湖南大學

HUNAN UNIVERSITY

课程实验报告

课　程　名　称：微处理器系统原理

实验项目名称：RISC-V 基本指令集模拟器设计与实现

专　业　班　级：计算机科学与技术 1602

姓　　名：陈智鋆

学　　号：201608010718

指　导　教　师：吴强

完　成　时　间：2019 年 9 月 10 日

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

CPU 指令集有 37 条指令

模拟器程序框架

考虑到 CPU 执行指令的流程为：

1. 取指

```
void show32Mess(){
    cout << endl << endl;
    for(int i=0; i<32; i++) {                                   //打印32个寄存器的值
        char tp = M[i];                                         //cout<<(int)tp<<endl;
        cout << "M[" << i << "]=0x"  << ((unsigned int)tp&0x000000ff)<<" ";
    }
    cout << endl << endl;
}

void showRegs() {                                               //用来打印寄存器的值，总共有32个寄存器,表
                                                                //打印pc的值
    cout << "PC=0x" << std::hex << PC << " " << "IR=0x" << std::hex << IR << endl;
    show32Mess();
    for(int i=0; i<32; i++) {                                   //打印32个寄存器的值
        cout << "R[" << i << "]=0x" << std::hex << R[i] << " ";
    }
    cout << endl<<endl;
}
```

2. 译码

```
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;
```

3. 执行（包括运算和结果写回）

部分模拟器程序的框架设计如下：

```
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
            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，
            break;
        case AUIPC:                                                                //0x17用于建立PC相对地址，使用U型格式，用0填充
            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
            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;
```

这里为测试程序,够测试出各个类型的指令，包括算术、逻辑、控制等，以及区分符号扩展和零扩展和一些易混淆的指令，详情如下：

```
void m_progMem(){
    writeWord(0, (0x666 << 12) | (2 << 7) | (LUI));//指令功能在第2个寄存器写入0x666
    writeWord(4, (1 << 12) | (3 << 7) | (AUIPC));//指令功能在第3个寄存器中写入PC+0x1000
    writeWord(8, (0x66 << 12) | (5 << 7) | (LUI));//指令功能在第5个寄存器写入6
    writeWord(12, (0x0<<25) | (5<<20) | (0<<15) | (SW << 12) | (0x1a << 7) | (STORE));//向(0号寄存器的值加上0x1a)地址写入5号寄存器中的值
    writeWord(16, (0x10<<20) | (0<<15) | (LBU<<12) | (4<<7) | (LOAD));//读取0x10地址上的1byte取最后8位写入4号寄存器
    writeWord(20, (0x0<<25) | (2<<20) | (0<<15) | (BGE<<12) | (0x8<<7) | (BRANCH));//判断0号寄存器和2号寄存器值的大小，如果大于等于则修改NextPC
}
```

指令详情见注释。

测试

测试记录

模拟器运行过程的截图如下：

AUIPC：

```
Registers bofore executing the instruction @0x4
PC=0x4 IR=0xfffff037
R[0]=0xfffff000 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 AUIPC
PC = 4
Imm31_12UtypeZeroFilled = 1000
Registers after executing the instruction
PC=0x8 IR=0x1097
R[0]=0xfffff000 R[1]=0x1004 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[1
0]=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]
```

取值 IR&0x0F80 0x1000, PC 的值，得到 0x1004，存入 R[1]。

LUI：

```
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[1
1]=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=0xfffff037
R[0]=0xfffff000 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]
Y
```

立即数的零扩展,放立即数到 rd 的高 20 位，低 12 位置 0。上述指令中立即数为 0xfffff，存入 R[0]。

SW：

```
Registers bofore executing the instruction @0x808
PC=0x808 IR=0xd0e3
R[0]=0xfffff000 R[1]=0x1004 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[1
0]=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: c, 1004
Registers after executing the instruction
PC=0x80c IR=0x112623
R[0]=0xfffff000 R[1]=0x1004 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[1
0]=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]
```

写入操作，将 R[1]的数据 1004 写入地址 0xc+R[2]。

SLT：

```
Registers bofore executing the instruction @0x828
PC=0x828 IR=0x308233
R[0]=0xfffff000 R[1]=0x1004 R[2]=0xffffffff0 R[3]=0xfff0 R[4]=0x10ff4 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]=0x
0 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 SLT
Registers after executing the instruction
PC=0x82c IR=0x40a2b3
R[0]=0xfffff000 R[1]=0x1004 R[2]=0xffffffff0 R[3]=0xfff0 R[4]=0x10ff4 R[5]=0x1 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]=0x
0 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]
Y
```

