CSC 526, Spring 2020, Assignment 4

Purpose: Booleans and Conditionals

Due: 11:59pm, Wednesday, February 26, 2020

Get the assignment code

These instructions assume that your course git repository is set up. Change into your course repository directory and enter the following commands.

```
git fetch assignments
git checkout assignments/master -- assignment4
git add assignment4
git commit -a
```

This will copy the `assignment4` directory into your working directory, start tracking the files in the `assignment4` directory, and commit those files to your local git repository.

Assignment Description

The purpose of this assignment is to augment the language with a boolean type and conditional expressions.

Syntax

```
<program> :=
    | (<expr>)

<expr> :=
    | <integer>
    | <identifier>
    | true
    | false
    | (let (<bindings>) <expr>)
    | (if <expr> <expr> <expr>)
    | (add1 <expr>)
    | (sub1 <expr>)
    | (+ <expr> <expr>)
    | (- <expr> <expr>)
    | (* <expr> <expr>)
    | (< <expr> <expr>)
    | (> <expr> <expr>)
    | (== <expr> <expr>)

<bindings> :=
    | (<identifier> <expr>)
    | (<identifier> <expr>) <bindings>
```

Representation of Values

The representation of values requires a definition. We will use the following representations to keep track of type information at runtime:

- `true` will be represented as the constant `0x0000000000000006`
- `false` will be represented as the constant `0x0000000000000002`
- numbers will be represented with a `1` in the rightmost bit, with the actual two's complement value shifted to the left by one.

Optional error handling cases:

- Unable to parse non-representable integer literal error: A literal integer in the source program, say `x`, is outside the range -2^{62} to $2^{62}-1$ (the range of 63-bit signed two's complement integers), should output "Non-representable integer `x`".

Note: the next assignment will perform runtime type checking, so you do not need to worry about that for this assignment.

New Assembly Instructions

The assembly instructions that you need for this assignment are:

- `label:` – create a location that can be jumped to with `jmp`, `jne`, and other jump commands
- `cmp <arg2> <arg1>` – compares the two arguments for equality. This sets the condition code in the machine to track if the arguments were equal, or if the left was greater than or less than the right. This information is used by conditional jump instructions.
- `jne <label>` – If the condition code says that the last comparison (`cmp`) was given equal arguments, then do nothing. Otherwise, immediately start executing instructions from the given label.
- `je <label>` – If the condition code says that the last comparison (`cmp`) was given non-equal arguments, then do nothing. Otherwise, immediately start executing instructions from the given label.
- `jmp <label>` – Unconditionally start executing instructions from the given label.
- `jl <label>` – Jump if the last comparison said the first value was less than the second. Otherwise, do nothing.
- `jg <label>` – Jump if the last comparison said the first value was greter than the second. Otherwise, do nothing.
- `jo <label>` – Jump if the last operation set the overflow condition code. Otherwise, do nothing.
- `jno <label>` – Jump if the last operation din not set the overflow condition code. Otherwise, do nothing.

Code Generation for If Expressions

When generating code for an if expression, we need to execute exactly one of the branches. A typical structure for doing this is to have two labels: one for the else case and one for the end of the if expression. The shape of the generated code may look like:

```
  cmp $rax, 0
  je else_branch
  ; commands for then branch
  jmp end_of_if
else_branch:
  ; commands for else branch
end_of_if:
```

Note that if we did not put the `jmp end_of_if` after the commands for the then branch, control would continue and evaluate the else branch as well.

When creating labels, we cannot repeatedly use the same label names. So, we need a way to generate new label names that will not conflict with existing label names. One way to do this is to make a function that takes a string and appends the value of a counter to it, which is increased during each call. This will work provided that the base strings passed to the function do not have numbers at the end. Using this idea, the example code from before might look like:

```
  cmp $rax, 0
  je else_branch1
  ; commands for then branch
  jmp end_of_if2
else_branch1:
  ; commands for else branch
end_of_if2:
```

Turning in the Assignment

To turn in the assignment execute the following git commands from within your repository:

```
git add <file>
git commit -a
git push origin master
```

where a `git add <file>` command is needed for every file that is required for building the assignment executable. Failure to add any required files will result in a failing grade for the