chip for the Commodore 64.

Because the VIC-20 ran the same version of Microsoft BASIC and included the same PETSCII character set as the PET before it and the Commodore 64 after it, the `10 PRINT` program executes flawlessly on the VIC-20, though no published versions of the maze program intended for the VIC-20 specifically are known to exist. If users had run `10 PRINT` or a variation on the VIC-20, they would have had a different aesthetic experience than a Commodore 64 user (figure 60.6). PETSCII was designed for the forty-column PET; on the twenty-two-column VIC-20 the characters are elongated, stretched as if one were watching an old 4:3 television show on a widescreen. The maze looks almost 3D, as if seen from the isometric point of view of Sega's 1982 hit arcade game *Zaxxon*.

Despite its modest memory, the VIC-20 was seen as a dramatic improvement over the PET computers, at a price that appealed to the home market. The VIC-20 was sold in retail stores (including K-Mart) to a broader market than previous computers had reached. It was the bestselling computer of 1982 (the year when the Commodore 64 was introduced), selling 800,000 units, but then it took a back seat to the more expensive but also much more powerful Commodore 64. While the VIC-20 was discontinued in 1985, the Commodore 64 was sold through 1994.

There is much to say about the Commodore 64 as one of the most popular home computers of all time, but for the sake of clarity it is important to focus on those elements of the Commodore 64 that come into play in `10 PRINT`, namely, its unique graphical character set, the VIC-II chip that implements the computer's graphic capabilities, and the ROM-based operating system, or KERNAL.

```
(222)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

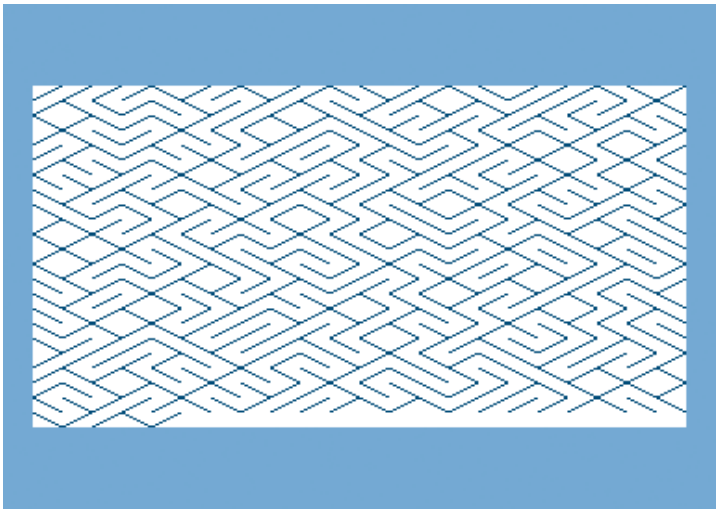


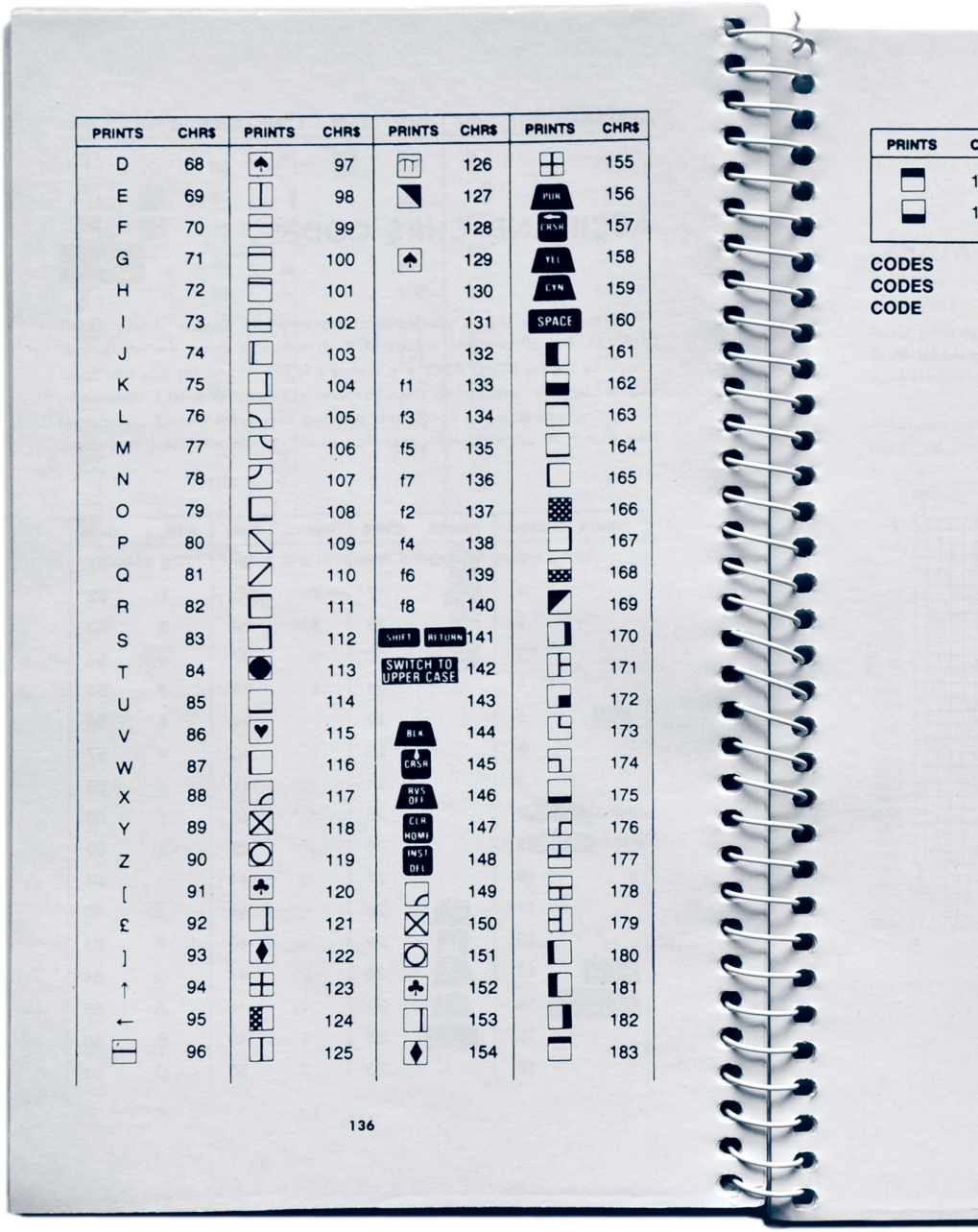
Figure 60.6
The `10 PRINT` maze on the 22 × 23 screen of the VIC-20.

PETSCII

While the PETSCII character set was not unique to the Commodore 64, it was an idiosyncrasy of Commodore computers; neither the Apple II nor the TRS-80 line of computers, which competed with Commodore's computers, offered an extended version of ASCII. A close examination of PETSCII, and particularly its implementation on the Commodore 64, is therefore helpful in appreciating `10 PRINT`.

The facts of PETSCII are simple: it is an extension of the 128-character ASCII (American Standard Code for Information Interchange) set; in addition to letters, numbers, and punctuation, it contains color codes (to turn text white, for example), screen control codes (such as RETURN or CLR), and graphical characters (lines, curves, arrows, boxes, and shaded patterns). These graphical characters are labeled on the PET, VIC-20, and Commodore 64 keyboards, and are easily accessed with the Commodore or SHIFT keys.

These facts are well known and well documented. Less obvious are a myriad of quirks about PETSCII on the Commodore 64. To begin with, the name PETSCII is unofficial. Commodore only ever referred to its character



PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
D	68		97		126		155
E	69		98		127	PUR	156
F	70		99		128	CRSR	157
G	71		100		129	YEL	158
H	72		101		130	CYN	159
I	73		102		131	SPACE	160
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84		113	SWITCH TO UPPER CASE	142		171
U	85		114		143		172
V	86		115	BLK	144		173
W	87		116	CRSR	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90		119	INST DEL	148		177
[	91		120		149		178
£	92		121		150		179
]	93		122		151		180
↑	94		123		152		181
←	95		124		153		182
	96		125		154		183

136

PRINTS C

CODES
CODES
CODE

Figure 60.7

Appendix F of the 1982 *Commodore 64 User's Guide* lists the mapping between numerical values and graphical symbols in PETSCII.