指令比较 R[1]、R[4]的大小，如果 R[1]>R[4],R[5]=1，反之为 0。

R[1]=0x1004，R[4]=0x10ff4，R[5]=0x1。

LH：

```
Registers after executing the instruction
PC=0x80c IR=0x112623
R[0]=0xfffff000 R[1]=0x1004 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[1
0]=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]
Registers bofore executing the instruction @0x80c
PC=0x80c IR=0x112623
R[0]=0xfffff000 R[1]=0x1004 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[1
0]=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 LH
Registers after executing the instruction
PC=0x810 IR=0x121103
R[0]=0xfffff000 R[1]=0x1004 R[2]=0xffffffff0 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[1
f]=0x0
Continue simulation (Y/n)? [Y]
Y
```

读出地址 0x1+R[4]，即 0x1 的半字，符号扩展后存入 R[2]，R[2]为 0xfffffff0。从地址 0x1 开始读，第一个字节是 0xf0，原数是 0xfffff000，说明是小端。

LHU：

```
Registers bofore executing the instruction @0x810
PC=0x810 IR=0x121103
R[0]=0xfffff000 R[1]=0x1004 R[2]=0xffffffff0 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[1
f]=0x0
Do LHU
Registers after executing the instruction
PC=0x814 IR=0x125183
R[0]=0xfffff000 R[1]=0x1004 R[2]=0xffffffff0 R[3]=0xfff0 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]
```

读出地址为 0x1+R[4]的半字，零扩展后存入 R[3]，R[3]为 0xfff0。

BGE

```
Registers bofore executing the instruction @0x8
PC=0x8 IR=0x1097
R[0]=0xfffff000 R[1]=0x1004 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[1
0]=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 BGE
Registers after executing the instruction
PC=0x808 IR=0xd0e3
R[0]=0xfffff000 R[1]=0x1004 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[1
0]=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]
Y
```

如果 R[0]中的数>=R[1]中的数，跳转到 0x808(0x1<<7+8)。

BNE

```
Registers bofore executing the instruction @0x814
PC=0x814 IR=0x125183
R[0]=0xffffff000 R[1]=0x1004 R[2]=0xfffffff0 R[3]=0xfff0 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 BNE
Registers after executing the instruction
PC=0x824 IR=0x219863
R[0]=0xffffff000 R[1]=0x1004 R[2]=0xfffffff0 R[3]=0xfff0 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]
Y
```

比较 R[2]和 R[3]的数据，不相等时，跳转到 PC+0x10，即跳转到 0x824。

ADD：

```
Registers bofore executing the instruction @0x824
PC=0x824 IR=0x219863
R[0]=0xffffff000 R[1]=0x1004 R[2]=0xfffffff0 R[3]=0xfff0 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 ADD
Registers after executing the instruction
PC=0x828 IR=0x308233
R[0]=0xffffff000 R[1]=0x1004 R[2]=0xfffffff0 R[3]=0xfff0 R[4]=0x10ff4 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]=0x
0 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]
```

ADD 指令 R[4]=R[1]+R[3]。

ORI：

```
Registers bofore executing the instruction @0x830
PC=0x830 IR=0x105313
R[0]=0xffffff000 R[1]=0x1004 R[2]=0xfffffff0 R[3]=0xfff0 R[4]=0x10ff4 R[5]=0x1 R[6]=0x7ffff800 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 ORI
Registers after executing the instruction
PC=0x834 IR=0xff36393
R[0]=0xffffff000 R[1]=0x1004 R[2]=0xfffffff0 R[3]=0xfff0 R[4]=0x10ff4 R[5]=0x1 R[6]=0x7ffff800 R[7]=0x7ffff8ff 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]
```

ORI 指令 R[6]与 0xff 按位相或得到的数存入 R[7]，R[6]=0x7ffff800，

R[7]=0x7ffff8ff。

SRLT：

```
Registers bofore executing the instruction @0x82c
PC=0x82c IR=0x40a2b3
R[0]=0xffffff000 R[1]=0x1004 R[2]=0xfffffff0 R[3]=0xfff0 R[4]=0x10ff4 R[5]=0x1 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]=0x
0 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 SRLI
Registers after executing the instruction
PC=0x830 IR=0x105313
R[0]=0xffffff000 R[1]=0x1004 R[2]=0xfffffff0 R[3]=0xfff0 R[4]=0x10ff4 R[5]=0x1 R[6]=0x7ffff800 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]
```

算数右移，上述指令中，将 R[0]中的值右移 1 位并存入 R[6]中。

R[0]=0xfffff000，R[6]=0x7ffff800。

收获与体会

通过编写模拟器程序对 RISC-V 基本指令集的模拟，对这些指令理解更加深刻，我们在计算机系统中只是了解到了这些指令的作用，但是不知道这些指令在硬件中是这样运作的，通过这次实验，知道了这些指令具体是怎么实现的，如逻辑右移、算术右移，零扩展、符号扩展等，对 cpu 的指令有了更好的理解。