



香港中文大學  
The Chinese University of Hong Kong

# CENG3420

## Lab 2-1: RISC-V RV32I Assembler

Chen BAI

Department of Computer Science & Engineering

Chinese University of Hong Kong

[cbai@cse.cuhk.edu.hk](mailto:cbai@cse.cuhk.edu.hk)

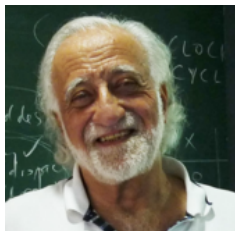
Spring 2022

- ① Introduction
- ② Introduction to RV32I
- ③ RISC-V-LC RV32I Specifications
- ④ Lab 2-1 Assignment

# Introduction

- Assembly language – symbolic (RISC-V, MIPS, ARM, X86, LC-3b, ...)
- Machine language – binary
- **Assembler** is a program that
  - turns symbols into machine instructions.
  - EX: `lc3b_asm`, `riscv64-unknown-elf-as`, ...
- **Simulator** is a program that
  - mimics the behavior of a processor
  - usually in high-level language
  - EX: `lc3b_sim`, `spike`, ...

- 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



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

Plus 4 special-purpose registers:

- Program Counter (**PC**)
- Instruction Register (**IR**)
- Memory Access Register (**MAR**)
- Memory Data Register (**MDR**)

### NOTICE

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

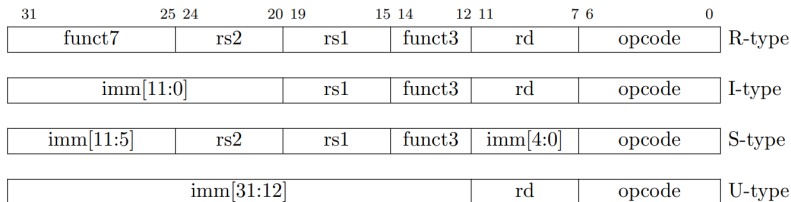
# Introduction to RV32I

- Instruction encoding width is 32-bit.
- Align on a four-byte boundary in memory
- Manipulate integer numbers.
- Our labs are based on RV32I [version 2.0](#).

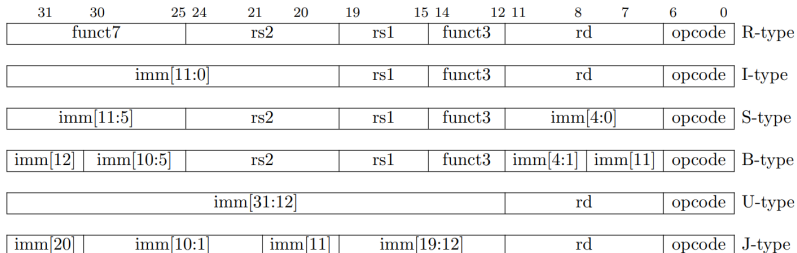


# Introduction to RV32I

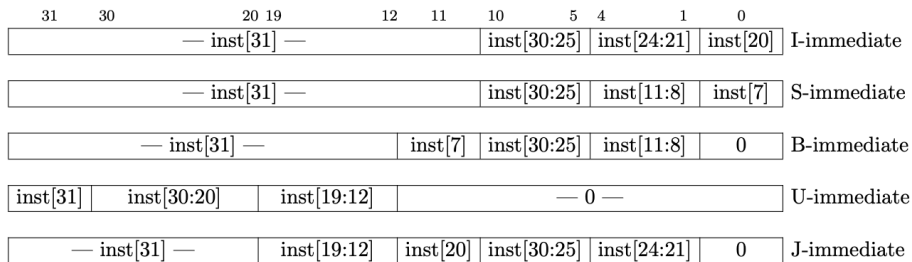
- Four core instruction formats



- Two variants of the instruction formats



Why does the RISC-V specification add two variant instructions B/J ?



