The Hidden Language of Computer Hardware and Software

C  1000011
O  1001111
D  1000100
E  1000101

Charles Petzold
## Contents

*Preface*  

| Chapter One | Best Friends | 3  |
| Chapter Two  | Codes and Combinations | 9  |
| Chapter Three | Braille and Binary Codes | 15 |
| Chapter Four | Anatomy of a Flashlight | 22 |
| Chapter Five | Seeing Around Corners | 32 |
| Chapter Six | Telegraphs and Relays | 40 |
| Chapter Seven | Our Ten Digits | 47 |
| Chapter Eight | Alternatives to Ten | 54 |
| Chapter Nine | Bit by Bit by Bit | 69 |
| Chapter Ten | Logic and Switches | 86 |
| Chapter Eleven | Gates (Not Bill) | 102 |
| Chapter Twelve | A Binary Adding Machine | 131 |
| Chapter Thirteen | But What About Subtraction? | 143 |
| Chapter Fourteen | Feedback and Flip-Flops | 155 |
| Chapter Fifteen | Bytes and Hex | 180 |
| Chapter Sixteen | An Assemblage of Memory | 190 |
| Chapter Seventeen | Automation | 206 |
| Chapter Eighteen | From Abaci to Chips | 238 |
| Chapter Nineteen | Two Classic Microprocessors | 260 |
| Chapter Twenty | ASCII and a Cast of Characters | 286 |
| Chapter Twenty-One | Get on the Bus | 301 |
| Chapter Twenty-Two | The Operating System | 320 |
| Chapter Twenty-Three | Fixed Point, Floating Point | 335 |
| Chapter Twenty-Four | Languages High and Low | 349 |
| Chapter Twenty-Five | The Graphical Revolution | 364 |

*Acknowledgments*  

*Index*  

385
Code rattled around in my head for about a decade before I started writing it. As I was contemplating Code and then writing it, and even after the book was published, people would ask me, “What’s the book about?”

I was always reluctant to answer this question. I’d mumble something about “a unique journey through the evolution of the digital technologies that define the modern age” and hope that would be sufficient.

But finally I had to admit it: “Code is a book about how computers work.”

As I feared, the reactions weren’t favorable. “Oh, I have a book like that,” some people would say, to which my immediate response was, “No, no, no, you don’t have a book like this one.” I still think that’s true. Code is not like other how-computers-work books. It doesn’t have big color illustrations of disk drives with arrows showing how the data sweeps into the computer. Code has no drawings of trains carrying a cargo of zeros and ones. Metaphors and similes are wonderful literary devices but they do nothing but obscure the beauty of technology.

The other comment I heard was, “People don’t want to know how computers work.” And this I’m sure is true. I personally happen to enjoy learning how things work. But I also like to choose which things I learn about and which I do not. I’d be hard pressed to explain how my refrigerator works, for example.

Yet I often hear people ask questions that reveal a need to know something about the inner workings of personal computers. One such common question is, “What’s the difference between storage and memory?”

That’s certainly a critical question. The marketing of personal computers is based on such concepts. Even novice users are expected to know how many megas of the one thing and gigas of the other thing will be necessary for their particular applications. Novice users are also expected to master the concept of the computer “file” and to visualize how files are loaded from storage into memory and saved from memory back to storage.

The storage-and-memory question is usually answered with an analogy: “Memory is like the surface of your desk and storage is like the filing cabinet.” That’s not a bad answer as far as it goes. But I find it quite unsatisfactory. It makes it sound as if computer architecture were patterned after an office. The truth is that the distinction between memory and storage...
is an artificial one and exists solely because we don’t have a single storage medium that is both fast and vast as well as nonvolatile. What we know today as “von Neumann architecture”—the dominant computer architecture for over 50 years—is a direct result of this technical deficiency.

Here’s another question that someone once asked me: “Why can’t you run Macintosh programs under Windows?” My mouth opened to begin an answer when I realized that it involved many more technical issues than I’m sure my questioner was prepared to deal with in one sitting.

I want *Code* to be a book that makes you understand these things, not in some abstract way, but with a depth that just might even rival that of electrical engineers and programmers. I also hope that you might recognize the computer to be one of the crowning achievements of twentieth century technology and appreciate it as a beautiful thing in itself without metaphors and similes getting in the way.

