Chapter 5: RISC-V in Practice

Contents

Introduction

In this chapter, we will get some hands-on experience programming RISC-V assembly language for the Linux operating system. Even if you’ve never programmed assembly before, this chapter will give you all the information you need to get started writing your first “Hello World” application. Feel free to follow along, or simply read through the material and watch the videos.

By the end of this chapter, you should be able to:

- Understand how to emulate a simple Linux system using QEMU;
- Write a simple “Hello World” program in RISC-V 64 bit assembly language.;
- Compile and run a RISC-V application in emulation.

RISC-V Assembly Language Overview

Required Documentation

First off, Chapter 2 of the RISC-V Unprivileged Specification goes into detail about the RV32I Base Integer Instruction Set, including a programming model and an explanation of instruction formats. While this information is not required for this course it is certainly helpful in understanding how the RISC-V architecture executes instructions.

For programming assembly instructions, we can use both the ABI reference documentation and the ASM manual to answer any questions we may have along the way. You can find those documents here:

- RISC-V Specifications [1];
- ABI Documentation [1];
- ASM Manual [2].

Again, none of this information is required knowledge for this chapter, but you can reference it if you have any questions not answered here.

Assembly Language Overview

This chapter will be a very high-level overview of RISC-V assembly instructions and will only cover a few of those instructions in practice. The hope is that this tutorial will give you the tools you need to continue your journey programming assembly language. If your goal is simply to understand the basics and develop applications in a higher-level language, this course will likely cover most of the information you need.

RISC-V is a “reduced instruction set” architecture, and as such, there are not many instructions to learn. In this tutorial, we only use 3 instructions: LA (load absolute address), ADDI (add immediate), and ECALL. The ECALL instruction is used to make a service request to the execution environment. We will only use two calls in our Hello World app, one to “write” and one to “exit”.

For a full list of instructions, you can see the RISC-V Unprivileged Specification Chapter 24 “RV32/64G Instruction Set Listings”. If you’d like to learn more about assembly language programming there are plenty of books and courses available. For more information, visit RISC-V website [3].

Getting Started with QEMU

Compiling Required Binaries

If you want to follow along with the videos, you can certainly do so. However, it should be noted that creating an emulation environment is no small task. We’d highly recommend you simply follow along for now unless you have experience with compiling the Linux kernel.

Instructions for compiling required binaries can be found in the "RISC-V - Getting Started Guide" [4].

Creating a Custom RISC-V System

If you are already comfortable with compiling the Linux kernel, QEMU, and software suites like BusyBox, you may want to take things a step further. There is a build system for creating Linux based root file systems and emulating them called the Yocto Project. RISC-V has a “layer” which can be used to create a completely custom Linux distribution. For more details see meta-riscv [5] on GitHub.

RISC-V Hello World

Environment Overview

Here is the hello world application we will be using:

```
-----code-----
# Simple RISC-V Hello World

.global _start

_start: addi  a0, x0, 1
    la    a1, helloworld
    addi  a2, x0, 13
    addi  a7, x0, 64
    ecall

    addi    a0, x0, 0
    addi    a7, x0, 93
    ecall

.data
helloworld:    .ascii "Hello World!\n"
-----end code-----
```

There are also two ways of compiling this code, either using GCC or calling “as” and “ld” directly:

```
-----code-----
# GCC
riscv64-linux-gnu-gcc -o rv-hello rv-hello.s -nostdlib -static

# AS & LD
riscv64-linux-gnu-as -march=rv64imac -o rv-hello.o rv-hello.s
riscv64-linux-gnu-ld -o rv-hello rv-hello.o
-----end code-----
```

List of sources

1. Specifications: [Electronic resource] // RISC-V. URL: https://riscv.org/technical/specifications/.
2. RISC-V Assembly Programmer’s Manual: [Electronic resource] // GitHub. URL: https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md.
3. RISC-V Learn: [Electronic resource] // RISC-V. URL: https://riscv.org/learn/.
4. Running 64- and 32-bit RISC-V Linux on QEMU: [Electronic resource] // RISC-V Getting Started Guide. URL: https://risc-v-getting-started-guide.readthedocs.io/en/latest/linux-qemu.html.
5. meta-riscv: [Electronic resource] // GitHub. URL: https://github.com/riscv/meta-riscv.