CS 241 Week 9 Tutorial

Code Generation

Spring 2018

1 Code Generation Conventions and Tips

These are a collection of non-mandatory tips which will most likely make it easier to write your code generator.

- All expressions should store their result in $3. This will make it much easier to combine small expressions into larger ones.
- When evaluating expressions, store any intermediate results on the stack rather than in registers. This is easier to reason about and makes it much harder to make mistakes.
- Define some helper functions that return segments of code that you will need to generate often. In particular, you may want functions which generate MIPS code for push and pop.
- Store the value 4 in $4 and 1 in $11, or some other suitable registers. You'll use these constants on a regular basis and it helps to have them consistently available.
- Output comments as part of your assembly code. It is extremely difficult to debug generated assembly, and even comments such as "addition starts/ends" or similar will help with debugging.

2 Pre- and Post-Increment Code Generation

Recall that C and C++ have the pre- and post-increment operators `++i` and `i++`. Suppose we added the following rules to the WLP4 grammar:

`factor` → `PLUS PLUS lvalue`

`factor` → `lvalue PLUS PLUS`

We will assume that scanning, parsing, and semantic analysis all work out, and that these operators can only be used on `INT` types. Write pseudocode to generate the correct MIPS output for each of these grammar rules.

3 Switch Statement Code Generation

Recall that C and C++ also have switch statements. Suppose we wished to add a similar statement with a slightly different syntax to WLP4:

```
switch(expr) {
    case(expr) {
        statements
    }
    case(expr) {
        statements
    }
    ...
    default {
        statements
    }
}
```

Here, case statements don't fall through, and the default case is mandatory. Furthermore, each case can contain an arbitrary expression rather than a constant. Write pseudocode to generate MIPS assembly code for the following production rules, once again assuming that all of scanning, parsing and semantic analysis are already handled.

$$
\begin{aligned}
\texttt{statement} &\rightarrow \texttt{SWITCH LPAREN expr RPAREN LBRACE cases default RBRACE} \\
\texttt{cases} &\rightarrow \texttt{cases case} \\
\texttt{cases} &\rightarrow \epsilon \\
\texttt{case} &\rightarrow \texttt{CASE LPAREN expr RPAREN LBRACE statements RBRACE} \\
\texttt{default} &\rightarrow \texttt{DEFAULT LBRACE statements RBRACE}
\end{aligned}
$$