



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

Department of Computer Science & Engineering

Chinese University of Hong Kong

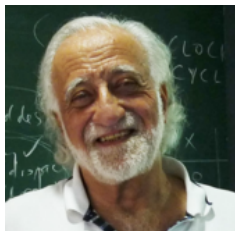
cbai@cse.cuhk.edu.hk

Spring 2022

- ① Introduction
- ② Finite State Machine
- ③ Lab 3-1 Assignment
- ④ Appendix

Introduction

- 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.



- Inspired from LC-3b.
- Focus on [RV32I](#).
- Programming language: C.
- Compatible with Lab2 (RV32I assembler & RV32I simulator).
- Single system bus distributed registers design.

RV32I ISA basic requirements:

- RISC-V 32 general-purpose registers.
- 32-bit data and address width.
- 25+ basic instructions.

Plus several special-purpose registers:

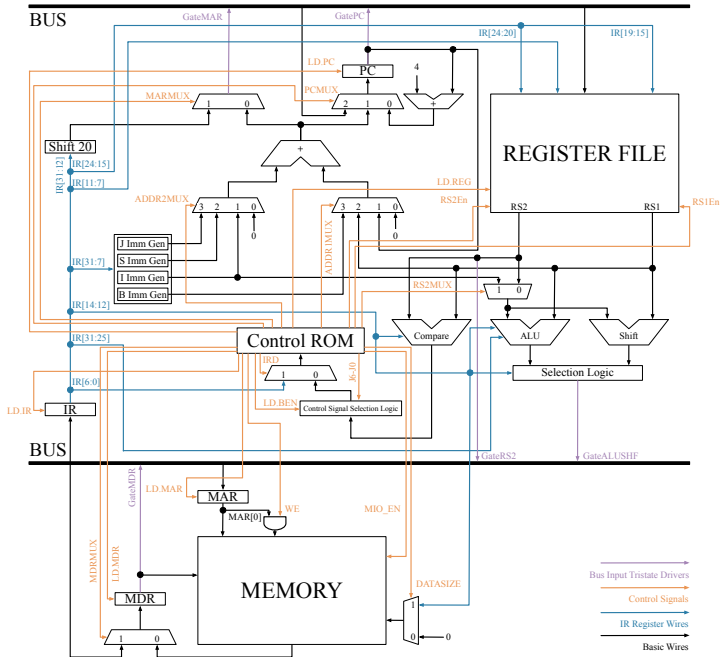
- Program Counter (**PC**)
- Instruction Register (**IR**)
- Memory Address Register (**MAR**)
- Memory Data Register (**MDR**)

Misc. registers:

- Branch flag register (**B**)
- Memory ready bit indicator (**READY**)