Immediate values encodings

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

- Integer Register-Immediate Instructions
- Integer Register-Register Instructions
- NOP Instructions

# 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 |

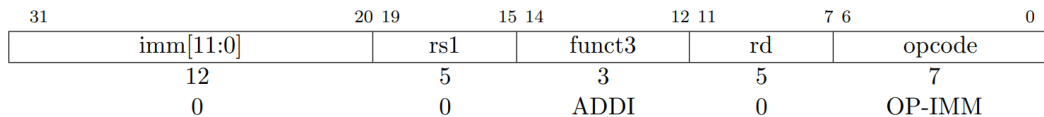
|           |       |            |       |        |      |        |
|-----------|-------|------------|-------|--------|------|--------|
| 31        | 25 24 | 20 19      | 15 14 | 12 11  | 7 6  | 0      |
| imm[11:5] |       | imm[4:0]   | rs1   | funct3 | rd   | opcode |
| 7         |       | 5          | 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  |

# 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     |   |

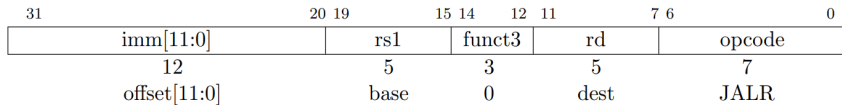
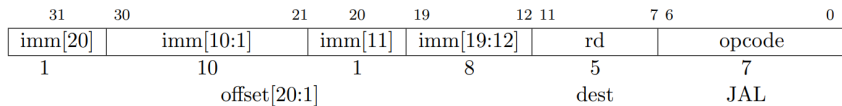
# NOP Instructions



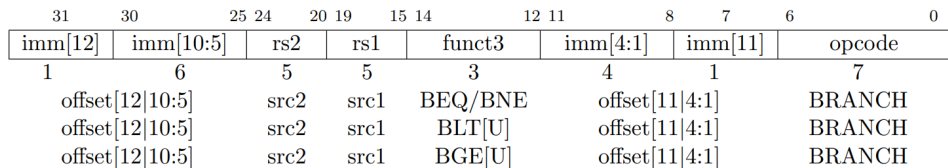
- Unconditional Jumps
- Conditional Branches



# Unconditional Jumps

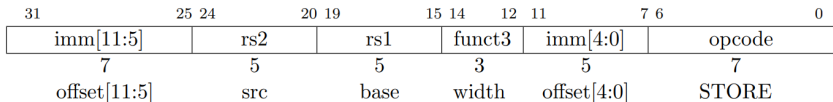
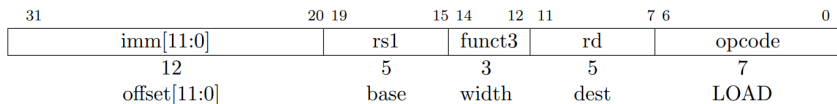


# Conditional Branches



The difference between RISC-V ISA and other ISA like MIPS, X86, ARM, SPARC?

# Load and Store Instructions



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

# Memory Ordering Instructions

|    |             |    |    |    |           |    |    |    |     |        |    |          |    |    |   |   |   |
|----|-------------|----|----|----|-----------|----|----|----|-----|--------|----|----------|----|----|---|---|---|
| 31 | 28          | 27 | 26 | 25 | 24        | 23 | 22 | 21 | 20  | 19     | 15 | 14       | 12 | 11 | 7 | 6 | 0 |
| fm | PI          | PO | PR | PW | SI        | SO | SR | SW | rs1 | funct3 | rd | opcode   |    |    |   |   |   |
| 4  | 1           | 1  | 1  | 1  | 1         | 1  | 1  | 1  | 5   | 3      | 5  | 7        |    |    |   |   |   |
| FM | predecessor |    |    |    | successor |    |    |    | 0   | FENCE  | 0  | MISC-MEM |    |    |   |   |   |

# Environment Call and Breakpoints

|         |  |       |  |        |       |    |     |        |   |
|---------|--|-------|--|--------|-------|----|-----|--------|---|
| 31      |  | 20 19 |  | 15 14  | 12 11 |    | 7 6 |        | 0 |
| funct12 |  | rs1   |  | funct3 |       | rd |     | opcode |   |
| 12      |  | 5     |  | 3      |       | 5  |     | 7      |   |
| ECALL   |  | 0     |  | PRIV   |       | 0  |     | SYSTEM |   |
| EBREAK  |  | 0     |  | PRIV   |       | 0  |     | SYSTEM |   |

