香港中文大學
The Chinese University of Hong Kong

CENG3420

Lab 3-1: RISC-V Litter Computer (RISC-V LC)

Chen BAI & Hongduo LIU
Department of Computer Science & Engineering
Chinese University of Hong Kong
cbai@cse.cuhk.edu.hk &
hdliu21@cse.cuhk.edu.hk

Spring 2023

Outline

Introduction

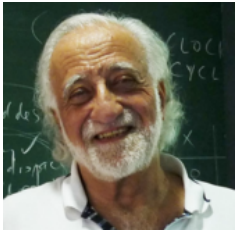
Use C programming language to finish lab assignments in following weeks.

- 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

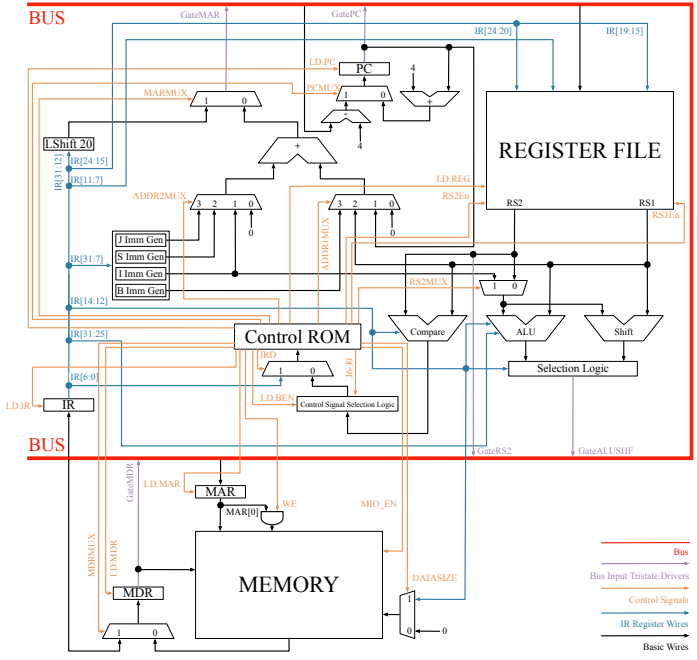
NOTICE

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.



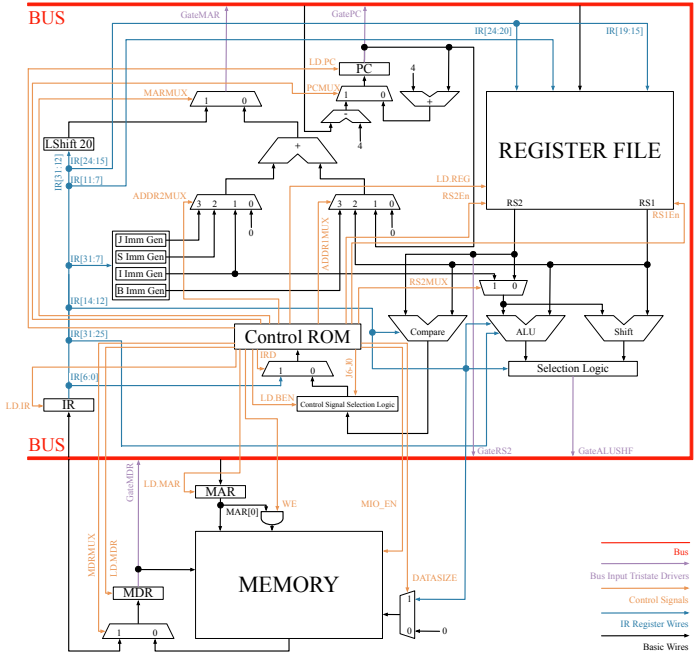
- We construct a microarchitecture for RISCV-LC.
- 32 general-purpose registers.
- Other registers
  - Program Counter (PC), Instruction Register (IR), Memory Address Register (MAR), Memory Data Register (MDR)
- Signals: Branch flag (B), Memory ready signal (READY)
- The behavior of RISCV-LC microarchitecture during a given clock cycle is completely determined by 33 control signals.
- No pipeline.
- Each instruction: Fetch $\rightarrow$ Decode $\rightarrow$ Execution $\rightarrow$ Write back.


