实验报告

实验名称（设计 RISC-V 的基本指令集 RV32I 的模拟器）

班级：物联 1601 学号：201608010527 姓名：旦增克珠

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

RV32I 指令集请见这里

模拟器程序框架

CPU 执行指令流程：

1.取指令【if】：根据 pc 指令地址，从存储器取出一条指令，同时 PC 根据指令长度自动产生下一条指令需要的指令地址，但遇到“地址转移”指令时，控制器把“转移地址”做些变换送入 PC。

2.指令译码【ID】：对上述得到的指令进行分析译码，确定指令需要完成的操作，从而产生相应的操作控制信号，用于驱动器执行状态的各种操作。

3.指令执行【EXE】：根据译码，具体执行指令动作，转移到结果写回状态。

4.存储器访问【MEM】：访问存储器的操作都在该步骤，给出存储器

的数据地址，把数据写入存储器中数据地址所指定的存储单元或从存储器得到数据地址单元中的数据。

5.结果写回【WB】：指令执行结果或者访问存储器中得到的数据写回相应的目的寄存器中。

对模拟器程序的框架设计如下：

```
294     while(c != 'n') {
295         cout << "Registers bofore executing the instruction @0x" << std::hex << PC << endl;
296         showRegs();                                                          //每次循环显示一下
297         IR = readWord(PC);                                                   //读取pc对应的指令
298         NextPC = PC + WORDSIZE;                                              //赋值下一个PC
299         decode(IR);                                                          //解析指令
300         switch(opcode) {                                                     //这个是在decode时
301             case LUI:                                                        // 执行的操作load
302                 cout << "Do LUI" << endl;
303                 R[rd] = Imm31_12UtypeZeroFilled;                             //这里rd是decode取
304                 break;
305             case AUIPC:                                                      //0x17用于建立PC相
306                 cout << "Do AUIPC" << endl;
307                 cout << "PC = " << PC << endl;
308                 cout << "Imm31_12UtypeZeroFilled = " << Imm31_12UtypeZeroFilled << endl;
309                 R[rd] = PC + Imm31_12UtypeZeroFilled;
310                 break;
311             case JAL:                                                        //0x6F,无条件跳?
312                 cout << "Do JAL" << endl;
313                 R[rd]=PC+4;
314                 NextPC = PC+ Imm20_1JtypeSignExtended;
315                 break;
316             case JALR:                                                       //0x67,无条件跳?
317                 cout << "DO JALR" << endl;
318                 R[rd]=PC+4;
319                 NextPC=R[rs1]+Imm20_1JtypeSignExtended;
320                 break;
321             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
322                 switch(funct3) {
323                     case BEQ://0x0当src1和src2寄存器相等的时候执行
324                         cout << "DO BEQ" << endl;
325                         if(src1==src2){
326                             NextPC = PC + Imm12_1BtypeSignExtended;
327                         }
328                         break;
329                     case BNE://0x1当src1和src2寄存器不相等的时候执行
330                         cout << "Do BNE " << endl;
331                         if(src1!=src2){
332                             NextPC = PC + Imm12_1BtypeSignExtended;
333                         }
334                         break;
335                     case BLT://0x4有符号比较当src1<src2时执行
336                         cout << "Do BLT" << endl;
337                         if((int)src1<(int)src2){
338                             NextPC = PC + Imm12_1BtypeSignExtended;
339                         }
340                         break;
341                     case BGE://0x5有符号比较当src1>=src2时执行
342                         cout << "Do BGE" << endl;
343                         cout<<"src1为 "<<src1<<endl;
344                         cout<<"src2为 "<<src2<<endl;
345                         cout<<"imm为 "<<Imm12_1BtypeSignExtended<<endl;
346                         if((int)src1 >= (int)src2)
347                             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  Loads copy a value from memory to register rd
        /*The LW instruction loads a 32-bit value from memory into rd. LH loads a 16-bit value from memory,
then sign-extends to 32-bits before storing in rd. LHU loads a 16-bit value from memory but then
zero extends to 32-bits before storing in rd. LB and LBU are defined analogously for 8-bit values.*/
            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;
                        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  Stores copy the value in register rs2 to memory. Stype有sr1?
        /*
        The SW, SH, and SB instructions store 32-bit, 16-bit, and 8-bit values from the low bits of registerrs2
        */
            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;
                    //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://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
                //cout << "this is EBREAK !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                    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
                                        //cout << "this is 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
                            /*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，?
                            //TODO: Fill code for the instruction here
                            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?
                            //TODO: Fill code for the instruction here
                            {
                                uint32_t temp = readWord(rs2)&0x00000fff;
                                uint32_t temp1 = rs1 & 0x000fffff;
//                              cout<<"temp值为0x"<<temp<<endl;
//                              cout<<"temp1值为0x"<<temp1<<endl;
//                              cout<<"rd的值为0x"<<rd<<endl;
//                              cout<<"写入rd的值为0x"<<(temp|temp1)<<endl;
                                writeWord(rd,(temp|temp1));
                                cout << "do CSRRS and the result is :" << "rd="<<readWord(rd)<<endl;
                                break;
                            }
                        case CSRRC://队友CSRRS和CSRRC，如果rs1==x0，则指令不会写CSR寄存器
                            /*读取CSR的值，0扩展，写入rd寄存器，整数寄存器rs1中的初始值被视为指定要在csr中清除的位位置?
                            //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;
//                                  cout<<"rd的值为0x"<<rd<<endl;
//                                  cout<<"rs2的值为0x"<<rs2<<endl;
//                                  cout<<"zmm的值为0x"<<zmm<<endl;
//                                  cout<<"tem的值为0x"<<tem<<endl;
                                    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)
                                {
//                                  cout<<"rd的值为0x"<<rd<<endl;
//                                  cout<<"rs2的值为0x"<<rs2<<endl;
//                                  cout<<"zmm的值为0x"<<zmm<<endl;
//                                  cout<<"tem的值为0x"<<tem<<endl;
                                    writeWord(rs2, zmm | tem);
                                }
                                cout << "do CSRRCI and the result is :" << "rd=" << readWord(rd) << endl;
                                break;
                            }
                        default:
```

```
672                     cout << "ERROR: Unknown funct3 in CSRX instruction " << IR << endl;
673                 }
674                 break;
675             default:
676                 cout << "ERROR: Unkown instruction " << IR << endl;
677                 break;
678         }
679
680         //Update PC
681         PC = NextPC;
682
683         cout << "Registers after executing the instruction" << endl;
684         showRegs();
685         cout << "Continue simulation (Y/n)? [Y]" << endl;
686         cin.get(c);
687         getchar();
688     }
689
690     freeMem();
691
```

测试

测试平台

部件	配置	备注
CPU	core(TM) i3-6100U	
内存	DDR3-8GB	
操作系统	Windows10 家庭版	64 位操作系统

测试记录

第一条指令运行后输出：

```
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]=0x
0 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
```

第二条指令运行后输出：

```
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]=0x
0 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=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]=0x
0 R[19]=0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
```

第三条指令运行后输出：

```
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]=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 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]=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
```

第四条指令运行后输出：

```
Registers bofore executing the instruction @0x10
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]=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 LBU
Registers after executing the instruction
PC=0x14 IR=0x1004203


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]=0x3 R[5]=0x66000 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
```

分析和结论

从测试记录来看，模拟器实现了对二进制指令的读入，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。