

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 1501 班

学号：201508010726

姓名：罗松俄珠

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

CPU 指令集

CPU 的指令集请见[这里](#)，其中基本指令集共有_47_条指令。其中的三条指令为我完成的指令,分别为：OR,AND,SRA

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（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 ADD:  
  
            cpu.regs[inst.rd] = cpu.regs[rs] + cpu.regs[rt];  
  
            break;  
  
        case /*其它操作码*/ :  
  
            /* 执行相关操作 */  
  
            break;  
  
        default:  
  
            cout << "无法识别的操作码： " << inst.opcode;    }}
```

其中 while 循环条件可以根据需要改为模拟终止条件。

具体指令内容如下

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 SRA:

```
        cout<<"DO SRA"<<endl;
        R[rd]=(int)src1>>src2;
        break;
```

测试平台

模拟器在如下机器上进行了测试：

部件	配置	
CPU	core i5-6500U	
内存	DDR3 4GB	
操作系统	Windows 7	

测试记录

模拟器的测试输入指令：

```
void progMem() {
```

```
    // Write starts with PC at 0
```

```

writeWord(4, (0x00<<25) | (2<<20) | (1<<15) | (OR<< 12) | (3 << 7) |
(ALURRR));

writeWord(8, (0x00<<25) | (2<<20) | (1<<15) | (AND << 12) | (3 << 7) |
(ALURRR));

writeWord(20, (0x20<<25) | (2<<20) | (1<<15) | (SRA << 12) | (3 << 7) |
(ALURRR));

}

```

SRA,OR,AND 指令的作用：

- SRA: 将 R[rs1]的值，算术右移，空出来的位置使用 R[rs2]的值填充，结果保存到地址为 R[rd]的通用寄存器中。
- OR: R[rs1]与 R[rs2]寄存器的值进行逻辑“或”运算，运算结果保存到地址为 Rrd 的通用寄存器中。
- AND:将地址为 R[rs1]寄存器的值，与地 R[rs2]的通用寄存器的值进行逻辑“与”运算，运算结果保存到地址为 R[rd]的通用寄存器中

模拟器运行过程的截图如下：

进行三条指令之前由执行过其他指令后的地址和各个寄存器的值

OR 指令运行后模拟器的输出

```
Do OR
Registers after executing the instruction
PC=0x8 IR=0x20e1b3
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]
Registers before executing the instruction 0x8
PC=0x8 IR=0x20e1b3
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 AND
半:
```

AND 指令运行后模拟器的输出

```
0x0 R[1a]=0x0 R[1b]=0x0 R[1c]=0x0 R[1d]=0x0 R[1e]=0x0 R[1f]=0x0
Do AND
Registers after executing the instruction
PC=0xc IR=0x20f1b3
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]
y
半:
```

SRA 指令运行后模拟器的输出

```
DO SRA
Registers after executing the instruction
PC=0x18 IR=0x4020d1b3
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]
半:
```

分析和结论

从测试记录来看，模拟器实现了对十六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。