Report - Phase 5: Code Generation

Yongun Seong 2017-19937

In this final phase, I implemented code generation, finally producing working assembly for SnuPL/2. Given that I was am not performing any optimization, this was mostly straightforward. In fact, the most challenging part was identifying and fixing all the bugs I had introduced in the previous phases.

Notably, I had to fix bugs related to correctly parsing procedures that call themselves, if/while blocks only generating the first statement, empty returns, and parsing parameters for procedure calls.

Once I had resolved all the identified issues, I implemented actually emitting the assembly, where all I had to do was fill out the marked out comment sections. The most interesting part was figuring out the structure of the stack frame.

My stack frame was structured like so, each block being 8 bytes:

```
[saved parameters N-7]
--- previous stack frame ---
return address
saved rbx
saved r12
saved r13
saved r14
saved r15
saved rbp <-- %rbp
[saved parameters 1-6]
[local variables] <-- %rsp
[downstream arguments]
```

When making a procedure call, the calling function sets the argument registers for parameters 1-6, and puts the rest of the parameters on the stack, in decreasing order. That is, just before the function call, the 7th parameter is at `(%rsp)`, the 8th parameter at `8(%rsp)`, and so forth. As arrays are passed as pointers, no parameter is ever larger than 8 bytes, and to make things simpler, I chose to allocate the full 8 bytes for each parameter. This produces self-consistent code, though it may break when calling external functions that take more than 6 arguments. Thankfully none of the given library functions take that many arguments.

On the caller side, each procedure prologue quickly puts each callee-saved register onto the stack, with `%rbp` being the last, and then sets `%rbp` to the top of the stack. This allows us to easily restore the stack by `mov %rbp, %rsp` when leaving the functio call, without having to keep track of the size of the current stack frame at all. Afterwards, the callee decrements `%rsp` to allocate space for the current frame, and puts the register-passed passed arguments to `8(%rbp)`, `16(%rbp)`, and so on.