模拟器实验报告

班级：物联 1601

学号：201608010628

姓名：曾彤芳

实验目标

完成一个模拟 RISC-V 的基本整数指令集 RV32I 的模拟器设计

实验要求

- 使用 C/C++编写模拟器代码
- 模拟器的输入是二进制的机器指令文件
- 模拟器的输出是 CPU 各个寄存器的状态和相关的存储器单元状态

实验内容

RISC-V 指令集介绍

RISC-V 指令集 是基于精简指令集计算(RISC)原理建立的开放指令集架构(ISA)，RISC-V 是在指令集不断发展和成熟的基础上建立的全新指令。RISC-V 指令集完全开源，设计简单，易于移植*nix 系统，模块化设计，完整工具链，同时有大量的开源实现和流片案例，已在社区得到大力支持。RISC-V(读作"RISC-FIVE")是基于精简指令集计算(RISC)原理建立的开放指令集架构(ISA)，V 表示为第五代 RISC(精简指令集计算机),表示此前已经四代 RISC 处理器原型芯片。每一代 RISC 处理器都是在同一人带领下完成，那就是加州大学伯克利分校的 David Patterson 教授。与大多数 ISA 相反，RISC-V ISA 可以免费地用于所有希望的设备中，允许任何人设计、制造和销售 RISC-V 芯片和软件。展示了此前的四代 RISC 处理器原型芯片。它虽然不是第一

个开源的的指令集(ISA)，但它很重要，因为它第一个被设计成可以根据具体场景可以选择适合的指令集的指令集架构。基于 RISC-V 指令集架构可以设计服务器 CPU，家用电器 cpu，工控 cpu 和用在比指头小的传感器中的 cpu。

RISC-V 指令集编码格式

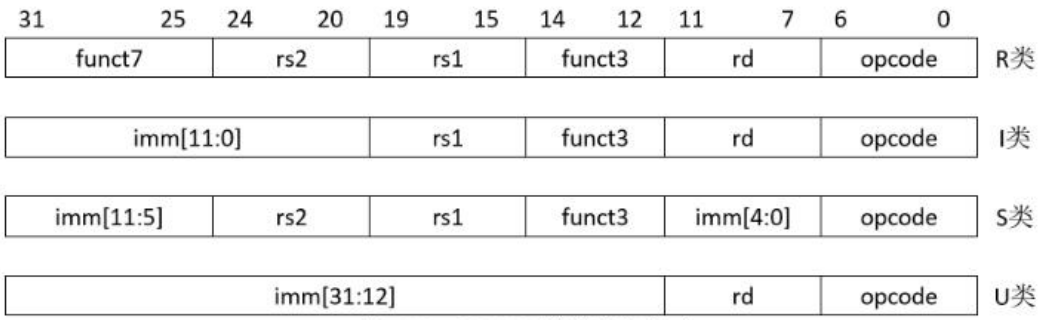


图 2.2：RISC-V 基本指令格式

RISC-V 指令

基本指令格式

四种基础指令格式 R/I/S/U

imm：立即数

rs1：源寄存器 1

rs2：源寄存器 2

rd：目标寄存器

opcode：操作码

整数计算

ADDI:将 12 位有符号立即数和 rs 相加，溢出忽略，直接使用结果的最低 32bit，并存入 rd

SLTI：如果 rs 小于立即数(都是有符号整数),将 rd 置 1,否则置 0

SLTIU：和 SLTI 一致，不过都是无符号数

ANDI/ORI/XORI：rs 与有符号 12 位立即数进行 and,or,xor 操作

ADD/SUB:rs1(+/-)rs2 => rd

SLT/SLTU: 如果 rs1<rs2，rd 写 1; 否则 rd 为 0

AND/OR/XOR: rs1 与 rs2 进行 and,or,xor 操作

SLL/SRL/SRA: 和"寄存器-立即数"指令一致，将 r2 的低 5 位作为立即数即可

NOP 指令:

实际上是 ADDI x0,x0,0

JAL：J 类指令，立即数+pc 为跳转目标，rd 存放 pc+4（返回地址）

跳转范围为 pc(+/-)1MB

JALR：I 类指令，rs+立即数为跳转目标，rd 存放 pc+4（返回地址）

实现远跳转

BEQ/BNE：rs1(==/!=)rs2, 分别在相等或者不等时，发生跳转

BLT：rs1 < rs2, 跳转

BGE：rs1 >= rs2, 跳转

LOAD：rs 作为基地址，加上有符号的偏移，读取到 rd 寄存器

STORE：rs1 作为基地址加上有符号的偏移，作为内存地址，写入内容为 rs2

CSRRW：Atomic Read/Write CSR

读取 CSR 的值存入 rd 寄存器，并将 rs 存入 CSR

CSRRS：Atomic Read and Set Bits in CSR

读取 CSR 的值存入 rd 寄存器，并根据 rs 中高位对 CSR 置 1

CSRRC：Atomic Read and Clear Bits in CSR

读取 CSR 的值存入 rd 寄存器，并根据 rs 中高位对 CSR 置 0

CSRRWI/CSRRSI/CSRRCI

将 CSRRW 类寄存器中的 rs 换成立即数

RDCYCLE：时钟周期计数

RDTIME：时间 tick 数

RDINSTRET：指令数

ECALL

EBREAK

模拟器设计代码

考虑到 CPU 执行指令的流程为：

1. 取指
2. 译码
3. 执行（包括运算和结果写回）

对模拟器程序的框架设计如下：

```
while(1) {
	inst = fetch(cpu.pc);
	cpu.pc = cpu.pc + 4;

	inst.decode();

	switch(inst.opcode) {
		case LUI:
			cout << "Do LUI" << endl;
			R[rd] = Imm31_12UtypeZeroFilled;
			break;
		case AUIPC:
			cout << "Do AUIPC" << endl;
			cout << "PC = " << PC << endl;
			cout << "Imm31_12UtypeZeroFilled = " << Imm31_12UtypeZeroFilled << endl;
			R[rd] = PC + Imm31_12UtypeZeroFilled;
			break;
		case BRANCH:
			switch(funct3) {
				case BNE:
					cout << "Do BNE " << endl;
					if(src1!=src2){
						NextPC = PC + Imm12_1BtypeSignExtended;
					}
					break;
				default:
					cout << "ERROR: Unknown funct3 in BRANCH instruction " << IR << endl;
			}
```

```
                break;
            case LOAD:
                switch(funct3) {
                    case LH:
                        cout << "Do LH " << endl;
                        unsigned int temp_LH,temp_LH_UP;
                        temp_LH=readHalfWord(src1+Imm11_0ItypeSignExtended);
                        temp_LH_UP=temp_LH>>15;
                        if(temp_LH_UP==1){
                            temp_LH=0xffff0000 | temp_LH;
                        }else{
                            temp_LH=0x0000ffff & temp_LH;
                        }
                        R[rd]=temp_LH;
                        break;
                    default:
                        cout << "ERROR: Unknown funct3 in LOAD instruction " << IR <<
endl;
                }
                break;
            case ALUIMM:
                switch(funct3) {
                    case ADDI:
                        cout <<       "Do ADDI" << endl;
                        R[rd]=src1+Imm11_0ItypeSignExtended;
                        break;
                    case SLLI:
                        cout << "Do SLLI " << endl;
                        R[rd]=src1<<shamt;
                        break;
                    default:
                        cout << "ERROR: Unknown funct3 in ALUIMM instruction " << IR <<
endl;
                }
                break;
            case ALURRR:
                switch(funct3) {
                    case SLT:
                        cout << "Do SLT " << endl;
                        if((int)src1<(int)src2){
                            R[rd]=1;
                        }else{
                            R[rd]=0;
                        }
```

```
                    break;
                default:
                    cout << "ERROR: Unknown funct3 in ALURRR instruction " << IR << endl;
                }
                break;
            default:
                cout << "无法识别的操作码：” << inst.opcode;
        }
}
```