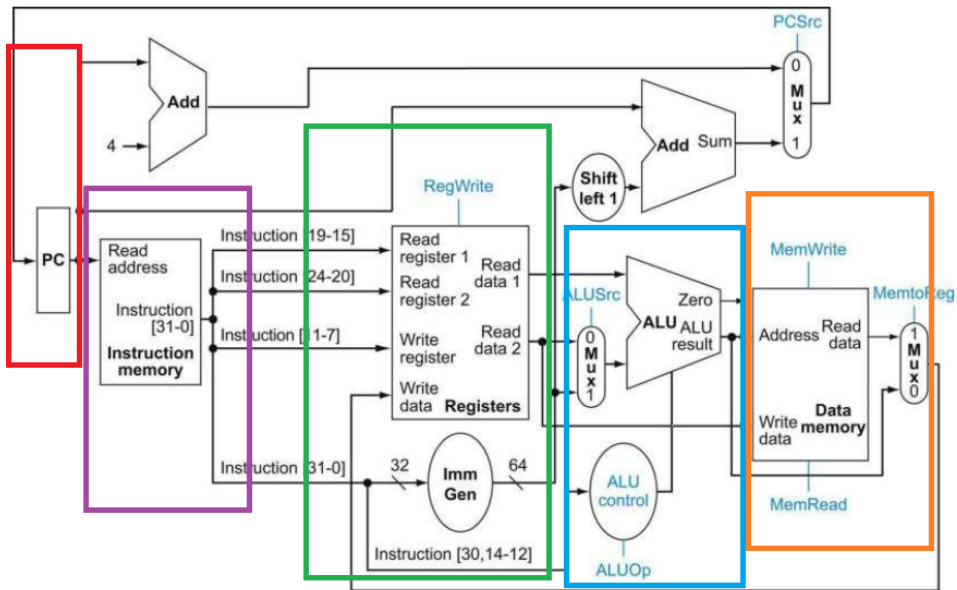
Introduction

RISC-V LC



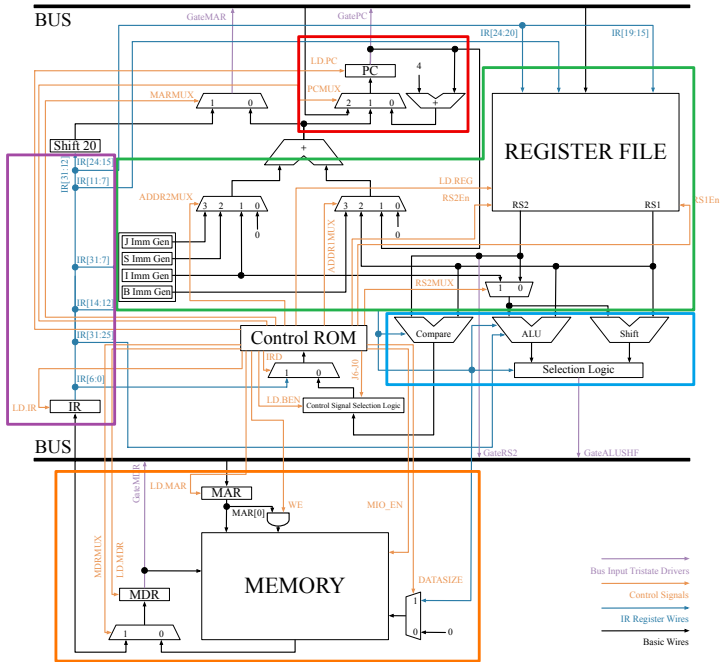
Introduction

RISC-V LC



Introduction

RISC-V LC



What will we do in Lab3?

Three **challenging** tasks:

- Figure out the finite state machine for RISC-V LC ([Lab 3-1](#)).
- Implement the memory-related operations ([Lab 3-2](#)).
- Implement the datapaths for RISC-V LC ([Lab 3-3](#)).

Finite State Machine

Definition

Finite State Machine (FSM) is an abstract machine that can be in exactly one of a finite number of states at any given time.

RISC-V LC FSM

RISC-V LC FSM determines the state of RISC-V LC at any given cycle.

- $PC \leftarrow PC + 4$
- $Memory \leftarrow MDR$
- $rd \leftarrow MDR$
- ...

- A state in RISC-V-LC is indexed by 7 bits (7 control signals, *i.e.*, J6, J5, J4, J3, J2, J1, J0).
- A state in RISC-V-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] controls the state transition.
- There are 22 different states for RISC-V-LC.

