香港中文大學
The Chinese University of Hong Kong

CENG3420

Lab 3-2: 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

Introduction

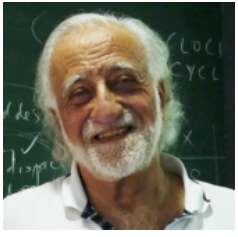
Use C programming language to finish lab assignments in following weeks.

- 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

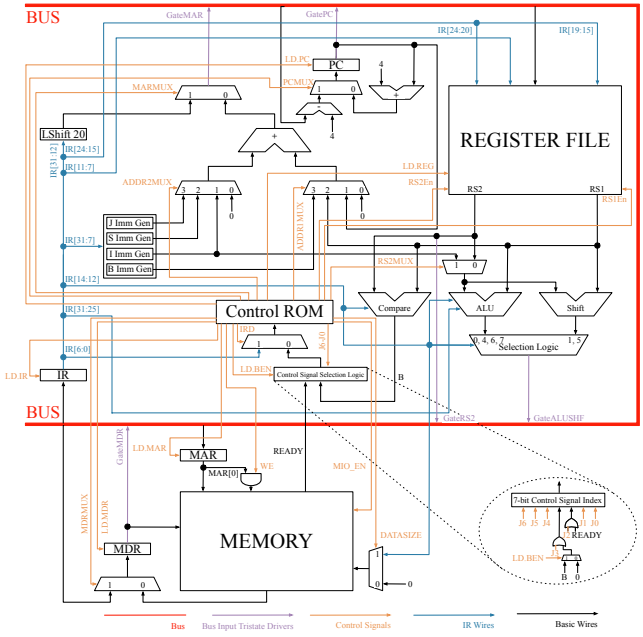
Lab2 & Lab3 are challenging!
Once you have passed Lab2 & Lab3, you will be more familiar with RV32I & a basic implementation!

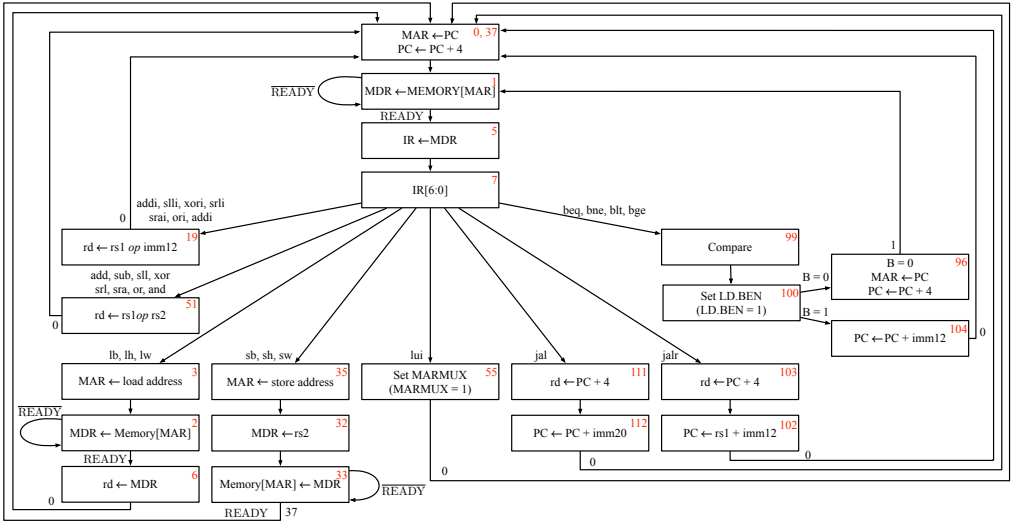
- LC-3b: **Little Computer 3, b** version.
- Relatively simple instruction set.
- Most used in teaching for CS & CE.
- Developed by Yale Patt@UT & Sanjay J. Patel@UIUC.



What will we do in Lab 3-2?

- Implement the memory-related operations: load & store.