Computers are constructed in a hierarchy, from transistors down at the bottom to the information displayed on our computer screens at the top. Moving up each level in the hierarchy—which is how *Code* is structured—is probably not as hard as most people might think. There is certainly a lot going on inside the modern computer, but it is a lot of very common and simple operations.

Although computers today are more complex than the computers of 25 years or 50 years ago, they are still fundamentally the same. That’s what’s so great about studying the history of technology: The further back in time you go, the simpler the technologies become. Thus it’s possible to reach a point where it all makes relatively easy sense.

In *Code*, I went as far back as I could go. Astonishingly, I found that I could go back into the nineteenth century and use early telegraph equipment to show how computers are built. In theory at least, everything in the first 17 chapters of *Code* can be built entirely using simple electrical devices that have been around for over a century.

This use of antique technology gives *Code* a fairly nostalgic feel, I think. *Code* is a book that could never be titled *The Faster New Faster Thing* or *Business @ the Speed of a Digital Nervous System*. The “bit” isn’t defined until page 68; “byte” isn’t defined until page 180. I don’t mention transistors until page 142, and that’s only in passing.

So, while *Code* goes fairly deep into the workings of the computer (few other books show how computer processors actually work, for example), the pace is fairly relaxed. Despite the depth, I tried to make the trip as comfortable as possible.

But without little drawings of trains carrying a cargo of zeros and ones.

*Charles Petzold*
*August 16, 2000*
code (kōd) ...

3.a. A system of signals used to represent letters or numbers in transmitting messages.

b. A system of symbols, letters, or words given certain arbitrary meanings, used for transmitting messages requiring secrecy or brevity.

4. A system of symbols and rules used to represent instructions to a computer…

— The American Heritage Dictionary of the English Language
Chapter Three

Braille and Binary Codes

Samuel Morse wasn’t the first person to successfully translate the letters of written language to an interpretable code. Nor was he the first person to be remembered more as the name of his code than as himself. That honor must go to a blind French teenager born some 18 years after Samuel Morse but who made his mark much more precociously. Little is known of his life, but what is known makes a compelling story.

Louis Braille was born in 1809 in Coupvray, France, just 25 miles east of Paris. His father was a harness maker. At the age of three—an age when young boys shouldn’t be playing in their fathers’ workshops—he accidentally stuck a pointed tool in his eye. The wound became infected, and the infection spread to his other eye, leaving him totally blind. Normally he would have been doomed to a life of ignorance and poverty (as most blind people were in those days), but young Louis’s intelligence and desire to learn were soon recognized. Through the intervention of the village priest and a schoolteacher, he first attended school in the village with the other children and at the age of 10 was sent to the Royal Institution for Blind Youth in Paris.
One major obstacle in the education of the blind is, of course, their inability to read printed books. Valentin Haüy (1745–1822), the founder of the Paris school, had invented a system of raised letters on paper that could be read by touch. But this system was very difficult to use, and only a few books had been produced using this method.

The sighted Haüy was stuck in a paradigm. To him, an A was an A was an A, and the letter A must look (or feel) like an A. (If given a flashlight to communicate, he might have tried drawing letters in the air as we did before we discovered it didn’t work very well.) Haüy probably didn’t realize that a type of code quite different from the printed alphabet might be more appropriate for sightless people.

The origins of an alternative type of code came from an unexpected source. Charles Barbier, a captain of the French army, had by 1819 devised a system of writing he called *écriture nocturne*, or “night writing.” This system used a pattern of raised dots and dashes on heavy paper and was intended for use by soldiers in passing notes to each other in the dark when quiet was necessary. The soldiers were able to poke these dots and dashes into the back of the paper using an awl-like stylus. The raised dots could then be read with the fingers.

The problem with Barbier’s system is that it was quite complex. Rather than using patterns of dots and dashes that corresponded to letters of the alphabet, Barbier devised patterns that corresponded to sounds, often requiring many codes for a single word. The system worked fine for short messages in the field but was distinctly inadequate for longer texts, let alone entire books.

Louis Braille became familiar with Barbier’s system at the age of 12. He liked the use of raised dots, not only because it proved easy to read with the fingers but also because it was easy to write. A student in the classroom equipped with paper and a stylus could actually take notes and read them back. Louis Braille diligently tried to improve the system and within three years (at the age of 15) had come up with his own, the basics of which are still used today. For many years, the system was known only within the school, but it gradually made its way to the rest of the world. In 1835, Louis Braille contracted tuberculosis, which would eventually kill him shortly after his 43rd birthday in 1852.

