



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!

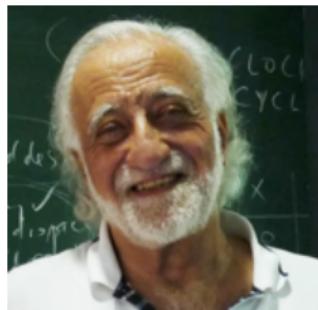
ISA Simulator

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

Introduction

Our Lab2 & Lab3 are Inspired by LC-3b

- 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

31	30	25 24	21	20	19	15 14	12 11	8	7	6	0	
	funct7		rs2		rs1	funct3		rd		opcode		R-type
		imm[11:0]			rs1	funct3		rd		opcode		I-type
	imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode			S-type
imm[12]	imm[10:5]		rs2		rs1	funct3	imm[4:1]	imm[11]		opcode		B-type
		imm[31:12]					rd		opcode			U-type
imm[20]	imm[10:1]	imm[11]	imm[19:12]				rd		opcode			J-type

RV32I instructions base formats.

Integer Computational Instructions

Integer Register-Immediate Instructions

31	20 19	15 14	12 11	7 6	0
imm[11:0]	rs1	funct3	rd	opcode	
12	5	3	5	7	
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM	
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM	

addi, andi, ori, xorI

31	25 24	20 19	15 14	12 11	7 6	0
imm[11:5]	imm[4:0]	rs1	funct3	rd	opcode	
7	5	5	3	5	7	
0000000	shamt[4:0]	src	SLLI	dest	OP-IMM	
0000000	shamt[4:0]	src	SRLI	dest	OP-IMM	
0100000	shamt[4:0]	src	SRAI	dest	OP-IMM	

slli, srli, srai

31	12 11	7 6	0
imm[31:12]	rd	opcode	
20	5	7	
U-immediate[31:12]	dest	LUI	
U-immediate[31:12]	dest	AUIPC	

lui

Integer Computational Instructions

Integer Register-Register Instructions

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

31	30	21	20	19	12 11	7 6	0
imm[20]	imm[10:1]	imm[11]	imm[19:12]		rd	opcode	
1	10	1	8	5		7	

offset[20:1] dest JAL

jal

31	20 19	15 14	12 11	7 6	0
imm[11:0]	rs1	funct3	rd	opcode	
12 offset[11:0]	5 base	3 0	5 dest	7 JALR	

jalr

31	30	25 24	20 19	15 14	12 11	8	7	6	0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode		
1	6	5	5	3	4	1	7		
offset[12 10:5]		src2	src1	BEQ/BNE	offset[11 4:1]		BRANCH		
offset[12 10:5]		src2	src1	BLT[U]	offset[11 4:1]		BRANCH		
offset[12 10:5]		src2	src1	BGE[U]	offset[11 4:1]		BRANCH		

beq, bne, blt, bge

Load and Store Instructions

31	20 19	15 14	12 11	7 6	0
imm[11:0]	rs1	funct3	rd	opcode	
offset[11:0]	base	width	dest	7 LOAD	

31	25 24	20 19	15 14	12 11	7 6	0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	
offset[11:5]	src	base	width	offset[4:0]	7 STORE	

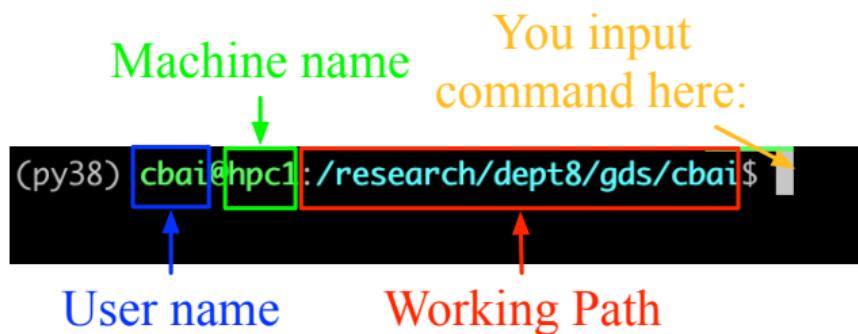
lb, lh, lw, sb, sh, sw

RISC-V LC ISA Simulator

RISC-V LC ISA Simulator

Basics 1 – Linux Terminal Tutorial

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-\(GCC\)](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 ISA Simulator

Assemble & Simulate

```
1 This program counts from 10 to 0, the result is in t2
2 # t2 = 55 / 0x00000037
3 la t0, ten
4 lw t1, 0(t0)
5 add t2, zero, zero
6 loop add t2, t2, t1
7 addi t1, t1, -1
8 bne t1, zero, loop
9 halt
10
11 ten .fill 10
```

count10.asm

1	0x000002b7	0x0
2	0x02028293	0x4
3	0x0002a303	0x8
4	0x000003b3	0xc
5	0x006383b3	0x10
6	0xffff30313	0x14
7	0xfe031ce3	0x18
8	0x0000707f	0x1c
9	0x0000000a	0x20

count10.bin

RISC-V LC Assembler.

RISC-V LC ISA Simulator

Assemble & Simulate

```
1 1 This program counts from 10 to 0, the result is in t2
2 # t2 = 55 / 0x00000037
3 la t0, ten
4 lw t1, 0(t0)
5 add t2, zero, zero
6 loop add t2, t2, t1
7 addi t1, t1, -1
8 bne t1, zero, loop
9 halt
10
11 ten .fill 10
```

count10.asm

Assemble

Assemble

Assemble

Assemble

1	x0000002b7
2	0x02028293
3	0x0002a303
4	0x000003b3
5	0x006383b3
6	0xffff30313
7	0xfe031ce3
8	0x0000707f
9	0x0000000a

count10.bin

Simulation

t0 = 0x20

t2 = 0

t1 == 0?

Memory[0x20] = 0xa

RISC-V LC ISA Simulator.

RISC-V LC ISA Simulator

Assemble & Simulate

How to launch the simulator?

Launch the simulator using the command

```
(py38) cbai@hpc1:/research/dept8/gds/cbai/ta/ceng343$ ./sim benchmarks/count10.bin
```

[INFO]: Welcome to the RISCV LC Simulator

```
[INFO]: read 36 words (144 bytes) from program into memory.
```

```
RISCV LC SIM > ?
```

```
----- RISCV LC SIM Help -----  
go          - run a program till the end  
run n       - execute a program for n instructions  
mdump low high - dump memory from low address to high address  
rdump        - dump the register & bus values  
?/h          - display the help menu  
quit         - exit the simulator
```

```
RISCV LC SIM > █
```

Help menu

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

RISC-V LC ISA Simulator

Assemble & Simulate

Implementation examples (in `sim.c`):

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

Lab 2-2 Assignment

Lab 2-2 Assignment

Assignment Content

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](#).

Lab 2-2 Assignment

Verification

Benchmarks

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

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

Verification

- `isa.bin` → $a_3 = -18/0xfffffee$ and $\text{MEMORY}[0x84 + 16] = 0xfffffee$
- `count10.bin` → $t_2 = 55/0x00000037$
- `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 (0xffffffffb)` to `-1 (0xfffffffff)`

Lab 2-2 Assignment

Submission

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**

Lab 2-2 Assignment

Tips

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