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
ADDR1MUX
J Imm Gen
S Imm Gen
I Imm Gen
B Imm Gen
IR[31:7]
IR[14:12]
RS2MUX
Control ROM
Compare
ALU
Shift
IR[31:25]
IR[6:0]
IR[5]
IR[6:0]
Selection Logic
LD.IR
IR
LD.BEN
Control Signal Selection Logic
BUS
GateRS2
GateALUSHF
GateMDR
LD.MAR
MAR
MAR[0]
WE
MIO_EN
MDRMUX
LD.MDR
MEMORY
DATASIZE
MDR
Bus
Bus Input Tristate Drivers
Control Signals
IR Register Wires
Basic Wires

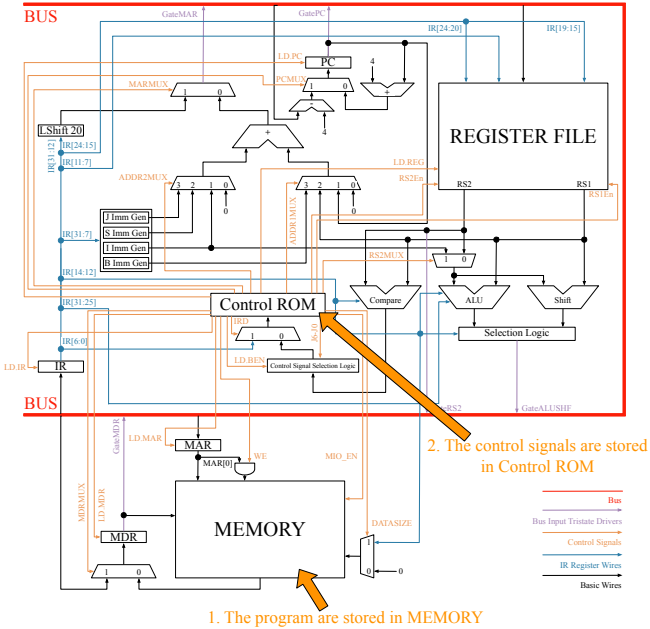
What will we do in Lab3?
Three challenging tasks:

- Implement the finite state machine for RISC-V LC (Lab 3-1).
- Implement the memory-related operations (load & store) (Lab 3-2).
- Implement the datapath for RISC-V LC Microarchitecture (Lab 3-3).

RISCV-LC Microarchitecture


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]
LD.REG
RS2En
RS2
RS1
RS1En
ADDR2MUX
ADDR1MUX
J Imm Gen
S Imm Gen
I Imm Gen
B Imm Gen
IR[31:7]
IR[14:12]
RS2MUX
Compare
ALU
Shift
IR[31:25]
Control ROM
IR[6:0]
IR[5]
IR[6:0]
Selection Logic
LD.IR
IR
LD.BEN
Control Signal Selection Logic
BUS
GateRS2
GateALUSHF
GateMDR
LD.MAR
MAR
MAR[0]
WE
MIO_EN
MDRMUX
LD.MDR
MEMORY
DATASIZE
MDR
Bus
Bus Input Tristate Drivers
Control Signals
IR Register Wires
Basic Wires

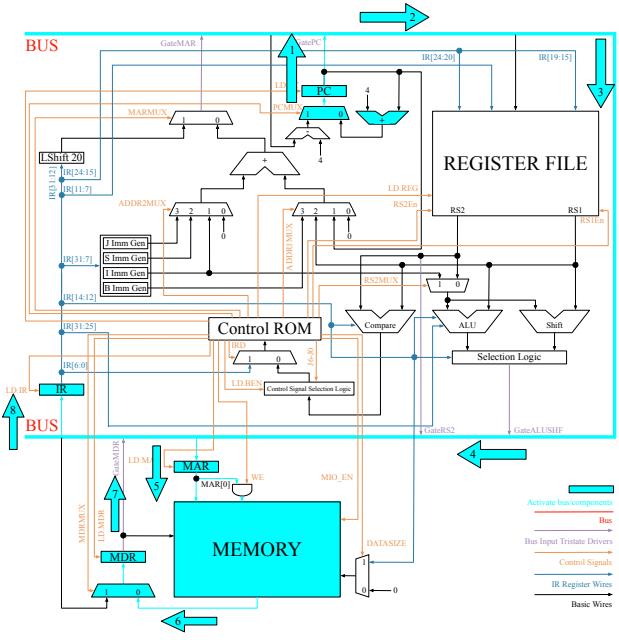
- Load the program to the memory.
- Load the control signal tables to the control read-only memory (ROM).
- Initialize RISCV-LC with the state 0 (states are discussed in the finite state machine section).
- PC is initialized with zero.



