湖南大學

HUNAN UNIVERSITY

CPU 模拟器实验报告

学生姓名　潘小天

学生学号　201608010309

专业班级　智能 1601

指导老师　吴强

完成日期　2019 年 9 月 5 日

一、 实验内容

完成一个模拟 RISC-V 的基本整数指令集 RV32I 的模拟器设计。

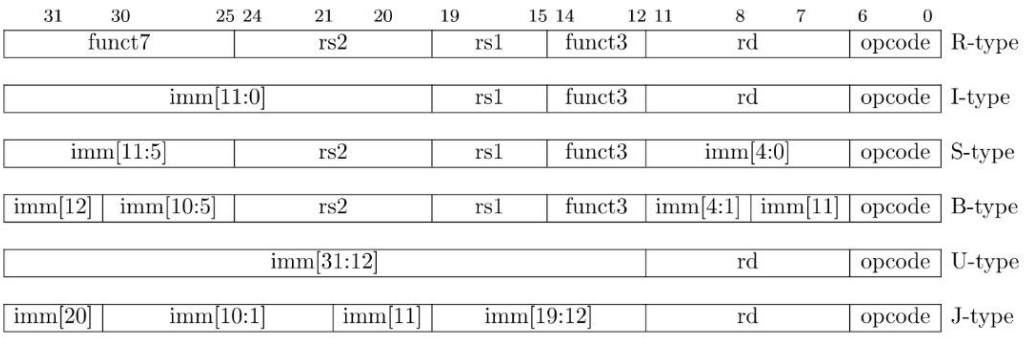
二、 实验要求

硬件设计采用 VHDL 或 Verilog 语言，软件设计采用 C/C++或 SystemC 语言，其它语言例如 Chisel、MyHDL 等也可选。
实验报告采用 markdown 语言，或者直接上传 PDF 文档
实验最终提交所有代码和文档

三、 实验过程及结果

RISC-V 指令集

RISC-指令集格式如下：



指令集的内容和具体实现参考教材。

具体实现

模拟器所模拟 CPU 的主要功能分别为取指、译码、执行
具体指令的实现如下：
指令分类：
U 类：LUI、ALUPC
J 类：JAL
B 类：BEQ、BNE、BLT、BGE、BLTU、BGEU
I 类：JALR、LB、LH、LW、LBU、LHU、ADDI、SLTI、SLTIU、XORI、ORI、ANDI、SLLI、SRLI、SRAI