HINTs are encoded as integer computational instructions with  $rd=x0$

| Instruction | Constraints  | Code Points  | Purpose                                 |
|-------------|--|--------------|---|
| LUI         | $rd=x0$  | $2^{20}$     | <i>Reserved for future standard use</i> |
| AUIPC       | $rd=x0$  | $2^{20}$     |   |
| ADDI        | $rd=x0$ , and either $rs1 \neq x0$ or $imm \neq 0$ | $2^{17} - 1$ |   |
| ANDI        | $rd=x0$  | $2^{17}$     |   |
| ORI         | $rd=x0$  | $2^{17}$     |   |
| XORI        | $rd=x0$  | $2^{17}$     |   |
| ADD         | $rd=x0$  | $2^{10}$     |   |
| SUB         | $rd=x0$  | $2^{10}$     |   |
| AND         | $rd=x0$  | $2^{10}$     |   |
| OR          | $rd=x0$  | $2^{10}$     |   |
| XOR         | $rd=x0$  | $2^{10}$     |   |
| SLL         | $rd=x0$  | $2^{10}$     |   |
| SRL         | $rd=x0$  | $2^{10}$     |   |
| SRA         | $rd=x0$  | $2^{10}$     |   |
| FENCE       | $pred=0$ or $succ=0$                               | $2^5 - 1$    |   |
| SLTI        | $rd=x0$  | $2^{17}$     | <i>Reserved for custom use</i>          |
| SLTIU       | $rd=x0$  | $2^{17}$     |   |
| SLLI        | $rd=x0$  | $2^{10}$     |   |
| SRLI        | $rd=x0$  | $2^{10}$     |   |
| SRAI        | $rd=x0$  | $2^{10}$     |   |
| SLT         | $rd=x0$  | $2^{10}$     |   |
| SLTU        | $rd=x0$  | $2^{10}$     |   |

# RISCV-LC RV32I Specifications

## NOTICE

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

- Label specification: do not support the colon, and labels should be on the same line with RV32I instructions
- No `.data` and `.text` directives
- Add one pseudo instruction: `LA`
- Add one self-customized directive: `.FILL`
- Add one halt instruction: `HALT`



Do not support the colon, and labels should be on the same line with RV32I instructions

```
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
```

The code snippet under our specifications

LA is translated to two RV32I instructions: `lui` and `addi`.

```
1      lui 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`

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

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

.FILL directive

# Lab 2-1 Assignment

## Get Latest Updates of the Lab

- Click <https://github.com/baichen318>.
- Follow my GitHub.  
**Follow me through GitHub, so that you can see any latest updates of the lab!**

The screenshot shows the GitHub profile page for user **Chen BAI** (username `baichen318`). The profile picture is a circular image of a large stadium. The **Follow** button is circled in red. The page displays the user's public repositories, including `FreePDK45`, `vim.config`, `Raspberry-PI-SmartCar`, `tvm`, `chippard`, and `onnx_tool`. At the bottom, there is a contribution grid for the last year.

Why GitHub? Team Enterprise Explore Marketplace Pricing

Search Sign in Sign up

Overview Repositories 10 Projects Packages Stars 25

**Chen BAI**  
`baichen318`

**Follow**

18 followers · 19 following

The Chinese University of Hong Kong  
<https://baichen318.github.io/>

**Achievements**

**Highlights**  
★ PhD  
Block or Report

**Popular repositories**

- FreePDK45** (Public)  
This is the FreePDK45 V1.4 Process Development Kit for the 45 nm technology  
HTML ☆ 5
- vim.config** (Public)  
My own custom configurations for the Vim editor  
Vim script ☆ 3
- Raspberry-PI-SmartCar** (Public)  
A smart car controlled by Raspberry Pi, can recognize images through TensorFlow  
Python ☆ 1
- tvm** (Public)  
Forked from apache/tvm  
Open deep learning compiler stack for cpu, gpu and specialised accelerators  
Python ☆ 3 🏆 1
- chippard** (Public)  
Forked from ucb-bar/chippard  
An Agile RISC-V SoC Design Framework with in-order cores, out-of-order cores, accelerators, and more  
C ☆ 2
- onnx\_tool** (Public)  
A general tool for ONNX models  
Python ☆ 1

**20 contributions in the last year**

|    | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec | Jan | Feb |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Me | ■   | ■   |     |     |     |     | ■   | ■   |     |     |     |     |     |
| W  | ■   | ■   |     |     |     |     | ■   | ■   |     |     |     |     |     |
| F  |     |     |     |     |     |     | ■   | ■   |     |     |     |     |     |

Learn how we count contributions Less More

### Get the RV32I Assembler

- `$ 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`

### Finish the RV32I assembler including 26+ instructions in asm.c as follows

- Pseudo instruction: la
- 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](#)

## 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: `$ make validate # generate reports inside tools`

## Submission Method:

Submit the source code and report **after** the whole lectures of Lab2 into **Blackboard**.



## 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