实验报告

实验名称：RISC-V 基本指令集模拟器设计与实现

班级：智能 1602

学号：201608010609

姓名：李鹏飞

实验目标

设计一个 CPU 模拟器，能模拟 CPU 指令集的功能

实验要求

·采用 C/C++编写程序
·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

1.RISC-V 指令集简介

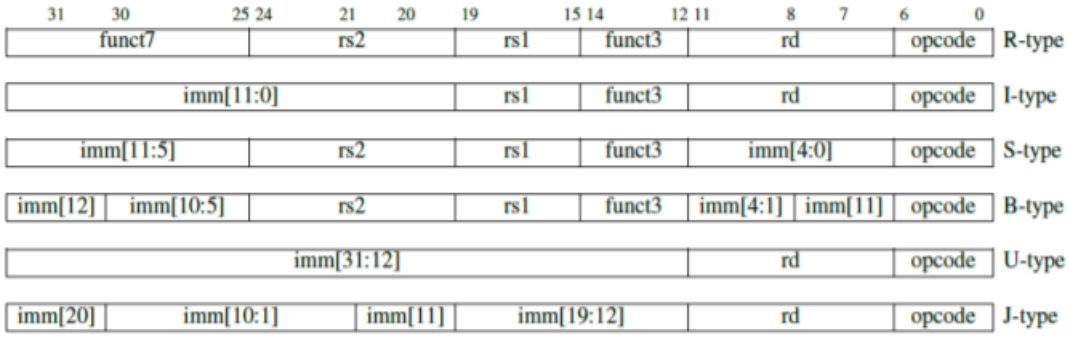
RISC-V（英文发音为"risk-five"）是一个全新的指令集架构，该架构最初由美国加州大学伯克利分校的 EECS 部门的计算机科学部门的 Krste Asanovic 教授、Andrew Waterman 和 Yunsup Lee 等开发人员于 2010 年发明。 其中"RISC"表示精简指令集，而其中"V"表示伯克利分校从 RISC I 开始设计的第五代指令集。

2010 年，加州大学伯克利分校的研究团队分析了 ARM、MIPS、SPARC、X86 等多种指令集，发现这些指令集不仅复杂度不断提升，且还存在知识产权风险，而处理器架构种类和处理能力并无直接关联。针对以上问题，该小组设计并推出了一套基于 BSD 协议许可的免费开放的指令集架构 RISC-V，其原型芯片也于 2013 年 1 月成功流片。RISC-V 指令集具有性能优越，彻底免费开放两大特征。RSIC-V 的设计目标是能够满足从微控制器到超级计算机等各种复杂程度的处理器需求，支持从 FPGA、ASIC 乃至未来器件等多种实现方式，同时能够高效地实现各种微结构，支持大量定制与加速功能，并与现有软件及编程语言可良好适配。RISC-V 产业生态正进入快速发展期。加州大学伯克利分校在 2015 年成立非盈利组织 RISC-V 基金会，该基金会旨在聚合全球创新力量共同构建开放、合作的软硬件社区，打造 RISC-V 生态系统。三年多来，谷歌、高通、IBM、英伟达、NXP、、西部数据、Microsemi、中科院计算所、麻省理工学院、华盛顿大学、英国宇航系统公司等 100 多个企业和研究机构先后加入了 RISC-V 基金会。

2.RISC-V 指令集内容

我们在这里编写的是 RV32I 指令集，其包含了六种基本指令格式，分别是：用于寄存器-寄存器操作的 R 类型指令，用于短立即数和访存 load 操作的 I 型指令，用于访存 store 操作的 S 型指令，用于条件跳转操作的 B 类型指令，用于长立即数的 U 型指令和用于无条件跳转的 J 型指令。