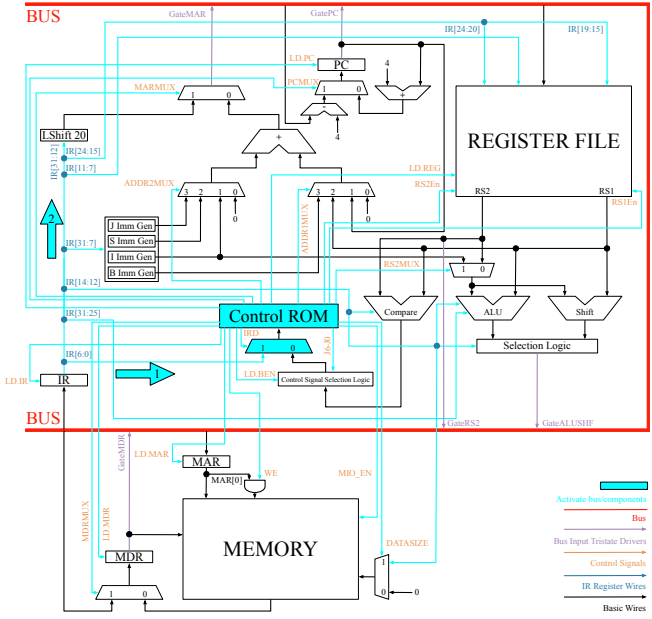
2. The control signals are stored in Control ROM

1. The program are stored in MEMORY

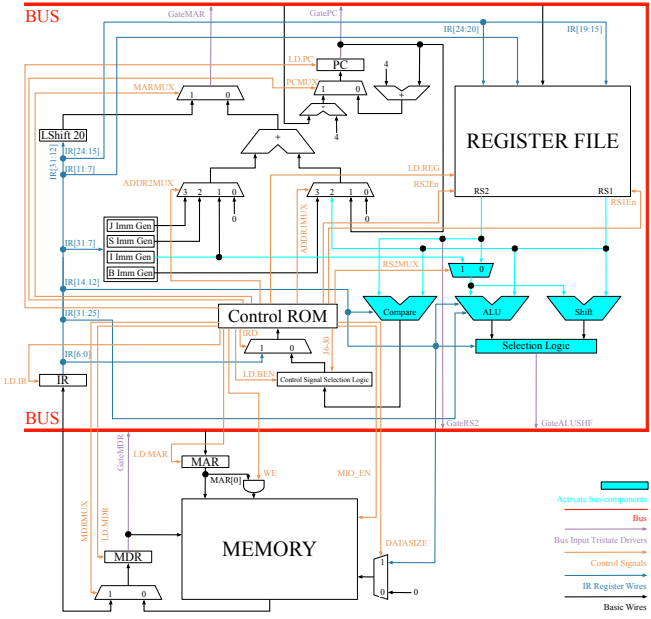
- PC $\rightarrow$ BUS $\rightarrow$ MAR
- Read the memory with the address value in MAR, and store the result in MDR: MEMORY[MAR] $\rightarrow$ MDR
- Load the instruction to IR: MDR $\rightarrow$ BUS $\rightarrow$ IR
- PC + 4 $\rightarrow$ PC


BUS
GateMAR
GatePC
LD.PC
PC
PCMUX
MARMUX
LShift 20
IR[31:12]
IR[24:15]
IR[11:7]
IR[24:20]
IR[19:15]
REGISTER FILE
LD.REG
RS2En
RS2
RS1
DSTLD
ADDR2MUX
ADDR1MUX
J Imm Gen
S Imm Gen
I Imm Gen
B Imm Gen
IR[31:7]
IR[14:12]
IR[31:25]
IR[6:0]
RS2MUX
Compare
ALU
Shift
Control ROM
IRD
Selection Logic
LD.BEN
Control Signal Selection Logic
LD.IR
IR
GateRS2
GateALUSHF
GateMDR
LD.MAR
MAR
MAR[0]
WE
MIO_EN
MDRMUX
LD.MDR
MDR
MEMORY
DATASIZE
Activate bus/components
Bus
Bus Input Tristate Drivers
Control Signals
IR Register Wires
Basic Wires