```
(224)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

set as ASCII; PETSCII was an informal name that came from the Commodore's users, not its engineers, that conflated PET and ASCII. The character set's creator, Leonard Tramiel, was not in favor of the name PETSCII, noting, "I never really liked that term since it was never much of a standard" (Bagnall 2010, 92).

The Order of PETSCII

Another uncertainty about PETSCII is the order of the characters in the PETSCII table (figure 60.7). Very few related graphical characters are numerically adjacent to each other, neighbors according to character code. In fact, many related images (sets of corners, playing card suit symbols, and mirror images) appear to be scattered throughout the table. A spade is CHR$(97) while a heart is CHR$(115). The upper-right quarter of a circle is CHR$(105) while the upper-left quarter is CHR$(117). A filled-in circle is CHR$(113), the outline form CHR$(119).

Why is the order of graphical characters in the PETSCII table so seemingly haphazard? The answer is that arrangement was dictated by the PET keyboard design, a hardware-driven decision. The original PET 2001 keyboard is a variant of the QWERTY arrangement, featuring the graphical characters of PETSCII alongside the regular keyboard letters (figure 60.8). The grid of keys became a canvas for displaying logical groupings of related symbols. Thus the four corners of a square are grouped on the keys for O, P, L, and :. Similarly, the four arcs of a circle are found on the U, I, J, and K keys, and the four suits of a card deck on A, S, Z, and X.

There are times when the visual grouping on the keyboard and the numerical character codes logically coincide, namely with alphabetically adjacent keys on the QWERTY keyboard: F, G, and H; J, K, and L; N and M (though the letters are reversed here); and O and P. In these four instances, the CHR$ codes associated with each character *are* numerically adjacent, as is not the case with many of the other graphical characters, which, while adjacent on the physical keyboard, are effectively scrambled by the QWERTY layout before being placed in the alphabetized PETSCII index.

