湖南大學

HUNAN UNIVERSITY

《微处理器设计（含实验）》

RISC-V 模型机设计

模拟器部分

实验报告

学生姓名 邱勒铭

学生学号 201608010702

专业班级 智能 1602

指导老师 吴强

完成日期 2019.12.21

一、实验名称：

基于 RISC-V 指令集的微处理器的模拟器设计

实验环境：Window 操作系统

编写语言：C++

调试 IDE：codeblock

二、实验内容：

使用软件程序设计语言编写模拟器，可以模拟执行 RISC-V 指令集中的指令。

实验目的：

在使用硬件设计微处理器时，由于硬件语言较为严格并且不易检查出程序的错误。需要使用软件编程语言设计一个编译器运行与硬件相同的指令与硬件设计的结果进行对比，从而及时发现错误所在。

四、实验原理:

如图所示为 RISC-V 的指令表：

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]			rs1	000	rd	1100111	JALR
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]		rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]		rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]		rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]		rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]		rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]			rs1	000	rd	0000011	LB
imm[11:0]			rs1	001	rd	0000011	LH
imm[11:0]			rs1	010	rd	0000011	LW
imm[11:0]			rs1	100	rd	0000011	LBU
imm[11:0]			rs1	101	rd	0000011	LHU
imm[11:5]		rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]		rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]			rs1	000	rd	0010011	ADDI
imm[11:0]			rs1	010	rd	0010011	SLTI
imm[11:0]			rs1	011	rd	0010011	SLTIU
imm[11:0]			rs1	100	rd	0010011	XORI
imm[11:0]			rs1	110	rd	0010011	ORI
imm[11:0]			rs1	111	rd	0010011	ANDI
0000000		shamt	rs1	001	rd	0010011	SLLI
0000000		shamt	rs1	101	rd	0010011	SRLI
0100000		shamt	rs1	101	rd	0010011	SRAI
0000000		rs2	rs1	000	rd	0110011	ADD
0100000		rs2	rs1	000	rd	0110011	SUB
0000000		rs2	rs1	001	rd	0110011	SLL
0000000		rs2	rs1	010	rd	0110011	SLT
0000000		rs2	rs1	011	rd	0110011	SLTU
0000000		rs2	rs1	100	rd	0110011	XOR
0000000		rs2	rs1	101	rd	0110011	SRL
0100000		rs2	rs1	101	rd	0110011	SRA
0000000		rs2	rs1	110	rd	0110011	OR
0000000		rs2	rs1	111	rd	0110011	AND
fm	pred	succ	rs1	000	rd	0001111	FENCE
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK

RISC-V 每条指令都有操作码 opcode，这是区分它们的关键。指令分为 6 类，每一类的操作码相同，除了 LUI,AUIPC,JAL,JALR 四条指令外，这 6 类指令都通过功能码进一步区分每条指令。

五、实验过程：

指令的执行分为取址，译码，执行三个步骤，由于我们这里是模拟器，不涉及到取址的

过程。因此直接对每一条指令进行译码执行即可。

1. 为方便对指令的操作码和功能码的描述，首先用宏定义它们，比如指令 LUI 的操作码为 00110111，十六进制为 0x37，则使用#define LUI 0x37 定义。

2. 内存使用一个 char 型数组表示。由于 RISC-V 是 32 位指令，因此我们使用 4 个字节存储一条指令。使用 Writeword 函数实现向内存写入指令。再使用 program 函数记录所有待执行的指令，并将它们都写入内存。

