EECS 665
Stephen Longofono
Lab 08 Writeup

Program Operation

The provided program works by invoking the cgen.y grammar to translate csem intermediate code into valid x86 assembly code. The test.c file contains a number of declarations for external assembly routines, which are defined by the grammar in test.t.s. On running the Makefile, the compiled parser uses a translation scheme approach to translate the given csem into an assembly file. This assembly file is linked into the executable generated for test.c, running the code we have generated.

Translation

Translating the code involved writing the translation scheme in the cgen.y file. For each of the specified operations in csem, I wrote code to move the first operand into the accumulator, and then print the assembly instruction with the second operand as an argument to perform the operation and implicitly place it in the accumulator.

Implementing functions was a little confusing until I reviewed the notes on project 2 about formals and function calls in intermediate code. C function calls are translated into formal argument declaration, call, and return. The first requires pushing the given formal ID onto the stack. The second requires storing the ID in the accumulator, printing a call instruction to that ID, and making room on the stack for the specified number of integers. This call instruction uses a relative address operator since the given address is not in reference to the program counter. Finally, the return code simply shifts the given value into the accumulator.

Difficulties

Dealing with x86 documentation was a nightmare; complete, correct, or readily understood: pick 1. That said, there was also some confusion about how the parser was written. I figured it out eventually, but with the given test code, it was possible to flip around the operands for subl and still get a correct answer. It would be more instructive in the future to put two separate instructions to draw attention to the difference.

I tried to use gdb to do some debugging for function calls, but it wasn't much help beyond identifying the specific instruction that failed. It was not clear how to use the given address for calling, but I figured it out with stackexchange and trial and error.