微处理器系统设计

实验名称　RISC-V 基本指令集模拟器设计与实现

学生姓名　陈德飞

学生学号　201608010729

专业班级　计科 1602

实验报告

实验目标

使用 c++完成一个模拟 RISC-V 的基本整数指令集 RV32I 的模拟器设计。

实验要求

软件设计采用 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

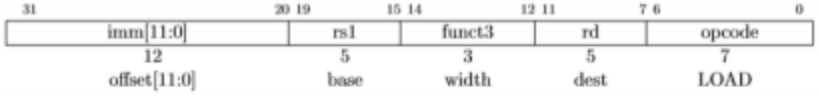
实验内容

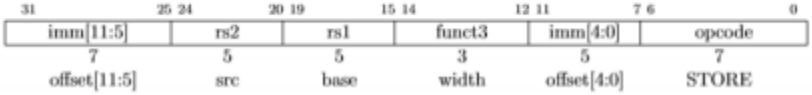
模拟器的大部分代码都已经给出，所以我们只要在了解代码的基础上稍作修改即可。代码运行的大致流程就是取址，译码，执行指令。

CPU 指令集如下所示:

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}	

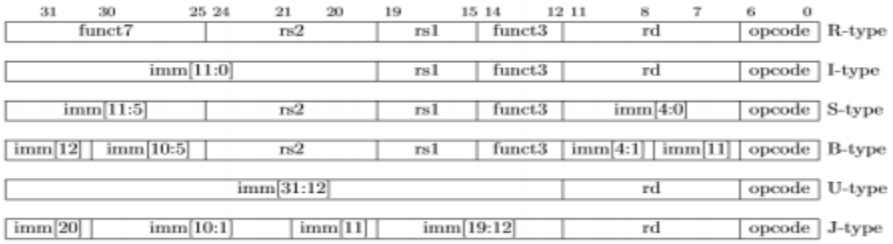
指令解析方式如下





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

指令类型如下:



整个模拟器的运行实质上就是一个 while 循环。

输入 n 的时候，模拟器停止运行。输入 y，模拟器会执行下一条指令。每执行一条指令就输入一次进行判断。每次循环都先进行取址。然后将下一次需要取址的地址设为 NextPC。对指令进行解析根据。解析完后通过判断语句执行相应的操作。

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。

至于指令具体的执行方式就是对用数组模拟出的寄存器和内存或 PC 等的值进行修改。由于是用 c++模拟出来的，所以这些修改都比较直接和简单。

实验结果:

