实验报告

RISC-V 基本指令集模拟器设计与实现

智能 1602 班 201608010707 潘曙辉

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

- 模拟器采用 C/C++或 SystemC 语言
- 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

实验内容

CPU 指令集

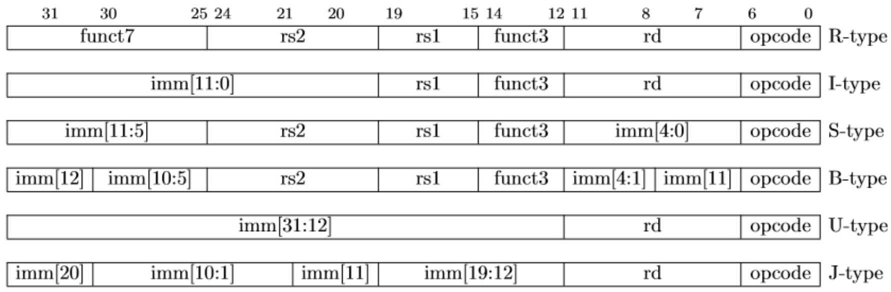
Instruction	Constraints	Code Points	Purpose
LUI	*rd*=x0	2^{20}	*Reserved for future standard use*
AUIPC	*rd*=x0	2^{20}	
ADDI	*rd*=x0, and either *rs1*≠x0 or *imm*≠0	$2^{17}-1$	
ANDI	*rd*=x0	2^{17}	
ORI	*rd*=x0	2^{17}	
XORI	*rd*=x0	2^{17}	
ADDIW	*rd*=x0	2^{17}	
ADD	*rd*=x0	2^{10}	
SUB	*rd*=x0	2^{10}	
AND	*rd*=x0	2^{10}	
OR	*rd*=x0	2^{10}	
XOR	*rd*=x0	2^{10}	
SLL	*rd*=x0	2^{10}	
SRL	*rd*=x0	2^{10}	
SRA	*rd*=x0	2^{10}	
ADDW	*rd*=x0	2^{10}	
SUBW	*rd*=x0	2^{10}	
SLLW	*rd*=x0	2^{10}	
SRLW	*rd*=x0	2^{10}	
SRAW	*rd*=x0	2^{10}	
FENCE	*pred*=0 or *succ*=0	$2^{5}-1$	
SLTI	*rd*=x0	2^{17}	*Reserved for custom use*
SLTIU	*rd*=x0	2^{17}	
SLLI	*rd*=x0	2^{11}	
SRLI	*rd*=x0	2^{11}	
SRAI	*rd*=x0	2^{11}	
SLLIW	*rd*=x0	2^{10}	
SRLIW	*rd*=x0	2^{10}	
SRAIW	*rd*=x0	2^{10}	
SLT	*rd*=x0	2^{10}	
SLTU	*rd*=x0	2^{10}	

31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	width	dest	LOAD

31 — 25	24 — 20	19 — 15	14 — 12	11 — 7	6 — 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7	5	5	3	5	7
offset[11:5]	src	base	width	offset[4:0]	STORE

Register operand				
Instruction	rd	rs1	read CSR?	write CSR?
CSRRW	x0	-	no	yes
CSRRW	!x0	-	yes	yes
CSRRS/C	-	x0	yes	no
CSRRS/C	-	!x0	yes	yes
Immediate operand				
Instruction	rd	uimm	read CSR?	write CSR?
CSRRWI	x0	-	no	yes
CSRRWI	!x0	-	yes	yes
CSRRS/CI	-	0	yes	no
CSRRS/CI	-	!0	yes	yes

指令的类型:



模拟器程序框架

cpu 执行指令的流程为

1. 取指
2. 译码
3. 执行

整个模拟器的运行封装在一个 while 循环中，当输入为 n 的时候，表示停止模拟器。

```
while(c != 'n') {
    cout << "Registers before executing the instruction @0x" << std::hex << PC << endl;
    //show32Mess();
```

每执行一条指令就输入是否继续执行，getchar()是用来消去回车的

```
        cin.get(c);
        getchar();
    }
```

每次循环依次取指，设置 NextPC，解析指令，根据解析的指令执行相应的操作，其中 IR 是指令寄存器，用来保存指令，PC 是程序计数器，用来指示指令在存储器中的位置。

```
IR = readWord(PC);
NextPC = PC + WORDSIZE;
decode(IR);
cout<<"this is IR test0 "<<IR<<endl;
switch(opcode) {
    case LUI:
```

