香港中文大學
The Chinese University of Hong Kong

CENG3420

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

Chen BAI & Shixin Chen
Department of Computer Science & Engineering
Chinese University of Hong Kong
cbai@cse.cuhk.edu.hk &
sxchen22@cse.cuhk.edu.hk

Spring 2023

Outline

Recap

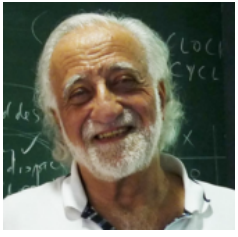
Use C programming language to finish lab assignments in following weeks.

- 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

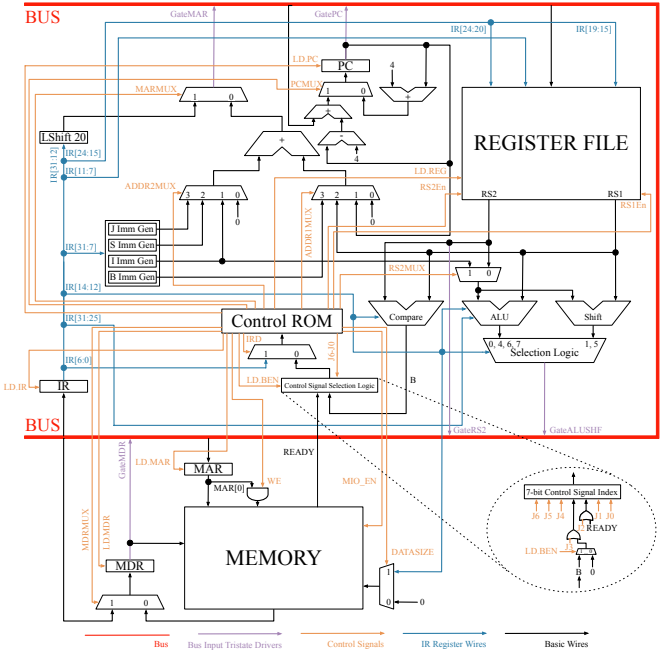
Lab2 & Lab3 are challenging!
Once you have passed Lab2 & Lab3, you will be more familiar with RV32I & a basic implementation!

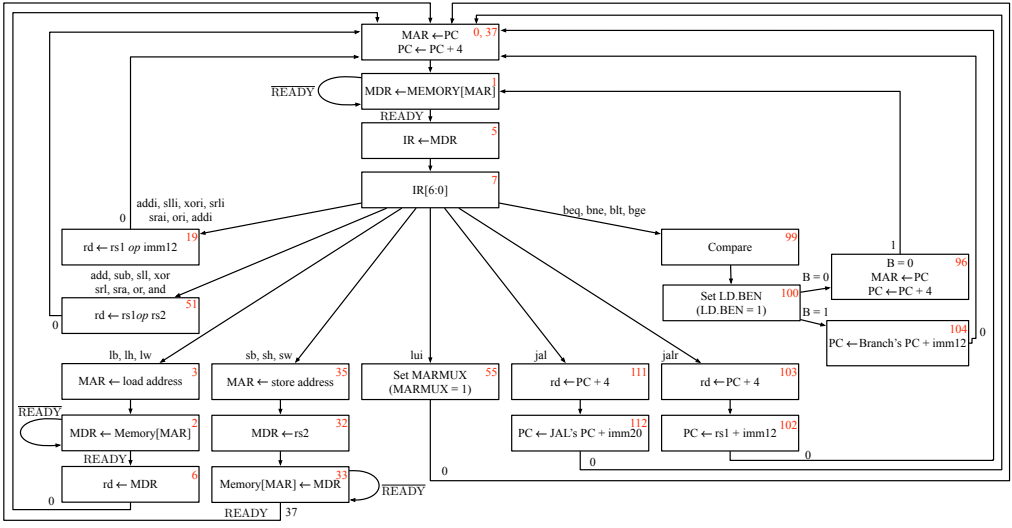
- LC-3b: **Little Computer 3, b** version.
- Relatively simple instruction set.
- Most used in teaching for CS & CE.
- Developed by Yale Patt@UT & Sanjay J. Patel@UIUC.



