

Assignment 10: removing structured control flow and 3-operand statements

ECEN 4553 & 5013, CSCI 4555 & 5525
Prof. Jeremy G. Siek

November 12, 2007

This week's assignment brings us closer to x86 assembly in two steps.

1. First we remove structured control-flow statements such as `if` and `while`. They will be replaced by `goto` statements and conditional `goto` statements.
2. The second step is replacing 3-operand statements with 2-operand statements, to better match the instruction format for x86.

1 Removing Structured Control Flow

The `goto` statement may be considered harmful [1], but modern computer architectures don't understand `if` statements and `while` loops, so it is the job of the compiler to translate `if` statements and `while` loops into `goto` statements, which closely correspond to jump instructions in x86 assembly.

First we consider removing the `if` statements. The `if` statement is responsible for two things: evaluating and testing a condition, and then transferring control to either the "then" or "else" branch. We will arrange it so that the condition expression always evaluates to an integer, which we will store into a temporary t_0 . We can then test if t_0 is zero and then goes to the "else" branch, and otherwise fall through to the "then" branch. The combination of an `if` statements that tests for zero with and a `goto` should be represented as a single AST node, and corresponds to a conditional jump in x86. When control reaches the end of the "then" branch, we don't want to fall through and execute the "else" branch, so we place a `goto` at the end of the "then" code that jumps over the "else" code. The `goto` should be represented as an AST node that corresponds to an unconditional jump in x86. The following shows the translation, in general, from an `if` statements to a combination of `gotos` and conditional `gotos`.

		$t_0 = e_0$
		if ($t_0 == 0$) goto ELSE0;
if (e_0)	\implies	<i>thenstmt</i>
<i>thenstmt</i>		goto ENDO;
else		ELSE0:
<i>elsestmt</i>		<i>elsestmt</i>
		END0:
		;

The while loop can also be translated using a combination of conditional and unconditional jumps. The code below shows the style of translation that is used by the GNU C compiler. There are, of course, many other possibilities. In this approach, the loop condition is placed after the loop body, so the first line is an unconditional jump to the condition code. After e_0 is evaluated, there is a conditional jump to the top of the loop body if the condition is true. Otherwise, execution falls through, exiting the loop.

		goto WHILECONDO;
		WHILEBODY0:
while (e_0)	\implies	<i>body</i>
<i>body</i>		WHILECONDO:
		$t_0 = e_0$
		if ($t_0 != 0$) goto WHILEBODY0;

New AST nodes:

```
# if (cond) goto label;
class ConditionalGoto:
    def __init__(self, cond, label):
        self.cond = cond
        self.label = label

# goto label;
class Goto:
    def __init__(self, label):
        self.label = label

# label:
class Label:
    def __init__(self, label):
        self.label = label
```

2 Removing 3-operand Instructions

The x86 instruction set is comprised of mostly 2-operand instructions, where at least one of the operands is a register and the other may be an address (location in memory) or a register. In our current flattened AST, our statements primitive operations, such as integer addition, in the following form:

```
int x, y, z;  
...  
x = y + z
```

This corresponds to a 3-operand instruction (two input and one output). So we need to transform statements like the above into two statements:

```
int x, y, z;  
...  
x = y  
x += z
```

In general, the conversion from 3-operand instructions to 2-operand instructions is achieved by the following transformation.

$$x = e_0 + e_1 \implies \begin{array}{l} x = e_0 \\ x += e_1 \end{array}$$

You do not need to do this translation for non-primitive operations, such as adding two pyobjs, because those operations are implemented in the C runtime functions, and should already be expressed as function calls (CallFunc nodes).

References

- [1] E. Dijkstra. Go to statement considered harmful. pages 27–33, 1979.