S 类：SB、SH、SW
R 类：ADD、SUB、SLL、SLT、SLTU、XOR、SRL、SRA、OR、AND

主要架构：

```
                while(c != 'n') {
            cout << "Registers bofore executing the instruction @0x" << std::hex <<
PC << endl;
            showRegs();
//每次循环显示一下寄存器
            IR = readWord(PC);
//读取 pc 对应的指令，一个指令是一个 Word，即 4byte
            NextPC = PC + WORDSIZE;
//赋值下一个 PC
            decode(IR);
//解析指令
            switch(opcode)
{
      //这个是在 decode 时的低 7 位的值，是操作码
                case LUI:
//  执行的操作 load upper imm，其实应该是加载指令吧
                    cout << "Do LUI" << endl;
                    R[rd] = Imm31_12UtypeZeroFilled;
//这里 rd 是 decode 取出来的值，是 IR 中高 20 位的值，这里使用的 Utype 指令，取的是
                    break;
                case AUIPC:
//0x17 用于建立 PC 相对地址，使用 U 型格式，用 0 填充最低的 12 位， 将该偏移量添加到 AUIPC 指令的地址，然后将结果放入寄存器
                    cout << "Do AUIPC" << endl;
                    cout << "PC = " << PC << endl;
                    cout << "Imm31_12UtypeZeroFilled = " <<
Imm31_12UtypeZeroFilled << endl;
                    R[rd] = PC + Imm31_12UtypeZeroFilled;
                    break;
                case JAL:
//0x6F,无条件跳转
                    cout << "Do JAL" << endl;
                    R[rd]=PC+4;
                    NextPC = PC+ Imm20_1JtypeSignExtended;
                    break;
                case JALR:
//0x67,无条件跳转，直接跳转指令，无条件跳转到由寄存器 rs1 指定的指令，并将下一条指令的地址保存到寄存器 rd 中
                    cout << "DO JALR" << endl;
```

```
                    R[rd]=PC+4;
                    NextPC=R[rs1]+Imm20_1JtypeSignExtended;
                    break;
                case BRANCH://0x63 分支指令 所有的 BRANCH 指令都用的是
B 类型格式，这条指令立即数就是代表偏移量
                    switch(funct3) {
                        case BEQ://0x0 当 src1 和 src2 寄存器相等的时候执行
                            cout << "DO BEQ" << endl;
                            if(src1==src2){
                                NextPC = PC + Imm12_1BtypeSignExtended;
                            }
                            break;
                        case BNE://0x1 当 src1 和 src2 寄存器不相等的时候执行
                            cout << "Do BNE " << endl;
                            if(src1!=src2){
                                NextPC = PC + Imm12_1BtypeSignExtended;
                            }
                            break;
                        case BLT://0x4 有符号比较当 src1<src2 时执行
                            cout << "Do BLT" << endl;
                            if((int)src1<(int)src2){
                                NextPC = PC + Imm12_1BtypeSignExtended;
                            }
                            break;
                        case BGE://0x5 有符号比较当 src1>=src2 时执行
                            cout << "Do BGE" << endl;
                            cout<<"src1 为 "<<src1<<endl;
                            cout<<"src2 为 "<<src2<<endl;
                            cout<<"imm                                    为
"<<Imm12_1BtypeSignExtended<<endl;
                            if((int)src1 >= (int)src2)
                                NextPC = PC + Imm12_1BtypeSignExtended;
                            break;
                        case BLTU://0x6
                            cout << "Do BLTU" << endl;
                            if(src1<src2){
                                NextPC=PC+Imm12_1BtypeSignExtended;
                            }
                            break;
                        case BGEU://0x7
                            cout<<"Do BGEU"<<endl;

                            if(src1>=src2){
                                NextPC=PC+Imm12_1BtypeSignExtended;
```

```
                    }
                    break;
                default://找不到相应的指令
                    cout << "ERROR: Unknown funct3 in BRANCH instruction " << IR << endl;
                }
                break;
            case LOAD://0x03 LOAD 被编码为 I 类型
                switch(funct3) {
                    case LB://加载一个 byte
                        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){//符号位扩展
                            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);//Itype 只有一个源 src1
                        temp_LH_UP=temp_LH>>15;
                        if(temp_LH_UP==1){//执行符号位扩展
                            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);// 这里为什么要用 readByte
```

```
                    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://没有找到指令
                    cout << "ERROR: Unknown funct3 in LOAD
instruction " << IR << endl;
                }
                break;
            case STORE://STORE 指令 STORE 被编码为 S 类型
                switch(funct3) {//sr1 指明了地址，sr2 指明了保存的值
                    case SB:
                        cout << "Do SB" << endl;
                        char sb_d1;
                        unsigned int sb_a1;
                        sb_d1=R[rs2] & 0xff;//最多只能写 8 位
                        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;//最多只能写 16 位
                        unsigned int x;
                        x = R[rs1] + Imm11_0StypeSignExtended;
                        writeHalfWord(x,j);
                        break;
                    case SW:
                        cout << "DO SW" << endl;
                        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://ALUIMM 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;//这里的 shamt 是从 sr2
取出的数据
                                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://ALURRR 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;//最多左移 32 位
        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://与指令
        cout << "Do AND" << endl;
                R[rd]=R[rs1]&R[rs2];
        break;

    case SRLA://右移指令
        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://FENCES 指令
                        switch(funct3) {
                            case FENCE:
                                //TODO: Fill code for the instruction here
                                break;
                            case FENCE_I:
                                //TODO: Fill code for the instruction here
                                cout<<"this is test IR "<<IR<<endl;
                                cout<<"fence_i,nop"<<endl;
                                break;
                            default:
                                cout << "ERROR: Unknown funct3 in FENCES
instruction " << IR << endl;
                        }
                        break;
                    case CSRX://CSRX 指令   Itype
                        switch(funct3) {
                            case CALLBREAK:
                                switch(Imm11_0ItypeZeroExtended) {
                                    case ECALL:
                                        //TODO: Fill code for the instruction here
                                        break;
                                    case EBREAK:
                                        {
                                            NextPC = ebreakadd;
                                            cout << "do ebreak and pc jumps to :" <<
ebreakadd << endl;
                                            break;
                                        }
                                    default:
                                        cout << "ERROR: Unknown imm11_0i in
CSRX CALLBREAK instruction " << IR << endl;
                                }
                                break;
                            case CSRRW://The CSRRW (Atomic Read/Write CSR)
```

