# Lab 4

Computer Architecture I ShanghaiTech University

## Goals

These exercises are intended to give you more practice with

- calling conventions, including prologue and epilogue.
- manipulating stack pointer in RISC-V.

Download source code from here.

## Exercises

### Exercise 1: calling convention checker

This exercise uses the file `ex1.s`.

A quick recap of RISC-V calling conventions: all functions that overwrite registers, which are preserved across function calls by convention must have a prologue. This prologue saves those register values to the stack at the start of the function. Along with the prologue is the epilogue, which restores those values for the function's caller. You can find a more detailed explanation along with some concrete examples in these notes.

Bugs due to calling convention violations can often be difficult to detect manually, so Venus provides a way to automatically report some of these errors at runtime.

Take a look at the contents of the `ex1.s` file, particularly at the `main`, `simple_fn`, `naive_mod`, `mul_arr`, and `helper_fn` functions. Enable the CC checker in Venus-Settings-Calling Convention, then run the program in the simulator. You should see an output similar to the following:

```
[CC Violation]: (PC=0x00000080) Usage of unset register t0! editor.S:57 addi a0, t0, 2025
[CC Violation]: (PC=0x0000008C) Setting of a saved register (s0) which has not been saved! editor.S:78 mv s0, a0
[CC Violation]: (PC=0x00000094) Setting of a saved register (s0) which has not been saved! editor.S:81 sub s0, s0, a1
......
```

Find the source of each of the errors reported by the CC checker and fix it. You can find a list of CC error messages, as well as their meanings, in the Venus reference.

Once you've fixed all the violations reported by the CC checker, the code might still fail: this is likely because there's still some remaining calling convention errors that Venus doesn't report. Since function calls in assembly language are basically just jumps, to avoid false positives, Venus can't report these violations without more information.

The fixes for all of these errors (both the ones reported by the CC checker and the ones it can't find) should be added near the lines marked by the `FIXME` comments in the starter code.

**Note:** Venus's calling convention checker will not report all calling convention bugs; it is intended to be used as a sanity check. Most importantly, it will only look for bugs in functions that are exported with the `.globl` directive - the meaning of `.globl` is explained in more detail in the Venus reference.

### Action Item

Resolve all the calling convention errors in `ex1.s`, and answer the following questions:

- What causes the errors in `simple_fn`, `naive_mod`, and `mul_arr` that were reported by the Venus CC checker?
- In RISC-V, we invoke functions by jumping to them and storing the return address in the `ra` register. Does calling convention apply to the jumps to the `naive_mod_loop` or `naive_mod_end` labels?

- Why do we need to store `ra` in the prologue for `mul_arr`, but not in any other function?
- Why wasn't the calling convention error in `helper_fn` reported by the CC checker? (Hint: it's mentioned above in the exercise instructions.)

### Testing

After fixing the errors with `FIXME` in `ex1.s`, run Venus locally with the command from the beginning of this exercise to make sure the behavior of the functions hasn't changed and that you've remedied all calling convention violations.

Once you have fixed everything, running the above Venus command should output the following:

```
Sanity checks passed! Make sure there are no CC violations.
Found 0 warnings!
```

#### Checkoff

- Show your TA your code and its test run.
- Provide answers to the questions. We encourage you to prepare your answers in advance before the lab check, but please refrain from directly reciting text materials during the lab check. Points will be deducted if there is any direct recitation of text materials written by yourself.

## Exercise 2: RISC-V recursion with `greatest_common_divider`

This exercise uses the file `ex2.s`.

In this exercise, you will complete an implementation of greatest common divider algorithm with recursion in RISC-V.

You will find it helpful to refer to the RISC-V green card to complete this exercise. If you encounter any instructions or pseudo-instructions you are unfamiliar with, use this as a resource.

Besides the common RV32I instructions, in this exercise, RV32M instruction set are also supported. Additionally, we provide you with the reference implementation of the greatest common divider algorithm (citation).

```
int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
```

Due to the fact that the samantics of C and RISCV assembly differ, we do not require that your code has the same semantics as the C code, given that your code roughly follows the C code and **you do use recursion**.

### Action Item

Complete the implementation of `greatest_common_divider` below the annotation `# YOUR CODE HERE #`. When you've finished the implementation, running the code in Venus locally should provide you with the following output:

```
greatest_common_divider(3, 12) = 3
Found 0 warnings!
```

#### Checkoff

- Show your TA your test run and make sure no CC violation is detected. Note: TAs will use a larger test case during Lab 4 check. The range of `a` and `b` is `(2, 20000]`.

---

The following TA(s) is(are) responsible for Lab4 this year.

*Songhui Cao* <caosh2022 AT shanghaitech.edu.cn>
*Guanghui Hu* <hugh2024 AT shanghaitech.edu.cn>