1 Code Generation

Consider the following grammar, which has been extended from that of Tutorial 8:

\[
\begin{align*}
\text{start} & \rightarrow \text{BOF} \text{ terns} \text{ EOF} \\
\text{terns} & \rightarrow \text{tern} \text{ COMMA} \text{ terns} \\
\text{terns} & \rightarrow \text{tern} \\
\text{tern} & \rightarrow \text{IF} \text{ terns} \text{ THEN} \text{ terns} \text{ ELSE} \text{ terns} \text{ ENDIF} \\
\text{tern} & \rightarrow \text{assign} \\
\text{assign} & \rightarrow \text{ID} \text{ BECOMES} \text{ assign} \\
\text{assign} & \rightarrow \text{expr} \\
\text{expr} & \rightarrow \text{expr} \text{ PLUS} \text{ term} \\
\text{expr} & \rightarrow \text{expr} \text{ MINUS} \text{ term} \\
\text{expr} & \rightarrow \text{term} \\
\text{term} & \rightarrow \text{term} \text{ EXPT} \text{ factor} \\
\text{term} & \rightarrow \text{factor} \\
\text{factor} & \rightarrow \text{LPAREN} \text{ terns} \text{ RPAREN} \\
\text{factor} & \rightarrow \text{ID} \\
\text{factor} & \rightarrow \text{NUM}
\end{align*}
\]

This grammar represents mathematical equations involving +, −, and ^ (exponentiation), allows both constants (NUM) and variables (ID), and allows:

- Assignment to variables (such as \(a = 3 + 4\)).
- If statements, where 0 is considered false and other numbers are considered true.
- Sequences of expressions \((a, b)\) computes \(a\) and then \(b\), and returns \(b\) as the result.

The variables \text{arg1} and \text{arg2} contain the two arguments supplied to \text{cs241-twoints}. All other variables are initialized to 0.

For example, a program in this language might look like this:

\[
\text{if} (a = 3) - \text{arg1} \text{ then } b = 3, a + b \text{ else } a - (b = 4)
\]

This program returns 6 \((3 + 3)\) if \text{arg1} \neq 3, and \(-1\) \((3 - 4)\) if \text{arg1} = 3.
Modify your solution to last week’s tutorial to:

- Support a format similar to .wlmi (see example below) where terminals are included in the input.
- Add code that supports the new language features.
- Change the behaviour of the factor → ID rule to work with variables rather than simply returning 1.

1.1 Helpful conventions

You may find the following conventions and starting points helpful:

- All variables are stored on the stack at the beginning of the program.
- $29 should point to just below the first variable on the stack, so that the first variable’s offset from $29 is −4.
- If labels are called elseifᵢ and endifᵢ whereᵢ ∈ {0, 1, . . .} is different for every if.

1.2 Example

Given the following input:

```
start BOF expr EOF
BOF bof
expr expr PLUS term
expr term
term factor
factor ID
ID arg1
PLUS +
term factor
factor ID
ID arg2
EOF eof
```

(This represents arg1 + arg2)

You might produce the following code (but will likely produce something significantly longer):

```
add $3, $1, $2
jr $31
```