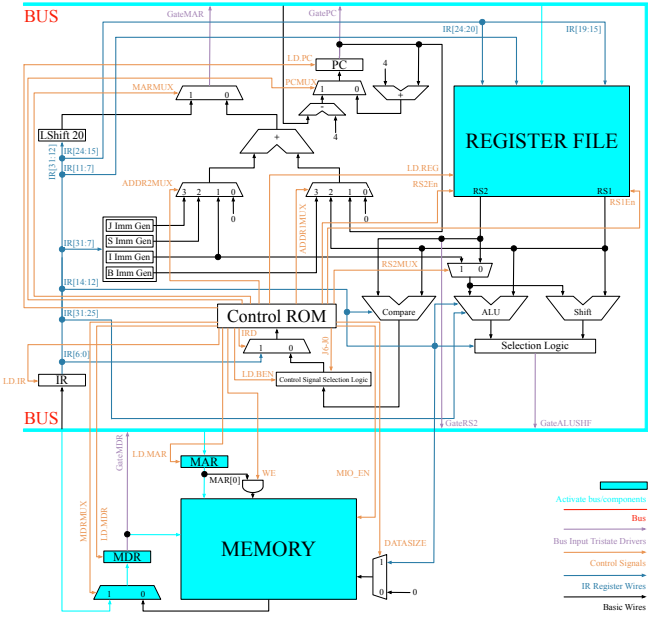
- Determine instruction types with IR[6:0], IR[14:12], *etc.*
- Control signals are generated according to IR[6:0] if IRD is asserted.
- Some immediate values can be directly extracted from IR.


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
RS1En
RS2
RS1
ADDR1MUX
J Imm Gen
S Imm Gen
I Imm Gen
B Imm Gen
IR[31:7]
IR[14:12]
IR[31:25]
IR[6:0]
RS2MUX
Compare
ALU
Shift
Control ROM
IRD
Selection Logic
LD.IR
IR
LD.BEN
Control Signal Selection Logic
BUS
GateRS2
GateALUSHF
GateMDR
LD.MAR
MAR
MAR[0]
WE
MIO_EN
MDRMUX
LD.MDR
MDR
MEMORY
DATASIZE
Activate bus/components
Bus
Bus Input Tristate Drivers
Control Signals
IR Register Wires
Basic Wires

- Execute instructions with the corresponing logics, *e.g.*, Compare, ALU, Shift, *etc.*
- Some instructions may use BUS.


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
ADDR1MUX
J Imm Gen
S Imm Gen
I Imm Gen
B Imm Gen
IR[31:7]
IR[14:12]
RS2MUX
Compare
ALU
Shift
IR[31:25]
Control ROM
IRD
IR[6:0]
Selection Logic
LD.IR
IR
LD.BEN
Control Signal Selection Logic
BUS
GateRS2
GateALUSHF
GateMDR
LD.MAR
MAR
MAR[0]
WE
MIO_EN
MDRMUX
LD.MDR
MDR
MEMORY
DATASIZE
Activate bus/components
Bus
Bus Input Tristate Drivers
Control Signals
IR Register Wires
Basic Wires

- Write results back to the register or memory.
- BUS are used to transfer results.


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
ADDR1MUX
J Imm Gen
S Imm Gen
I Imm Gen
B Imm Gen
IR[31:7]
IR[14:12]
IR[31:25]
IR[6:0]
RS2MUX
Compare
ALU
Shift
Control ROM
IRD
Selection Logic
LD.BEN
Control Signal Selection Logic
LD.IR
IR
BUS
GateRS2
GateALUSHF
GateMDR
LD.MAR
MAR
MAR[0]
WE
MIO_EN
MEMORY
MDR
MDR.MUX
LD.MDR
DATASIZE
Activate bus/components
Bus
Bus Input Tristate Drivers
Control Signals
IR Register Wires
Basic Wires