测试平台:win10-64 位

运行的指令都由函数 progMem()写入到内存中，等待后面的循环进行调用。

这里运行前五行指令进行展示

writeWord(0, (0x123 << 12) | (2 << 7) | (LUI));//指令功能在第 2 个寄存器写入 0x123

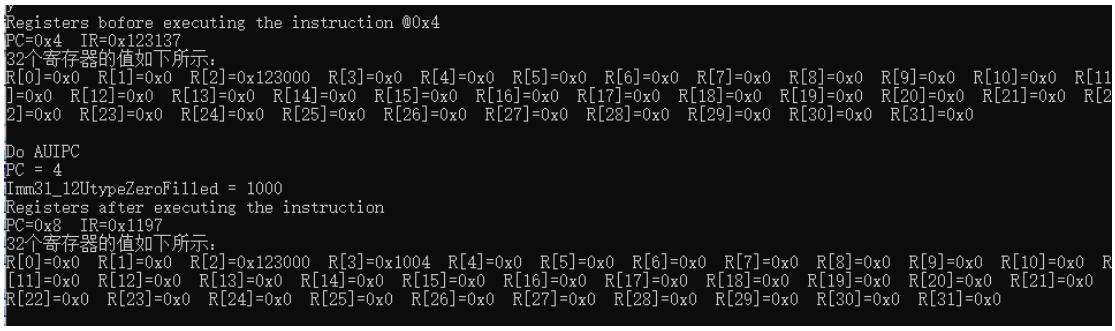
```
Registers bofore executing the instruction @0x0
PC=0x0  IR=0x0
32个寄存器的值如下所示：
R[0]=0x0  R[1]=0x0  R[2]=0x0  R[3]=0x0  R[4]=0x0  R[5]=0x0  R[6]=0x0  R[7]=0x0  R[8]=0x0  R[9]=0x0  R[10]=0x0  R[11]=0x0
  R[12]=0x0  R[13]=0x0  R[14]=0x0  R[15]=0x0  R[16]=0x0  R[17]=0x0  R[18]=0x0  R[19]=0x0  R[20]=0x0  R[21]=0x0  R[22]=0x
0  R[23]=0x0  R[24]=0x0  R[25]=0x0  R[26]=0x0  R[27]=0x0  R[28]=0x0  R[29]=0x0  R[30]=0x0  R[31]=0x0

Do LUI
Registers after executing the instruction
PC=0x4  IR=0x123137
32个寄存器的值如下所示：
R[0]=0x0  R[1]=0x0  R[2]=0x123000  R[3]=0x0  R[4]=0x0  R[5]=0x0  R[6]=0x0  R[7]=0x0  R[8]=0x0  R[9]=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[20]=0x0  R[21]=0x0  R[2
2]=0x0  R[23]=0x0  R[24]=0x0  R[25]=0x0  R[26]=0x0  R[27]=0x0  R[28]=0x0  R[29]=0x0  R[30]=0x0  R[31]=0x0

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

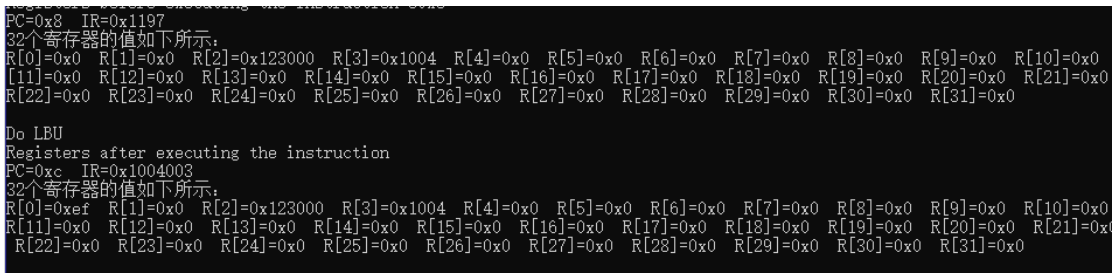
可以看到第 2 个寄存器的值由 0 变成了 0x123

writeWord(4, (1 << 12) | (3 << 7) | (AUIPC));//指令功能在第 3 个寄存器中写入 PC+0x1000

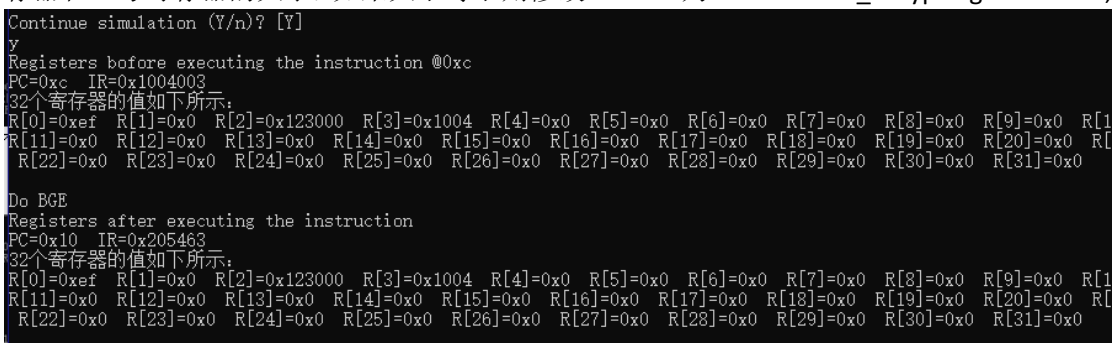


可以看到第三个寄存器的值变成了 1000+4

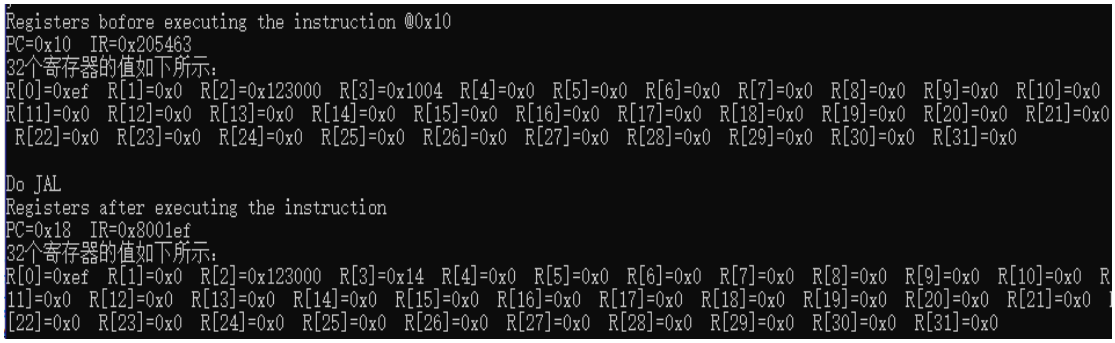
writeWord(8, (0x10<<20) | (0<<15) | (LBU<<12) | (1<<7) | (LOAD));//读取 0x10 地址上的 1byte 取最后 8 位写入 1 号寄存器



writeWord(12, (0x0<<25) | (2<<20) | (0<<15) | (BGE<<12) | (0x8<<7) | (BRANCH));//判断 0 号寄存器和 2 号寄存器的大小，如果大于等于则修改 NextPC 为 PC + Imm12_1BtypeSignExtended;



writeWord(16,(0<<31)|(4<<21)|(0<<20)|(0<<12)|(3<<7) | (JAL));　　//无条件跳转



实验总结与心得体会

本实验只需要理解各个指令是具体是做了什么，是读取还是写入，分清是对 PC 操作还是对寄存器操作就不难了。

通过本次实验，我对 RISC-V 架构有了很多的了解。也对 cpu 的各个指令有了更清楚的认识。

本次实验中遇到了很多问题，尤其是位运算很多，还有指针转换，不过做完实验后发现，虽然看上去很难，但只要细心，就不会太大的问题了。