下面是 readWord 的具体实现，读出某一个地址连续的 4byte，可以用这个函数来读取指令，因为一条指令刚好是 4byte。

```
uint32_t readWord(unsigned int address) {
    if(address >= MSize-WORDSIZE) {
        cout << "ERROR: Address out of range in readWord" << endl;
        return 0;
    }
    return *((uint32_t*)&(M[address]));
}
```

下面是 decode 的具体实现，根据上面的指令的类型格式取出指令中某些位，比如 imm11_5s 表示的是 S 类型指令中立即数 5 到 11 位的数据，在后面和 imm4_0s 一起构成了 S 类型指令中的立即数 Imm11_0StypeSignExtended。

```
void decode(uint32_t instruction) {
    // Extract all bit fields from instruction
    opcode = instruction & 0x7F;
    rd = (instruction & 0x0F80) >> 7;
    cout<<"this is rd "<<rd<<endl;
    rs1 = (instruction & 0xF8000) >> 15;
    zimm = rs1;
    rs2 = (instruction & 0x1F00000) >> 20;
    shamt = rs2;
    funct3 = (instruction & 0x7000) >> 12;
    funct7 = instruction >> 25;
    imm11_0i = ((int32_t)instruction) >> 20;
    csr = instruction >> 20;
    imm11_5s = ((int32_t)instruction) >> 25;
    imm4_0s = (instruction >> 7) & 0x01F;
    imm12b = ((int32_t)instruction) >> 31;
    imm10_5b = (instruction >> 25) & 0x3F;
    imm4_1b = (instruction & 0x0F00) >> 8;
    imm11b = (instruction & 0x080) >> 7;
    imm31_12u = instruction >> 12;
    imm20j = ((int32_t)instruction) >> 31;
    imm10_1j = (instruction >> 21) & 0x3FF;
    imm11j = (instruction >> 20) & 1;
    imm19_12j = (instruction >> 12) & 0x0FF;
    pred = (instruction >> 24) & 0x0F;
    succ = (instruction >> 20) & 0x0F;

    // ========================================================================
    // Get values of rs1 and rs2
    src1 = R[rs1];
    src2 = R[rs2];

    // Immediate values
    Imm11_0ItypeZeroExtended = imm11_0i & 0x0FFF;
    Imm11_0ItypeSignExtended = imm11_0i;

    Imm11_0StypeSignExtended = (imm11_5s << 5) | imm4_0s;

    Imm12_1BtypeZeroExtended = imm12b & 0x00001000 | (imm11b << 11) | (imm10_5b << 5) | (imm4_1b << 1);
    Imm12_1BtypeSignExtended = imm12b & 0xFFFFF000 | (imm11b << 11) | (imm10_5b << 5) | (imm4_1b << 1);

    Imm31_12UtypeZeroFilled = instruction & 0xFFFFF000;

    Imm20_1JtypeSignExtended = (imm20j & 0xFFF00000) | (imm19_12j << 12) | (imm11j << 11) | (imm10_1j << 1);
    Imm20_1JtypeZeroExtended = (imm20j & 0x00100000) | (imm19_12j << 12) | (imm11j << 11) | (imm10_1j << 1);
    // ========================================================================
}
```

具体指令的实现如下(以 LUI,AUIPC,JAL,JALR 为例)

可以看到 LUI 和 AUIPC 是写寄存器的指令,LUI 是把立即数写入 rd 寄存器,AUIPC 把程序计数器和一个立即数相加的写入 rd 寄存器,这个指令的作用是构造 PC 相对地址。JAL 和 JALR 是无条件跳转指令，是通过对 NextPC 赋值来实现的。我自己理解 JAL 是相对跳转即相对 PC 跳转，而 JALR 是绝对跳转，即跳转到由 rs1 指定的指令上去，我们可以先对某一个寄存器赋值，然后再调用 JALR 指令跳转到我们想跳转到的地方。

```
switch(opcode) {
    case LUI:
        cout << "Do LUI" << endl;
        R[rd] = Imm31_12UtypeZeroFilled;
        break;
    case AUIPC:
        cout << "Do AUIPC" << endl;
        cout << "PC = " << PC << endl;
        cout << "Imm31_12UtypeZeroFilled = " << Imm31_12UtypeZeroFilled << endl;
        R[rd] = PC + Imm31_12UtypeZeroFilled;
        break;
    case JAL:
        cout << "Do JAL" << endl;
        R[rd]=PC+4;
        NextPC = PC+ Imm20_1JtypeSignExtended;
        break;
    case JALR:
        cout << "DO JALR" << endl;
        R[rd]=PC+4;
        NextPC=R[rs1]+Imm20_1JtypeSignExtended;
        break;
```

