香 港 中 文 大 學
The Chinese University of Hong Kong

CENG3420

Lab 3-3: RISC-V Litter Computer (RISC-V LC)

Lancheng ZOU & Shixin Chen
Department of Computer Science & Engineering
Chinese University of Hong Kong
lczou23@cse.cuhk.edu.hk &
sxchen22@cse.cuhk.edu.hk

Spring 2023

Outline

1. Recap
2. RISC-V LC Execution Model
3. Implementations
4. Lab 3-3 Assignment

Recap

Introduction

Overview

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

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

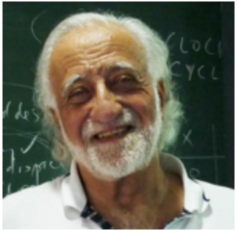
NOTICE

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

Introduction

Our Lab2 & Lab3 are Inspired by LC-3b

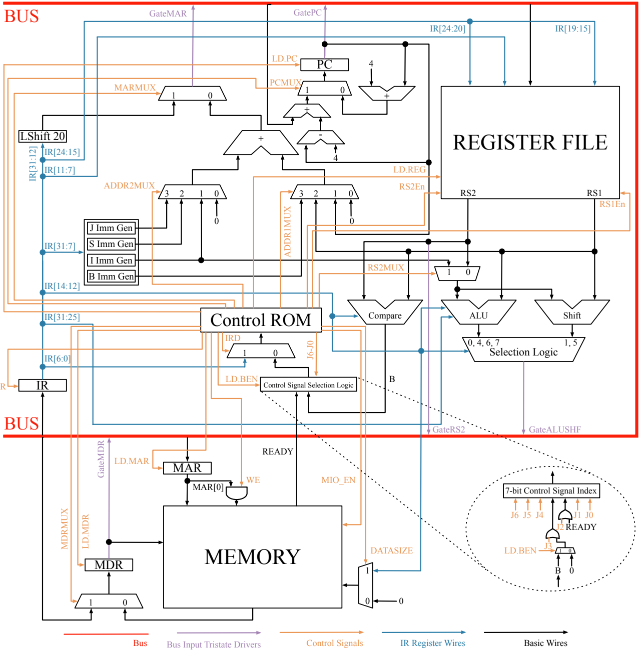
- LC-3b: **Little Computer 3, b** version.
- Relatively simple instruction set.
- Most used in teaching for CS & CE.
- Developed by Yale Patt@UT & Sanjay J. Patel@UIUC.



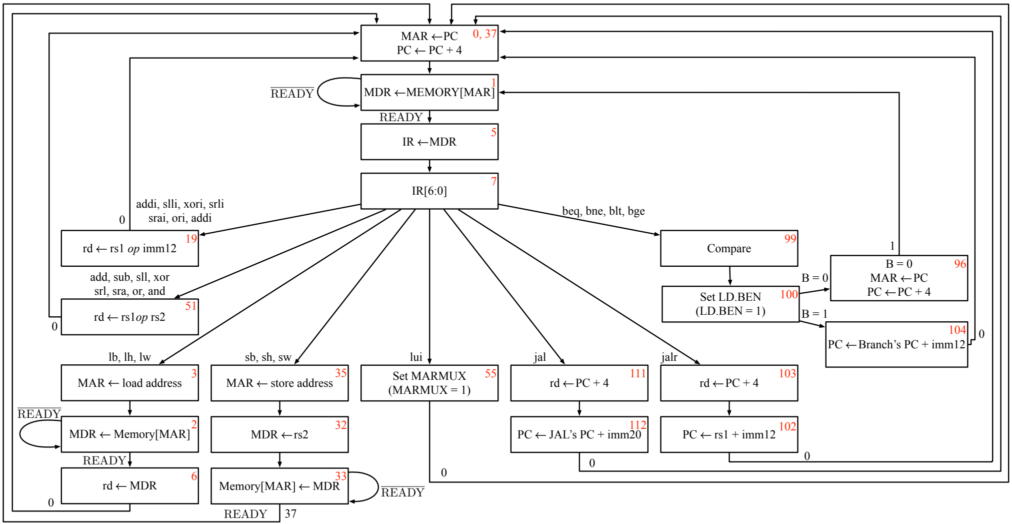
What will we do in Lab 3-3?

- Implement the BUS driver & datapath.