3.RISC-V 指令集编码格式



4.RISC-V 指令

Category Name	Fmt	RV32I Base	
Shifts			
Shift Left Logical	R	SLL	rd,rs1,rs2
Shift Left Log.Imm.	I	SLLI	rd,rs1,shamt
Shift Right Logical	R	SRL	rd,rs1,rs2
Shift Right Log.Imm.	I	SRLI	rd,rs1,shamt
Shift Right Arithmetic	R	SRA	rd,rs1,rs2
Shift Right Arith.Imm.	I	SRAI	rd,rs1,shamt
Arithmetic			
ADD	R	ADD	rd,rs1,rs2
ADD Immediate	I	ADDI	rd,rs1,imm
SUBtract	R	SUB	rd,rs1,rs2
Load Upper Imm	U	LUI	rd,imm
Add Upper Imm to PC	U	AUIPC	rd,imm
Logical			
XOR	R	XOR	rd,rs1,rs2
XOR Immediate	I	XORI	rd,rs1,imm
OR	R	OR	rd,rs1,rs2
OR Immediate	I	ORI	rd,rs1,imm
AND	R	AND	rd,rs1,rs2
AND Immediate	I	ANDI	rd,rs1,imm

Category Name	Fmt	RV32I Base	
Compare			
Set<	R	SLT	rd,rs1,rs2
Set<Immediate	I	SLTI	rd,rs1,rs2
Set<Unsigned	R	SLTU	rd,rs1,rs2
Set<Imm Unsigned	I	SLTIU	rd,rs1,imm
Branches			
Branch=	B	BEQ	rs1,rs2,imm
Branch≠	B	BNE	rs1,rs2,imm
Branch<	B	BLT	rs1,rs2,imm
Branch≥	B	BGE	rs1,rs2,imm
Branch<Unsigned	B	BLTU	rs1,rs2,imm
Branch≥Unsigned	B	BGEU	rs1,rs2,imm
Jump&Link			
J&L	J	JAL	rd,imm
Jump&Link Register	I	JALR	rd,rs1,imm
Synch			
Synch thread	I	FENCE	
Synch Instr&Data	I	FENCE.I	
Environment			
CALL	I	ECALL	
BREAK	I	EBREAK	
Control Status Register(CSR)			
Read/Write	I	CSRRW	rd,csr,rs1
Read&Set Bit	I	CSRRS	rd,csr,rs1
Read&Clear Bit	I	CSRRC	rd,csr,rs1
Read/Write Imm	I	CSRRWI	rd,csr,imm
Read&Set Bit Imm	I	CSRRSI	rd,csr,imm
Read&Clear Bit Imm	I	CSRRCI	rd,csr,imm
Loads			
Load Byte	I	LB	rd,rs1,imm
Load Halfword	I	LH	rd,rs1,imm
Load Byte Unsigned	I	LBU	rd,rs1,imm
Load Half Unsigned	I	LHU	rd,rs1,imm
Load Word	I	LW	rd,rs1,imm
Stores			
Store Byte	S	SB	rs1,rs2,imm
Store Halfword	S	SH	rs1,rs2,imm
Store Word	S	SW	rs1,rs2,imm