如下为 writeword 函数和 program 函数的示例实现部分：

```
void WriteWord(uint32_t addr,uint32_t data){ //写值入存储器
    if(addr>=Msize-wordsize) {
        cout<<"ERROR:地址范围超出内存容量"<<endl;
        return;
    }
    *((uint32_t*)&(M[addr]))=data;
}
```

```
void Program(){
    /*U类指令*/
    WriteWord(0,(0x12345<<12)|(1<<7)|(LUI));
    WriteWord(4,(0x2<<12)|(2<<7)|(AUIPC));
    /*J类指令*/
    WriteWord(8,(0<<31)|(4<<21)|(0<<20)|(0<<12)|(3<<7)|(JAL));
    WriteWord(16,(12<<20)|(5<<15)|(0<<12)|(4<<7)|(JALR));//下一条跳转到10*2+4处，rd=4中存 0x14;

    /*B类指令*/
    //BEQ，r5=r6=0,所以跳转，令立即数=4，PC=24+4*2=32;注意立即数的符号位为0
    WriteWord(24,(0<<31)|(0<<25)|(6<<20)|(5<<15)|(0<<12)|(4<<8)|(0<<7)|(BType));
    //BNE，r3!=r6,所以跳转，令立即数=6，PC=32+6*2=44;注意立即数的符号位为0
    WriteWord(32,(0<<31)|(0<<25)|(6<<20)|(3<<15)|(1<<12)|(6<<8)|(0<<7)|(BType));
    //BLT，r6<r3,所以跳转，令立即数=4，PC=44+4*2=52;注意立即数的符号位为0
    WriteWord(44,(0<<31)|(0<<25)|(3<<20)|(6<<15)|(4<<12)|(4<<8)|(0<<7)|(BType));
    //BGE，r3>r6,所以跳转，令立即数=4，PC=52+4*2=60;注意立即数的符号位为0
    WriteWord(52,(0<<31)|(0<<25)|(6<<20)|(3<<15)|(5<<12)|(4<<8)|(0<<7)|(BType));
    //BLTU，r6<r3,所以跳转，令立即数=4，PC=60+4*2=68;注意立即数的符号位为0
    WriteWord(60,(0<<31)|(0<<25)|(3<<20)|(6<<15)|(6<<12)|(4<<8)|(0<<7)|(BType));
    //BGEU，r3>r6,所以跳转，令立即数=4，PC=68+4*2=76;注意立即数的符号位为0
    WriteWord(68,(0<<31)|(0<<25)|(6<<20)|(3<<15)|(7<<12)|(4<<8)|(0<<7)|(BType));
```

Writeword 为原子操作，第一个参数为写入的地址，第二个参数为写入的内容。

3. 内存里有了指令就可以拿来译码操作了。首先将每条指令分解成不同的部分，比如操作码和功能码。还有源目的寄存器的标号，以及立即数等等。有了这些微指令就可以确定一条指令并对其进行操作。

译码 decode 函数：

```
void Decode(unsigned int IR){ //指令译码
    opcode= IR & 0x7f;//0111 1111截取后七位操作码
    rd= (IR>>7)& 0x1f;//将操作码移位7位后截取后5位，这样是32位吗？
    r1= (IR>>15)&0x1f;
    r2= (IR>>20)&0x1f;
    func3=(IR>>12)&0x7;
    func7=(IR>>25)&0x7f;

    imm31_12U = (IR>>12)& 0xfffff;//取出无符号的前20位(U类)

    imm31J=(IR>>31) & 1; //取出符号位
    imm30_21J=(IR>>21) & 0x3ff;
    imm20J=(IR>>20) &1;
    imm19_12J=(IR>>12) & 0xff;

    imm31_20JR=IR>>20;//JALR取高12位，并且默认移位扩展32位为有符号的
    /*B类指令*/
    imm31B=imm31J;
    imm30_25B=(IR>>25)& 0x3f;//？？？0x3f
    imm11_8B=(IR>>8)&0xf;
    imm7B=(IR>>7)&0x1;
    /*L类指令*/
    imm31_20L=IR>>20; //取高12位，默认扩展32位有符号的??修改成I类类似
    /*S类指令*/
    imm31S=imm31J;
    imm30_25S=(IR>>25)&0x3f;
    imm11_7S=(IR>>7) & 0x1f;
    /*I类指令*/
    imm_sign_31_20I=(int)IR>>20;
    shamt=(IR>>20)&0x1f;

    /************各类拼接**************/
    imm31_12U_0 = imm31_12U<<12;//用0填充低12位，用于U类指令
    imm_sign_31_12J=(imm31J<<20)&0xfff00000|(imm19_12J<<12)|(imm20J<<11)|(imm30_21J);//JAL
    imm_sign_31_25B_11_7B=(imm31B<<12)&0xffff000|(imm7B<<11)|(imm30_25B<<5)|(imm11_8B);
    imm_sign_31_25S_11_7S=(imm31S<<12)&0xffff000|(imm30_25S<<5)|imm11_7S; //S类有符号扩展

}
```

4. 主函数里通过对不同的指令进行译码，将得到的内容进行相应的操作，得到对应的结果，

例如 LUI 指令，将立即数写入对应的寄存器：

```
switch(opcode){
    case LUI:{
        cout<<"执行LUI指令:将立即数作为高20位，低12位用0填充，结果放进rd寄存器"<<endl;
        R[rd]=imm31_12U_0;
        break;
    }
```