Not coincidentally, `10 PRINT` uses the NM pair—because it is visible on the interface, because it is elegant and concise in the code, and because the output is surprising, given the context of mazelike computer graphics at the time. There are other pairs of keys that share graphically

Figure 60.8
The PET 2001 keyboard had PETSCII graphics symbols printed on the front of the corresponding keys. The graphics were arranged spatially on the keyboard. For example, notice the arrangement ╱ and ╲, side by side on the N and M keys.

related characters (the right angles on the O and P keys, for example), but only NM will produce something more structural than textural, with pleasing large-scale variation.

Taking a closer look at the graphical characters on the N and M keys—CHR$(206) and CHR$(205), respectively—reveals more details about PETSCII. First, there are the numbers themselves. The ASCII chart included in appendix F of the *Commodore 64 User's Guide* lists the values of ╱ and ╲ as CHR$(110) and CHR$(109), yet the title of this book uses CHR$(205) as its touchstone, and the first two published versions of the program, in the very same *Commodore 64 User's Guide* and *Run* magazine, also use CHR$(205) as their base. The *Commodore 64 User's Guide* notes that "CODES 192-223 SAME AS 96-127" (Commodore 1982, 137), meaning that 109 and 110 are exactly the same as 205 and 206. But why? Why do early versions of the program use the upper character values (205 and 206), especially when the PETSCII chart that appears in the manual itself only lists the 109 and 110 values?

A likely explanation can be found in the way the Commodore 64 responds to `PRINT ASC("X")`, a technique used to determine the ASCII character code of any printable character. If a user were seeking the character code of a graphic symbol she saw on her keyboard, say, the heart on

```
{226}  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

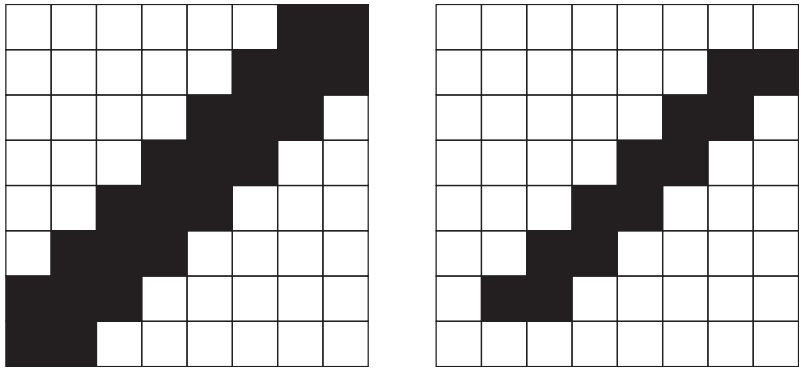


Figure 60.9
PETSCII character 206 (left) goes edge to edge within the grid, while character 47 (right), the forward slash, leaves space on the top and bottom for better spacing when used within a block of text.

the S key, or more to the point, the diagonal line on the N key, she could type `PRINT ASC("/")` and the computer would respond with "206." So, a possible implication of 205/206 being used in `10 PRINT` is that users were more likely to experiment with the keyboard in front of them than to look up codes in the back of the manual. Through the `ASC` function, BASIC became a self-contained pedagogical instrument itself, making outside manuals and guides less necessary.

The Shape of PETSCII

There is yet more to discover about the two graphical characters that appear in `10 PRINT`. Like all PETSCII characters, the two characters are plotted out on an 8 × 8 matrix of pixels. Whereas regular alphanumerical characters are generally confined to a 7 × 7 portion of the matrix, leaving a single-pixel "border" between characters, many of the graphical characters extend to the edge of their 8 × 8 grid. Consider the close-up of `CHR$(206)` in figure 60.9. Its distinct features become apparent when compared to the typographical symbol that most closely resembles it, the forward slash, or `CHR$(47)`.

`CHR$(206)` is three pixels wide in its body and terminates on either

end in a point, a thinning of the line that accounts for the divot that appears whenever two of the same characters are connected in the `10 PRINT` maze. The `CHR$(47)` slash, meanwhile, is a uniform two pixels wide. The difference between the graphical character and the typographical symbol is a mere one pixel along some of the edges, but it is significant. The shape of `CHR$(206)`—as well as the shape of its mirror image, `CHR$(205)`—is essential to the texture of the maze.

THE VIC-II CHIP

While the PETSCII character set remained the same from the PET to the VIC-20 and through to the Commodore 64, the means of displaying those characters—the chip controlling the graphics—changed dramatically over time. Despite its name, the 6567 (NTSC)/6569 (PAL) VIC-II graphics chip was not merely an improvement upon the VIC chip in the VIC-20. It was a complete redesign, led by Charpentier, the MOS engineer behind the first VIC. Home videogame systems, particularly Mattel's Intellivision, were the chief inspirations of the designers at MOS, who set out to create the most advanced graphics chip on the market (Bagnall 2010, 318).

The specifications of the final version of the chip were impressive for the time: three different forty-column text modes, two bitmap modes of 320 × 200 pixels each, eight hardware-driven sprites, hardware-supported screen scrolling, and a sixteen-color palette (Bauer 1996). The influence of videogames can clearly be seen in the VIC-II's built-in side and vertical scrolling (by seven pixels at a time) and the VIC-II's handling of sprites. Far more sophisticated than the sprites in the Atari 2600, the VIC-II sprites are 24 × 21 pixels and can be multicolored. The VIC-II chip can detect collisions between sprites; it can also detect when sprites have collided with other graphical data on the screen or individually specified raster lines (the horizontal scan lines on the CRT or television screen).

Text on the VIC-II

Despite its advanced sprite handling, though, the text modes of the VIC-II chip are the most relevant to `10 PRINT`. The text or character-based modes occupy one kilobyte of screen memory, and consist of forty columns