模拟器程序框架

考虑到 CPU 执行指令的流程为：

1.取指

2.译码

3.执行

我们将模拟器的框架设计如下：

```
/* code */
allocMem(4096); //开辟内存空间
progMem();  //编辑内存
PC = 0;
char c = 'Y';
while(c != 'n') {
    cout << "Registers bofore executing the instruction @0x" << std::hex << PC << endl;
    showRegs();  //显示寄存器
    IR = readWord(PC);      //读取程序计数器中的内容，交给指令寄存器
    NextPC = PC + WORDSIZE;    //下一个地址是 PC+WORDSIZE 决定的
    decode(IR);  //对指令进行译码
    switch(opcode) {    //判断
        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 JAL:
            cout << "Do JAL" << endl;
            R[rd]=PC+4;
            NextPC = PC+ Imm20_1JtypeSignExtended;
            break;
        case JALR:
            cout << "DO JALR" << endl;
            R[rd]=PC+4;
            NextPC=R[rs1]+Imm20_1JtypeSignExtended;
            break;
        case BRANCH:  //分支指令
            switch(funct3) {
                case BEQ:
                    cout << "DO BEQ" << endl;
```

```
                if(src1==src2){
                        NextPC = PC + Imm12_1BtypeSignExtended;
                }
                break;
        case BNE:
                cout << "Do BNE " << endl;
                if(src1!=src2){
                        NextPC = PC + Imm12_1BtypeSignExtended;
                }
                break;
        case BLT:
                cout << "Do BLT" << endl;
                if((int)src1<(int)src2){
                        NextPC = PC + Imm12_1BtypeSignExtended;
                }
                break;
        case BGE:
                cout << "Do BGE" << endl;
                if((int)src1 >= (int)src2)
                        NextPC = PC + Imm12_1BtypeSignExtended;
                break;
        case BLTU:
                cout << "Do BLTU" << endl;
                if(src1<src2){
                        NextPC=PC+Imm12_1BtypeSignExtended;
                }
                break;
        case BGEU:
                cout<<"Do BGEU"<<endl;

                if(src1>=src2){
                        NextPC=PC+Imm12_1BtypeSignExtended;
                }
                break;
        default:
                cout << "ERROR: Unknown funct3 in BRANCH instruction " << IR << endl;
    }
    break;
case LOAD:   //load 指令
    switch(funct3) {
        case LB:
                cout << "DO LB" << endl;
                unsigned int LB_LH,LB_LH_UP;
                cout << "LB Address is: " << src1+Imm11_0ItypeSignExtended << endl;
```

```
        LB_LH=readByte(src1+Imm11_0ItypeSignExtended);
        LB_LH_UP=LB_LH>>7;
        if(LB_LH_UP==1){
            //LB_LH=0xffffff00 & LB_LH;
            LB_LH=0xffffff00 | LB_LH;
        }else{
            LB_LH=0x000000ff & LB_LH;
        }
        R[rd]=LB_LH;
        break;
    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;
    case LW:
        cout << "Do LW" << endl;
        unsigned int temp_LW,temp_LW_UP;
        temp_LW=readByte(src1+Imm11_0ItypeSignExtended);
        temp_LW_UP=temp_LW>>31;
        if(temp_LW_UP==1){
            temp_LW=0x00000000 | temp_LW;
        }else{
            temp_LW=0xffffffff & temp_LW;
        }
        R[rd]=temp_LW;
        break;
    case LBU:
        cout << "Do LBU" << endl;
        R[rd] = readByte(Imm11_0ItypeSignExtended + src1) & 0x000000ff;
        break;
    case LHU:
        cout << "Do LHU" << endl;
        R[rd] = readByte(Imm11_0ItypeSignExtended + src1) & 0x0000ffff;
        break;
    default:
        cout << "ERROR: Unknown funct3 in LOAD instruction " << IR << endl;
```

```
        }
        break;
case STORE:    //存储指令
        switch(funct3) {
                case SB:
                        cout << "Do SB" << endl;
                        char sb_d1;
                        unsigned int sb_a1;
                        sb_d1=R[rs2] & 0xff;
                        sb_a1 = R[rs1] +Imm11_0StypeSignExtended;
                        writeByte(sb_a1, sb_d1);
                        break;
                case SH:
                        cout<<"Do SH"<<endl;
                        uint16_t j;
                        j=R[rs2]&0xffff;
                        unsigned int x;
                        x = R[rs1] + Imm11_0StypeSignExtended;
                        writeHalfWord(x,j);
                        break;
                case SW:
                        cout << "DO SW" << endl;
                        //unsigned int imm_temp;
                        uint32_t _swData;
                        _swData=R[rs2] & 0xffffffff;
                        unsigned int _swR;
                        _swR = R[rs1] + Imm11_0StypeSignExtended;