Today, enhanced versions of the Braille system compete with tape-recorded books for providing the blind with access to the written word, but Braille still remains an invaluable system and the only way to read for people who are both blind and deaf. In recent years, Braille has become more familiar in the public arena as elevators and automatic teller machines are made more accessible to the blind.

What we’re going to do in this chapter is dissect Braille code and see how it works. We don’t have to actually learn Braille or memorize anything. We just want some insight into the nature of codes.

In Braille, every symbol used in normal written language—specifically, letters, numbers, and punctuation marks—is encoded as one or more raised
dots within a two-by-three cell. The dots of the cell are commonly numbered 1 through 6:

1 〇〇 4
2 〇〇 5
3 〇〇 6

In modern-day use, special typewriters or embossers punch the Braille dots into the paper.

Because embossing just a couple pages of this book in Braille would be prohibitively expensive, I’ve used a notation common for showing Braille on the printed page. In this notation, all six dots in the cell are shown. Large dots indicate the parts of the cell where the paper is raised. Small dots indicate the parts of the cell that are flat. For example, in the Braille character

•

dots 1, 3, and 5 are raised and dots 2, 4, and 6 are not.

What should be interesting to us at this point is that the dots are binary. A particular dot is either flat or raised. That means we can apply what we’ve learned about Morse code and combinatorial analysis to Braille. We know that there are 6 dots and that each dot can be either flat or raised, so the total number of combinations of 6 flat and raised dots is $2 \times 2 \times 2 \times 2 \times 2 \times 2$, or $2^6$, or 64.

Thus, the system of Braille is capable of representing 64 unique codes. Here they are—all 64 possible Braille codes:
If we find fewer than 64 codes used in Braille, we should question why some of the 64 possible codes aren’t being used. If we find more than 64 codes used in Braille, we should question either our sanity or fundamental truths of mathematics, such as 2 plus 2 equaling 4.

To begin dissecting the code of Braille, let’s look at the basic lowercase alphabet:

```
ab cde fg hi j
kl mnp qrs t
uvwxyz
```

For example, the phrase “you and me” in Braille looks like this:

```

You and me
```

Notice that the cells for each letter within a word are separated by a little bit of space; a larger space (essentially a cell with no raised dots) is used between words.

This is the basis of Braille as Louis Braille devised it, or at least as it applies to the letters of the Latin alphabet. Louis Braille also devised codes for letters with accent marks, common in French. Notice that there’s no code for u, which isn’t used in classical French. (Don’t worry. The letter will show up eventually.) At this point, only 25 of the 64 possible codes have been accounted for.

Upon close examination, you’ll discover that the three rows of Braille illustrated above show a pattern. The first row (letters a through j) uses only the top four spots in the cell—dots 1, 2, 4, and 5. The second row duplicates the first row except that dot 3 is also raised. The third row is the same except that dots 3 and 6 are raised.

Since the days of Louis Braille, the Braille code has been expanded in various ways. Currently the system used most often in published material in English is called Grade 2 Braille. Grade 2 Braille uses many contractions in order to save trees and to speed reading. For example, if letter codes appear by themselves, they stand for common words. The following three rows (including a “completed” third row) show these word codes:
Thus, the phrase “you and me” can be written in Grade 2 Braille as this:

```
\[ \text{(none) but can do every from go have (none) just} \]
\[ \text{knowledge like more not (none) people quite rather so that} \]
\[ \text{us very it you as and for of the with} \]
```

Thus, the phrase “you and me” can be written in Grade 2 Braille as this:

```
\[ \text{\textbf{you and me}} \]
```

So far, I’ve described 31 codes—the no-raised-dots space between words and the 3 rows of 10 codes for letters and words. We’re still not close to the 64 codes that are theoretically available. In Grade 2 Braille, as we shall see, nothing is wasted.