Recap
RISC-V LC Microarchitecture



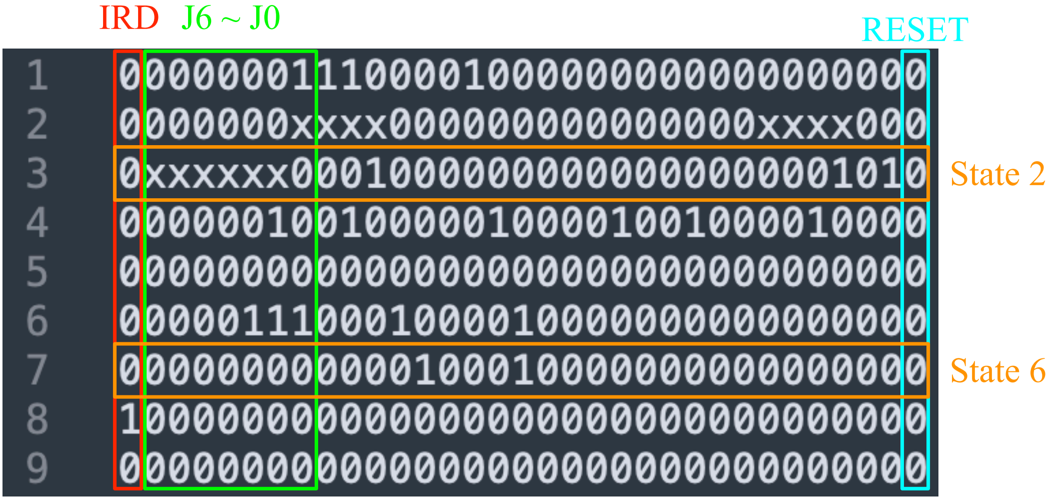
Recap
RISC-V LC Finite State Machine



RISC-V LC Execution Model

RISC-V LC Execution Model

Intialization

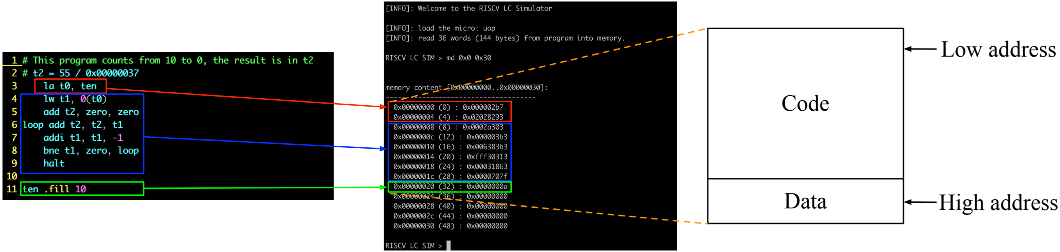


```
000000011100000100000000000000000
0000000xxxx000000000000000xxxx000
0xxxxxx00010000000000000000001010
000000100100000100001001000010000
000000000000000000000000000000000
000001110001000010000000000000000
000000000000010001000000000000000
100000000000000000000000000000000
000000000000000000000000000000000
```

Micro-ops specifications

RISC-V LC Execution Model

The Data Structure Organization in Memory

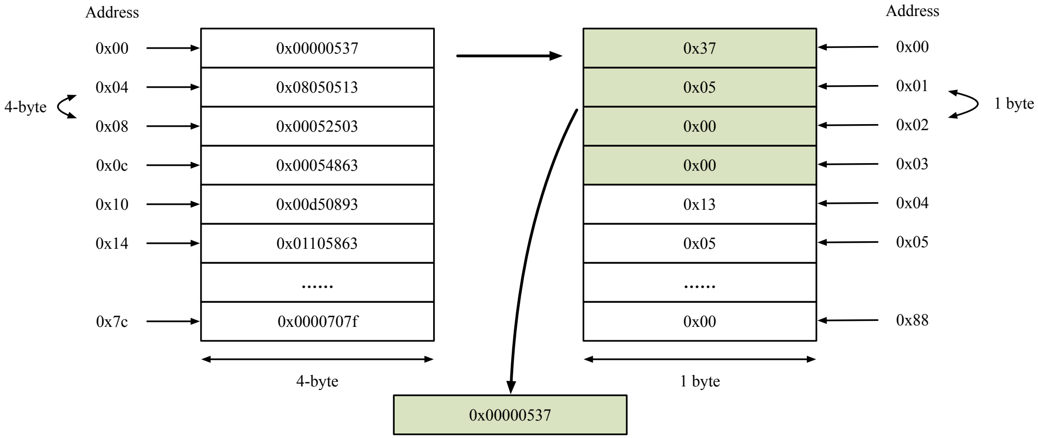


```
# This program counts from 10 to 0, the result is in t2
# t2 = 55 / 0x00000037
    la t0, ten
    lw t1, 0(t0)
    add t2, zero, zero
loop add t2, t2, t1
    addi t1, t1, -1
    bne t1, zero, loop
    halt

ten .fill 10
```

```
[INFO]: Welcome to the RISCV LC Simulator

[INFO]: load the micro: uop
[INFO]: read 36 words (144 bytes) from program into memory.

RISCV LC SIM > md 0x0 0x30


memory content [0x00000000..0x00000030]:
-------------------------------------
  0x00000000 (0) : 0x000002b7
  0x00000004 (4) : 0x02028293
  0x00000008 (8) : 0x0002a303
  0x0000000c (12) : 0x000003b3
  0x00000010 (16) : 0x006383b3
  0x00000014 (20) : 0xfff30313
  0x00000018 (24) : 0x00031863
  0x0000001c (28) : 0x0000707f
  0x00000020 (32) : 0x0000000a
  0x00000024 (36) : 0x00000000
  0x00000028 (40) : 0x00000000
  0x0000002c (44) : 0x00000000
  0x00000030 (48) : 0x00000000

RISCV LC SIM > 
```