                        cout << "SW Addr and Data are: " << _swR << ", " << _swData << endl;
                        writeWord(_swR, _swData);
                        break;
                default:
                        cout << "ERROR: Unknown funct3 in STORE instruction " << IR << endl;
        }
        break;
case ALUIMM:    //ALU 立即数指令
        switch(funct3) {
                case ADDI:
                        cout <<       "Do ADDI" << endl;
                        R[rd]=src1+Imm11_0ItypeSignExtended;
                        break;
                case SLTI:
                        cout << "Do SLTI" << endl;
                        if(src1<Imm11_0ItypeSignExtended)
                                R[rd] = 1;
```

```
                else
                        R[rd] = 0;
                break;
        case SLTIU:
                cout << "Do SLTIU" << endl;
                if(src1<(unsigned int)Imm11_0ItypeSignExtended)
                        R[rd] = 1;
                else
                        R[rd] = 0;
                break;
        case XORI:
                cout << "Do XORI" << endl;
                R[rd]=(Imm11_0ItypeSignExtended)^R[rs1];
                break;
        case ORI:
                cout<<"Do ORI"<<endl;
                R[rd]=R[rs1]|Imm11_0ItypeSignExtended;
                break;
        case ANDI:
                cout << "DO ANDI"<<endl;
                R[rd]=R[rs1]&Imm11_0ItypeSignExtended;
                break;
        case SLLI:
                cout << "Do SLLI " << endl;
                R[rd]=src1<<shamt;
                break;
        case SHR:
                switch(funct7) {
                        case SRLI:
                                cout << "Do SRLI" << endl;
                                R[rd]=src1>>shamt;
                                break;
                        case SRAI:
                                cout << "Do SRAI" << endl;
                                R[rd] = ((int)src1) >> shamt;
                                break;
                        default:
                                cout << "ERROR: Unknown (imm11_0i >> 5) in ALUIMM SHR instruction " <<
IR << endl;
                }
                break;
        default:
                cout << "ERROR: Unknown funct3 in ALUIMM instruction " << IR << endl;
        }
```

```
                break;
            case ALURRR:    //ALU 寄存器指令
                switch(funct3) {
                    case ADDSUB:
                        switch(funct7) {
                            case ADD:
                                cout << "Do ADD" << endl;
                                R[rd]=R[rs1]+R[rs2];
                                break;
                            case SUB:
                                cout<<" Do SUB"<<endl;
                                R[rd]=R[rs1]-R[rs2];
                                break;
                            default:
                                cout << "ERROR: Unknown funct7 in ALURRR ADDSUB instruction " << IR <<
endl;
                        }
                        break;
                    case SLL:
                        cout<<"DO SLL"<<endl;
                        unsigned int rsTransform;
                        rsTransform=R[rs2]&0x1f;
                        R[rd]=R[rs1]<<rsTransform;
                        break;
                    case SLT:
                        cout << "Do SLT " << endl;
                        if((int)src1<(int)src2){
                            R[rd]=1;
                        }else{
                            R[rd]=0;
                        }
                        break;
                    case SLTU:
                        cout << "Do SLTU" << endl;
                        if(src2!=0){
                            R[rd]=1;
                        }else{
                            R[rd]=0;
                        }
                        break;
                    case XOR:
                        cout << "Do XOR " << endl;
                        R[rd]=R[rs1]^R[rs2];
                        break;
```

```
                case OR:
                    cout << "Do OR" << endl;
                        R[rd]=R[rs1]|R[rs2];
                    break;
                case AND:
                    cout << "Do AND" << endl;
                        R[rd]=R[rs1]&R[rs2];
                    break;

                case SRLA:
                    switch(funct7) {
                        case SRL:
                        cout<<"DO SRL"<<endl;