BUS
GateMAR
GatePC
IR[24:20]
IR[19:15]
LD.PC
PC
4
PCMUX
MARMUX
REGISTER FILE
L.Shift 20
IR[31:12]
IR[24:15]
IR[11:7]
LD.REG
ADDR2MUX
RS2En
RS2
RS1
RS1En
ADDR1MUX
J Imm Gen
S Imm Gen
I Imm Gen
B Imm Gen
IR[31:7]
IR[14:12]
RS2MUX
Compare
ALU
Shift
Control ROM
IR[31:25]
IRD
IR[6:0]
JB_sel
0, 4, 6, 2
1, 5
Selection Logic
LD.IR
IR
LD.BEN
Control Signal Selection Logic
B
BUS
GateRS2
GateALUSHF
LD.MAR
MAR
READY
GateMDR
MAR[0]
WE
MIO_EN
MDRMUX
LD.MDR
MEMORY
DATASIZE
MDR
7-bit Control Signal Index
J6 J5 J4 J3 J2 J1 J0
READY
LD.BEN
B
Bus
Bus Input Tristate Drivers
Control Signals
IR 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
add, sub, sll, xor
srl, sra, or, and
rd ← rs1op rs2
51
0
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
PC ← PC + imm12
104
0
lb, lh, lw
MAR ← load address
3
READY
MDR ← Memory[MAR]
2
READY
rd ← MDR
6
0
sb, sh, sw
MAR ← store address
35
MDR ←rs2
32
Memory[MAR] ← MDR
33
READY
READY
37
lui
Set MARMUX
(MARMUX = 1)
55
0
jal
rd ←PC + 4
111
PC ← PC + imm20
112
0
jalr
rd ←PC + 4
103
PC ← rs1 + imm12
102
0

- A state in RISCV-LC is indexed by 7 bits (7 control signals, *i.e.*, J6, J5, J4, J3, J2, J1, J0).
- A state in RISCV-LC is consist of 33 bits (33 control signals).
- A control read-only memory (ROM) stores all states.
- Control Signal Selection Logic & IR[6:0] (IRD decides the multiplexor) controls the state transition.
- There are 22 different states for RISCV-LC.

RISCV-LC Execution Model

Notice

- A design with a single bus and distributed registers.
- A single memory holds both instructions and data.
- The first instruction is placed at the address 0x0.
- The data are placed at the end of all instructions.
- PC is initialized as 0x0.
- No pipeline, and no caches.

Source codes ↔ Machine codes ↔ Organization in memory

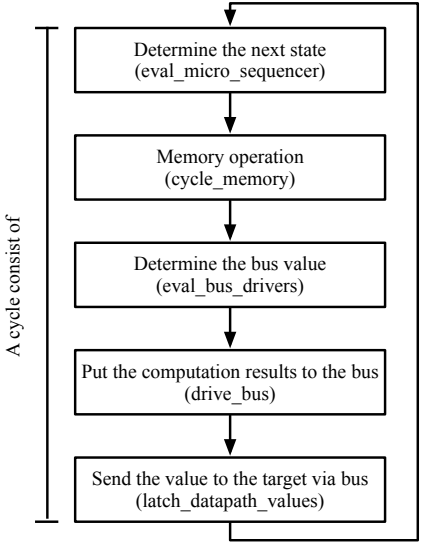
1. PC $\rightarrow$ BUS
2. BUS $\rightarrow$ MAR
3. PC $+4 \rightarrow$ PC
4. Memory[MAR] $\rightarrow$ MDR
5. MDR $\rightarrow$ BUS
6. BUS $\rightarrow$ IR
7. Generate control signals according to IR[6:0]

The detail information is shown in Lab 3-1 slides.

