Chapter 15: The end of it all

July 20, 2016

The endings of programming language roads: 3 more prevalent models

- Graph reduction (ML family of functional languages)
- Stack-based (Pleasant, Forth, SECD-based functional languages)
- Basic blocks: use registers effectively, place activation records generally in a stack (most compiled languages) or in a heap

Phases to final assembly

- The book suggests a plausible list:
  - Scanner (front)
  - Parser (front)
  - Semantic analysis (front)
  - Intermediate code generation (middle or back)
  - Machine independent optimization (back)
  - Target code generation (back)
  - Machine-specific code optimization (back)

Intermediate forms

- IFs
  - High level: tend to be “tree-ish”
  - Lower level: tend to be more linear (three address/quadruples are common) (LLVM, CIL, . . .)

Code generation

- The text goes through steps to generate target code for the GCD program from the beginning of the book, using an interesting combination of stack manipulation via a register formulation.

Basic blocks

- A basic block is just a set of always sequential instructions (no jumps in or out).

Register spills

- What happens when you run out of real registers? You have to move something to memory; that's called a "spill".

Address space organization

- PIC, relocatable code, executable code, and linking:
  - Position independent code needs no relocation for items in the code unit, although external references will still need some sort of scheme (import and export tables)
  - Relocatable code needs a relocation table in addition to an import table to handle locally relocatable information
  - Executable code has resolved all relocation issues and can be processed by the processor.
  - Generally this resolution process is called “linking” and is done by a “linker” or “loader” (see the discussion at the bottom of page 797 about distinctions that might be perceived for the two terms.)

Sections and segments

- Sections exist in executables as instructions to the kernel as to how to lay out an executable's segments in memory.
- Sections can be BSS, or data, or read-only data, or executable code, or symbol table information, or debug information, or thread-local storage, or whatever else the compiler writer wants. Additionally, dynamic segments can be created from operations like dlopen(3) or mmap(2); typically dlopen(3) type operations are used for shared libraries, and pure mmap(2) calls are used for memory allocation.

Dynamic linking

- Problematic solution to old problems, and its security implications are frightening. Dynamic Linking considered harmful, (for the other side, see No static linking.)