First, we can use the codes for letters \textit{a} through \textit{j} combined with a raised dot 6. These are used mostly for contractions of letters within words and also include \textit{w} and another word abbreviation:

```
\[ \text{ch gh sh th wh cd er ou ow w} \]
\[ \text{(or “will”)} \]
```

For example, the word “about” can be written in Grade 2 Braille this way:

```
\[ \text{\textbf{about}} \]
```

Second, we can take the codes for letters \textit{a} through \textit{j} and “lower” them to use only dots 2, 3, 5, and 6. These codes are used for some punctuation marks and contractions, depending on context:

```
\[ \text{ea bb cc dis en to gg his in was} \]
\[ , ; : . ! ( ) “ ” \]
```

The first four of these codes are the comma, semicolon, colon, and period. Notice that the same code is used for both left and right parentheses but that two different codes are used for open and closed quotation marks.
We’re up to 51 codes so far. The following 6 codes use various unused combinations of dots 3, 4, 5, and 6 to represent contractions and some additional punctuation:

\[
\begin{array}{cccc}
\vdots & \vdots & \vdots & \vdots \\
\text{st} & \text{ing} & \text{ble} & \text{ar} & \# & \text{com} \\
/ & & & & \text{-}
\end{array}
\]

The code for “ble” is very important because when it’s not part of a word, it means that the codes that follow should be interpreted as numbers. These number codes are the same as those for letters a through j:

\[
\begin{array}{cccccccccccc}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \\
\end{array}
\]

Thus, this sequence of codes

\[
\begin{array}{cccc}
\vdots & \vdots & \vdots & \vdots \\
\text{BL} & \text{E} & \# & \text{C} \\
\end{array}
\]

means the number 256.

If you’ve been keeping track, we need 7 more codes to reach the maximum of 64. Here they are:

\[
\begin{array}{ccccccc}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g} \\
\end{array}
\]

The first (a raised dot 4) is used as an accent indicator. The others are used as prefixes for some contractions and also for some other purposes: When dots 4 and 6 are raised (the fifth code in this row), the code is a decimal point in numbers or an emphasis indicator, depending on context. When dots 5 and 6 are raised, the code is a letter indicator that counterbalances a number indicator.

And finally (if you’ve been wondering how Braille encodes capital letters) we have dot 6—the capital indicator. This signals that the letter that follows is uppercase. For example, we can write the name of the original creator of this system as

\[
\begin{array}{cccccccc}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\text{H} & \text{i} & \text{s} & \text{I} & \text{S} & \text{B} & \text{R} & \text{A} \\
\end{array}
\]

This is a capital indicator, the letter l, the contraction ou, the letters i and s, a space, another capital indicator, and the letters b, r, a, i, l, l, and e. (In actual use, the name might be abbreviated even more by eliminating the last two letters, which aren’t pronounced.)
In summary, we’ve seen how six binary elements (the dots) yield 64 possible codes and no more. It just so happens that many of these 64 codes perform double duty depending on their context. Of particular interest is the number indicator and the letter indicator that undoes the number indicator. These codes alter the meaning of the codes that follow them—from letters to numbers and from numbers back to letters. Codes such as these are often called *precedence*, or *shift*, codes. They alter the meaning of all subsequent codes until the shift is undone.

The capital indicator means that the following letter (and only the following letter) should be uppercase rather than lowercase. A code such as this is known as an *escape* code. Escape codes let you “escape” from the humdrum, routine interpretation of a sequence of codes and move to a new interpretation. As we’ll see in later chapters, shift codes and escape codes are common when written languages are represented by binary codes.
Index

Note: Page numbers in italics refer to illustrations.

Numbers
6800 microprocessor, 260–61, 281–83, 284
8080 microprocessor, 260–83, 284