Implementations

In “riscv-lc.c”:

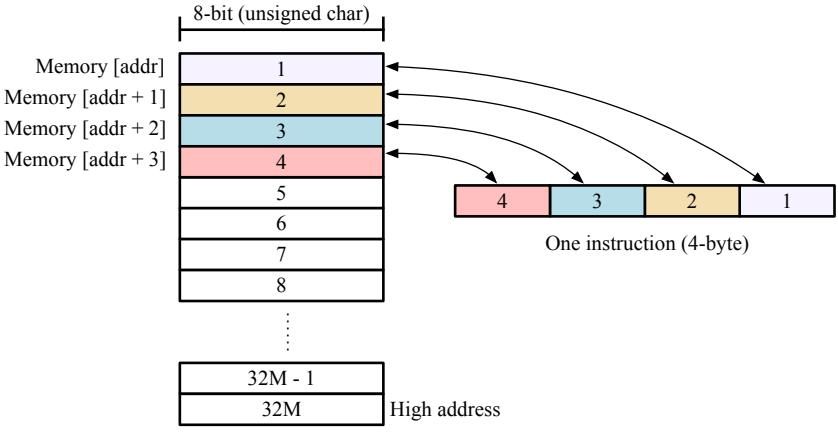
```
/*
 * execute a cycle
 */
void cycle() {
    /*
     * core steps
     */
    eval_micro_sequencer();
    cycle_memory();
    eval_bus_drivers();
    drive_bus();
    latch_datapath_values();

    CURRENT_LATCHES =
        NEXT_LATCHES;

    CYCLE_COUNT++;
}
```



Operations in one clock cycle.

In “riscv-lc.h”

```
/* Main memory */
#define MEM_CYCLES 5
#define BYTES_IN_MEM 0x2000000
unsigned char MEMORY[BYTES_IN_MEM];
/* `MEM_VAL` saves the output of the main memory at each cycle
    */
int MEM_VAL;
```



Memory organization: little endian.

In “functions.c”, we select the memory operation per byte, half word, or word if “data_size” is asserted.

```
int datasize_mux(unsigned int data_size, int funct3, int zero) {
    if (data_size) {
        switch(data_size) {
            case 0:
                return zero;
            case 1:
                return ~(funct3 & 0x3);
        }
    } else
        return zero;
}
```

The potential implementation:

```
// 8-bit
MEMORY[CURRENT_LATCHES.MAR] = MASK7_0(CURRENT_LATCHES.MDR);
// 16-bit
MEMORY[CURRENT_LATCHES.MAR] = MASK7_0(CURRENT_LATCHES.MDR);
MEMORY[CURRENT_LATCHES.MAR + 1] = MASK15_8(CURRENT_LATCHES.MDR);
// ...
...
```

The potential implementation:

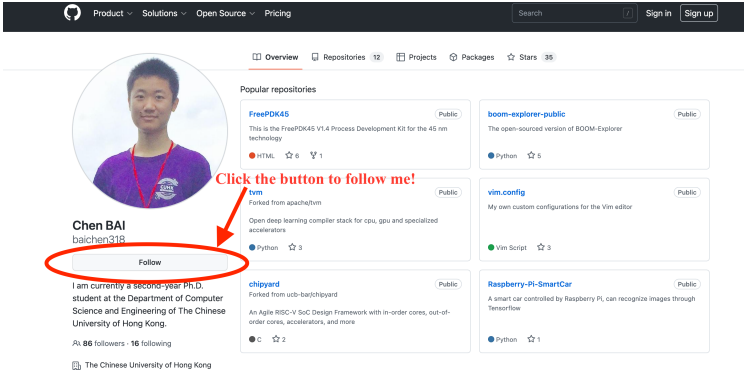
```
// 8-bit
val = sext_unit(MEMORY[CURRENT_LATCHES.MAR], 8);
// 16-bit
val = sext_unit((MEMORY[CURRENT_LATCHES.MAR + 1] << 8) + MEMORY[
    CURRENT_LATCHES.MAR], 16);
// ...
...
```

```
// LD_REG
REGS[mask_val(CURRENT_LATCHES.IR, 11, 7)] = BUS;
// Why do we load a register with a value from bit 7 to bit 11?
```

Lab 3-2 Assignment

Lab 3-2 Assigment
Pre-requisites

- 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 lab3.2

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 cycle_memory
- Finish latch_datapath_values

The unimplemented codes are commented with Lab3-2 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 the zip file (including codes and a report) after the whole lectures of Lab3 into **Blackboard**.

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