The linearized control flow graph.

The inner loop here is only 8 instructions long, compared to 20 in our naive L2.

We may split and merge basic blocks; add and delete blocks; change the control flow.

Like the algorithm that builds predict sets for a top one or more pass(es) over the tree.

Like static semantic checking, these pass(es) can be expressed with a machine independent AST.

Conversion to low level instructions in basic blocks of the control flow graph:

much of the decision making is driven by which values are known to be live at a given point in the flow graph (or other IF) typically uses basic blocks of the control flow graph:

The control flow graph is "straightforward" to generate from the AST:

Consider the GCD example from the Chap. 1 of the book.

One family of IFs deserves separate mention:

NB: there are no hard boundaries between these levels.

"level," or degree of abstractness:

- low
- medium
- high

It's common for a compiler to have more than one intermediate level, or degree of abstractness:

We've used special register names (a1 and a2) to pass values to subroutines.

These are assumed to be unlimited in number.

Reprise compiler phases:

23 Sept 2020

Scanner (lexical analysis)

Parser and AST generation

Semantic analysis

Code generation (intermediate and target)

Machine-independent code improvement

Dependent code improvement

"Middle end"

"Back end"

"Front end"

Character stream

Token stream

Abstract syntax tree (AST)

Instruction scheduling

Modified flow graph

Annotated AST

Token sequence

The AST (we know how to generate this now):

int main() {
  rv := r1
  rv := rv
  rv := r2
  if less than
  rv := rv
  goto
}

More on this in Chap 17 (not covered this semester)

Another view of the AST:

if you reorder instructions, the number of registers needed may change.

which virtual registers are known to be usable at a given point in the flow graph?

which values are known to be usable at a given point in the flow graph?

reschedule to fill new load delays, so long as it doesn’t change the set of instructions

which changes the set of instructions

lots of instructions, but tiny time

add and delete blocks; change the control flow

Like the algorithm that builds predict sets for a top one or more pass(es) over the tree.

Like static semantic checking, these pass(es) can be expressed with an AST.