A
abacus, 238
adding machines, 131–42, 148–49, 194, 207–8
Aiken, Howard, 243, 354
ALGOL programming language, 354–60, 362, 363
algorithms, 50, 236, 354–63
al-Khwarizmi, Muhammed ibn-Musa, 50
Allen, Paul, 362
Altair, 283–84, 302, 304, 362
ALU (Arithmetic Logic Unit), 232, 269
Ampère, André Marie, 28
analog computers, 231
Analytical Engine, 101, 240, 241, 362
API (application programming interface), 330–32, 371, 373, 381
Apple Computer
Apple II, 284, 366, 372
Apple Computer, continued
Apple Lisa, 370
Macintosh, 285, 334, 370, 372–73, 375, 378, 381
argument, 233
Aristotle, 86, 87, 91
Arithmetic Logic Unit (ALU). See ALU (Arithmetic Logic Unit)
Art of Computer Programming series (Knuth), 360
ASCII (American Standard Code for Information Interchange), 286–300, 311–13, 315
operating systems and, 320–21, 323–24, 327–31
assembly language, 236
associative law, 88–89, 92, 103
“As We May Think” (Bush), 364–65, 369
AT&T (American Telephone and Telegraph), 246, 248, 333. See also Bell Telephone Laboratories
Atanasoff, John V., 244
automation, 206–37
bandwidth, 310
Barbier, Charles, 16, 101, 240–42
Bardeen, John, 247
base two logarithm, 76
BASIC, 361–62
Baudot code, 288–90, 295
Baudot, Emile, 288
BCD (binary-coded decimal), 271, 292, 296–97, 338
Bell, Alexander Graham, 248, 377
Bell Systems Technical Journal, 246
Bell Telephone Laboratories, 243, 246–47, 249, 333, 362, 373, 377, 380. See also AT&T (American Telephone and Telegraph)
big-endian method, 283
binary-coded decimal (BCD). See BCD (binary-coded decimal)
binary (base two) number system, 61–85, 182
adding machines and, 131–43
automation and, 208
bytes and, 181
conversion to/from, 184–85
flip-flops and, 177
signed/unsigned numbers in, 154
switches and, 95–96
bits (binary digits). See also bytes carry, 136
least-significant (rightmost), 141, 142
logic gates and, 104
most-significant (leftmost), 141
origin of the term, 67–68
overview of, 69–85
photographic film and, 76–79, 88
sign, 153
sum, 136
use of the term, by Shannon, 103, 246

Boole, George, 87, 87, 95, 101, 129–30, 359. See also Boolean algebra
Boolean algebra, 87–103, 130, 246, 269–70, 359–60, 363. See also Boole, George
Braille, 5, 31, 242, 287. See also
Braille, Louis
basic description of, 14–21
binary digits and, comparison of, 70
simplicity of, in comparison to
Morse code, 85
Braille, Louis, 15, 15, 16, 18. See also Braille
Brattain, Walter, 247
Bricklin, Dan, 366
British Broadcasting Corporation, 7
buffers, 128–29
bugs, 236, 274, 275
Burks, Arthur W., 245
bus, 301–19
Bush, Vannevar, 364, 369, 380
Busicom, 258
Byron, Augusta Ada, 240, 362
bytes, 180–89. See also bits (binary digits)
definition of, 180
high-order (leftmost), 216–17
low-order (rightmost), 216, 217, 222

C programming language, 362–63
calculators, 188, 231, 239
Carroll, Lewis (Charles Dodgson), 86
cathode-ray tubes (CRTs). See CRTs (cathode-ray tubes)
CD-ROM (CD Read-Only Memory), 378, 380
CDs (compact discs). See compact discs (CDs)
Census data, 241–42
character sets, 286–300. See also ASCII (American Standard Code for Information Interchange)
Index

chip, 250–59

clocks, 209, 222, 263. See also oscillators
  flip-flops and, 158, 166–68, 170–78
  memory and, 191–92
  speed of, 258, 259, 261

closed architecture, 303

CMOS (complementary metal-oxide semiconductor), 251, 256

COBOL, 361

coincidence (equivalence) gate, 136

Colossus computer, 244

comments, 235–36

common connections, 34, 36

commutative rule, 88, 89

compact discs (CDs), 43, 376–78, 380

complementary metal-oxide semiconductor. See CMOS (complementary metal-oxide semiconductor)

compression, 375–76, 379–80

computability, concept of, 244, 258

conditional jumps, 228

conductors, 28, 35, 38, 39

Constitution of the United States, 40, 241

contacts, electrical, 34–35

control signals, 214–15

counters, 177–78, 232

CPUs (central processing units). See microprocessors

CRTs (cathode-ray tubes), 309–15, 320, 365, 368–70

current, 28–29

cybernetics, 246

cyberspace, 246

D

Daguerre, Louis, 40
daguerreotypes, 40
data paths, 180

De Morgan, Augustus, 129–30

De Morgan’s Laws, 129–30

decimal (base ten) number system, 47–53, 336–37
  adding machines and, 132
  alternatives to, 54–68
  bits and, 69–70
  bytes and, 181
  conversion to/from, 184–89
  flip-flops and, 177
  floating-point numbers and, 341
  hexadecimal number system and, 184–89
  subtraction and, 147

decoders, 121–22, 129, 197–200

Dickson, William, 314

Difference Engine, 101, 240

digital data, 231

Digital Equipment Corporation, 354

Dodgson, Charles (Lewis Carroll), 86

DRAM (dynamic random access memory), 308–9

Dummer, Geoffrey, 250

dynamic random access memory (DRAM). See DRAM (dynamic random access memory)

