Problems and solutions:

Below is a list of some problems encountered during the project and how we approached to solve them.

Problem:

SPROCKELL only has 5 registers, expressions cannot parse values to their parent expressions using only 5 registers. The expression (1+(1+(1+(1+(1+(1+1)))))) will need more than 5 registers when using registers to parse values.

Solution:

Use the stack to parse values of expression to parent expressions using push to store the value of an expression and pop to retrieve the value of a child expression. The stack has a size of 128. This means that the compiled SPROCKELL code will fail when expressions are used with a depth of more than 128. However, it is safe to assume that this will never be case.

Problem:

Using push and pup to parse the values of expression will cause unnecessary SPROCKELL instructions for parsing values to parent expressions.

For example, the expression (1+1) will be compiled to the following instructions:

Const 1 RegA, Push RegA, Const 1 RegA, Push RegA, Pop RegA, Pop RegB, Compute Add RegB RegA RegA

The instructions Push RegA, Pop RegA have zero effect when executed like this and can be removed

Solution:

We’ve solved this problem by letting each compilation of an expression node store its value to regA and then push regA to the stack.

The parent expression can then choose to remove the last instruction of an child expression(this will always be the push instruction) and use regA as the value of the child expression.

Problem:

Jump/branch statements have to know to which target it has to jump, which are only know after these statements are added to the program.

Solution:

We have solved this by keeping a reference to the Target object that was added to an instruction as an operand (initially pointing to -1). This target object is updated when the target index is known which causes the target in the instruction to be updated as well because it is the same object.

Problem:

When implementing break/continue statements we had to find a way to make it compatible with nested while/for loops.

Solution:

Maintain two stacks of targets: one for continue targets and one for break targets. When visiting while/for statements, targets get pushed onto the stacks. At the end of the visit the targets get popped from the stacks. Using the peek() method, break/continue statements can get the correct targets of the correct while/for statement.