六、实验操作：

运行程序，每次执行一条指令：

1. 执行 LUI 指令：

```
----------------在执行指令前PC=0x0-------------------
PC=0x0  IR=0x0
32个寄存器值(16进制)分别为：
R[1]=0  R[2]=0  R[3]=0  R[4]=0  R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0
R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0
R[31]=0 R[32]=0

执行LUI指令:将立即数作为高20位，低12位用0填充，结果放进rd寄存器
---------------执行指令后寄存器的值-------------------
PC=0x4  IR=0x123450b7
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=0  R[3]=0  R[4]=0  R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0 R[14]=0
R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0
R[30]=0 R[31]=0 R[32]=0
```

2. 执行 ALUPC 指令：

```
-----------------在执行指令前PC=0x4--------------------
PC=0x4  IR=0x123450b7
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=0  R[3]=0  R[4]=0  R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0 R[14]=0
R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0
R[30]=0 R[31]=0 R[32]=0

执行ALUPC指令：将立即数作为高20位，低12位用0填充，结果加上此时PC值放入rd寄存器，PC值本身不变
----------------执行指令后寄存器的值--------------------
PC=0x8  IR=0x2117
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=0  R[4]=0  R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

3. 执行 JAL 指令：

```
-----------------在执行指令前PC=0x8--------------------
PC=0x8  IR=0x2117
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=0  R[4]=0  R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行JAL指令：将立即数有符号扩展*2+pc作为新的pc值，并将原pc+4放进rd寄存器
----------------执行指令后寄存器的值--------------------
PC=0x10  IR=0x8001ef
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=0  R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

4. 执行 JALR 指令：

```
-----------------在执行指令前PC=0x10--------------------
PC=0x10  IR=0x8001ef
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=0  R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行JALR指令：将指令高12位作为立即数有符号扩展*2+r1作为新的pc值，并将原pc+4放进rd
----------------执行指令后寄存器的值--------------------
PC=0x18  IR=0xc28267
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

5. 执行 BEQ 指令：

```
-----------------在执行指令前PC=0x18--------------------
PC=0x18  IR=0xc28267
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行BEQ指令：如果r1里值=r2里值，将立即数有符号填充高20位*2+PC作为PC值
----------------执行指令后寄存器的值--------------------
PC=0x20  IR=0x628463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

6. 执行 BLT 指令：

```
-----------------在执行指令前PC=0x2c--------------------
PC=0x2c  IR=0x619663
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行BLT指令：进行有符号比较,如果r1里值<r2里值，将立即数有符号填充高20位*2+PC作为PC值
----------------执行指令后寄存器的值--------------------
PC=0x34  IR=0x334463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

7. 执行 BNE 指令：

```
----------------在执行指令前PC=0x20--------------------
PC=0x20  IR=0x628463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行BNE指令：如果r1里值!=r2里值，将立即数有符号填充高20位*2+PC作为PC值
----------------执行指令后寄存器的值--------------------
PC=0x2c  IR=0x619663
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

8. 执行 BGE 指令：

```
----------------在执行指令前PC=0x34--------------------
PC=0x34  IR=0x334463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行BGE指令：进行有符号比较,如果r1里值>r2里值，将立即数有符号填充高20位*2+PC作为PC值
----------------执行指令后寄存器的值--------------------
PC=0x3c  IR=0x61d463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

9. 执行 BLTU 指令：

```
----------------在执行指令前PC=0x3c--------------------
PC=0x3c  IR=0x61d463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行BLTU指令：进行无符号比较,如果r1里值<r2里值，将立即数有符号填充高20位*2+PC作为PC值
----------------执行指令后寄存器的值--------------------
PC=0x44  IR=0x336463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

10. 执行 BGEU 指令：

```
----------------在执行指令前PC=0x44--------------------
PC=0x44  IR=0x336463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行BGEU指令：进行无符号比较,如果r1里值>r2里值，将立即数有符号填充高20位*2+PC作为PC值
----------------执行指令后寄存器的值--------------------
PC=0x4c  IR=0x61f463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

11. 执行 LB 指令：

```
----------------在执行指令前PC=0x4c--------------------
PC=0x4c  IR=0x61f463
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=0  R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0 R[13]=0
R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0
R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行LB指令：将指令高12位作为立即数有符号扩展+r1寄存器的值，作为地址，读取存储器相应地址中的字节并扩展到32位放在rd寄存器
----------------执行指令后寄存器的值--------------------
PC=0x50  IR=0x3f418283
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0
R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0
R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