E

EBCDIC character code, 295–97, 356

Ebert, Roger, 73–74, 96

Eccles, William Henry, 161

Eckert, J. Presper, 244, 246

Eckert-Mauchly Computer Corporation, 246

Edison, Thomas, 30, 314, 375

EDVAC (Electronic Discrete Variable Automatic Computer), 245

electromagnets, 41, 44–46, 156–57, 317
  logic gates and, 106, 108–10
  memory and, 205
  electrons, 23, 27–29, 33–34, 37

Engelbart, Douglas, 369, 370
ENIAC (Electronic Numerical Integrator and Computer), 244, 245
error checking, 81, 82–83
escape code, 21
even parity, 81
expansion slots, 302

Gates, Bill, 102, 362
Gibson, William, 246
gigabyte, 202
GNU project, 334
Goldstine, Herman H., 245

Goto instruction, 226
graphical user interface (GUI). See GUI (graphical user interface)
ground, use of the term, 34–35
guard patterns, 80–81
GUI (graphical user interface), 370, 371
Gunter, Edmund, 239

Fairchild Semiconductor Corporation, 250
feedback, 155–79
Feynman, Richard, 360
file systems, 325, 333
film critics, 73–75, 85, 96
fixed-point numbers, 335–48
Fleming, John Ambrose, 243
flip-flops, 155–79, 249–50
automation and, 207, 226
chips and, 254–55
edge-triggered, 170–73, 178–79, 226–27
level-triggered, 166, 170, 173, 191
memory and, 191
floating-point numbers, 335–38
floppy disks, 318
foreign languages, 47, 181, 298, 300
Forest, Lee de, 243
Formal Logic (De Morgan), 129
FORTRAN, 354, 360, 361
French Telegraph Service, 288
frequency dividers, 175–76
function tables, 162, 169

Gates, Bill, 102, 362
Gibson, William, 246
gigabyte, 202
GNU project, 334
Goldstine, Herman H., 245

H
hard disks, 318–19
hardware, use of the term, 232
Harvard Mark I/II computers, 243
Haüy, Valentin, 16, 42
Hertz, Heinrich Rudolph, 159, 175, 310
hexadecimal (base 16) number system, 183–89, 234, 288–97, 321, 349
high-level programming language, 349–63
Hindu-Arabic (Indo-Arabic) number system, 50–53
Hoff, Ted, 258
Hollerith, Herman, 241, 241
Hollerith cards, 241–42
Hopper, Grace Murray, 243, 354

I
IBM (International Business Machines), 180–81, 242, 246, 261, 284–85, 332–33
ASCII and, 295
floating-point hardware and, 247
peripheral devices and, 303–4, 314, 317–18
punch cards, 295–97, 295, 317, 361
video displays and, 366–67
IC (integrated circuit), 250–59, 301. See also chip
information
   as a choice among two or more possibilities, 72–73
   retention of, through flip-flop circuits, 161
   theory, 246
initialization code, 323
input devices, 105, 231, 261–62
integrated circuit (IC). See chip; IC (integrated circuit)
Intel, 258–61, 284, 303, 309, 320, 327, 332, 348
International Telecommunication Union (ITU), 288
interrupts, 280–81, 332
inverters, 119, 129, 149–50, 157
   automation and, 215–16
   memory and, 195
ITU (International Telecommunication Union). See International Telecommunication Union (ITU)

J
Jacquard, Joseph Marie, 239
Jacquard loom, 239–41
Java, 381–82
Jobs, Steven, 284, 370
Jordan, F. W., 161

K
Kemeny, John, 361
Keuffel & Esser Company, 239
keyboard, 315–17, 322–24, 349, 369
Kilby, Jack, 250
Kildall, Gary, 326
kilobytes, 201–3
kinetograph devices, 314
Knuth, Donald, 360
Kurtz, Thomas, 361

