香港中文大學
The Chinese University of Hong Kong

CENG3420

Lab 2-2: RISC-V RV32I Simulator

Shixin Chen& Mingjun Li
Department of Computer Science & Engineering
Chinese University of Hong Kong
sxchen22@cse.cuhk.edu.hk&
mjli23@cse.cuhk.edu.hk

Spring 2024

Outline

1. Introduction
2. RV32I Instructions
3. Lab 2-2 Assigment

Introduction

Use C programming language to finish lab assignments in following weeks.

- Lab 2.1 – implement an RISCV-LC Assembler
- Lab 2.2 – implement an RISCV-LC ISA Simulator
- Lab 3.x – implement an RISCV-LC Simulator

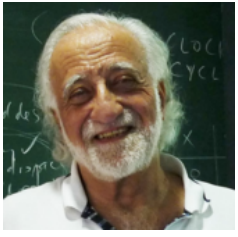
NOTICE

Lab2 & Lab3 are challenging!
Once you have passed Lab2 & Lab3, you will be more familiar with RV32I & a basic implementation!

ISA Simulator

- Mimic the behavior of an instruction execution.
- ISA Simulator input: a binary file generated in Lab 2.1.

- LC-3b: **Little Computer 3, b** version.
- Relatively simple instruction set.
- Most used in teaching for CS & CE.
- Developed by Yale Patt@UT & Sanjay J. Patel@UIUC.

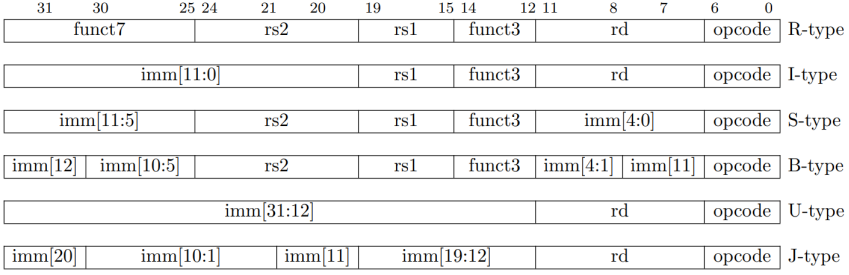


RV32I Instructions

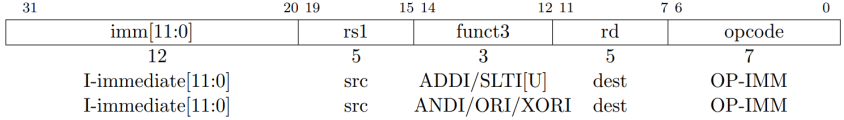
Introduction to RV32I

- The instruction encoding width is 32-bit.
- Each instruction is aligned with a **four-byte** boundary in memory.
- RV32I only manipulates integer numbers and no multiplication or division.
- Our labs are based on the official RISC-V ISA manual: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf.

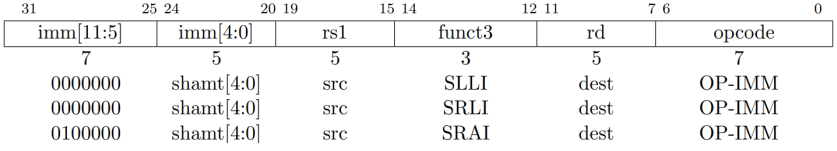
RV32I Instructions



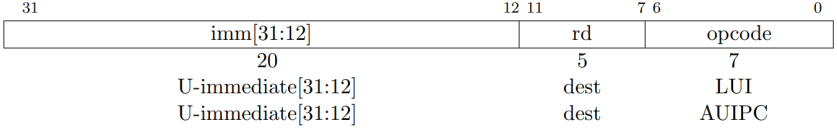
RV32I instructions base formats.



addi, andi, ori, xori



slli, srli, srai

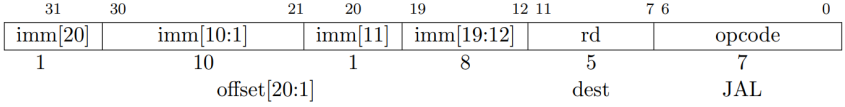


lui

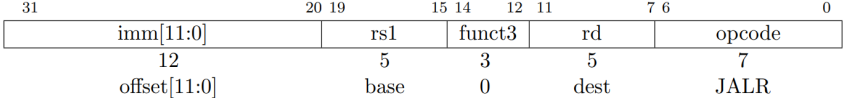
31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