                            R[rd]=R[rs1]>>R[rs2];
                            break;
                        case SRA:
                            cout<<"DO SRA"<<endl;
                            R[rd]=(int)src1>>src2;
                            break;
                        default:
                            cout << "ERROR: Unknown funct7 in ALURRR SRLA instruction " << IR <<
endl;
                    }
                    break;
                default:
                    cout << "ERROR: Unknown funct3 in ALURRR instruction " << IR << endl;
            }
            break;
        case FENCES:
            switch(funct3) {
                case FENCE:
                    //TODO: Fill code for the instruction here
                    break;
                case FENCE_I:
                    //TODO: Fill code for the instruction here
                    cout<<"fence_i,nop"<<endl;
                    break;
                default:
                    cout << "ERROR: Unknown funct3 in FENCES instruction " << IR << endl;
            }
            break;
        case CSRX:
            switch(funct3) {
                case CALLBREAK:
```

```
                switch(Imm11_0ItypeZeroExtended) {
                        case ECALL:
                                //TODO: Fill code for the instruction here
                                break;
                        /*      case EBREAK:
                                {//TODO: Fill code for the instruction here
                                        PC = ebreakadd;
                                        cout << "do ebreak and pc jumps to :" << ebreakadd << endl;
                                        break;
                                }*/     //无法程序调用
                        default:
                                cout << "ERROR: Unknown imm11_0i in CSRX CALLBREAK instruction " << IR
<< endl;
                }
                break;
        case CSRRW:
                //TODO: Fill code for the instruction here
                break;
        case CSRRS:
                //TODO: Fill code for the instruction here
                {
                    uint32_t temp = readWord(rs2)&0x00000fff;
                      uint32_t temp1 = rs1 & 0x000fffff;
                      writeWord(rd,(temp|temp1));
                      cout << "do CSRRS and the result is :" << "rd="<<readWord(rd)<<endl;
                      break;
                }
        case CSRRC:
                //TODO: Fill code for the instruction here
                break;
        case CSRRWI:
                //TODO: Fill code for the instruction here
                {
                    if (rd == 0) break;
                      else
                      {
                              uint32_t zmm = imm11j& 0x000001f;
                              uint32_t tem = readWord(rs2) & 0x00000fff;
                              writeWord(rd, tem);
                              writeWord(rs2, zmm);
                              cout << "do CSRRWI and the result is :" << "rd=" << readWord(rd) << endl;
                              break;
                      }
                }
```

```
                        case CSRRSI:
                            //TODO: Fill code for the instruction here
                            break;
                        case CSRRCI:
                            //TODO: Fill code for the instruction here
                            {   uint32_t zmm = imm11j & 0x000001f;
                                uint32_t tem = readWord(rs2) & 0x00000fff;
                                if (readWord(rd) != 0)
                                {
                                    writeWord(rs2, zmm | tem);
                                }
                                cout << "do CSRRCI and the result is :" << "rd=" << readWord(rd) << endl;
                                break;
                            }
                        default:
                            cout << "ERROR: Unknown funct3 in CSRX instruction " << IR << endl;
                    }
                    break;
                default:
                    cout << "ERROR: Unkown instruction " << IR << endl;
                    break;
            }

            // Update PC
            PC = NextPC;

            cout << "Registers after executing the instruction" << endl;
            showRegs();

            cout << "Continue simulation (Y/n)? [Y]" << endl;
            cin.get(c);
            getchar();    //清除回车
        }
        freeMem();
```