L
labels, 234–35
language
   foreign, 47, 181, 298, 300
   high-level programming, 349–63
   machine (machine code), 232, 236, 321, 349–63
   speech and, 5
   use of the term, 232
last-in-first-out (LIFO) storage. See LIFO (last-in-first-out) storage
Law of Contradiction, 91
Laws of Thought, The (Boole), 101
Leibniz, Gottfried Wilhelm von, 87, 239
LIFO (last-in-first-out) storage, 273
Linux, 334
little-endian method, 283
logarithms, 76, 231, 236, 238–40, 340, 346–47
logic, 85, 86–101, 285. See also logic gates
   character sets and, 295
   tables, 194–95, 197
logic gates, 102–30, 214, 307–8
   binary addition and, 134–38
   chips and, 252–55
   flip-flops and, 159–64, 167, 169
   memory and, 191, 195, 197, 205
   semiconductors and, 249
   subtraction and, 148, 150, 151
   vacuum tubes and, 243–44, 249
Longfellow, Henry Wadsworth, 70–72
Lowell, Amy, 312

M
machine code (machine language), 232, 236, 321, 349–63
magnetic storage, 317–18
Maltin, Leonard, 74–75
Marquez, Gabriel Garcia, 5
Mathematical Analysis of Logic, The (Boole), 129
“Mathematical Theory of Communication, A” (Shannon), 246
Mauchly, John, 244
McCarthy, John, 363
Memex, 364, 380
memory, 190–205, 243, 355–56. See also RAM (random access memory)
1-bit, 167
basic description of, 231
high-level programming language and, 349
magnetic core, 245
mercury delay line, 245
microprocessors and, 261–85
operating systems and, 320, 321
peripheral devices and, 319
storage and, difference between, 319
memory-mapped I/O, 280
microprocessors, 231–32, 246–48. See also 6800 microprocessor; 8080 microprocessor
high-level programming language and, 349–50, 352
invention of, 250–59
memory and, 261–85
operating systems and, 320
peripheral devices and, 302, 303, 305, 317
single-chip, 261
two classic, 260–85
Microsoft Corporation, 102, 362
MS-DOS operating system, 332, 333, 354–55, 365, 371, 372
Windows operating system, 334, 372–73, 378
MIT (Massachusetts Institute of Technology), 103, 333, 364, 367
mnemonics, 232–33, 264, 267, 349
Moore, Gordon E., 251, 258. See also Moore’s Law
Moore’s Law, 251, 285, 309
Morse, Samuel, 9, 15, 40, 40, 42, 44, 101, 240. See also Morse code
Morse code, 1–14, 31–32, 40–46, 382. See also Morse, Samuel
binary digits and, comparison of, 70
character sets and, 287, 289
invention of, 9–10
telegraph system and, 33–39
UPC codes and, 80, 83–85
MOS Technology, 284
motherboard, 302
Motorola, 259–60, 281, 283, 285, 348, 370
MS-DOS (Microsoft Disk Operating System), 332, 333, 354–55, 365, 371, 372
Multics, 333
multitasking, 334
Murray code, 288
Murray, Donald, 288

N
nanoseconds, 253
Napier, John, 238, 239
Napier’s Bones, 239
National Semiconductor, 260
negation, 146
negative transition, 173
networks, 103, 104
Neumann, John von, 245, 245, 246, 363, 367, 372
neutrons, 23–24
Newton, Isaac, 87
nibble, definition of, 181
nines’ complement, 144–45
Nobel Prize, 247
noise, 72
Noyce, Robert, 250, 253, 258
NPN transistors, 248
number systems, 47–70, 335–48. See also specific number systems
Nyquist, Harry, 377
OCR (optical character recognition), 80, 376
octal (base eight) number system, 55–60, 63, 181–82
odd parity, 81
Ohm, Georg Simon, 29
Ohm’s Law, 39
One Hundred Years of Solitude (Marquez), 5
ones’ complement, 146, 150–51, 154
open architecture, 303
operands, 87, 92–94
operating systems, 319, 320–34, 370–71
optical character recognition (OCR). See OCR (optical character recognition)
Organon (Aristotle), 86
Orlando, Tony, 69, 71
oscillators, 157–59, 173–76, 178, 209, 222, 262
oscilloscope, 367
Oughtred, William, 239
output devices, 105, 231, 261–62