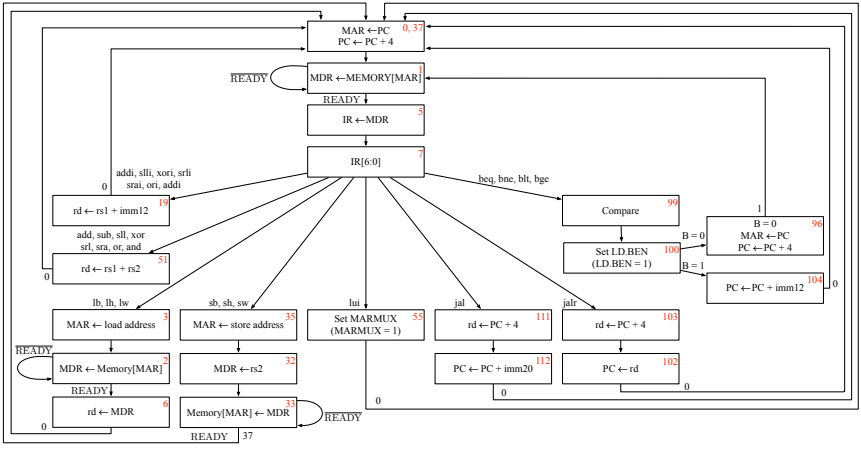
Finite State Machine

Finite State Machine

- Time is divided into clock cycles.
- At each clock cycle, the behavior of RISCV-LC is defined by 33 control signals.
- At each clock cycle, 33 control signals are also responsible for determining the behavior of the following clock cycle.
- We use a terminology state to represent a valid combination of 33 control signals.
- Each state is encoded with a unique number. The number of states equals the number of behaviors of RISCV-LC.
- We define a finite state machine to describe all states and their relations.
- There are 22 different states for RISCV-LC in total.

Definition

Finite State Machine (FSM) is a graph. It describes states (behaviors) and relations.



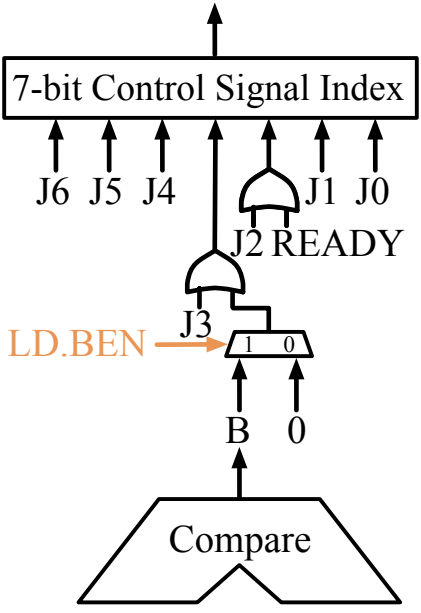
- Each box is a state, and numbers colored in red denote the state number.
- Edges are relations between states.
- RISCV-LC is initialized with the state 0.
- State 127 is a HALT state, and it is ignored in the figure.

Index	Control Signals	Descriptions
1	IRD	mux. signal determines the next state
2	J6, J5, J4, J3, J2, J1, J0	signals determine the state transition
9	LD.PC	mux. signal determines PC load address
10	LD.MAR	enable signal for loading the address from the bus
11	LD.MDR	enable signal for loading the value to MDR
12	LD.IR	enable signal for loading the instruction from the bus
13	LD.REG	enable signal for storing the register value from the bus
14	LD.BEN	mux. signal determines the usage of B in Control Signal Selection Logic
15	GatePC	enable signal for bus input tristate driver of PC
16	GateMAR	enable signal for bus input tristate driver of MAR
17	GateMDR	enable signal for bus input tristate driver of MDR
18	GateALUSHF	enable signal for bus input tristate driver of ALUSHF
19	GateRS2	enable signal for bus input tristate driver of RS2
20	PCMUX	mux. signal for selecting the address for PC
21	ADDR1MUX	mux. signal for selecting B format imm., RS1, PC, and zero
23	ADDR2MUX	mux. signal for selecting J format imm., S format imm., I format imm., and zero
25	MARMUX	mux. signal for selecting values for MAR
26	MDRMUX	mux. signal for selecting values for MDR
27	RS2MUX	mux. signal for selecting I format imm, and RS2
28	RS2En	enable signal for outputting RS2
29	RS1En	enable signal for outputting RS1
30	MIO_EN	enable signal for the main memory
31	WE	enable signal for writing the main memory
32	DATASIZE	mux. signal for selecting the bit width
33	RESET	reset signal for RISC-V LC

Table: Control Signals

Mux. signal is a multiplex selection signal.