```
{228}  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

and twenty-five rows of characters, namely 1,000 characters in total. As `10 PRINT` writes the maze across the screen, row by row, it plots one of its two PETSCII characters in each space on the 40 × 25 grid, and just for a fraction of a second, 1,000 characters do fill the entire screen—in what might be considered an illusory consummation of the maze—before the text scrolls upward, leaving two more twenty-five-character rows to fill.

This point is key to understanding the dynamic between the aesthetic quality of the maze and the computer process by which it is plotted. While the code for `10 PRINT` specifies one of two characters to display on the screen, it says nothing about where on the screen the chosen character should appear. That placement is defined by the VIC-II chip. More specifically, the placement of either `CHR$(205)` or `CHR$(206)` depends on the Commodore 64's screen memory map. To the user, the screen appears as a 40 × 25 grid, but to the VIC-II graphics chip, the screen is a series of memory slots, or locations. The first slot, 1024, is at grid location 0,0 and pixel location 0,0. Memory location 1025 maps to the space after this, to the right, and so on. Any character value that is stored in a memory slot will be displayed at the corresponding screen position. The large border that surrounds the maze is not addressable by the VIC-II; the thirty-two pixel borders on the left and right and thirty-five pixel borders on the top and bottom were created in consideration of the wide variation within cathode ray tube televisions of the era. The CRT screen of different televisions framed the pixels differently, making only a subset of pixels in the center reliable for display. Running `10 PRINT` in a software emulator, of course, eliminates the need for such a border, though the Commodore 64's KERNAL nevertheless draws it.

The VIC-II also defines the way `10 PRINT` scrolls upward across the screen. The maze is programmed to loop endlessly, so there must be a contingency available for when the cursor has filled the entire screen grid with characters and there is no next row. In addition to wrapping text automatically, the VIC-II also automatically scrolls the contents of the screen when the cursor is on the bottom row and attempts to move down. Though the screen appears to scroll up two lines after hitting the last character slot on the screen, from the Commodore 64's perspective only one line is advanced; the Commodore 64's physical screen is forty characters wide, but its logical screen width is eighty characters. While the continual scrolling might seem to be intuitive, it is not necessarily the only way it could have

been done. A different environment could simply stop the program when the cursor reaches the last location on the screen, or return the cursor to the first row of the first column and begin again, overwriting the characters that had already appeared on the screen.

Designing New Characters

An intriguing feature of the VIC-II is its ability to use RAM-programmable characters instead of the PETSCII characters permanently stored in the character generator ROM. The *Commodore 64 Programmer's Reference Guide* explains how the VIC-II can be pointed to a location in RAM to use as a new character set, giving users control over "an almost infinite set of symbols" (Commodore 1982, 104). It is possible, therefore, to modify `10 PRINT`, substituting alternate `CHR$(205)` and `CHR$(206)` characters for the default PETSCII ones. Recall that the stroke of both of these characters is three pixels wide. What might a single-pixel diagonal line look like as the fundamental building block of the maze?

With the VIC-II, that question can be answered. Using the `POKE` command, a program can create and store two new bitmaps into the locations of characters 205 and 206:

```
5 PRINT CHR$(142)
10 POKE 52,48:POKE 56,48:CLR
20 POKE 56334,PEEK(56334) AND 254
30 POKE 1, PEEK(1) AND 251
40 FOR I = 0 TO 511:POKE I+12288,PEEK(I+53248):NEXT
50 POKE 1, PEEK(1) OR 4
60 POKE 56334,PEEK(56334) OR 1
70 FOR I = 0 TO 7:POKE I+12904,2^I:NEXT
80 FOR I = 0 TO 7:POKE I+12912,2^(7-I):NEXT
90 POKE 53272,(PEEK(53272) AND 240) + 12
100 PRINT CHR$(205.5+RND(1)); : GOTO 100
```