instruction atomically swaps values in the CSRs and integer registers

/*CSRRW 指令读取旧的 CSR 的值，把它 0 扩展后写入整数寄存器 rd，rs1 的初始值写入 CSR 中，如果 rd 为 0，则说明不能对 CSR 做任何操作*/

```
                break;
            case CSRRS:
```

/*CSRRS 读取 CSR 中的值，0 扩展，然后将其写入到整型寄存器 rd，rs1 的初始值被当做一个位掩码指定要在 CSR 中要设置的位位置，

如果 csr 位可写，rs1 中的任何高位都将导致在 csr 中设置相应的位。csr 中的其他位不受影响（尽管 csr 在写入时可能会产生副作用）。*/

```
                {
                    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:

                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:
                break;
            case CSRRCI:
                {
                    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;
                }
```

测试输入：

模拟器输入：

```
void decode(uint32_t instruction) {//decode 是译码的意思，RV32I 指令 4 个字节
    // Extract all bit fields from instruction   从指令中提取所有位字段
    opcode = instruction & 0x7F;//获取低 7 位，即 0~6 位
    rd = (instruction & 0x0F80) >> 7;//获取从低至高第 7~11 位
    rs1 = (instruction & 0xF8000) >> 15;//获取第 15~19 位，得到第一个寄存器
    zimm = rs1;//zimm 是我们定义的一个 unsigned int，把 rs1 赋值给了它
    rs2 = (instruction & 0x1F00000) >> 20;//获取第 20~24 位，得到第二个寄存器
    shamt = rs2;//shamt 是我们定义的一个 unsigned int，  把 rs2 赋值给了它
    funct3 = (instruction & 0x7000) >> 12;//获取第 12~14 位
    funct7 = instruction >> 25;//获取 25~31 位?
    imm11_0i = ((int32_t)instruction) >> 20;//转化成有符号的再移动，对应着 Itype
类型的地址
    csr = instruction >> 20;//获取 20~31 位，应该与上面的 imm11_0i 差不多，不
过是无符号类型的
    imm11_5s = ((int32_t)instruction) >> 25;//获取第 25~31 位数据，对应着 Stype
类型的地址
    imm4_0s = (instruction >> 7) & 0x01F;//获取第 7~11 位数据，对应 Stype 类型
的地址
    imm12b = ((int32_t)instruction) >> 31;//获取第 31 位数据，对应 Btype 类型的
地址
    imm10_5b = (instruction >> 25) & 0x3F;//获取第 25~30 位数据，对应 Btype 类
型的地址
    imm4_1b = (instruction & 0x0F00) >> 8;//第 8~11 位，对应 Btype 类型的地址
    imm11b = (instruction & 0x080) >> 7;//第 7 位，对应 Btype 类型的地址
```

imm31_12u = instruction >> 12;//第 12~31 位，对应 Utype 类型的地址

imm20j = ((int32_t)instruction) >> 31;//第 31 位，对应 jtype 类型的地址

imm10_1j = (instruction >> 21) & 0x3FF;//第 21~31 位，对应 jtype 类型的地址

imm11j = (instruction >> 20) & 1;//第 20 位，对应 jtype 类型的地址

imm19_12j = (instruction >> 12) & 0x0FF;//第 12 到 19 位，对应 jtype 类型的地址

pred = (instruction >> 24) & 0x0F;

succ = (instruction >> 20) & 0x0F;

