3 Runtime library

We require the code you produce to be able to interface with the runtime we provide, and to interoperate with other functions we may create for testing. For this reason we require you to follow the ABI specification, and in particular to implement System V calling conventions. You've already done most of the work required to meet the ABI spec in PA4. In this assignment, you'll take care of the details that were kept abstract in the IR. In particular, you will need to generate code that respects ABI rules about caller- and callee-saved registers.

4 Quality of assembly code

We do not expect you to implement optimizations or high-quality register allocation for this assignment; the goal here is to produce working programs. It's fine to spill every `TEMP` in a function to the stack. However, we do expect you to implement nontrivial *instruction selection*. Your tiles should make use of x86-64 instruction set features like complicated addressing modes and in-memory operands.

5 Assembling your code

It may help you to take a look at the assembly lab and its corresponding example code to get a better idea of how to assemble and link your generated assembly code.

In particular, in the released runtime/ there is a script linketa.sh that should be useful for assembling your code as follows:

```
./linketa.sh -o binary foo.s
```

Running that command in the VM generates a binary file called `binary` which you can run by running

```
./binary
```

6 Command-line interface

A general form for the command-line interface is as follows:

```
etac [options] <source files>
```

Unless noted below, the expected behaviors of previously available options are as defined in the previous assignment. `etac` should support any reasonable combination of options. For this assignment, the following options are possible:

- `--help`: Print a synopsis of options.
- `--lex`: Generate output from lexical analysis.
- `--parse`: Generate output from syntactic analysis.