This program causes diagonal lines a single pixel thick to be substituted for the standard PETSCII characters. (The two characters are written to memory in lines 70 and 80.) After this is done, `10 PRINT` (or in this new form, `100 PRINT`) produces a maze that is remarkably similar but that neverthe-

```
{230}  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

THE SID CHIP

While the features of the Commodore 64 that made `10 PRINT` possible are chiefly BASIC, PETSCII, and the VIC-II graphics chip, it would be a disservice to the Commodore 64 to ignore another component that made the computer such a critical and popular success: the MOS Technology 6581 Sound Interface Device (SID) chip. Designed by Bob Yannes, the SID chip was a remarkable advance for its time. A three-voice synthesizer with variable pitch, amplitude, and harmonic tone controls, the SID made the Commodore a formidable music maker and game machine. With the SID, programmers could easily specify waveforms such as sawtooth or noise, as well as independently manage the attack, decay, sustain, and release times of the three oscillators (providing the three different voices) in the chip. Furthermore, the three voices could be used in conjunction with each other to create complex melodies, harmonies, and rhythms.

What is most interesting about the SID chip for the purposes of `10 PRINT` is that the third oscillator—the only of the three oscillators whose output can be fed back into the CPU—can be used for number generation. Poking SID memory location 54299 produces numbers from 0 to 255, while the waveform controls the sequence of those numbers. For example, a triangle waveform yields a cycling through every number from 0 to 255 and back down to 0, the rate controlled by the oscillator's frequency setting (Nelson 1987, 24). More relevant to `10 PRINT` is that the noise waveform produces random numbers, with the rate of the random number generation determined by the frequency of voice 3. Thus, even though the SID plays no part in `10 PRINT`, it could have a role in a similar program, and does, as evidenced by the assembly program "threadbare" that is discussed later.

less has a noticeably different appearance (see figure 60.10). The maze seems to have a sketched or stitched quality. The points on the ends of the original characters 205 and 206 are gone, so the computer screen's grid of characters is not accentuated by them. While the different lines can evoke drawing (as of a maze on paper) and craft, their more continuous nature and the greater difference between figure and ground makes the resulting output appear even more mazelike to many viewers.

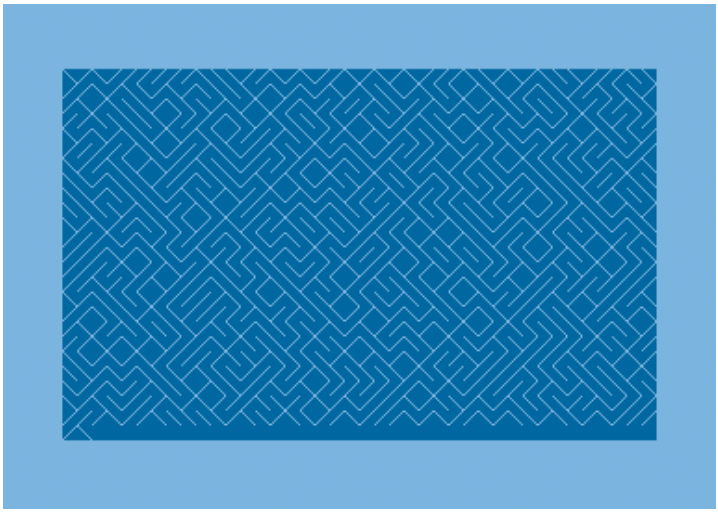


Figure 60.10
`10 PRINT` with the two standard characters replaced with custom-designed, single-pixel lines.

THE KERNAL

The various components of the Commodore 64 discussed in this book—the `RND` function, BASIC, PETSCII, the VIC-II chip—are all held together by the machine's KERNAL, its underlying operating system. A misspelling of the word "kernel" that has stuck ever since it first appeared on draft documentation for the VIC-20 (Bagnall 2010, 330), the KERNAL controls all input, output, and memory management of the Commodore 64. Any keyboard input, any screen output, any interaction at all with the computer's RAM or ROM is governed by the rules of the KERNAL. It is the brainstem of the machine, its core, its always-present, unyielding, and unchangeable center. Residing in the last eight KB of the Commodore 64's ROM ($E000–$FFFF), the KERNAL is made up of a series of input and output routines, which can be found on the "Jump Table." Any command issued to the computer in BASIC (such as the `10 PRINT` program) is "translated" by the BASIC interpreter into a language that the CPU can understand, namely assembly language, which calls routines in the Jump Table.

The KERNAL is intended to make machine language coding easier,

```
(232)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

providing a stable set of instructions and registers a programmer can address. Yet as enabling as the KERNAL may be, it is also structuring and limiting, the basis of the Commodore 64.

A View from Assembly Language

Writing a maze-generation program in BASIC leaves the programmer free from concerns about memory management, keyboard interrupts, screen outputs, and so on. All those things are provided. This is not the case when talking to the machine using a "low-level" language. In fact, Friedrich Kittler (1995) has famously argued that high-level languages essentially *obscure* the operations of the hardware. Skipping the BASIC interpreter or any other high-level language means the programmer must manipulate the microprocessor, memory, inputs, and outputs directly. Machine language itself exemplifies low-level programming, but since a machine language program is nothing but a series of numbers, it is not a very suitable language for humans. Low-level programming is typically done in assembly language instead. In assembly, the programmer provides instructions specific to the microprocessor, for example to load a value from a particular memory location into a particular processor register, or to perform a mathematical operation upon a memory location. In assembly, the programmer need not recall the numerical equivalents of such instructions, but only human-readable mnemonics for them—which are stored in the Commodore 64's KERNAL.

Recall that the microprocessor at the heart of the Commodore 64 is a modified 6502 chip. While it is not necessary to know everything about the 6502 to appreciate either the Commodore 64 or `10 PRINT`, it's worth noting that the chip essentially has three functions: it moves values between memory and one of three microprocessor registers (named X, Y, and Accumulator, abbreviated A); it executes mathematical operations on values in the accumulator; and it changes the address at which program execution takes place. The first type of operation is for loading or storing data (for example, the assignment `N = 1` in BASIC), the second type is a typical mathematical operation (say, + or − in BASIC), and the third corresponds to jumps and subroutine calls (analogous to `GOTO` and `GOSUB` in BASIC).