测试

测试平台

部件	配置
CPU	core i5-6300U
内存	12GB
操作系统	windows 10

测试记录

我用于测试的指令集如下

```
void m_progMem(){
    writeWord(0, (0x666 << 12) | (2 << 7) | (LUI));//指令功能在第2个寄存器写入0x666000
    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0x66000
    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为 PC + Imm12_1BtypeSignExtended;
}
```

用于显示的函数有两个，分别是显示前 32 个内存地址和所有寄存器的函数(在该函数中调用显示内存地址的函数)， 这个函数分别在每条指令执行前和执行后调用

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

```
void showRegs() {
    cout << "PC=0x" << std::hex << PC << " " << "IR=0x" << std::hex << IR << endl;
    show32Mess();
    for(int i=0; i<32; i++) {
        cout << "R[" << i << "]=0x" << std::hex << R[i] << " ";
    }
    cout << endl<<endl;
}
```

```
while(c != 'n') {
    cout << "Registers bofore executing the instruction @0x" << std::hex << PC << endl;
    showRegs();
    IR = readWord(PC);
    NextPC = PC + WORDSIZE;
    decode(IR);
    switch(opcode) {
        case LUI:
```

```
    cout << "Registers after executing the instruction" << endl;
    showRegs();
    cout << "Continue simulation (Y/n)? [Y]" << endl;
    cin.get(c);
    getchar();
}
```

第一条指令，在第 2 个寄存器写入 0x666000

```
writeWord(0, (0x666 << 12) | (2 << 7) | (LUI));
```

可以看到程序打印出了指令执行前后的 PC,IR 值，内存值和寄存器值。显示出了执行的指令，也可以看到第 2 个寄存器的值由 0 变为了 0x666。

```
D:\大4\小学期\mycode\m_cpusim.exe
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

Continue simulation (Y/n)? [Y]
y
```

第二条指令，在第 3 个寄存器中写入 PC+0x1000

```
writeWord(4, (1 << 12) | (3 << 7) | (AUIPC));
```

```
D:\大4\小学期\mycode\m_cpusim.exe
Continue simulation (Y/n)? [Y]
y
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

Continue simulation (Y/n)? [Y]
y
```

第三条指令，在第 5 个寄存器写入 0x66000

```
writeWord(8, (0x66 << 12) | (5 << 7) | (LUI));
```

```
Continue simulation (Y/n)? [Y]
y
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]=0x
0 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]=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
```

第四条指令，向(0 号寄存器的值加上 0x1a)地址写入 5 号寄存器中的值

```
writeWord(12, (0x0<<25) | (5<<20) | (0<<15) | (SW << 12) | (0x1a << 7) | (STORE));
```

```
Continue simulation (Y/n)? [Y]
y
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

Continue simulation (Y/n)? [Y]
y
```

第五条指令，读取 0x10 地址上的 1byte 取最后 8 位写入 4 号寄存器

```
Continue simulation (Y/n)? [Y]
y
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

Continue simulation (Y/n)? [Y]
y
```

第六条指令，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 为 PC + Imm12_1BtypeSignExtended，这里因为 0 号寄存器为 0x0，2 号寄存器

为 0x666000 所以不会修改 NextPC 的值。

```
writeWord(20, (0x0<<25) | (2<<20) | (0<<15) | (BGE<<12) | (0x8<<7) | (BRANCH));
```

```
D:\大4小学期\mycode\m_cpusim.exe
Continue simulation (Y/n)? [Y]
y
Registers bofore executing the instruction @0x14
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

Do BGE
src1为 0
src2为 666000
imm为 8
Registers after executing the instruction
PC=0x18 IR=0x205463


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

Continue simulation (Y/n)? [Y]
```

分析和结论

其实只要理解了各个指令是具体是做了什么，是读取还是写入，是对寄存器操作还是对内存操作还是对 NextPC 操作，剩下的就是编程了。