12. 执行 LH 指令：

```
----------------在执行指令前PC=0x50--------------------
PC=0x50  IR=0x3f418283
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=0  R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0 R[12]=0
R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0
R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行LH指令：将指令高12位作为立即数有符号扩展+r1寄存器的值，作为地址，读取存储器相应地址中的2个字节并扩展到32位放在rd寄存
器
----------------执行指令后寄存器的值--------------------
PC=0x54  IR=0x3f419303
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0
R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0
R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

13. 执行 LW 指令：

```
----------------在执行指令前PC=0x54--------------------
PC=0x54  IR=0x3f419303
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=0  R[8]=0  R[9]=0  R[10]=0 R[11]=0
R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0
R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行LW指令：将指令高12位作为立即数有符号扩展+r1寄存器的值，作为地址，读取存储器相应地址中的4个字节放在rd寄存器
----------------执行指令后寄存器的值--------------------
PC=0x58  IR=0x3f41a383
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=0  R[9]=0  R[10]=0
R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0
R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

14. 执行 LBU 指令：

```
----------------在执行指令前PC=0x58--------------------
PC=0x58  IR=0x3f41a383
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=0  R[9]=0  R[10]=0
R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0
R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行LBU指令：将指令高12位作为立即数有符号扩展+r1寄存器的值，作为地址，读取存储器相应地址中的字节并无符号扩展到32位放在rd
寄存器
----------------执行指令后寄存器的值--------------------
PC=0x5c  IR=0x3f41c403
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=0  R[10]=0
R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0
R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

15. 执行 LHU 指令：

```
----------------在执行指令前PC=0x5c--------------------
PC=0x5c  IR=0x3f41c403
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=0  R[10]=0
R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0
R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行LHU指令：将指令高12位作为立即数有符号扩展+r1寄存器的值，作为地址，读取存储器相应地址中的2个字节并无符号扩展到32位放
在rd寄存器
----------------执行指令后寄存器的值--------------------
PC=0x60  IR=0x3f41d483
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=0 R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0
R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

16. 执行 SB 指令：

```
----------------在执行指令前PC=0x60--------------------
PC=0x60  IR=0x3f41d483
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=0 R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0
R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行SB指令：将立即数有符号扩展32位与r1寄存器相加，作为存储器地址，将r2寄存器中值低8位存进存储器
执行指令后相应内存值为0xfe
----------------执行指令后寄存器的值--------------------
PC=0x64  IR=0x20758023
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=0 R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0
R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

17. 执行 SLTIU 指令：

```
----------------在执行指令前PC=0x74--------------------
PC=0x74  IR=0xfff1a593
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=0 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0
R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行SLTIU指令:进行无符号比较，如果r1<imm(有符号扩展)，rd=1
----------------执行指令后寄存器的值--------------------
PC=0x78  IR=0xfff1b613
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0
R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

18. 执行 XORI 指令：

```
----------------在执行指令前PC=0x78--------------------
PC=0x78  IR=0xfff1b613
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=0 R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0
R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行XORI指令:进行异或操作，rd=r1^imm(符号扩展到32位)
----------------执行指令后寄存器的值--------------------
PC=0x7c  IR=0xff33c693
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0
R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

19. 执行 ORI 指令：

```
----------------在执行指令前PC=0x7c--------------------
PC=0x7c  IR=0xff33c693
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=0 R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0 R[23]=0
R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行ORI指令:进行或操作，rd=r1|imm(符号扩展到32位)
----------------执行指令后寄存器的值--------------------
PC=0x80  IR=0xf0116713
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0
R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

20. 执行 ANDI 指令：

```
----------------在执行指令前PC=0x80--------------------
PC=0x80  IR=0xf0116713
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=0 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0
R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行ANDI指令:进行与操作，rd=r1&imm(符号扩展到32位)
----------------执行指令后寄存器的值--------------------
PC=0x84  IR=0xf417793
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0
R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

21. 执行 SLLI 指令：

```
----------------在执行指令前PC=0x84--------------------
PC=0x84  IR=0xf417793
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=0 R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0 R[22]=0
R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行SLLI指令：进行左移操作，后面填充0，rd=r1<<shamt
----------------执行指令后寄存器的值--------------------
PC=0x88  IR=0x431813
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=ffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0
R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