add, and, or, xor, sll, srl, sub, sra

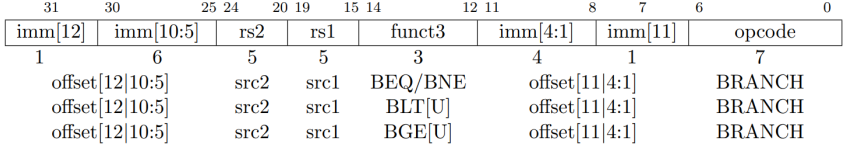
Control Transfer Instructions
Jumps & Conditional Branches



jal

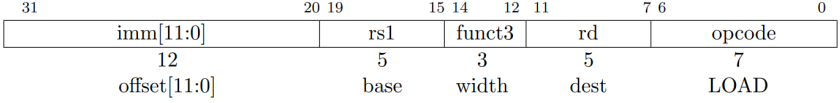


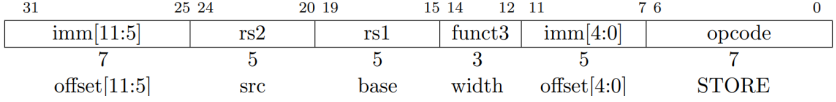
jalr



beq, bne, blt, bge

Load and Store Instructions

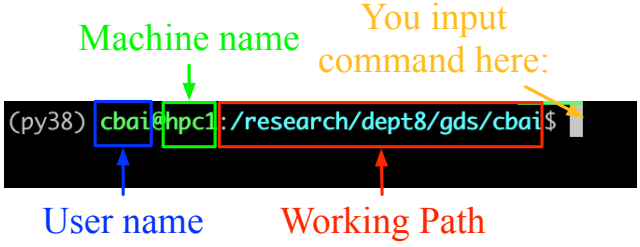




lb, lh, lw, sb, sh, sw

RISC-V LC ISA Simulator

Linux Terminal:



The user interface of Linux terminal

A live demo to illustrate:

- `ls`: list files/directories.
- `cd`: change the working directory.
- `rm`: remove a file.
- `mv`: move a file/rename a file.
- `cat`: show file contents.
- `file`: check the file type.
- `man`: help menu for commands.
- For more information: `https://ubuntu.com/tutorials/command-line-for-beginners`.

A live demo to illustrate:

- `git clone`: clone the code repository.
- `git checkout`: checkout a git branch.
- For more information:
  - `https://www.w3schools.com/git/`
  - `https://git-scm.com/docs/gittutorial`

A live demo to illustrate:

- How to compile C codes into machine codes with GCC?
  - https://medium.com/@laura.derohan/compiling-c-files-with-gcc-step-by-step-8e78318052
    https://www.wikihow.com/Compile-a-C-Program-Using-the-GNU-Compiler-(GCC)
- How to automate the compilation process with Makefile?
  - https://makefiletutorial.com/
  - https://www.tutorialspoint.com/makefile/index.htm

Get the RISC-V LC ISA Simulator

In the terminal of your computer, use these commands:

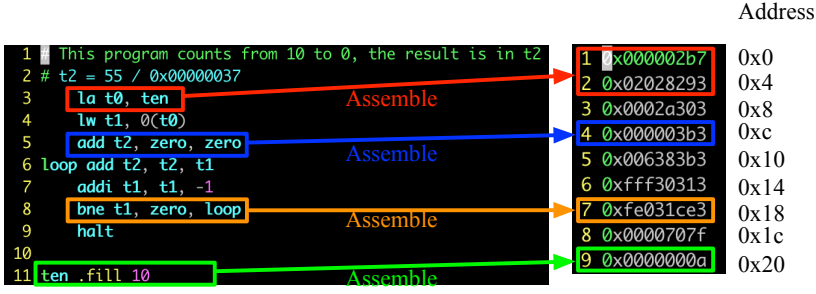
- git clone https://github.com/MingjunLi99/ceng3420.git
- cd ceng3420
- git checkout lab2.2

Compile (Linux/MacOS environment is suggested)

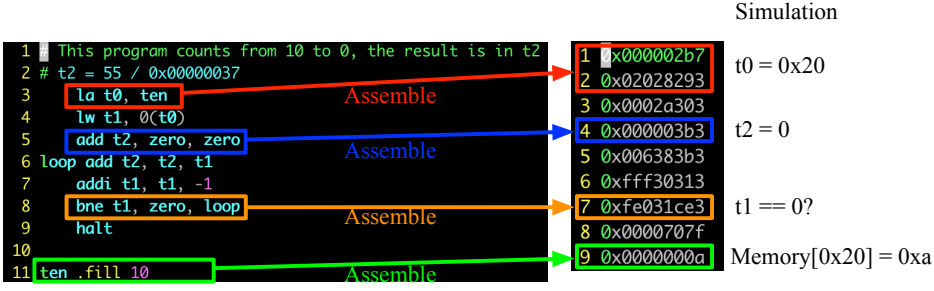
- make

Run the Simulator

- ./sim benchmarks/count10.bin # the simulator can execute successfully if you have implemented it.