What will we do in Lab 3-3?

- Implement the BUS driver & datapath.


BUS
GateMAR
GatePC
IR[24:20]
IR[19:15]
LD.PC
PC
4
PCMUX
MARMUX
REGISTER FILE
LShift 20
IR[31:12]
IR[24:15]
IR[11:7]
ADDR2MUX
LD.REG
RS2En
RS2
RS1
RS1En
J Imm Gen
S Imm Gen
I Imm Gen
B Imm Gen
ADDR1MUX
IR[31:7]
IR[14:12]
RS2MUX
Compare
ALU
Shift
IR[31:25]
Control ROM
IRD
IR[6:0]
J6-J0
0, 4, 6, 7
1, 5
Selection Logic
LD.IR
IR
LD.BEN
Control Signal Selection Logic
B
BUS
GateRS2
GateALUSHF
READY
LD.MAR
MAR
GateMDR
MAR[0]
WE
MIO_EN
7-bit Control Signal Index
J6 J5 J4
J3 J2 J1 J0
READY
MDRMUX
LD.MDR
MEMORY
DATASIZE
LD.BEN
MDR
B
Bus
Bus Input Tristate Drivers
Control Signals
IR Register Wires
Basic Wires

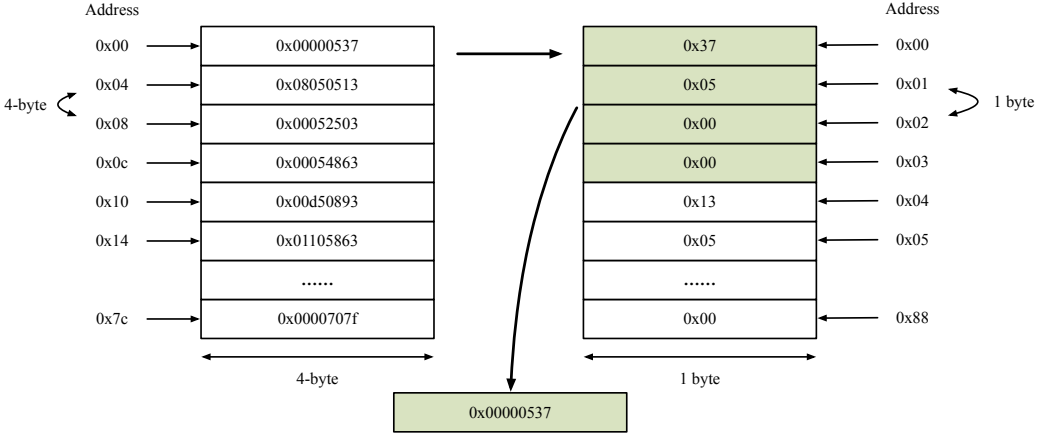

MAR ←PC
PC ← PC + 4
0, 37
READY
MDR ←MEMORY[MAR]
1
READY
IR ←MDR
5
IR[6:0]
7
addi, slli, xori, srli
srai, ori, addi
0
rd ← rs1 op imm12
19
beq, bne, blt, bge
Compare
99
1
B = 0
MAR ←PC
PC ←PC + 4
96
B = 0
Set LD.BEN
(LD.BEN = 1)
100
B = 1
add, sub, sll, xor
srl, sra, or, and
rd ← rs1op rs2
51
0
104
PC ←Branch's PC + imm12
0
lb, lh, lw
sb, sh, sw
lui
jal
jalr
MAR ← load address
3
MAR ← store address
35
Set MARMUX
(MARMUX = 1)
55
rd ←PC + 4
111
rd ←PC + 4
103
READY
MDR ← Memory[MAR]
2
MDR ←rs2
32
PC ← JAL's PC + imm20
112
PC ← rs1 + imm12
102
READY
rd ← MDR
6
Memory[MAR] ← MDR
33
READY
0
0
0
0
READY
37

RISC-V LC Execution Model

Micro-ops specifications

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.