22. 执行 SRLI 指令：

```
----------------在执行指令前PC=0x88--------------------
PC=0x88  IR=0x431813
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=0 R[18]=0 R[19]=0 R[20]=0 R[21]=0
R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行SRLI指令：进行逻辑右移操作，前面填充0，rd=r1>>shamt
----------------执行指令后寄存器的值--------------------
PC=0x8c  IR=0x435893
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=0 R[19]=0 R[20]=0
R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

23. 执行 SRAI 指令：

```
----------------在执行指令前PC=0x8c--------------------
PC=0x8c  IR=0x435893
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=0 R[19]=0 R[20]=0
R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行SRAI指令：进行算数右移操作，前面填充最高位，rd=r1>>shamt
----------------执行指令后寄存器的值--------------------
PC=0x90  IR=0x40435913
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=0
R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

24. 执行 ADD 指令：

```
----------------在执行指令前PC=0x90--------------------
PC=0x90  IR=0x40435913
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=0
R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行ADD指令：rd=r1+r2，取低32位，忽略溢出
----------------执行指令后寄存器的值--------------------
PC=0x94  IR=0xa189b3
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

25. 执行 SUB 指令：

```
----------------在执行指令前PC=0x94--------------------
PC=0x94  IR=0xa189b3
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=0 R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0

执行SUB指令：rd=r1-r2，取低32位，忽略溢出
----------------执行指令后寄存器的值--------------------
PC=0x98  IR=0x40350a33
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0
```

26. 执行 SLL 指令：

```
----------------在执行指令前PC=0x98--------------------
PC=0x98  IR=0x40350a33
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=0 R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0 R[32]=0


执行SLL指令：r1向左移动 r2值的低5位次，将结果放在rd中
----------------执行指令后寄存器的值--------------------
PC=0x9c  IR=0x639ab3
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0
R[32]=0
```

27. 执行 SLT 指令：

```
----------------在执行指令前PC=0x9c--------------------
PC=0x9c  IR=0x639ab3
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=0 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0
R[32]=0

执行SLT指令：有符号比较，如果r1<r2,将1写入rd中
----------------执行指令后寄存器的值--------------------
PC=0xa0  IR=0x72ab33
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0
R[32]=0
```

28. 执行 SLTU 指令：

```
----------------在执行指令前PC=0xa0--------------------
PC=0xa0  IR=0x72ab33
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=0 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0
R[32]=0

执行SLTU指令：无符号比较，如果r1<r2,将1写入rd中
----------------执行指令后寄存器的值--------------------
PC=0xa4  IR=0x53bbb3
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=1 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0
R[32]=0
```

29. 执行 XOR 指令：

```
----------------在执行指令前PC=0xa4--------------------
PC=0xa4  IR=0x53bbb3
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=1 R[24]=0 R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0 R[31]=0
R[32]=0

执行XOR指令：rd=r1^r2
----------------执行指令后寄存器的值--------------------
PC=0xa8  IR=0x324c33
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=1 R[24]=18        R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0
R[31]=0 R[32]=0
```

30. 执行 SRL 指令：

```
----------------在执行指令前PC=0xa8--------------------
PC=0xa8  IR=0x324c33
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=1 R[24]=18        R[25]=0 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0
R[31]=0 R[32]=0

执行SRL指令：r1右移r2低5位次，高位补0
----------------执行指令后寄存器的值--------------------
PC=0xac  IR=0x6adcb3
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=1 R[24]=18        R[25]=2 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0
R[31]=0 R[32]=0
```

31. 执行 SRA 指令：

```
----------------在执行指令前PC=0xac--------------------
PC=0xac  IR=0x6adcb3
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=1 R[24]=18        R[25]=2 R[26]=0 R[27]=0 R[28]=0 R[29]=0 R[30]=0
R[31]=0 R[32]=0

执行SRA指令：r1右移r2低5位次，高位补符号位
----------------执行指令后寄存器的值--------------------
PC=0xb0  IR=0x406add33
32个寄存器值(16进制)分别为：
R[1]=12345000   R[2]=2004       R[3]=c  R[4]=14 R[5]=fffffffe   R[6]=fffff6fe   R[7]=1234f6fe   R[8]=fe R[9]=f6fe
R[10]=b R[11]=0 R[12]=1 R[13]=edcb090d  R[14]=ffffff05  R[15]=4 R[16]=ffff6fe0  R[17]=fffff6f   R[18]=ffffff6f  R[19]=17
        R[20]=ffffffff  R[21]=80000000  R[22]=1 R[23]=1 R[24]=18        R[25]=2 R[26]=fffffffe  R[27]=0 R[28]=0 R[29]=0
R[30]=0 R[31]=0 R[32]=0
```