运行结果：

第一条指令输出结果：

```
Registers bofore executing the instruction @0x0
PC=0x0 IR=0x0


M[0]=0x37 M[1]=0x61 M[2]=0x66 M[3]=0x0 M[4]=0x97 M[5]=0x11 M[6]=0x0 M[7]=0x0 M[8]=0xb7 M[9]=0x62 M[a]=0x6 M[
b]=0x0 M[c]=0x23 M[d]=0x2d M[e]=0x50 M[f]=0x0 M[10]=0x3 M[11]=0x42 M[12]=0x0 M[13]=0x1 M[14]=0x63 M[15]=0x54
 M[16]=0x20 M[17]=0x0 M[18]=0x0 M[19]=0x0 M[1a]=0x0 M[1b]=0x0 M[1c]=0x0 M[1d]=0x0 M[1e]=0x0 M[1f]=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 LUI
Registers after executing the instruction
PC=0x4 IR=0x666137


M[0]=0x37 M[1]=0x61 M[2]=0x66 M[3]=0x0 M[4]=0x97 M[5]=0x11 M[6]=0x0 M[7]=0x0 M[8]=0xb7 M[9]=0x62 M[a]=0x6 M[
b]=0x0 M[c]=0x23 M[d]=0x2d M[e]=0x50 M[f]=0x0 M[10]=0x3 M[11]=0x42 M[12]=0x0 M[13]=0x1 M[14]=0x63 M[15]=0x54
 M[16]=0x20 M[17]=0x0 M[18]=0x0 M[19]=0x0 M[1a]=0x0 M[1b]=0x0 M[1c]=0x0 M[1d]=0x0 M[1e]=0x0 M[1f]=0x0

R[0]=0x0 R[1]=0x0 R[2]=0x666000 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]=0
x0 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]
```

执行 LUI 指令用于立即数的零扩展，在第二个寄存器写入 0x666。

第二条指令输出结果：

```
Registers bofore executing the instruction @0x4
PC=0x4 IR=0x666137


M[0]=0x37 M[1]=0x61 M[2]=0x66 M[3]=0x0 M[4]=0x97 M[5]=0x11 M[6]=0x0 M[7]=0x0 M[8]=0xb7 M[9]=0x62 M[a]=0x6 M[
b]=0x0 M[c]=0x23 M[d]=0x2d M[e]=0x50 M[f]=0x0 M[10]=0x3 M[11]=0x42 M[12]=0x0 M[13]=0x1 M[14]=0x63 M[15]=0x54
 M[16]=0x20 M[17]=0x0 M[18]=0x0 M[19]=0x0 M[1a]=0x0 M[1b]=0x0 M[1c]=0x0 M[1d]=0x0 M[1e]=0x0 M[1f]=0x0

R[0]=0x0 R[1]=0x0 R[2]=0x666000 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]=0
x0 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 AUIPC
PC = 4
Imm31_12UtypeZeroFilled = 1000
Registers after executing the instruction
PC=0x8 IR=0x1197


M[0]=0x37 M[1]=0x61 M[2]=0x66 M[3]=0x0 M[4]=0x97 M[5]=0x11 M[6]=0x0 M[7]=0x0 M[8]=0xb7 M[9]=0x62 M[a]=0x6 M[
b]=0x0 M[c]=0x23 M[d]=0x2d M[e]=0x50 M[f]=0x0 M[10]=0x3 M[11]=0x42 M[12]=0x0 M[13]=0x1 M[14]=0x63 M[15]=0x54
 M[16]=0x20 M[17]=0x0 M[18]=0x0 M[19]=0x0 M[1a]=0x0 M[1b]=0x0 M[1c]=0x0 M[1d]=0x0 M[1e]=0x0 M[1f]=0x0

R[0]=0x0 R[1]=0x0 R[2]=0x666000 R[3]=0x1004 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
```