其中 while 的循环条件我们可以自行更改

测试

测试平台

模拟器在如下机器上进行了测试

部件	配置	备注
CPU	Core i5-6700U	
内存	DDR4 12GB	
操作系统	Windows10 家庭版	

测试记录

模拟器输入如下：

```
void progMem() {
        // Write starts with PC at 0
        writeWord(0, (0xfffff << 12) | (2 << 7) | (LUI));   //imm,rd,opcode;这里是加载顶端立即数操作，内容是 0xfffff137
        writeWord(4, (1 << 12) | (5 << 7) | (AUIPC));   //imm,rd,opcode;顶端立即数加至 PC，0x00001297
        writeWord(8, (0x20<<25) | (5<<20) | (0<<15) | (SW << 12) | (0 << 7) | (STORE));//imm,rs2,rs1,funct3,imm,opcode;
        writeWord(12, (0x400<<20) | (0<<15) | (LB<<12) | (3<<7) | (LOAD));
        writeWord(16, (0x400<<20) | (0<<15) | (LBU<<12) | (7<<7) | (LOAD));
        writeWord(20, (0x0<<25) | (2<<20) | (0<<15) | (BGE<<12) | (0x8<<7) | (BRANCH));
        writeWord(28, (0x8<<20) | (3<<15) | (SLTIU<<12) | (8<<7) | (ALUIMM));
        writeWord(32, (SRAI<<25) | (0x2<<20) | (0x2<<15) | (SHR<<12) | (9<<7) | (ALUIMM));
        writeWord(36,(0x400)<<20|(1<<15)|(JALR<<12)|(4<<7)|(JALR));
        writeWord(40, (0x20<<25) | (7<<20) | (0<<15) | (SH << 12) | (9 << 7) | (STORE));
        writeWord(44, (0x0<<25) | (4<<20) | (1<<15) | (BGEU<<12) | (0x8<<7) | (BRANCH));
        writeWord(48, (0x400<<20) | (2<<15) | (ORI<<12) | (4<<7) | (ALUIMM));
        writeWord(52, (SUB<<25) | (4<<20) | (2<<15) | (SUB << 12) | (9 << 7) | (ALURRR));
        writeWord(56, (1<<31) |(0<<25) | (8<<20) | (0<<15) | (BLTU << 12) | (0 << 11) |(0 << 7) | (BRANCH));
        writeWord(60, (0x20<<25) | (8<<20) | (0<<15) | (SB << 12) | (0 << 7) | (STORE));
        writeWord(64, (0x100<<20) | (3<<15) | (XORI << 12) | (9 << 7) | (ALUIMM));
        writeWord(68, (ADD<<25) | (3<<20) | (1<<15) | (ADDSUB << 12) | (10 << 7) | (ALURRR));
        writeWord(72, (1 << 31) |(1 << 23) |(1 << 22) |(1 << 12) | (7 << 7) | (JAL));
        writeWord(0, 0x0013ab73);// CSRRS
        writeWord(4, 0x0013db73);//CSRRWI
        writeWord(8, 0x0013fb73);//CSRRCI
        writeWord(12, 0x0000100f);//FENCE_I
        writeWord(16, 0x00100073);//EBREAK
}
```

模拟器运行的截图如下：

第一条指令模拟器运行后的输出：

```
Registers bofore executing the instruction @0x0
PC=0x0 IR=0x0
R[0]=0x0 R[1]=0x0 R[2]=0x0 R[3]=0x0 R[4]=0x0 R[5]=0x0 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0x0 R[b]=
0x0 R[c]=0x0 R[d]=0x0 R[e]=0x0 R[f]=0x0 R[10]=0x0 R[11]=0x0 R[12]=0x0 R[13]=0x0 R[14]=0x0 R[15]=0x0 R[16
]=0x0 R[17]=0x0 R[18]=0x0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
do CSRRS and the result is :rd=3af
Registers after executing the instruction
PC=0x4 IR=0x13ab73
R[0]=0x0 R[1]=0x0 R[2]=0x0 R[3]=0x0 R[4]=0x0 R[5]=0x0 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0x0 R[b]=
0x0 R[c]=0x0 R[d]=0x0 R[e]=0x0 R[f]=0x0 R[10]=0x0 R[11]=0x0 R[12]=0x0 R[13]=0x0 R[14]=0x0 R[15]=0x0 R[16
]=0x0 R[17]=0x0 R[18]=0x0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
```

