Code Generation

Part VI

Phases In a Compiler

- A modern compiler is usually partitioned into three major parts:
  1. The front-end – Responsible for validating that the program is legal in the source language.
  2. The "middle-end" – Optimizations.
  3. The back-end – Generating a program in the target language, usually some form of machine code.

More on Phases

- The first phase has to be tailored for the particular language to compile but is really independent of the machine to compile to.
- The last phase has to be tailored for the machine to compile to but is to a large extent independent of the source language.
- Some optimizations are dependent on the source language or the target language, but most optimizations are independent.

Intermediate Representation

- An intermediate representation (IR) can make the front-end and the back-end truly independent.
- This will make it possible to reuse parts of the compiler in other projects.
  - By plugging in different back-ends we can get compilers for different machines (SPARC, x86, JVM, EVM)
  - By plugging in different front-ends we can get compilers for different languages (Expressive, Java, C, C++)

Intermediate Representation

- The intermediate representation has to be carefully designed.
  - It should support constructs for several different programming languages (functions, objects, types).
  - It should be easy to map to several different target machines.
- There are several common ways to structure the IR.
  - 3-address instructions, Control Flow Graphs (CFG), Expression-Trees, Static Single Assignment (SSA)

Optimization

- The IR is ideal for optimizing on:
  - Unnecessary syntactical constructs have been removed.
  - Strange machine specific peculiarities have not been introduced.
- By using the IR for our optimization phases we can reuse optimizations in other compilers we write. ;)

Part VI

Project Note

- In the project we are not going to do any optimizations (or porting).
- Hence, for Expressive we will not use an intermediate representation.
- Instead we will generate (virtual) machine code directly.

Part VI

The Target Language

- In order to generate code we need to know the target language.
- In our project the target language is the assembly language of a virtual machine called EVM (the Expressive Virtual Machine).
- Before describing code generation in detail we will take a detour and look at the EVM.