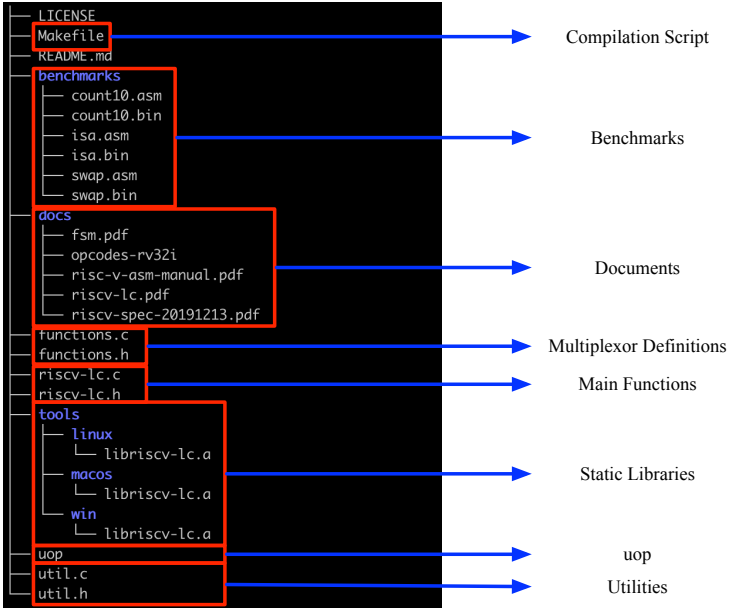
add a4, a2, a0

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

- J2 is asserted automatically once READY is asserted.

- MDR $\rightarrow$ BUS
  - In state 5, J2, J1, J0 are asserted, GateMDR is asserted.
- BUS $\rightarrow$ IR
  - In state 5, LD_IR is asserted.
- Generate control signals according to IR[6:0]
  - In state 7, IRD is asserted.
- R-type addition: a2 + a0
  - In state 51, J6 $\sim$ 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 $\rightarrow$ State 0
  - We deassert J6 $\sim$ J0 to transfer to state 0.

Implementations


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

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++;
}
```

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;
```

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

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

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)
```

```
            ),
            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)
```

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

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)]
        ),
```

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

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);
```

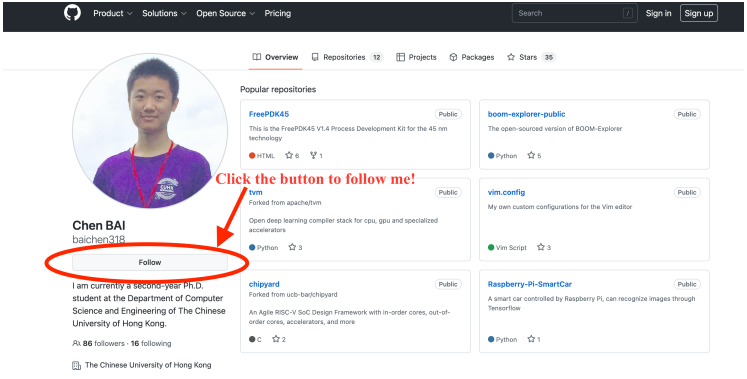
In “riscv-lc.c”, function “drive_bus”:

```
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")
            ;
```

```
    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

- We use `git` to manager our codes, please access `https://github.com/`, and register your account.
- Click `https://github.com/baichen318`.
- Click the **Follow** button.
  Follow me through GitHub, so that you can see any latest updates of the lab!



Get RISC-V LC

- $ git clone https://github.com/baichen318/ceng3420.git
- $ 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.

In **riscv-lc.c**,

- Finish eval_bus_drivers
- Finish drive_bus

These unimplemented codes are commented with Lab3-3 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:

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.c` (*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.

Deadline: 23:59, 3 May. (Wed)

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