Prerequisites For Labs

CS 211 Advanced Computer Architecture, Fall 2023

1. RISC-V CPU Simulator

In this lab, we will use a RISC-V CPU simulator, which is modified based on Hao He's 5-stage pipelining RISC-V CPU. You can download it from the following link.

DOWNLOAD `cs211_23f_lab_sim_framework.zip` HERE (campus only)

2. Cross-compilier `riscv-gnu-toolchain`

> *This part is based on the description in the original repository.*

RISC-V Simulator can run binary file that is compiled in RISC-V. To compile RISC-V binary on a non RISC-V system, cross-compile is required. We give following instructions on build and usage of `riscv-gnu-toolchain`.

2.1. Get the build tool

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain
$ git submodule update --init
```

> *Note: Even though the instructions in the repo don't require `submodule` update, we still recommend it due to the network connection here. You may take a long time pulling these repos. We took 30 minutes for that, as a reference.*

HINT

If you want the file only, download `riscv-gnu-toolchain_230920.zip` (campus only) and unzip to build it.

You may need some dependencies:

```
# Here we only show the installation in Ubuntu Linux 22.04 distribution.
Please refer to the original repo if you are using other distribution.
$ sudo apt update
$ sudo apt install autoconf automake autotools-dev curl python3 python3-pip
libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo
gperf libtool patchutils bc zlib1g-dev libexpat-dev ninja-build git cmake
libglib2.0-dev
```

You need to specify install path, and the ISA extension during build configuration.

```
# Compile
# prefix determines the install path, change them to install to a path you
want
# For RV64I and RV64M
./configure --prefix=/opt/riscv --with-arch=rv64im
# For RV64I, RV64M and RV64F
./configure --prefix=/opt/riscv --with-arch=rv64imf

# Build
# use -j[task_number] to specify the number of parallel tasks during the
build
# e.g. make -j10
# e.g. make -j$(nproc) use all cores in the system
make
```

2.2. Use the build tool

After finishing build, you can find the tool in the `bin/` folder under the path you specified in 2.1.

You may want to use `riscv64-unknown-elf-gcc` and `riscv64-unknown-elf-objdump`.

The usage of `riscv64-unknown-elf-gcc` is just like `gcc`. But the ELF file the simulator receives cannot be dynamically linked. So you may want to compile your program as below.

```
# Use RV64I and RV64M
riscv64-unknown-elf-gcc -march=rv64im [FileName].c -o [FileName].riscv

# Use RV64I, RV64M and RV64F
riscv64-unknown-elf-gcc -march=rv64imf [FileName].c -o [FileName].riscv

# You may want the test/lib.c to help build your own testcase
riscv64-unknown-elf-gcc -march=rv64imf [FileName].c test/lib.c -o
[FileName].riscv
```

You can generate human-readable assembly code by `riscv64-unknown-elf-objdump`.

```
# Disassembly by riscv64-unknown-elf-objdump, and outout it to the .s file
riscv64-unknown-elf-objdump -d [FileName].riscv > [FileName].s
```

This may be greatly helpful for debugging in company with `dump.txt` the simulator generates.

Lab 0

CS 211 Advanced Computer Architecture, Fall 2023

NOTICE
Please be noticed that Lab 0 takes 3% of the overall score for CS 211, Fall 2023.

DEADLINE

2023/10/13 (Fri) 23:59:59

1. Purpose

- Learn the structure of the simulator
- Learn the basics of 5-level pipelining in-order CPU
- Add RV64-F instructions to the simulator

2. RISC-V ISA

Here are some resources introducing RISC-V ISA:

- RISC-V Handbook (Chinese) [PDF]
- RISC-V ISA Manual [PDF]

You are required to implement the following instructions:

- `FMV.W.X` (previously called `FMV.S.X`)
- `FMV.X.W` (previously called `FMV.X.S`)
- `FCVT.S.W`
- `FCVT.W.S`
- `FLW`
- `FSW`
- `FADD.S`
- `FSUB.S`
- `FMUL.S`
- `FDIV.S`
- `FSQRT.S`
- (bonus) `FMADD.S`

- (bonus) `FMSUB.S`

If you do not understand how these instructions proceed, you may want to use Venus. To use Venus, please refer to this.

REMINDER

For most floating point including round model (rm), you are not required to implement all models. You can use ANY round model and skip the static round. Therefore, you are not required to implement controller and status register. 32 floating point register suffices.

3. Code

For code changes in lab 0, you will mainly modify:

- `src/Simulator.h`
- `src/Simulator.cpp`

To understand the code in the simulator, you need basic knowledge of 5-stage pipelining CPU, such as data hazards. You may refer to the doc of the original simulator, CS 110 slides and CS 61C materials.

Based on the original simulator, we made some modifications to reflect the different components latency in execute stage. You can get the component type by `getComponentUsed()`.
You are required to add new instructions into `getComponentUsed()`.
The different types and latencies of execute stage components are given by `executeComponent` and `latency[]` respectively. You may not want to modify it.

4. Test

We provide a case (`riscv-elf/test_float.riscv`). However, you need to create at least one more test case. For bonus instructions particularly, you are required to add new test case(s) where necessary for evaluation.

We will test the correctness of your modification with extra test cases.

5. Report

You need to prepare a report for this lab. The lab report must include at least the following sections:

- Analysis on the original code structure

- Description of your design
- Evaluation and analysis on your results, test case(s), etc.

6. Hand in

The submission shall include the following three parts:

- Source code that can be built and executed
- 1 to 3 test cases and a README on build and run them
- Lab report in `pdf` format

You shall pack the source code, test cases and lab report into a `zip`.

WARNING

THE FILE NAME OF YOUR `zip` MUST BE `[Your ID][Mail Prefix].zip`.

i.e. `2023114514wangdch12023.zip`

You can use the following command to avoid `.git/` or `build/` in your `zip` file.

```
$ cd [Your Simulator Directory]
$ zip ../[FileName].zip -r . --exclude=".git/*" --exclude="build/*"
```

You are recommended to arrange your submission like the following layout:

```
2023114514wangdch12023.zip
├── report.pdf # Your report
├── source_code # Directory of your source codes
│   └── ... # Make sure to not include .git/ and build/
│
└── test_program
    └── ...
```

SUBMISSION

Hand in your zip at
Gradescope. The entry code of CS 211 for Gradescope is `4G25GD`.

For each student, we will evaluate only the last submission before the deadline.

We will not evaluate any submission that passes the deadline.

Please hand in your submission before 2023/10/13 (Fri) 23:59:59.

7. (Appendix) Useful aliases and `PATH` setting

```
# Define the directory of `gcc` and the simulator, specify them as your
wish.
$ export GCC_DIR="/opt/riscv/bin"
$ export SIM_DIR="/path/to/sim_framework/build"

# Push `gcc` and the simulator onto $PATH
$ export PATH_OLD="$PATH"; export PATH="$PATH:$GCC_DIR:$SIM_DIR"
# To restore old $PATH,
$ export PATH="$PATH_OLD"

# A quick alias for build in anywhere
$ alias mksim="pushd /path/to/sim_framework/build; make Simulator; popd"

# These aliases work only when your `gcc` can be searched under $PATH
$ alias gccrv="riscv64-unknown-elf-gcc"
$ alias objdumprv="riscv64-unknown-elf-objdump"
$ gccrv -v # shows version if these two aliases above work
```

Here, you may just type `mksim` to build `Simulator` in any folders, without frequently changing your working directory.
All these changes will disappear if you close your terminal. However, some editors like VSCode server may extend the lifetime of your terminal.