执行 AUIPC 指令，将 IR&0x0F80 得到的值 0x1000,再加上当前 PC 的值，得 0x1004，存入 R[3]。

第三条指令输出结果：

```
Registers bofore executing the instruction @0x8
PC=0x8 IR=0x1197


M[0]=0x37 M[1]=0x61 M[2]=0x66 M[3]=0x0 M[4]=0x97 M[5]=0x11 M[6]=0x0 M[7]=0x0 M[8]=0xb7 M[9]=0x62 M[a]=0x6 M[
b]=0x0 M[c]=0x23 M[d]=0x2d M[e]=0x50 M[f]=0x0 M[10]=0x3 M[11]=0x42 M[12]=0x0 M[13]=0x1 M[14]=0x63 M[15]=0x54
 M[16]=0x20 M[17]=0x0 M[18]=0x0 M[19]=0x0 M[1a]=0x0 M[1b]=0x0 M[1c]=0x0 M[1d]=0x0 M[1e]=0x0 M[1f]=0x0

R[0]=0x0 R[1]=0x0 R[2]=0x666000 R[3]=0x1004 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 LUI
Registers after executing the instruction
PC=0xc IR=0x662b7


M[0]=0x37 M[1]=0x61 M[2]=0x66 M[3]=0x0 M[4]=0x97 M[5]=0x11 M[6]=0x0 M[7]=0x0 M[8]=0xb7 M[9]=0x62 M[a]=0x6 M[
b]=0x0 M[c]=0x23 M[d]=0x2d M[e]=0x50 M[f]=0x0 M[10]=0x3 M[11]=0x42 M[12]=0x0 M[13]=0x1 M[14]=0x63 M[15]=0x54
 M[16]=0x20 M[17]=0x0 M[18]=0x0 M[19]=0x0 M[1a]=0x0 M[1b]=0x0 M[1c]=0x0 M[1d]=0x0 M[1e]=0x0 M[1f]=0x0

R[0]=0x0 R[1]=0x0 R[2]=0x666000 R[3]=0x1004 R[4]=0x0 R[5]=0x66000 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0
x0 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
```

在 R[5]中写入 6。

第四条指令输出结果：

```
Registers bofore executing the instruction @0xc
PC=0xc IR=0x662b7


M[0]=0x37 M[1]=0x61 M[2]=0x66 M[3]=0x0 M[4]=0x97 M[5]=0x11 M[6]=0x0 M[7]=0x0 M[8]=0xb7 M[9]=0x62 M[a]=0x6 M[
b]=0x0 M[c]=0x23 M[d]=0x2d M[e]=0x50 M[f]=0x0 M[10]=0x3 M[11]=0x42 M[12]=0x0 M[13]=0x1 M[14]=0x63 M[15]=0x54
 M[16]=0x20 M[17]=0x0 M[18]=0x0 M[19]=0x0 M[1a]=0x0 M[1b]=0x0 M[1c]=0x0 M[1d]=0x0 M[1e]=0x0 M[1f]=0x0

R[0]=0x0 R[1]=0x0 R[2]=0x666000 R[3]=0x1004 R[4]=0x0 R[5]=0x66000 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0
x0 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 SW
SW Addr and Data are: 1a, 66000
Registers after executing the instruction
PC=0x10 IR=0x502d23


M[0]=0x37 M[1]=0x61 M[2]=0x66 M[3]=0x0 M[4]=0x97 M[5]=0x11 M[6]=0x0 M[7]=0x0 M[8]=0xb7 M[9]=0x62 M[a]=0x6 M[
b]=0x0 M[c]=0x23 M[d]=0x2d M[e]=0x50 M[f]=0x0 M[10]=0x3 M[11]=0x42 M[12]=0x0 M[13]=0x1 M[14]=0x63 M[15]=0x54
 M[16]=0x20 M[17]=0x0 M[18]=0x0 M[19]=0x0 M[1a]=0x0 M[1b]=0x60 M[1c]=0x6 M[1d]=0x0 M[1e]=0x0 M[1f]=0x0

R[0]=0x0 R[1]=0x0 R[2]=0x666000 R[3]=0x1004 R[4]=0x0 R[5]=0x66000 R[6]=0x0 R[7]=0x0 R[8]=0x0 R[9]=0x0 R[a]=0
x0 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
```

执行 SW 指令写入操作，向地址 0xc+R[1]，写入 R[3]的数据，即 1004。

分析：

经过测试结果可以看出，指令的输入模拟以及相对应的模拟结果，可以看出实现了相对应的功能，得到了相应的结果。

四、 实验体会

通过本次实验，我对于 RISC-V 指令集有了一定的理解，除此之外对于 CPU 对于指令的执行过程以及指令的实现完成有了更加深刻的认识，并且也学习了如何进行对于 CPU 的模拟编写。