parity, 81
Pascal, Blaise, 239
Pascal programming language, 362
Paterson, Tim, 332
Pentium microprocessors, 284–85, 348.
See also microprocessors
petabyte, 202
Pfleumer, Fritz, 317
phonograph, 376
photographic film, 76–79, 85
pixels, 311–13, 367–68, 370
Polonius, 144
pop, 273
positional number systems, 50
Poulsen, Valdemar, 317
“Preliminary Discussion of the Logical Design of an Electronic Computing Instrument” (Burks, Goldstine, and Neumann), 245
printing telegraph, 288
protocols, 381
push, 273
quaternary number system, 60–61, 63
Radio Shack, 38, 39, 110, 244, 284
RAM (random access memory). See also memory; RAM arrays automation and, 208–15, 219, 220–32, 236–37
basic description of, 198–99
microprocessors and, 258, 261, 263, 272–73, 284
operating systems and, 320, 325, 328
peripheral devices and, 301–2, 304–8, 312–13, 314, 317, 319
quantities of, 201–3
as volatile memory, 205
RAM arrays, 199–205, 249, 250, 256.
See also RAM (random access memory)
microprocessors and, 263, 284
peripheral devices and, 304–5, 308
random access memory (RAM). See RAM (random access memory)
read-only memory (ROM). See ROM (read-only memory)
registers, 264–67, 275–79, 282
Remington Rand, 246, 317, 354
resistance, 28–30, 37–39
resolution, 311, 314
Revere, Paul, 70–72
Ritchie, Dennis M., 333, 362
ROM (read-only memory), 312–13, 324–25, 332

S
scanning devices, 79–81, 83
Scheutz, Edvard, 241
Scheutz, George, 241
semiconductors, 247–49, 260
Shannon, Claude, 103, 105, 130, 246
Shockley, William, 247, 249
Shockley Semiconductor Laboratories, 249
short circuits, 30
Sieve of Eratosthenes, 359, 360
signed binary numbers, 154
Siskel, Gene, 73–74
Socrates, 86, 91–92
software
  engineers, 232
  use of the term, 232
  solid-state electronics, 248
  speaker wire, 38–39
SRAM (static random access memory), 308–9
stable states, 161
stacks, 273–76, 282
static random access memory (SRAM). See SRAM (static random access memory)
Stibitz, George, 243, 246, 380
Stroustrup, Bjarne, 373
Sun Microsystems, 381
syllogism, 86, 91
“Symbolic Analysis of Replay and Switching Circuits” (Shannon), 103, 105, 130
synchronicity, 158

T
Tabulating Machine Company, 242
tabulation machines, 241–42
TANSTAAFL principle, 222
tape systems, 317–18
Technical Reference manual (IBM), 303
telegraph systems, 33–46, 101, 105–6, 242. See also Morse code
telephone systems, 72, 75–76, 242, 317
teletypewriters, 288–90
television screens, 310–11, 314
tens’ complement, 152, 153–54
terabyte, 202
Texas Instruments, 250, 251, 257
Thompson, Ken, 333
Torvalds, Linus, 334
transistors, 142, 247–50, 260–61, 285, 305. See also TTL (transistor-transistor logic)
trigonometry, 231, 236, 239
ture/false values, 85–86, 93
TTL (transistor-transistor logic), 251–56, 305, 308. See also transistors
TTL Data Book for Design Engineers, 251–56
Tukey, John Wilder, 68
Turing, Alan M., 244, 258–59
Turing Test, 244

U
Unicode, 300
UNIVAC (Universal Automatic Computer), 246, 354
Universal Product Code (UPC). See UPC (Universal Product Code)
UNIX, 246, 333–34, 362
UPC (Universal Product Code), 79–85
V
vacuum tubes, 37–38, 142, 243, 247, 249
variables, 355–56
video displays, 311–15, 321, 324, 332, 334, 349, 366–70, 372
virtual memory, 334
VisiCalc, 366–67
voltage, 27–30, 37–39, 43
flip-flops and, 157, 159
logic gates and, 107–9, 113–14, 120
Volto, Count Alessandro, 28
von Neumann architecture, 245
von Neumann bottleneck, 245

W
Warnock, John, 374
Watson, Thomas J., 242
Watt, James, 31
white space, 234
Wiener, Norbert, 246
Wilson, Flip, 371
Windows (Microsoft). See Microsoft Corporation, Windows operating system
Wirth, Niklaus, 362
Wozniak, Stephen, 284
WYSIWYG (What You See Is What You Get), 371

X
Xerox PARC, 369, 370, 372

Z
Zenith, 251
Zuse, Conrad, 243