Like every BASIC program, `10 PRINT` is high-level. It relies on abstracted operations like `PRINT` and `RND` to perform complex tasks that

would require considerably greater effort to accomplish at a low level. For this reason, it is useful to compare the BASIC version of `10 PRINT` on the Commodore 64 with its equivalent in 6502 assembly. Doing so will help clarify what features of the program are unique to its BASIC implementation.

`10 PRINT` seems to be a "native" BASIC program, meaning it was originally written in BASIC for the Commodore 64, not first rendered in assembly and then reimplemented in BASIC. No canonical assembly program is known to exist. As with literary translation or artistic adaptation, there are multiple ways to recast a computer program from one language into another, even on a relatively simple system like the Commodore 64, and even with a relatively simple program like `10 PRINT`. Along the way to developing a production for the demoscene party @party, in June 2010, an assembly port of `10 PRINT` called "threadbare" was created.

```
    *= $1000      ; starting memory location

    lda #$80      ; set this value in:
    sta $d40f     ; the noise speed hi SID register
    sta $d412     ; and the noise waveform SID register
loop              ; label for loop location
    lda $d41b     ; load a random value
    and #1        ; lose all but the low bit
    adc #$6d      ; value of "\" PETSCII
    jsr $ffd2     ; output character via KERNAL routine
    bne loop      ; repeat
```

This short program may look arcane, even to someone familiar with BASIC. Yet it can be explained without too much difficulty, step by step, by following each instruction in the order in which it is processed.

`*= $1000`

This line tells the Commodore 64 where to put the program in memory, so that it can be run by the user. In this case, hexadecimal $1000 equals decimal 4,096, meaning the user can enter `SYS 4096` at the READY prompt to execute this program.

```
(234)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

`lda #$80`

This instruction has two parts, not counting the comment: The opcode lda and the operand $80. All instructions have at least an opcode—an operation code that corresponds to something the 6502 processor can carry out. Not all opcodes need take an operand, although all the ones in this program do. Some of these operands are a single byte long, some are two bytes long.

`lda` is the opcode for *load into the accumulator,* and when used with # it loads the numeric value of the operand. In other cases in this program, `lda` and the corresponding opcode `sta` (*store from the accumulator*) use the operand as an address. Here, no lookup occurs; the immediate hexadecimal value $80 (decimal 128) is placed into the 6502's accumulator.

`sta $d40f`

`sta $d412`

These two instructions store the value held in the accumulator (`sta`) in two different memory locations. The operand is used as an address, to look up a location in memory. These memory locations are mapped to registers of the SID, the Commodore 64's sound chip.

`loop`

While all other lines of this program are indented, the "loop" line is flush left. This is not a mere typographical convention. The assembler treats lines that begin with whitespace as instructions and lines that do not as labels, which designate positions in the program. When the assembler encounters a label such as "loop," it turns the label into a memory address that corresponds with the current position in the program. Then, on another pass through the source code, the assembler replaces references to the label with the correct sixteen-bit address. This label does not appear directly as machine code in the assembled program; the address of this location is, instead, used later, at the very end of the program.

`lda $d41b`

Once the SID registers have been initialized, every time the program loads a value from the memory address $d41b, a new eight-bit random value will be provided. This instruction does one such load, bringing a random number into the accumulator.

`and #1`

The two diagonal-line characters are neighbors on the PETSCII chart, their values differing by one. Only one bit of randomness is needed to select one or the other. Generating a random number from the SID chip provides a much larger eight-bit number, which varies between 0 and 255. In order to change this number into a single bit—either a zero or a one—this instruction shears off all but the last bit by ANDing it with the decimal value 1. For example, here the binary number 10101011 (171 in decimal) is reduced to 00000001:

```
    %10101011
AND %00000001
=============
    %00000001
```

After this instruction, the accumulator will contain either the value 1 (as in the example above) or 0 (if the last bit of the original value was 0).

`adc #$6d`

The value obtained in the previous step (0 or 1) is added in this step to the hexadecimal value $6d (decimal 109), which corresponds to the PETSCII character used in the canonical BASIC `10 PRINT`. Note that though `adc` stands for *add with carry,* this instruction won't ever perform a carry. This addition will result in either 109 or 110. The value $cd (decimal 205) could have been used instead, as this character is the same as 109.

`jsr $ffd2`

All that's left is to output the character, either 109 or 110, to the screen. This instruction jumps to a subroutine (`jsr`) at memory location $ffd2. That routine, known as CHROUT and part of the KERNAL, takes care of putting the character on the screen at the current cursor location.

`bne loop`

Until this point is reached, the program will have output only a single character. The goal, of course, is a program that prints characters continuously until the user interrupts it. This instruction branches back to the label "loop" earlier in the program, from which point execution will continue by


```
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

getting a new random value. The `bne` instruction is actually "branch if not equal," which will check to see if the processor's zero flag is set, and if not, it will complete the branch. In the case of the current program, the zero flag will never be set, so the branch will always be taken.

