

CDA 4205L Lab #1: Introduction to RISC-V and RARS

University of South Florida

Lab date: Jan. 15

Report due: One week after the lab, on Canvas

Welcome to CDA 4205L Lab #1! The purpose of this lab is to introduce you to (or refresh your memory of) RISC-V assembly and a RISC-V Assembler and Runtime Simulator (RARS).

Prelab

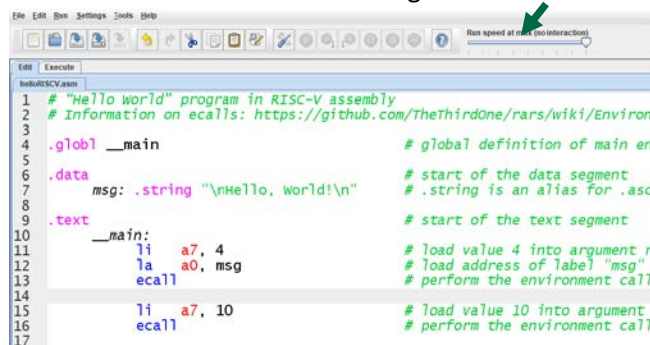
RISC-V is an open-source instruction set architecture (ISA). It was initially developed by UC Berkeley, but in the past decade, many other groups have contributed to the project, and several companies have started deploying physical processors that use the standard. In the lecture portion of this course, we will go over the principles of ISA design and important tradeoffs. In the lab portion of this course, we will go over practical implementation details.

Two very important tools for any ISA are the assembler and the runtime simulator. The assembler takes human-readable code and translates it into machine code, represented in binary or hexadecimal. The runtime simulator is a platform for executing machine instructions and provides insight into the state of the processor at any given time. A simulator can also provide an environment for doing things like file reads/writes, printing text to the console, or inputting values to your program. These are called “system calls” or “environment calls” depending on the platform, and are shortened to “syscalls” or “ecalls”.

In this lab, you will become familiar with one particular RISC-V assembler and runtime simulator called RARS, and experiment with some important ecalls provided by the platform.

Lab

1. Download and install RARS, which can be found here: <https://github.com/TheThirdOne/rars>. Note that you will need the Java runtime installed as well.
2. Download the *HelloRISCV.asm* code from Canvas (in Files > Labs > Lab 01 directory), and open the file in RARS. You should see something like this:



```
1 # "Hello World" program in RISC-V assembly
2 # Information on ecalls: https://github.com/TheThirdOne/rars/wiki/Environ
3
4 .globl __main                                # global definition of main en
5
6 .data                                        # start of the data segment
7     msg: .string "\nHello, world!\n"        # .string is an alias for .asc
8
9 .text                                        # start of the text segment
10
11     __main:
12         li a7, 4                            # load value 4 into argument r
13         la a0, msg                          # load address of label "msg"
14         ecall                               # perform the environment call
15
16         li a7, 10                           # load value 10 into argument
17         ecall                               # perform the environment call
```

3. Click the “Assemble” button (marked with the green arrow). From here, you will be able to run the code to completion (the “play” button next to the “assemble” button) or, alternatively, step

through the code line by line and see values change in the memory window (the smaller “play” button with the number 1).

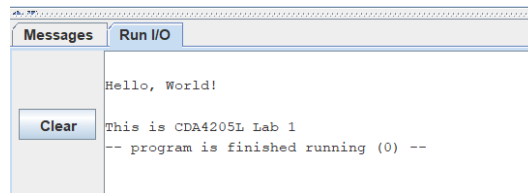
4. Modify the code so that it prints your name instead of “Hello World!”. Re-assemble, and re-run the code.

T1: Take a screenshot of the RARS console output. Include this in your report.

5. Observe that the provided code uses two ecalls: 4 and 10. The documentation for the ecalls is available here: <https://github.com/TheThirdOne/rars/wiki/Environment-Calls>.
6. Modify the code so that, in addition to your name, it prints “This is CDA4205L Lab 1”

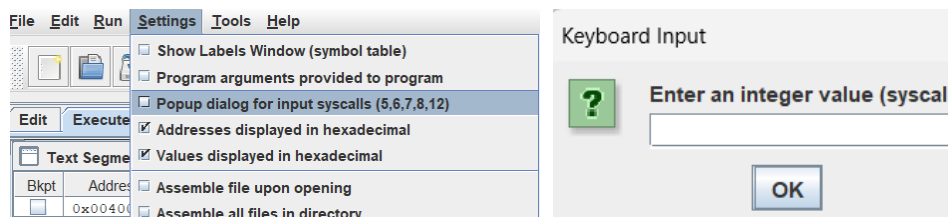
T2: What do ecalls 4 and 10 do? What is the purpose of register a7 and a0?

- a. Declare a new string in the .data section of the code. This string should be “\nThis is CDA4205L Lab “ (without lab number for now). You can label this string “msg2”.
- b. Print this string using the appropriate ecall.
- c. Refer to the ecall documentation. Find the ecall number for “PrintInt”. Use this to print the lab number.
- d. The console output should look like this (but instead of “Hello, World!” it should print your name).



T3: Take a screenshot of the RARS console output. Include this in your report.

7. Instead of hard coding the lab number into the program, modify the code so that the user can input this number at runtime.
 - a. First, enable the setting in RARS “Popup dialog for input syscalls (5, 6, 7, 8, 12)”. Terms syscall and ecall are used interchangeably. If your program uses one of these syscalls, you should get a popup at runtime where you can enter the desired value.



- b. Refer again to the ecall documentation. Find the ecall number for “ReadInt”. Use this to read the lab number from the user, then use PrintInt to print this value to the console.

T4: Take a screenshot of the RARS console output. Include this in your report.

In-class work - Complete at least **Task 1**, show it to the TA for review and submit it on Canvas before the lab session ends. Submit your in-class work as a Word document (.docx) format.

Final-Report - Submit your report pdf with answers to all questions and screenshots. Also, submit the final assembly file(s). Combine these in a .zip file. It is due in 1 week, before the next lab.