测试

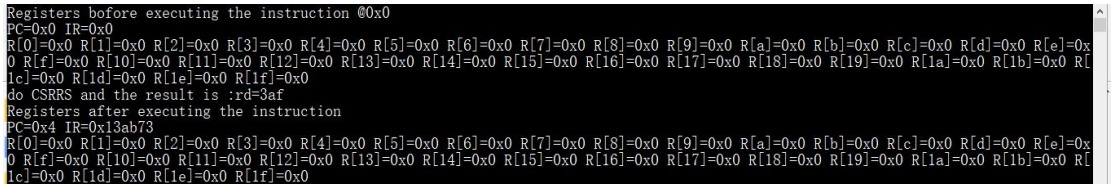
测试平台

模块	配置	备注
CPU	Core i5-6700U	
操作系统	Windows10	

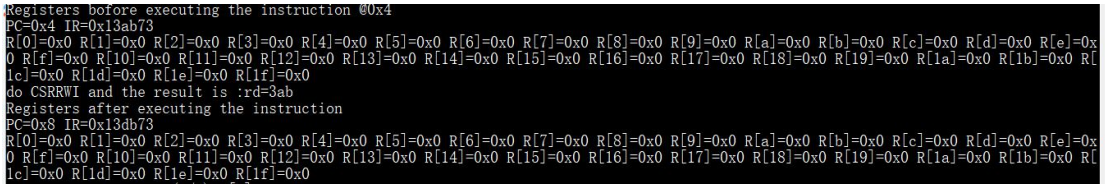
测试结果

运行程序后的输出的前 6 条指令如下：

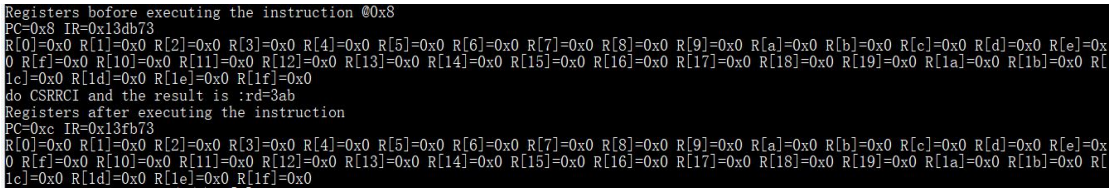
第一条指令：



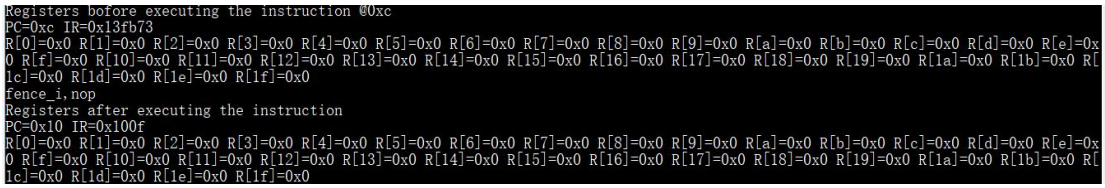
第二条指令：



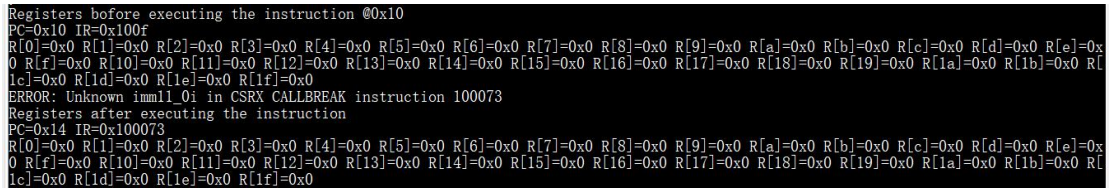
第三条指令：



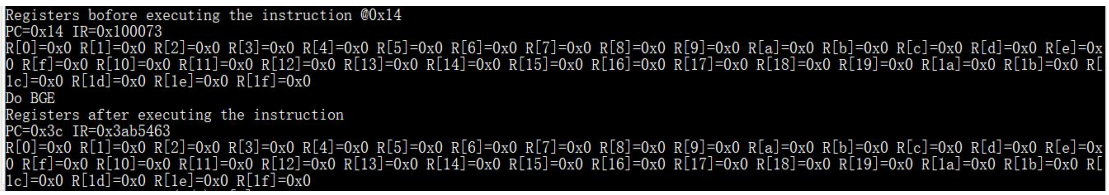
第四条指令：



第五条指令：



第六条指令：



从测试记录来看，完成一个模拟 RISC-V 的基本整数指令集 RV32I 的模拟器设计，完成了实验目标。