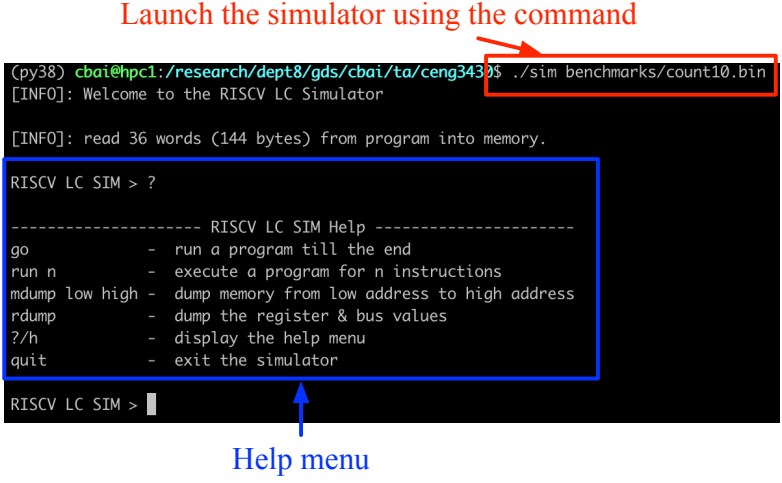


RISC-V LC Assembler.



RISC-V LC ISA Simulator.

How to launch the simulator?



An overview of the simulator.

RISC-V LC ISA simulator options:

- go: execute the program till the end.
- run n: execute the program by n instructions, *e.g.*, run 3
- mdump low high: dump the memory content, *e.g.*, mdump 0x0 0x30
- rdump: dump the registers
- h: help menu
- quit: exit the simulator

Implementation examples (in `sim.c`):

- `handle_addi`
- `handle_add`
- `handle_beq`
- `handle_lb`

Lab 2-2 Assignment

Finish the RISCV LC simulator including 24 instructions in sim.c

- Integer Register-Immediate Instructions: slli, xori, srli, srai, ori, andi, lui
- Integer Register-Register Operations: sub, sll, xor, srl, sra, or, and
- Unconditional Jumps: jalr, jal
- Conditional Branches: bne, blt, bge
- Load and Store Instructions: lh, lw, sb, sh, sw

These unimplemented codes are commented with Lab2-2 assignment.

Benchmarks

Verify your codes with these benchmarks (inside the `benchmarks` directory)

- isa.bin
- count10.bin
- swap.bin
- add4.bin

Verification

- isa.bin → a3 = -18/0xffffffee and MEMORY[0x84 + 16] = 0xffffffee
- count10.bin → t2 = 55/0x00000037
- swap.bin → NUM1 (memory address: 0x00000034) changes from 0xabcd to 0x1234 and NUM2 (memory address: 0x00000038) changes from 0x1234 to 0xabcd
- add4.bin → BL (memory address: 0x00000038) changes from -5 (0xfffffffb) to -1 (0xffffffff)

Submission Method

We will open TWO submit windows on **Blackboard** (for lab2-1 and lab2-2).
Submit two zip files to corresponding window:

- `name-sid-lab2-1.zip` (e.g. `zhangsan-1234567890-lab2-1.zip`)
- `name-sid-lab2-2.zip` (e.g. `zhangsan-1234567890-lab2-2.zip`)

In each zip file, please contain corresponding contents:

- The source codes of the corresponding lab (lab 2-1 or lab2-2).
- One report. (name format: `name-sid-report-lab2-1.pdf` or `name-sid-report-lab2-2.pdf`) The report summarizes your implementations and all screenshots of the corresponding lab results (lab 2-1 or lab2-2).

Deadline: 23:59, 26 Mar (Tue), 2024

Tips

Inside `docs`, there are three valuable documents for your reference!

- opcodes-rv32i: RV32I opcodes
- riscv-spec-20191213.pdf: RV32I specifications
- risc-v-asm-manual.pdf: RV32I assembly programming manual