Source codes ↔ Machine codes ↔ Organization in memory

RISC-V LC Execution Model
Little Endian One-Byte Addressed Memory



RISCV-LC adopts little endian byte addressed memory.

RISC-V LC Execution Model I

Example of the R-type Instruction

add a4, a2, a0

- PC → BUS
    - In state 0, GatePC is asserted.
- BUS → MAR
    - In state 0, LD_MAR is asserted.
- PC +4 → PC
    - In state 0, PCMUX is deasserted, and LD_PC is asserted.
- State 0 → State 1
    - In state0, we assert J0.
- Memory[MAR] → MDR
    - In state 1, the step will take `MEM_CYCLES` clocks.)
    - J0, LD_MDR, MIO_EN are asserted.
- State 1 → State 5
    - Once the memory finishes the read, **READY is asserted automatically.**

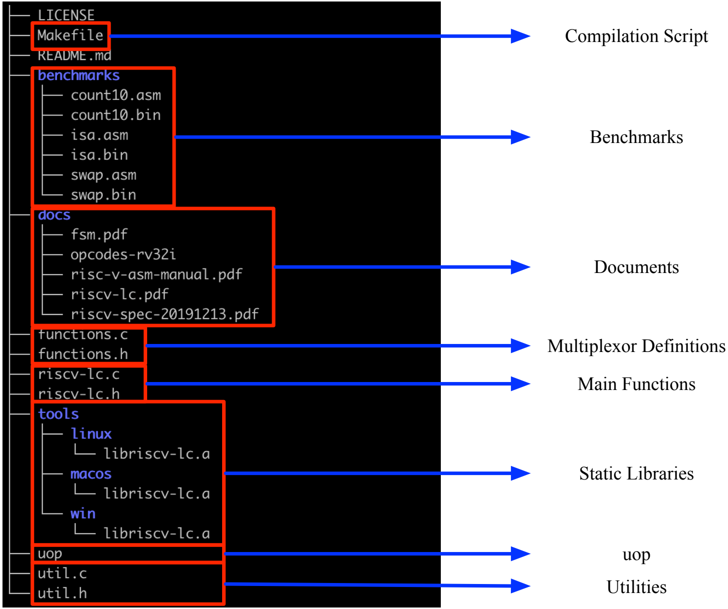
RISC-V LC Execution Model II

Example of the R-type Instruction

  - J2 is asserted automatically once READY is asserted.
- MDR → BUS
  - In state 5, GateMDR is asserted.
- BUS → IR
  - In state 5, LD_IR is asserted.
- State 5 → State 7
  - J2, J1, J0 are asserted.
- Generate control signals according to IR[6:0]
  - In state 7, IRD is asserted.
- R-type addition: a2 + a0
  - In state 51, J6 ∼ J0 are deasserted, RS2En, RS1En are asserted.
- R-type addition: results write back to a4
  - In state 51, LD_REG, GateALUSHF are asserted.
- State 51 → State 0
  - We deassert J6 ∼ J0 to transfer to state 0.

