No, I’m not interested in developing a powerful brain. All I’m after is just a mediocre brain, something like the President of the American Telephone and Telegraph Company.
—Alan Turing

Turing’s hope has not been realized; the CEO of AT&T was a pretty smart cookie. Furthermore, if Dijkstra is right, the nature of computer intelligence is constituted in the ability to manipulate symbols and not conscious choice and awareness.

However, Turing made a discovery in 1936 all the same. Obeying an algorithm is itself obeying an algorithm...which manages to be obvious, or profound, or stupid, or all three: “To follow the rules, follow the rules for following the rules.”

In the last chapter, you learned how the quickBasicEngine generates code. In this chapter, I will discuss the details of assembling code with jumps and with Go To instructions and related issues having to do with assemblers.

The compiler generates qbToken objects and stores them in a collection. These qbToken objects reference each other using labels, and in a rather clerical (but rather tricky) operation, these labels must be translated to numeric addresses.

A more exciting operation, discussed in the second half of this chapter, is the simulation of a computer by an interpreter. Rather surprisingly, even a complex computer architecture can be completely imitated by another computer architecture (even one less powerful or less complex) using software. In most cases, the simulation will be slower than a native implementation, but this is not necessarily the case when the computer doing the imitation is several orders of magnitude faster.

This chapter will discuss assembly in the context of the simple assembler embedded in the QuickBasic compiler. I will then discuss the design of the onboard Nutty Professor interpreter, a software machine for executing the qbPolish objects emitted by the compiler.

Assemblers

Let’s take a look at assemblers in general and in their historical context, and then examine the simple assembler embedded in the quickBasicEngine.
Assemblers, in General

Assemblers have been around since the earliest days of computers, although paradoxically, assemblers may postdate compilers. This is because the earliest computer scientists, including Charles Babbage, John von Neumann, and Konrad Zuse, did not work as lowly programmers. Instead, they prepared equations for the earliest programmers to enter by keying or setting switches.

Of the three pioneers I have mentioned, Konrad Zuse also developed in the early 1940s the PlanCalcul, which was a prototype of a high-level "compiled" language, and this predates the first assemblers: the first compilers predated the first assemblers (cf. *A History of Modern Computing, Second Edition* by Paul E. Ceruzzi [MIT Press, 2003]).

Later in the 1940s, Grace Murray Hopper (an early ENIAC programmer and an officer in the United States Navy) started to "reuse" the code for common equations by borrowing tapes containing ENIAC codes and lending her own code in return. This activity related more to the early compilers than to assemblers, but Hopper's team was, as noted in Chapter 1, the first to see the economic value of saving programmer time.

The first assemblers were developed by working programmers to avoid having to code in straight binary machine language. In fact, John von Neumann (the Hungarian émigré mathematician who, at Princeton's Institute for Advanced Study, is credited with the stored program concept) did not think that an expensive and rare computer should be used at all to make its programmer's life easier.

Nonetheless, the earliest commercial mainframes of the 1950s were shipped with assemblers after managers discovered that programming was much more time-consuming than originally thought, and because compilers were harder to develop at the time than assemblers (modern compiler theory not yet having been developed).

These early assemblers required the early programmer to specify actual machine operations, but allowed him or her to identify storage locations with mnemonic names. Such assemblers took over the job of assigning numeric locations to the names.

From Machine Language to Assembler Language

In January 1970, I took one of the first computer classes offered for academic credit at Roosevelt University, Chicago, which was taught by the great Max Plager, still on the math faculty at Roosevelt. As I mention in Chapter 1, this class was conducted during a lot of university upset and chaos.

Max had us students code our first program in actual machine language so we would appreciate assemblers and Fortran more.