CSC488
ASSIGNMENT 5
CODE GENERATOR

DANIEL BLOEMENDAL
OREN WATSON

Contents

1. Instructions

1.1. **Bounds checking.**

1.2. **Code dump & syntax highlighting.**

2. Design

2.1. **Overview.** The overarching theme in the design of the code generator was to avoid exposing the code generator class `CodeGen` to the complexities and finer details of the underlying machine. To that end, an assembler was developed that hides the complexity of addressing code via a label system and provides an enhanced instruction set, simplifying the emitted code in `CodeGen`. It should be noted that the assembler is also entirely decoupled from the rest of the code generator and stands on its own. The assembler is covered in more detail in `doc/ASSEMBLER.pdf`. In addition, the complexities of managing major scopes, their displays, and ensuring that minor scopes are merged into their enclosing major scopes, is dealt with by the `Frame` and `Table` classes.

2.2. **Calling Convention.** A key issue that we debated at length was the method by which function and procedure calls would be implemented. We decided that the most important thing in functions was for the calling code to have the result value at the top of the stack when it returns, with no cleanup, so that the result could immediately be worked with. Therefore, the function cleans up its own arguments and places its result into a reserved place at the bottom of the call frame.

To call a function, the calling code reserves a place for the result, pushes the return address and arguments, and then jumps to the function code. The function code then saves the display pointer for its level, sets the display, and allocates space for its locals. The frame therefore has the following structure, shown in figure 1, with the stack growing up. It should noted that the base address $D[LL]$ refers to the display set during the prolog of a function via the `SAVECTX LL` intermediate instruction. Here, LL refers to the lexical level of the major scope.

Figure 1. Call frame

Result	$D[LL] - N - 3$
Return address	$D[LL] - N - 2$
Argument 1	$D[LL] - N - 1$
...	...
Argument N	$D[LL] - 2$
Previous display	$D[LL] - 1$
Locals	$D[LL]$