Jason Fong

Brent Lee

Final Paper

Our compiler implementation includes a 32-bit architecture (everything includes integers, booleans, etc. are all represented in 32 bits). For the front end of the compiler, a file written in the Mini language gets put into the lexical analyzer. The parser, which is the next step, outputs the information in tokens as an Abstract Syntax Tree (AST). The static semantics that are used after the parsing, when given the AST, basically returns nothing, but prints out an error and exits if the static type-checking finds a type error. The method we implemented is passing down the return type all the way through functions, and anytime a return statement is found, it compares the expected return type and the expression in the return statement. All functions require a return statement with the only exceptions of expecting a void type. If there are branches, the type checker had to pass the return type through the true body and the false body whether it be for conditional statements or while statements. The return type gets passed down to every part of the code. The output of the entire AST then gets turned into a control flow graph, which is the intermediate representation before becoming assembly code.

The purpose of the control flow graph is to “show the flow” of the program. It is a directed graph where each “node” (block) is set up to contain a set of instructions up until the flow is “disturbed” such as the case with a conditional statement and a while statement. An intermediate representation (IR) is used in compilers so that the compiler systems, such as LLVM in our compiler’s case, can be used by different source languages to generate code for many different target architectures. Static single assignment (SSA Form or SSA) is a representation of the code such that each variable is assigned exactly once. A benefit of SSA is that at the end of a conditional, where the “if block” ends and the “else block” ends, no matter what path it takes, a variable would only have to be using the same variable and a phi instruction, which checks which path the code took. Another benefit is the simplification of the IR to help make easier some optimizations. We did not implement any optimizations for the compiler; however, we did implement the code generation and register allocation. For this to happen, we had to eliminate the phi instruction used in LLVM. To do this, we had to add a temporary register at the bottom of each previous block if there is a conditional or a break in the flow. That register was then used in the current block for assignment and use. The phi instruction had to be eliminated because ARM does not contain any phi instructions.