
# CS 240 – Computer Architecture

## Lab 1:
## C Programming
## and
## RISC-V Toolchain / Compilation

# Scope of the Lab Assignment

In this lab, you will write a program in C, compile it for RISC-V and run your very first RISC-V program. Since we you don't have physical RISC-V based computer (you either have x86 or ARM-based computer), we are going to use instruction set simulator called **Spike**. Using **Spike**, you can test, run, and debug C or assembly programs without having the physical RISC-V based computer.

**Objective:** Get familiar with basic tools in the RISC-V toolchain, and understand how to simulate the execution of RISC-V program by using Spike.

## Exercise 1: Using RISC-V toolchain and compiling a C program into RISC-V executable

**Step 1 -** Write a C program that finds the *greatest common divisor* of two numbers, *a* and *b*, according to the Euclidean algorithm. The values *a* and *b* should be statically defined variables in the program. Name the program **GCD.c**. Here is some additional information about the Euclidean algorithm: https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm. You can also simply google "Euclidean algorithm".

**Step 2 -** Compile the program into RISC-V using the RISC-V toolchain (riscv32-unknown-elf-gcc your_program.c -o name_of_the_program).

- For mac users to target 32-bit
- riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 your_code.c - your_output

**Step 3 -** Try to read the content of binary generated (e.g., by using "cat" command). How does the output look like?

**Step 4 -** Use "file" command to see the details of the executable (i.e., file [name of the executable]). Explain the output.

**Step 5 -**Use riscv32-elf-readelf (i.e., riscv32-elf-readelf -d [name of the executable]) and explain the output.

**Step 6 -** Use riscv32-elf-objdump (i.e., riscv32-elf-objdump -D [name of the executable]) to disassemble the binary and explain the output.

**Step 7 -** Run the program on Spike (i.e., spike [name of the executable]). What error did you get and why? Explain.

**Step 8 -** Run the program on Spike with proxy kernel (i.e., spike pk [name of the executable]). Explain the output.

- To target 32-bit: spike --isa=rv32i pk your_executable

## Exercise 2: RISC-V Assembly File Generation and Linking:

**Step 1 -** Generate RISC-V assembly file (.S file) using RISC-V toolchain (i.e., riscv32-unknown-elf-gcc -S your_program.c -o output_file.S) and report the content of the assembly file generated. How does it compare to the output of riscv32-elf-objdump that you obtained in Exercise 1.

**Step 2 -** Use assembler to generate RISCV machine code (i.e., riscv32-unknown-elf-as your_program.S -o output_file.o) which will create an object file which is not ready to execute.

**Step 3 -** Use linker to link other object files and libraries (i.e., riscv32-unknown-elf-ld output_file.o) to generate a final executable.

**Step 4 -** Run the executable on Spike with proxy kernel (i.e., spike pk [name of the executable]). Explain the output.

**Step 5 -** Run the program on Spike in conjunction with spike -d (debug mode) to step through the program instruction by instruction. Report how each instruction affects registers and memory.

## What to Push into Github?

You should push any source code that you write for the exercises. Also, you should push the results of the steps given in the exercises.

## How to get Graded for your lab?

Once you completed the exercises and pushed your files and results into github, ask TAs for grading. TAs will go over your results and may ask specific questions for you to answer. Based on your results and your answers to the questions asked by TAs, your grade will be published on LMS.