7 Code Generation and Optimization

Finally, the compilation system reaches the phase of code generation. The input of the code generator is the intermediate code which was produced during syntactic or semantic analysis. The output of the code generator is a sequence of instructions which facilitates the execution of the source code by a specific hardware (see Figure 7.1).

Fig. 7.1. Code generation

Thus, code generation depends not only on the chosen intermediate representation, but also on the target hardware and the instruction set of that hardware. However, code generation is probably the most important part of a compiler, since good code can be much faster in execution than badly generated code.

The generated code can be either

- *absolute code*, or
- *relocatable code*.

Absolute code means the generation of actual machine code with complete memory addresses, while relocatable code means the generation of code with address offsets. Thus, relocatable code can subsequently be linked with other object codes, e.g. library subroutines or any other separately compiled module.
Although it is difficult to formulate general algorithms for code generation (because of the machine dependent treatment of the topic) a few aspects which can be found in almost any code generator will be introduced in this Chapter. Before, however, it is essential to consider storage allocation techniques.

### 7.1 Storage Allocation

It was shown in the previous Chapters that the declaration of a name and the subsequent usage of this name result in symbol table manipulations. For code generation the information of the symbol table must be transformed into storage addresses. Thus, storage allocation in terms of code generation means the assignment of memory space for the storage of names as well as compiler defined auxiliary variables and additional information for procedures, such as the return address.

It must be distinguished between

- *static allocation*, and
- *dynamic allocation*.

During static allocation the storage space for names will be assigned during the compilation of the source program. This means that all names of the source code are bound to a fixed storage location and, therefore, exist during the whole lifetime of the program.

In contrast to this, dynamic allocation means that storage space for a name can be allocated and deallocated during the run-time of a program. This means that the names of a procedure are not bound to the same storage location each time the procedure is activated and especially that recursive procedures are possible. While FORTRAN is a typical example of a programming language with static storage allocation, block-oriented languages like PASCAL are typical examples for dynamic storage allocation.

Dynamic storage allocation allows different techniques:

- *stack allocation*, or
- *heap allocation*.

The first is used for recursive procedures, while heap allocation is useful for handling dynamic data structures, such as linked lists or trees.