第二条指令模拟器运行后的输出：

```
Registers bofore executing the instruction @0x4
PC=0x4 IR=0x13ab73
R[0]=0x0 R[1]=0x0 R[2]=0x0 R[3]=0x0 R[4]=0x0 R[5]=0x0 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0x0 R[b]=
0x0 R[c]=0x0 R[d]=0x0 R[e]=0x0 R[f]=0x0 R[10]=0x0 R[11]=0x0 R[12]=0x0 R[13]=0x0 R[14]=0x0 R[15]=0x0 R[16
]=0x0 R[17]=0x0 R[18]=0x0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
do CSRRWI and the result is :rd=3ab
Registers after executing the instruction
PC=0x8 IR=0x13db73
R[0]=0x0 R[1]=0x0 R[2]=0x0 R[3]=0x0 R[4]=0x0 R[5]=0x0 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0x0 R[b]=
0x0 R[c]=0x0 R[d]=0x0 R[e]=0x0 R[f]=0x0 R[10]=0x0 R[11]=0x0 R[12]=0x0 R[13]=0x0 R[14]=0x0 R[15]=0x0 R[16
]=0x0 R[17]=0x0 R[18]=0x0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
Continue simulation (Y/n)? [Y]
```

第三条指令模拟器运行后的输出：

```
Registers bofore executing the instruction @0x8
PC=0x8 IR=0x13db73
R[0]=0x0 R[1]=0x0 R[2]=0x0 R[3]=0x0 R[4]=0x0 R[5]=0x0 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0x0 R[b]=
0x0 R[c]=0x0 R[d]=0x0 R[e]=0x0 R[f]=0x0 R[10]=0x0 R[11]=0x0 R[12]=0x0 R[13]=0x0 R[14]=0x0 R[15]=0x0 R[16
]=0x0 R[17]=0x0 R[18]=0x0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
do CSRRCI and the result is :rd=3ab
Registers after executing the instruction
PC=0xc IR=0x13fb73
R[0]=0x0 R[1]=0x0 R[2]=0x0 R[3]=0x0 R[4]=0x0 R[5]=0x0 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0x0 R[b]=
0x0 R[c]=0x0 R[d]=0x0 R[e]=0x0 R[f]=0x0 R[10]=0x0 R[11]=0x0 R[12]=0x0 R[13]=0x0 R[14]=0x0 R[15]=0x0 R[16
]=0x0 R[17]=0x0 R[18]=0x0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
Continue simulation (Y/n)? [Y]
```

第四条指令模拟器运行后的输出：

```
Registers bofore executing the instruction @0xc
PC=0xc IR=0x13fb73
R[0]=0x0 R[1]=0x0 R[2]=0x0 R[3]=0x0 R[4]=0x0 R[5]=0x0 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0x0 R[b]=
0x0 R[c]=0x0 R[d]=0x0 R[e]=0x0 R[f]=0x0 R[10]=0x0 R[11]=0x0 R[12]=0x0 R[13]=0x0 R[14]=0x0 R[15]=0x0 R[16
]=0x0 R[17]=0x0 R[18]=0x0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
fence_i,nop
Registers after executing the instruction
PC=0x10 IR=0x100f
R[0]=0x0 R[1]=0x0 R[2]=0x0 R[3]=0x0 R[4]=0x0 R[5]=0x0 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0x0 R[b]=
0x0 R[c]=0x0 R[d]=0x0 R[e]=0x0 R[f]=0x0 R[10]=0x0 R[11]=0x0 R[12]=0x0 R[13]=0x0 R[14]=0x0 R[15]=0x0 R[16
]=0x0 R[17]=0x0 R[18]=0x0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
Continue simulation (Y/n)? [Y]
```

分析和结论

从测试记录来看，模拟器实现了对二进制指令文件的读入、指令功能的模拟，CPU 和存储器状态的输出。

根据分析结果，可以认为编写的模拟器实现了所要求的功能，完成了实验目标。