Recitation 5: SSA

23 February

Static Single Assignment Form

Recall the Fibonacci sequence:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \qquad\qquad n > 1 \end{aligned}$$

Check out this lil program that computes the nth Fibonacci number:

```
int fib(int n) {
  if (n == 0) return 0;
  int a = 0;
  int b = 1;
  int i = 1;
  while (i < n) {
    int c = b;
    b = a + b;
    a = c;
    i++;
  }
  return b;
}
```

Checkpoint 0

Translate this program into abstract assembly, organized as basic blocks with parametrized labels.

Checkpoint 1

Use generation counters to convert this basic block assembly into SSA form.

Checkpoint 2

Rewrite the SSA program using Φ-functions instead of parametrized labels (except for the first basic block fib).

If the parametrized label `foo`(x_i) : can be jumped to from 2 different lines **goto** `foo`(x_j) and **goto** `foo`(x_k), then we would switch to a non-parametrized label `foo` : but add the instruction $x_i \leftarrow \Phi(x_j, x_k)$ to the start of the basic block under `foo` :.

Checkpoint 3

Now minimize the Φ-function SSA program. Recall that we do this by repeatedly removing instructions of the form

$$t_i = \Phi(t_{x_1}, ..., t_{x_k})$$

whenever there exists a j such that all the x_n are either i or j, then replacing all instances of t_i with t_j.

Checkpoint 4

A very useful optimization that is made easy to implement by transforming programs into SSA form is copy and constant propagation. Since by definition each variable is only defined once in the program, whenever we see

- $x \leftarrow c$, we can replace all instances of x with c
- $x \leftarrow y$, we can replace all instances of x with y

Additionally, although it will not come up on the specific example we're working on for this checkpoint, whenever we see

- $\Phi(c, c)$, we can replace it with c
- $\Phi(x, x)$, we can replace it with x

Now apply this optimization to the minimized Φ-function SSA program from above.

Checkpoint 5

Convert the optimized Φ-function SSA program back into abstract assembly without Φ-functions, aka the de-SSA transformation.

For every occurence of $x \leftarrow \Phi(y, z)$ at the start of some basic block b, delete it, and instead insert $x \leftarrow y$ and $x \leftarrow z$ respectively at the end of each of b's predecessor blocks. Note that this will result in x having multiple assignment sites, hence why the program is no longer in Static Single Assignment (SSA) form.

Tips and Hints for Lab3

- **Header Files in L3:** Unlike in C, header files in L3 (and above) are only used to declare types and external functions. If a function is declared in a header file, then it may not be defined in the program – it is declared as *external*. External functions are defined in C source files, which are linked together with the assembly produced by your compiler.
- **RBP:** You are not required to use %rbp as a base pointer, so you are allowed to treat it like a normal callee-saved register in your compiler.
- **Code Review:** Code Review happens one week after Lab3 is due. So if you haven't polished the style of your compiler and added a README describing the design of various passes of your compiler, now would be a good time to start. We are looking for good coding style and comments, modular design, and that both of you are familiar with all components of the implementation.
- **SSA:** While SSA is an optimization that you won't need to (and probably shouldn't) implement until Lab 5, it's good to have an understanding of what SSA is and how it behaves. But as usual, just work on getting a correct compiler that runs reasonable well, you'll get to work entirely on optimizing your compiler for Lab 5!