It would have been more straightforward to use the jump (`jmp`) instruction, assembly's equivalent of `GOTO`. However, `bne` was used because it results in a completed program that is one byte smaller. Because `jmp` can move the program counter to any location, it requires a sixteen-bit address as an operand. In contrast, `bne` can change the flow of the program to a location at most 128 bytes earlier or 128 bytes later; its operand is an eight-bit offset relative to the location of the instruction.

The completed assembly version of `10 PRINT` elucidates several features of the program from the low-level perspective of the platform. Most crucially, the high-level abstractions of the BASIC program prove to be just as abstracted in the low-level assembly rendition. There are two such abstractions of note in the original, `PRINT` and `RND`, which constitute the majority of the program's computational work. Carrying out either one in assembly by coding them "from scratch" would be a more arduous task. Consider this common routine for generating a pseudorandom eight-bit number in 6502 assembly:

```
Rand8
   lda random           ; get seed
   asl                  ; shift byte
   bcc Rand8.no_eor     ; branch if flag not set
   eor #$CF             ; otherwise literal $CF
Rand8.no_eor
   sta random           ; save next seed
```

Each assembly instruction (`lda`, `asl`, etc.) uses a single byte in the program, and in this case those instructions that have operands (`random`, `#$CF`) have one-byte operands. This results in a routine nine bytes in size, or 25 percent of the space needed for the entire `10 PRINT` program in BASIC (given that each character of BASIC takes up a byte).

While the MOS Technology 6502 processor requires this nine-byte subroutine to generate a random number, the Commodore 64 itself does not, due to a combination of seemingly unrelated affordances of its KER-

NAL and hardware. It's a simple matter with the Commodore 64 to use a random function, which although obviously used in BASIC, is found not in the BASIC ROM, but in the eight kilobytes of the Commodore 64 KERNAL, at address $e097. The assembly programmer can jump to that subroutine with `jsr $e097`, which will have the same effect as using `RND(1)` in BASIC.

A more unusual approach to random number generation—and the one that is taken in "threadbare"—involves the Commodore 64 sound chip, the SID (see sidebar). Apart from its sonic functions, the SID has the ability to generate random values. To do so, the programmer selects the noise waveform on the SID's third oscillator and sets that voice's frequency to a nonzero value. Setting a higher frequency value will cause the noise values to change more rapidly, yielding a greater variety of random numbers. The first three instructions of the preceding assembly program accomplish these settings:

```
lda #$80      ; set this value in:
sta $d40f     ; the noise speed hi SID register
sta $d412     ; and the noise waveform SID register
```

After this code has run, the program can get a new eight-bit random number by reading from memory location $d41b. While the code looks a little messier than does a simple call to `RND` in BASIC, the result is equally abstract from the programmer's perspective—it is simply abstracted to a different place, namely the SID chip instead of the KERNAL. This method of producing pseudorandom values is unusual, but certainly not unheard of. It is even documented in the *Commodore 64 Programmer's Reference Guide* (Commodore 1982, 202). Interestingly, this substitute for BASIC's `RND(1)` or the KERNAL's `jsr $e097` renders "threadbare" unusable on the VIC-20. That Commodore 64 predecessor did not include a SID chip, meaning it lacked this means of generating pseudorandom numbers. This incompatibility highlights the differences between a high-level language like BASIC, which will run `10 PRINT` on any of Commodore's computers, and a low-level language like assembly, which relies much more heavily on the specifics of the machine.

Drawing a character to the screen is an equally complex task that can prove challenging in 6502 assembly on the Commodore 64. `10 PRINT`

```
{238}  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

places every character in the maze after the previous cursor position, making the maze appear to lay itself out column by column, row by row. To reproduce this behavior manually in 6502 assembly, the programmer would seem to have considerable work: determining the start of a screen, pausing, moving ahead one more position on the screen, repeating until the screen is filled, and then implementing a scrolling mechanism.

But as with the SID random number solution, the Commodore 64's KERNAL provides a much simpler solution. One subroutine of the KERNAL sends the PETSCII character value currently in the 6502 processor's accumulator to the current output device (the screen by default). That subroutine, CHROUT, lives at memory address $ffd2, and it can be executed in assembly by jumping to that address. This is precisely what "threadbare" does, after loading a random value and manipulating it to ensure that it will be one of the two slash characters that comprise the maze:

