Chapter 8
Assemblers, Disassemblers, Linkers, and Loaders

An assembler, like a compiler, is a converter from source code to target code, so many of the usual compiler construction techniques are applicable in assembler construction; they include lexical analysis, symbol table management, and back-patching. There are differences too, though, resulting from the relative simplicity of the source format and the relative complexity of the target format.

8.1 The tasks of an assembler

Assemblers are best understood by realizing that even the output of an assembler is still several steps away from a target program ready to run on a computer. To understand the tasks of an assembler, we will start from an execution-ready program and work our way backwards.

8.1.1 The running program

A running program consists of four components: a code segment, a stack segment, a data segment, and a set of registers. The contents of the code segment derive from the source code and are usually immutable; the code segment itself is often extendible to allow dynamic linking. The contents of the stack segment are mutable and start off empty. Those of the data segment are also mutable and are prefilled from the literals and strings from the source program. The contents of the registers usually start off uninitialized or zeroed.

The code and the data relate to each other through addresses of locations in the segments. These addresses are stored in the machine instructions and in the prefilled part of the data segment. Most operating systems will set the registers of the hardware memory manager unit of the machine in such a way that the address spaces
of the code and data segments start at zero for each running program, regardless of where these segments are located in real memory.

8.1.2 The executable code file

A run of a program is initiated by loading the contents of an executable code file into memory, using a loader. The loader is usually an integrated part of the operating system, which makes it next to invisible, and its activation is implicit in calling a program, but we should not forget that it is there. As part of the operating system, it has special privileges. All initialized parts of the program derive from the executable code file, in which all addresses should be based on segments starting at zero. The loader reads these segments from the executable code file and copies them to suitable memory segments; it then creates a stack segment, and jumps to a predetermined location in the code segment, to start the program. So the executable code file must contain a code segment and a data segment; it may also contain other indications, for example the initial stack size and the execution start address.

8.1.3 Object files and linkage

The executable code file derives from combining one or more program object files and probably some library object files, and is constructed by a linker. The linker is a normal user program, without any privileges. All operating systems provide at least one, and most traditional compilers use this standard linker, but an increasing number of compiling systems come with their own linker. The reason is that a specialized linker can check that the proper versions of various object modules are used, something the standard linker, usually designed for FORTRAN and COBOL, cannot do.

Each object file carries its own code and data segment contents, and it is the task of the linker to combine these into the one code segment and one data segment of the executable code file. The linker does this in the obvious way, by making copies of the segments, concatenating them, and writing them to the executable code file, but there are two complications here. (Needless to say, the object file generator and the linker have to agree on the format of the object files.)

The first complication concerns the addresses inside code and data segments. The code and data in the object files relate to each other through addresses, the same way those in the executable code file do, but since the object files were created without knowing how they will be linked into an executable code file, the address space of each code or data segment of each object file starts at zero. This means that all addresses inside the copies of all object files except the first one have to be adjusted to their actual positions when code and data segments from different object files are linked together.