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
Department of Computer Science & Engineering
Chinese University of Hong Kong
`cbai@cse.cuhk.edu.hk`

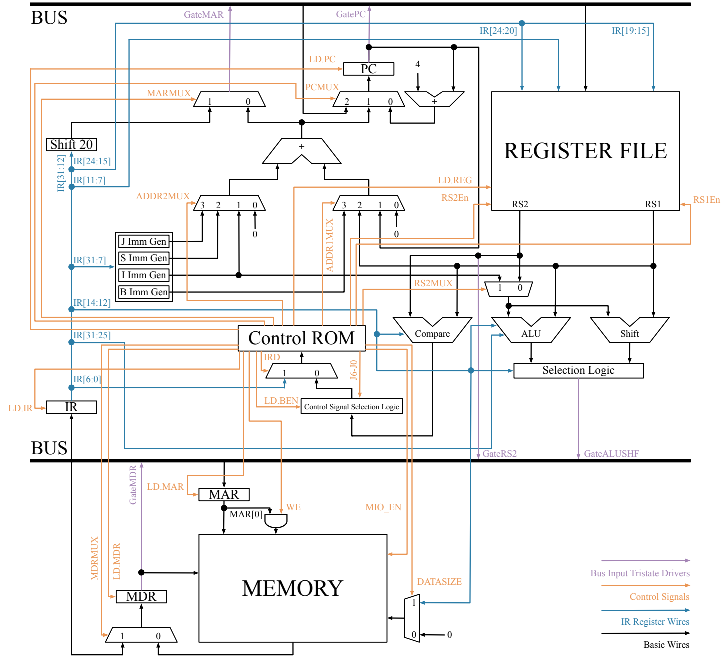
Spring 2022

Outline

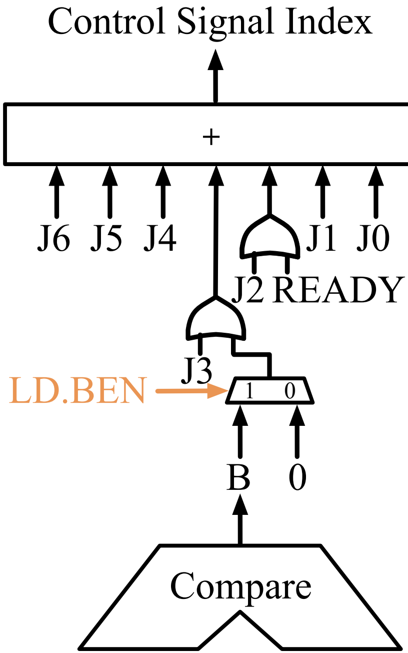
Introduction

Introduction

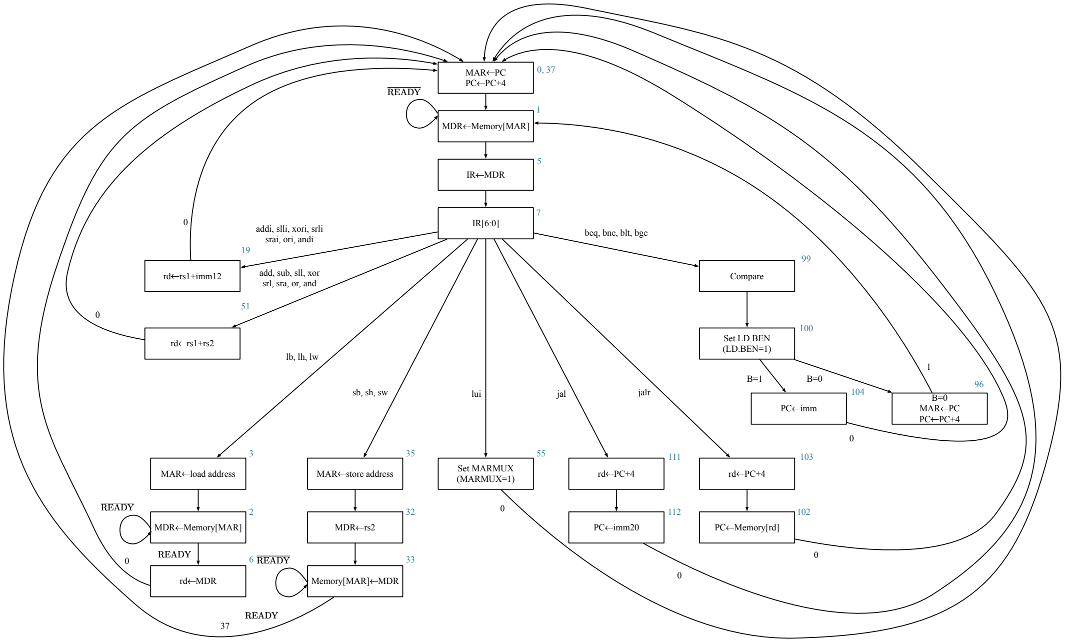
RISC-V LC



Control Signal Selection Logic



Introduction
RISC-V LC



Introduction
Finite State Machine in RISCV-LC

- 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] controls the state transition.
- There are 22 different states for RISCV-LC.

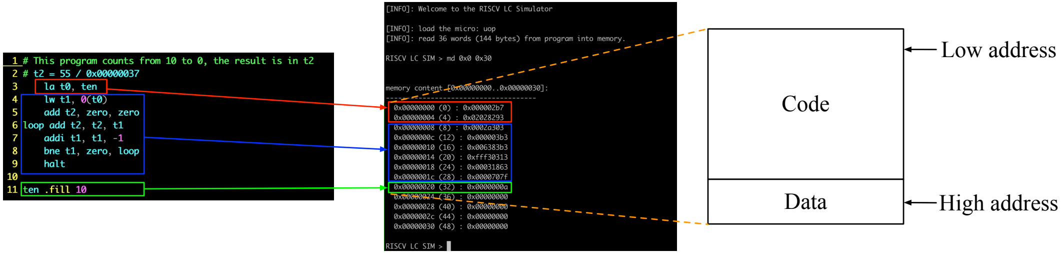
Workflow

Notice

- Single bus distributed registers design.
- 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.

Workflow

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

Workflow
Initial Steps

- PC $\rightarrow$ BUS
- BUS $\rightarrow$ MAR
- PC $+4 \rightarrow$ PC
- Memory[MAR] $\rightarrow$ MDR
- MDR $\rightarrow$ BUS
- BUS $\rightarrow$ IR
- Generate control signals according to IR[6:0]

Implementations

Implementations 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++;
}
```

Implementations I
Memory Specifications

```
/* 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;
```

Implementations I
Data Size Selection

```c
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;
}
```

Implementations I
Write Memory

```