Finite State Machine

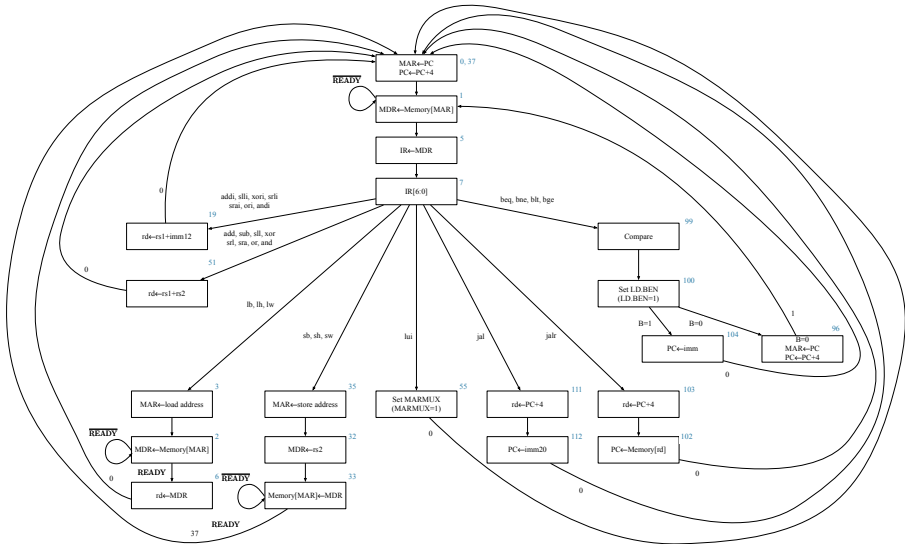
Control Signals

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

Finite State Machine

Finite State Machine in RISC-V-LC

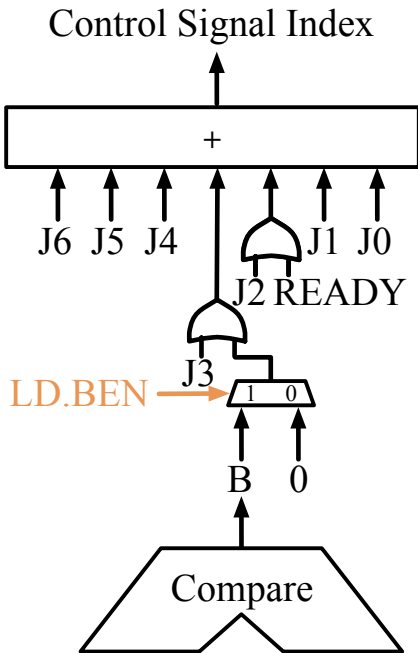


Notice

State 127 is a HALT state, and it is ignored in the state transition figure.

Finite State Machine

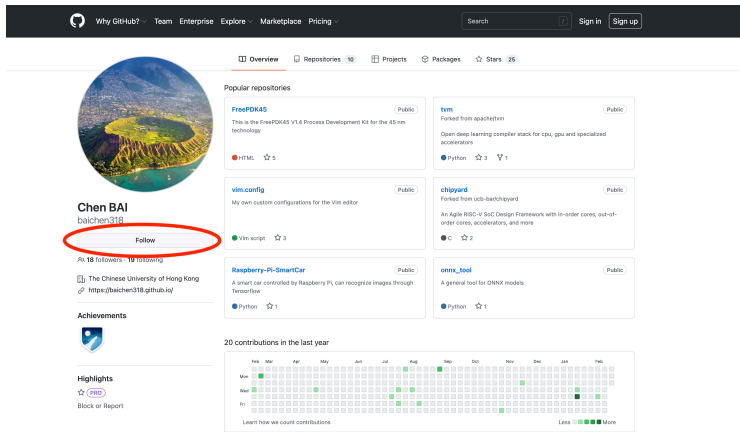
Control Signal Selection Logic



Lab 3-1 Assignment

Get Latest Updates of the Lab

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



The screenshot shows the GitHub profile page for user 'baichen318'. The profile name 'Chen BAI' and the username 'baichen318' are visible. A red circle highlights the 'Follow' button. The page displays several popular repositories:

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

At the bottom, there is a '20 contributions in the last year' section with a calendar grid showing activity from February to February.

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> # RISC-V-LC can execute successfully if you have implemented it.`

Benchmarks

Verify your codes with these benchmarks (inside the `benchmarks` directory)

- `isa.bin`
- `count10.bin`
- `swap.bin`

Verification

- `isa.bin` → `a3 = -18/0xffffffff` and `MEMORY[0x84 + 16] = 0xffffffff`
- `count10.bin` → `t2 = 55/0x00000037`
- `swap.bin` → `NUM1` changes from `0xabcd` to `0x1234` and `NUM2` changes from `0x1234` to `0xabcd`

Submission Method:

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

Tips

Inside `docs`, there are three 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

Appendix I

Control Signals Specifications

```
#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
```

Appendix II

Control Signals Specifications

```
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;
```

Appendix I

Critical Latches/Registers Specifications

```
/*
 * '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;
}
```

Appendix II

Critical Latches/Registers Specifications

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

Appendix I

Bus Input Tristate Drivers & BUS

```
/* 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;
```


Appendix I

Operations in One Clock Cycle

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