Implementations

Implementations
Repo. Organization



```
├── LICENSE
├── Makefile
├── README.md
├── benchmarks
│   ├── count10.asm
│   ├── count10.bin
│   ├── isa.asm
│   ├── isa.bin
│   ├── swap.asm
│   └── swap.bin
├── docs
│   ├── fsm.pdf
│   ├── opcodes-rv32i
│   ├── risc-v-asm-manual.pdf
│   ├── riscv-lc.pdf
│   └── riscv-spec-20191213.pdf
├── functions.c
├── functions.h
├── riscv-lc.c
├── riscv-lc.h
├── tools
│   ├── linux
│   │   └── libriscv-lc.a
│   ├── macos
│   │   └── libriscv-lc.a
│   └── win
│       └── libriscv-lc.a
├── uop
├── util.c
└── util.h
```

Compilation Script

Benchmarks

Documents

Multiplexor Definitions

Main Functions

Static Libraries

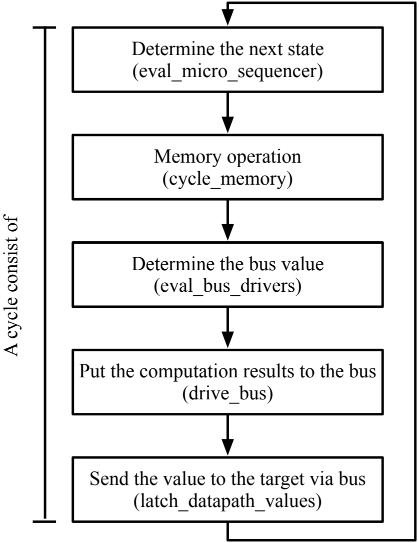
uop

Utilities

Implementations I
Operations in One Clock Cycle

In “riscv-lc.c”:

```
/*
 * execute a cycle
 */
void cycle() {
    /*
     * core steps
     */
    eval_micro_sequencer();
    cycle_memory();
    eval_bus_drivers();
    drive_bus();
    latch_datapath_values();

    CURRENT_LATCHES =
        NEXT_LATCHES;

    CYCLE_COUNT++;
}
```



Operations in one clock cycle.

Implementations I

Five Input Tristate Drivers

In “riscv-lc.c”, function “eval_bus_drivers”:

```
value_of_GatePC = 0;
value_of_GateMAR = 0;
value_of_GateMDR = 0;
value_of_GateALUSHF = 0;
value_of_GateRS2 = 0;
```

Implementations I

Three Intermediate Values for Data Path

In “riscv-lc.c”, function “eval_bus_drivers”:

```
int value_of_MARMUX = 0,
value_of_alu,
value_of_shift_function_unit = 0;
```

Implementations I

Implementation of `value_of_MARMUX`

In “riscv-lc.c”, function “eval_bus_drivers”:

```
value_of_MARMUX = addr2_mux(
    get_ADDR2MUX(CURRENT_LATCHES.MICROINSTRUCTION),
    0,
    sext_unit(mask_val(CURRENT_LATCHES.IR, 31, 20), 12),
    sext_unit(
        s_format_imm_gen_unit(
            mask_val(CURRENT_LATCHES.IR, 11, 7),
            mask_val(CURRENT_LATCHES.IR, 31, 25)
        ),
        12
    ),
    sext_unit(
        j_format_imm_gen_unit(
            mask_val(CURRENT_LATCHES.IR, 31, 31),
            mask_val(CURRENT_LATCHES.IR, 30, 21),
            mask_val(CURRENT_LATCHES.IR, 20, 20),
            mask_val(CURRENT_LATCHES.IR, 19, 12)
```

Implementations II

Implementation of value_of_MARMUX

```
        ),
        20
    )
) + addr1_mux(
    get_ADDR1MUX(CURRENT_LATCHES.MICROINSTRUCTION),
    0,
    CURRENT_LATCHES.PC,
    rs1_en(
        get_RS1En(CURRENT_LATCHES.MICROINSTRUCTION),
        0,
        CURRENT_LATCHES.REGS[mask_val(CURRENT_LATCHES.IR, 19,
           15)]
    ),
    sext_unit(
        b_format_imm_gen_unit(
            mask_val(CURRENT_LATCHES.IR, 7, 7),
            mask_val(CURRENT_LATCHES.IR, 11, 8),
            mask_val(CURRENT_LATCHES.IR, 30, 25),
            mask_val(CURRENT_LATCHES.IR, 31, 31)
```

Implementations III

Implementation of `value_of_MARMUX`

```
            ),
            12
        )
    );
```

