



香港中文大學
The Chinese University of Hong Kong

CENG3420

Lab 2-1: RISC-V RV32I Assembler

Chen BAI & Yuxuan ZHAO

Department of Computer Science & Engineering
Chinese University of Hong Kong

cbai@cse.cuhk.edu.hk &

yxzhao21@cse.cuhk.edu.hk

Spring 2023

Outline

- ① Introduction
- ② Introduction to RV32I
- ③ RISCV-LC Code Specifications
- ④ RISCV-LC Assembler
- ⑤ Lab 2-1 Assignment

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!

Introduction

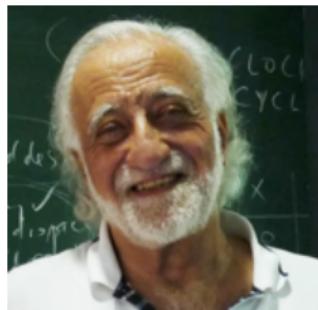
Assembler & Simulator

- Assembly language – symbolic (we have learned in Lab1)
- Machine language – binary
- **Assembler** is a program that
 - turns symbols into machine instructions, e.g., riscv64-unknown-elf-as
- **Simulator** is a program that
 - mimics the behavior of a processor
 - usually written in high-level language, e.g., spike

Introduction

Our Lab2 & Lab3 are Inspired by LC-3b

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



In Lab2-2, our RV32I ISA Simulator integrates

- RISC-V 32 general-purpose registers
- 32-bit data and address
- 25+ instructions (including pseudo instructions)

In Lab3, 4 more special-purpose registers will be added:

- Program Counter ([PC](#))
- Instruction Register ([IR](#))
- Memory Access Register ([MAR](#))
- Memory Data Register ([MDR](#))

Introduction to RV32I

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](https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf).

- Integer Computational Instructions
- Control Transfer Instructions
- Load and Store Instructions
- Memory Ordering Instructions
- Environment Call and Breakpoints
- HINT Instructions

Introduction to RV32I

- Four core instruction formats

31	25 24	20 19	15 14	12 11	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[31:12]			rd	opcode			U-type

- Two variants of the instruction formats

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

Introduction to RV32I

31	30	20 19	12	11	10	5	4	1	0	
— inst[31] —						inst[30:25]	inst[24:21]	inst[20]	I-immediate	
— inst[31] —						inst[30:25]	inst[11:8]	inst[7]	S-immediate	
— inst[31] —						inst[7]	inst[30:25]	inst[11:8]	0	B-immediate
inst[31]	inst[30:20]	inst[19:12]	— 0 —						U-immediate	
— inst[31] —			inst[19:12]	inst[20]	inst[30:25]	inst[24:21]	0			J-immediate

Immediate values encodings

If a value, e.g., 0xabcd, is encoded with I, S, B, U, or J format, can you write how it is encoded?

Introduction to RV32I

Integer Computational Instructions

- Integer Register-Immediate Instructions
 - addi, slti, andi, ori, xori, slli, srli, srai, lui
- Integer Register-Register Instructions
 - add, slt, and, or, xor, sll, srl, sub, sra

Introduction to RV32I

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	

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	

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	

Introduction to RV32I

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	

Introduction to RV32I

Control Transfer Instructions

- Unconditional Jumps
 - jal, jalr
- Conditional Branches
 - beq, bne, blt, bge

Introduction to RV32I

Unconditional Jumps

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

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

Introduction to RV32I

Conditional Branches

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	

Introduction to RV32I

Load and Store Instructions

- Load
 - `lb`, `lh`, `lw`
- Store
 - `sb`, `sh`, `sw`

Introduction to RV32I

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	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]	STORE	

The EEI will define whether the memory system is little-endian or big-endian. In RISC-V, endianness is byte-address invariant.

RISCV-LC Code Specifications

NOTICE

To make labs easy, I have added some self-defined directives.