```
jsr $ffd2   ; output character via kernal routine
```

The output of the assembly program is essentially identical to that of `10 PRINT`, although the program runs a bit more quickly because the microprocessor is receiving machine instructions directly, rather than as translations of BASIC statements. "threadbare" is shorter than its BASIC cousin (twenty-two bytes for the assembly version, compared to thirty-six bytes, or characters, for the BASIC program). While "threadbare" is clearly more esoteric and less human-readable than its BASIC predecessor, its implementation reveals that the abstraction that makes the emergent elegance of `10 PRINT`'s output possible in such a small set of instructions is not entirely a feature of the BASIC interpreter, but also depends on the underlying hardware and operating system of the Commodore 64.

Though `10 PRINT` is an example of a robust one-liner that can be reimplemented in other languages and platforms, it is a program deeply tied to the material specifications of the Commodore 64, a bestselling personal computer that played a pivotal role in establishing a place for computers and programming in certain users' homes. While discussion in this book has so far focused on the code of `10 PRINT` and its effects, this chapter reveals the imbrication of code and platform and ways in which specific code can become a means of discussing the platform and its affordances.

"THREAD," A TINY DEMOSCENE PRODUCTION

The demoscene is a programmer subculture centered on the design and manipulation of real-time audiovisual software. The origins of demoscene can be found in the cracking of eight-bit software for systems such as the Apple II, Commodore 64, and ZX Spectrum in order to remove copy protection. The individual or groups who cracked a particular piece of software would distribute the modified program with a signature of some sort (text-based or graphical) that displayed as the program loaded. Over time, these signatures began to include animated effects with sound. Eventually, productions growing from these additions were released apart from commercial software and called intros or (if they were more elaborate) demos. The hallmark of the demoscene is its emphasis on technical achievement and pushing the limits of earlier hardware systems. The demoscene also maintains interest in technically excellent systems from decades past, such as the Commodore 64: more than a hundred demos were programmed for the system in 2011 and music is continually being written for the system as well.

A demoscene production that was developed along with "threadbare" is a program called "thread"; it adds a progression through random colors to the drawing of the maze. This program, which is only thirty-one bytes long, shows some of the ways that a short assembly program can be extended. It takes advantage of some features of assembly, such as easy access to the zero page, which would have been much more difficult to incorporate in BASIC.

In "thread," the loop in the earlier program is elaborated in this way:

```
flourish
   tay
   lda ($f9),y     ; load color
   sta $0286       ; set char color
   lda $d41b       ; random
   and #1          ; lose all but low bit
   adc #$6d        ; value of one diag
   ; now either left or right diag
   jsr $ffd2       ; output character
   inx
```

```
(240)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

```
bne flourish    ; do 256 times...
inc $f9         ; shift to new region
```

10 PRINT was not intended to be a demo; it was not created within the demoscene, or with competition of any kind in mind. Nevertheless, the program's abstract, full-screen graphics bear similarity to the animated effects that characterize demoscene productions. While those features could be attributed to the canonical, BASIC version of 10 PRINT, "thread" adds a simple form of color-cycling. The method by which this small alteration in the program's visual output is accomplished likewise embraces the spirit of the demoscene. While the color shift appears dramatic (at least in the context of a simple thirty-one-byte program like this one), it is created by two assembly instructions totaling five bytes:

```
lda ($f9),y         ; load color
sta $0286           ; set char color
```

This portion of the program loads an arbitrary value from memory and stores it in the memory location that sets the character color. While far simpler than some of the feats of demoscene programs, this small act is suggestive of the competitive nature of the subculture: an attempt to produce impressive results with limited resources.

Another feature of "thread" distinguishes it from the BASIC rendition of 10 PRINT: it was written in a different social context. BASIC programming on home computers like the Commodore 64 almost always involved sharing, often through magazines and face-to-face computer club meetings. But demos are often written in the context of demoparties, events that hundreds of people may attend and that typically last several days. Participants program, socialize, share tricks, collaborate on programs, and watch and vote on the output of productions. "thread" was produced at a small-scale party of this sort.

Within the demoscene, it is a typical pastime to try to compress similar programs into less and space. Indeed, "thread" was created in the hopes of reducing the program to thirty-two bytes or below—bit-boundaries or powers of two offer popular ways to set goals for demos. There is a whole category for thirty-two byte demos on the demoscene community website pouet.net. The version of

"thread" above just makes the cut: it is thirty-one bytes—small by any reasonable measure. But subsequent to the appearance of "thread" and "threadbare," other members of the C64 demoscene community went on to fashion even smaller versions that produce the same output as `10 PRINT` in an impressive eighteen bytes. This was accomplished in the program "Thread Up," written in February 2012 by 4-Mat of the demoscene groups Ate Bit and Orb: <http://noname.c64.org/csdb/release/?id=106005>. A follow-up a few days later, in March, by Wisdom of Crescent is called "Thread Down" and squeezed the same essential effect into sixteen bytes, half our original limit: <http://noname.c64.org/csdb/release/?id=106044>. The obvious question: can you make a smaller version?

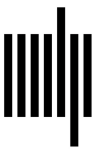
```
(242)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

10 PRINT CHR$(205.5+RND(1)); : GOTO 10

By: Nick Montfort, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample, Noah Vawter

Citation:

10 PRINT CHR$(205.5+RND(1)); : GOTO 10
By: Nick Montfort, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample, Noah Vawter
DOI: 10.7551/mitpress/9040.001.0001
ISBN (electronic): 9780262305501
Publisher: The MIT Press
Published: 2014



The MIT Press

This book was designed and typeset by Casey Reas using Avenir by Adrian Frutiger, C64 by Style, and TheSansMono by LucasFonts. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

10 PRINT CHR$(205.5+RND(1)); : GOTO 10 / Nick Montfort . . . [et al.].
p. cm.—(Software studies)
Includes bibliographical references and index.
ISBN 978-0-262-01846-3 (hardcover : alk. paper)
1. BASIC (Computer program language)—History. I. Montfort, Nick.
QA76.73.B3A14 2013
005.26'2—dc23

2012015872

10 9 8 7 6 5 4 3 2