TAC: Three-Address Code

This document summarizes a simple three-address code (TAC) targeted as an intermediate code representation in a compiler. You will likely wish to change or extend this instruction set to develop a TAC language for use as an intermediate format in your IC compiler. There are many potential design choices, and you should not treat this specification as final. See Dragon 6 (especially 6.2) or EC 5 for further discussions of three-address code and other intermediate representations.

TAC is a “flat” language without nested expressions. Every instruction references or addresses three or fewer distinct variables (a, b, c, etc., more formally called addresses), constants (301, false, "hello" etc.), or labels (markers in the code: L, etc.).

Instruction Forms

There are four basic types of instructions.

- **Arithmetic and Logic Instructions.**
  Basic instruction forms include:
  - unary operators a = OP b, where OP may be a unary operator: -, !
  - binary operators a = b OP c, where OP can be
    - an arithmetic operator: +, -, /, *
    - a logic operator: &&, ||
    - a comparison operator: ==, !=, <, <=, >, >=

- **Data Movement Instructions.**
  - Copy: a = b
  - Load/store: a = *b, *a = b
  - Array load/store: a = b[i], a[i] = b
  - Field load/store: a = b.f, a.f = b

- **Branch Instructions.**
  - Label: label L
  - Unconditional jump: jump L
  - Conditional jump: cjump a L (jump to L if a is true)

- **Function Call Instructions.**
  - Call with no result: call f(a_1, ..., a_n)
  - Call with result: a = call f(a_1, ..., a_n)

(Note: this TAC design abstract the representation of parameter passing, stack frames, etc. These details will emerge when doing machine code generation.)