- Label specification: no colon and one code line cannot contain only labels.
- No .data and .text directives
- Add one pseudo instruction: LA
- Add one self-customized directive: .FILL
- Add one halt instruction: HALT

RISC-V-LC Code Specifications

Label specification

Label specification: no colon and one code line cannot contain only labels.

```
1      la a0, AL
2      lw a0, 0(a0)
3      blt a0, zero, L1
4 ▼ L1 addi a7, a0, 13
5      bge zero, a7, L1
```

Specification examples

RISCV-LC Code Specifications

LA pseudo instruction

Add one pseudo instruction: LA. LA is translated to two RV32I instructions: [lui](#) and [addi](#).

```
1      la a0, A # lui a0, 0x0; addr = 0x0
2          # addi a0, a0, 0x8; addr = 0x4
3  A .FILL -2  # addr = 0x8
```

Translate [la](#) to [lui](#) and [addi](#)

RISC-V-LC Code Specifications

FILL self-customized instruction

.FILL is similar to .byte, .word, etc.

```
30
31    AL      .FILL  -2
32    BL      .FILL  -9
33
```

.FILL directive

RISCV-LC Assembler

Lab 2-1 Assignment

Pre-requisites

- Learn Linux/MacOS terminal commands, and we prefer using Linux/MacOS
- Install git in your computer: `sudo apt-get install git`
- Learn git from the reference: <https://git-scm.com/docs/gittutorial>
- Learn Makefile from the reference: <https://makefiletutorial.com/>

Lab 2-1 Assignment

Pre-requisites

Get Latest Updates of the Lab

- Click <https://github.com/baichen318>.
- Click the **Follow** button.

Follow me through GitHub, so that you can see any latest updates of the lab!

The screenshot shows Chen Bai's GitHub profile page. At the top, there is a navigation bar with links for Product, Solutions, Open Source, and Pricing. On the right side of the header are search, sign in, and sign up buttons. Below the header, there is a large circular profile picture of Chen Bai, followed by his name "Chen BAI" and GitHub handle "baichen318". A "Follow" button is located below his name, which is circled in red and has a red arrow pointing to it with the text "Click the button to follow me!". To the left of the "Follow" button is a bio section: "I am currently a second-year Ph.D. student at the Department of Computer Science and Engineering of The Chinese University of Hong Kong." Below this bio, it says "86 followers · 16 following". At the bottom of the bio, there is a link to "The Chinese University of Hong Kong". The main content area of the page is titled "Popular repositories" and lists several repositories: "FreePDK45", "boom-explorer-public", "tvm", "vim.config", "chipyard", and "Raspberry-Pi-SmartCar". Each repository card includes a thumbnail, name, description, language, star count, and fork count.

Lab 2-1 Assignment

Pre-requisites – Get RISC-V-LC Assembler

Please visit the website for more information:

<https://github.com/baichen318/ceng3420>

Get the RV32I Assembler

In the terminal of your computer, type these commands:

- git clone https://github.com/baichen318/ceng3420.git
- cd ceng3420
- git checkout lab2.1

Compile (Linux/MacOS environment is suggested)

- make

Run the assembler

- ./asm benchmarks/isa.asm isa.bin # you can check the output machine code: isa.bin if you have implemented the assembler

Lab 2-1 Live Demo

Now, let me take a live demo – by walking through the code repo to learn the assembler.

- Know the codes organizations.
- addi example in **asm.c**
- add example in **asm.c**
- beq example in **asm.c**
- lb example in **asm.c**

Lab 2-1 Assignment

Lab 2-1 Assignment

Finish the RV32I assembler including 25 instructions in asm.c as follows

- 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: lb, lh, lw, sb, sh, sw

These unimplemented codes are commented with [Lab2-1 assignment](#)

Lab 2-1 Assignment

Verification of Implementations

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

- [isa.asm](#)
- [count10.asm](#)
- [swap.asm](#)

Compare your output with [.bin](#) file. If both of them are the same, you are correct!

A quick verification command in the terminal: make validate # generate reports inside the tools directory.

Submission Method:

Submit a zip file including source codes and a report **after** the whole lectures of Lab2 into **Blackboard**.

Lab 2-1 Assignment

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