Implementations I

Implementation of `value_of_alu`

In “riscv-lc.c”, function “eval_bus_drivers”:

```
value_of_alu = alu(
    mask_val(CURRENT_LATCHES.IR, 14, 12),
    mask_val(CURRENT_LATCHES.IR, 31, 25),
    rs1_en(
        get_RS1En(CURRENT_LATCHES.MICROINSTRUCTION),
        0,
        CURRENT_LATCHES.REGS[mask_val(CURRENT_LATCHES.IR, 19,
           15)]
    ),
    rs2_mux(
        get_RS2MUX(CURRENT_LATCHES.MICROINSTRUCTION),
        rs2_en(
            get_RS2En(CURRENT_LATCHES.MICROINSTRUCTION),
            0,
            CURRENT_LATCHES.REGS[mask_val(CURRENT_LATCHES.IR,
               24, 20)]
        ),
```

Implementations II

Implementation of value_of_alu

```
            sext_unit(mask_val(CURRENT_LATCHES.IR, 31, 20), 12)
        )
    );
```

Implementations I

Intermidate Variables for Bus Drivers

In “riscv-lc.c”, function “drive_bus”:

```
int _GateMAR = get_GateMAR(CURRENT_LATCHES.MICROINSTRUCTION);
int _GateALUSHF = get_GateALUSHF(CURRENT_LATCHES.
   MICROINSTRUCTION);
int _GatePC = get_GatePC(CURRENT_LATCHES.MICROINSTRUCTION);
int _GateRS2 = get_GateRS2(CURRENT_LATCHES.MICROINSTRUCTION);
int _GateMDR = get_GateMDR(CURRENT_LATCHES.MICROINSTRUCTION);
```

Implementations I
Drive Bus

In “riscv-lc.c”, function “drive_bus”:

```
switch ((_GateMDR << 4) + (_GateRS2 << 3) + (_GatePC << 2) + (
   _GateALUSHF << 1) + (_GateMAR)) {
    case 0:
        BUS = 0;
        break;
    case 1:
        error("Lab3-3␣assignment:␣when␣value␣=␣1,␣BUS␣=␣?;\n")
           ;
    case 2:
        error("Lab3-3␣assignment:␣when␣value␣=␣1,␣BUS␣=␣?;\n")
           ;
    case 4:
        error("Lab3-3␣assignment:␣when␣value␣=␣1,␣BUS␣=␣?;\n")
           ;
    case 8:
        error("Lab3-3␣assignment:␣when␣value␣=␣1,␣BUS␣=␣?;\n")
           ;
```

Implementations II

Drive Bus

```
        case 16:
            error("Lab3-3␣assignment:␣when␣value␣=␣1,␣BUS␣=␣?;\n")
               ;
        default:
            BUS = 0;
            warn("unknown␣gate␣drivers␣for␣BUS\n");
    }
```

Lab 3-3 Assignment

Lab 3-3 Assignment

Pre-requisites

Get RISC-V LC

- $ git clone https://github.com/MingjunLi99/ceng3420
- $ cd ceng3420
- $ git checkout lab3.3

Compile (Linux/MacOS environment is suggested)

- $ make

Run the RISC-V LC

- $ ./riscv-lc <uop> <*.bin> # RISCV-LC can execute successfully if you have implemented it.

Lab 3-3 Assignment
Assignment Content

In **riscv-lc.c**,

- Finish eval_bus_drivers
- Finish drive_bus

These unimplemented codes are commented with Lab3-3 assignment.

Lab 3-3 Assignment

Verification

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:

Submit one zip file into **Blackboard**, including

- Three source codes zip files (the entire `riscv-lc` source codes with your implementations).
  - Your implementations of the source codes, *i.e.*, three riscv-lc.c source codes for three parts of Lab 3, and your *uop* should be renamed. The source codes are renamed to `name-sid-lab3-1.c`, `name-sid-lab3-2.c`, `name-sid-lab3-3.c`, and `name-sid-uop` (*e.g.*, `zhangsan-1234567890-lab3-1.c`, `zhangsan-1234567890-lab3-2.c`, *etc.*).
- One report. (name format: `name-sid-lab3.pdf`) The report ONLY includes screenshots (or console results) of all Lab 3 results, *i.e.*, the results of Lab 3-1, Lab 3-2, and Lab 3-3.

Lab 3-3 Assignment
Tips

Tips

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

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

Lab 3-3 Assignment

Submission

Submission Method:

Submit the zip file (including codes and a report) after the whole lectures of Lab3 into **Blackboard**.