// 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);
// ...
...
```

Implementations I
Read Memory

```
// 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);
// ...
...
```

Implementations I
Load Values

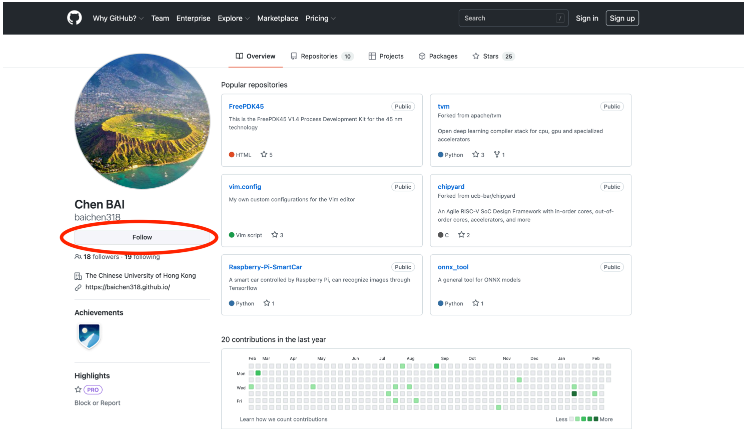
```
// LD_REG
REGS[mask_val(CURRENT_LATCHES.IR, 11, 7)] = BUS;
// Why do we load a register with a value from bit 7 to bit 11?
```

Lab 3-2 Assignment

Lab 3-2 Assignment
Pre-requisites

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!



Chen BAI
baichen318

18 followers · 19 following

The Chinese University of Hong Kong
https://baichen318.github.io/

Popular repositories

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

20 contributions in the last year

Learn how we count contributions

Lab 3-2 Assignment
Pre-requisites

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.

Lab3.2 Assignment
Assignment Content

In **riscv-lc.c**,

- Finish cycle_memory
- Finish latch_datapath_values

These unimplemented codes are commented with Lab3-2 assignment

Lab 3-2 Assignment
Verification

Benchmarks

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

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

Verification

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

Lab 3-2 Assignment
Submission

Submission Method:

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

Lab 3-2 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