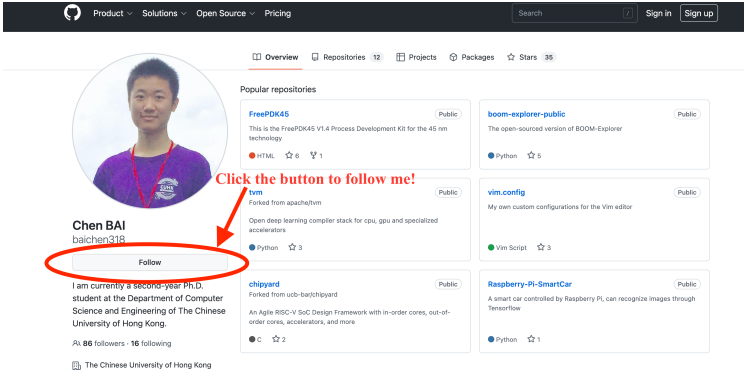
- A state is indexed by 7 bits, *i.e.* J6, J5, J4, J3, J2, J1, J0. Theoretically, we can use 128 states in total, but we only use 22 states. Unused states are reserved for future extensions.
- A control read-only memory (ROM) stores all states.
- Control Signal Selection Logic & IR[6:0] controls the state transition.


7-bit Control Signal Index
J6
J5
J4
J1
J0
J2 READY
J3
LD.BEN
1
0
B
0
Compare

- In assignments, a file named `uop` stores all control signals, which composes a 128×33 matrix.
- 1 means set the signal for the state, and 0 means do not set the signal for the state.
- x means you need to determine whether you need to set the signal or not set the signal.

Lab 3-1 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!



Lab 3-1 Assignment
Compile & Run RISCV-LC Simulator

Get RISC-V LC

- $ git clone https://github.com/baichen318/ceng3420.git
- $ cd ceng3420
- $ git checkout lab3.1

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.

- Implement all missing control signals in **uop**.
  - Fill *x* with the correct 0 or 1 according to our FSM, as shown in page 23.

```
00000001110000100000000000000000
0000000xxxx00000000000000xxxx000
0xxxxxx00010000000000000001010
```

- Implement the code that the register x0 is hard-wired to zero in **riscv-lc.c**.

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

Appendix

```
#define CONTROL_STATE_ROWS 128
#define INITIAL_STATE_NUMBER 0
/* definition of the bit for the control signals */
enum {
  IRD, // 1
  J6, J5, J4, J3, J2, J1, J0, // 2-8
  LD_PC, // 9
  LD_MAR, // 10
  LD_MDR, // 11
  LD_IR, // 12
  LD_REG, // 13
  LD_BEN, // 14
  GatePC, // 15
  GateMAR, // 16
  GateMDR, // 17
  GateALUSHF, // 18
```

```
  GateRS2, // 19
  PCMUX, // 20
  ADDR1MUX1, ADDR1MUX0, // 21-22
  ADDR2MUX1, ADDR2MUX0, // 23-24
  MARMUX, // 25
  MDRMUX, // 26
  RS2MUX, // 27
  RS2En, // 28
  RS1En, // 29
  MIO_EN, // 30
  WE, // 31
  DATASIZE, // 32
  RESET, // 33
  CONTROL_SIGNAL_BITS
} CONTROL_SIGNALS;
/* The control store ROM */
```

```
int CONTROL_STATE[CONTROL_STATE_ROWS][
   CONTROL_SIGNAL_BITS];
```

```
/*
 * definitions of special base addresses
 * 'CODE_BASE_ADDR': the base address to locate source codes
 * 'TRAPVEC_BASE_ADDR': exceptions & interruptions' base
    address
 */
#define CODE_BASE_ADDR 0x0
#define TRAPVEC_BASE_ADDR 0x400000
```

```
/* 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;
```

```
/*
 * 'struct_system_latches' stores the status of 'PC' and other
     registers
 */
typedef struct {
    /* program counter */
    int PC;
    /* register file  */
    int REGS[NUM_RISCV_LC_REGS];
    /* memory data register */
    int MDR;
    /* memory address register */
    unsigned int MAR;
    /* instruction register */
    unsigned int IR;
    /* branch flag register */
    unsigned int B;
    /* memory ready bit indicator */
    int READY;
```

```
    /* micro-code / microintruction */
    int MICROINSTRUCTION[CONTROL_SIGNAL_BITS];
    /* state number: provided for computer system debugging */
    int STATE_NUMBER;
} struct_system_latches;
```

```
/* bus input tristate drivers */
int value_of_GatePC;
int value_of_GateMAR;
int value_of_GateMDR;
int value_of_GateALUSHF;
int value_of_GateRS2;
/* value of the bus */
int BUS;
```

```
/*
 * 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++;
}
```