1. **简介**

我们组基于C语言实现了一个RISCV32I基本指令集的指令系统，这些指令分别是LUI,AUIPC,JAL,JALR,BEQ,BNE,BLT,BGE,BLTU,BGEU,LB,LH,LW,LBU,LHU,SB,SH,SW,ADDI,SLII,SLTIU,XORI,ORI,ANDI,SLLI,SRLI,SRAI,ADD,SUB,SLL,SLT,SLTU,XOR,SRL,SRA,OR,AND,SUB,SLL,SLT,SLTU,XOR,SRL,SRA,OR,AND,FENCE,FENCE.I,ECALL,EBREAK,CSRRW,CSRRS,CSRRC,CSRRWI,CSRRSI,CSRRCI。

**代码文件结构：**

----RISCV32I

|----vairable.h 变量声明头文件

|----operation.h 操作常量头文件

|----memory.c 内存读写代码文件

|----register.c 寄存器读写代码文件

|----decode.c 译码代码文件

|----op\_function.c 操作函数代码文件

|----execution.c 指令集运行代码文件

|----test.c 测试代码文件

**编写环境为：**Visual Studio 2015，并且向下兼容

CodeBlocks需要新建项目然后在项目内添加所有的文件，就可以编译

1. **运行与测试**

该程序的入口在test.c文件中的main函数处。该函数中就包含了所有指令的测试样例。这个函数首先申请了内存和寄存器的空间，然后将其初始化为0。然后定义了需要用到的内存和寄存器，并为其赋值。这之后就可以运行测试的指令了。所有的测试指令都已经写好，并都加了注释符号。我们提供了两种指令运行模式：单指令模式和多指令模式。

**单指令模式：**

在这个模式下每次仅运行一条指令，由于内存均被初始化为0，所以当一条指令运行完毕，PC的下一个地址存的是0，测试会终止。如果需要运行哪条指令，就将该指令的注释符号去掉，编译运行即可。我们接下来用两个例子来进行具体的说明。

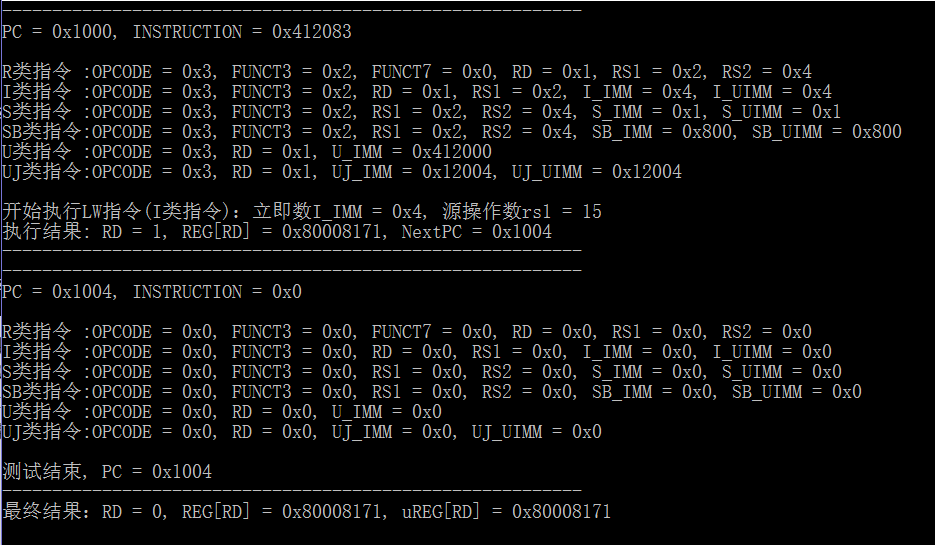
例1：

若要测试LW指令，我们找到注释的LW指令，然后去掉其注释符号，得到：



Write\_mem\_uword(uword PC, uword insn)表示将指令insn存入内存地址为PC的内存中。指令的insn是根据RISCV32规定的格式构造的。指令0x00412083的OPCODE是0x3，FUNCT3是0x2，这说明该指令为LW指令，是I类指令，立即数为0x4，源操作数RS1寄存器为0x2号，目的寄存器为0x1号。

然后编译运行test.c，得到：



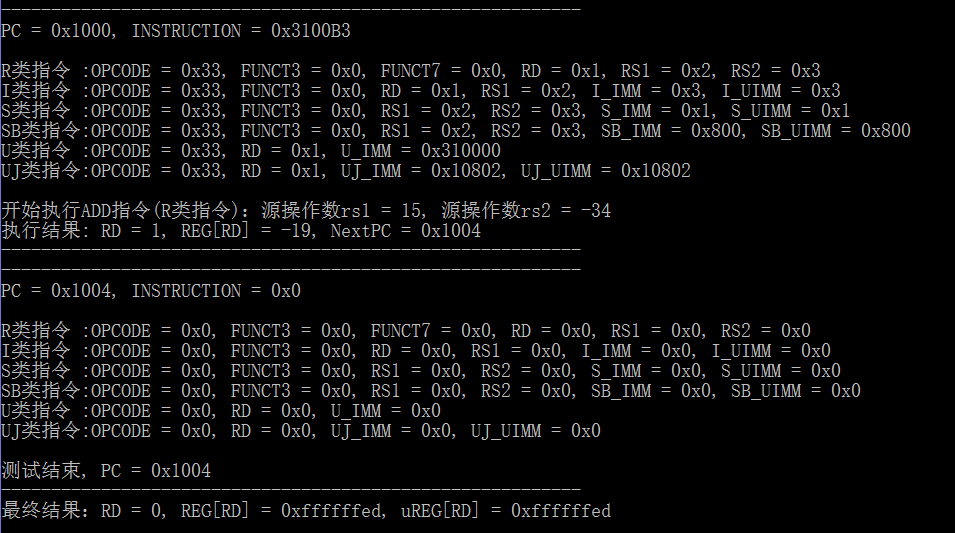
可以看出，LW指令将0x13内存中的数0x80008171存入了RD寄存器中。然后PC+4，而该地址中没有指令，所以测试终止。

例2：

现在我们测试ADD指令，去掉ADD指令注释，得到：



编译运行，得到：



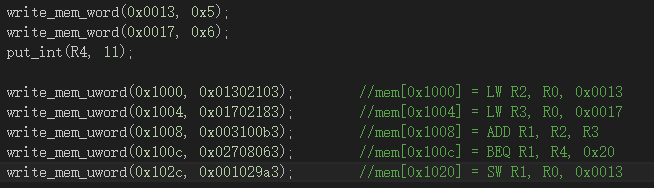
我们可以看到reg[1] = reg[2](=15) + reg[3](=-34) = -19，而最终结果中表示的是寄存器中存储的结果的格式。

需要说明的是，首先我们需要测试有符号立即数的符号扩展正确性：在JAL指令处测试UJ立即数、在JALR指令处测试I立即数、在BEQ指令处测试SB立即数、在SB指令处测试S立即数。而每个立即数的符号扩展的正确性只检查这一次；然后是对于有代表性的跳转指令进行了不同条件下的跳转模式；最后是对位移函数的扩展测试。具体的测试内容请参见test.c测试代码的注释。

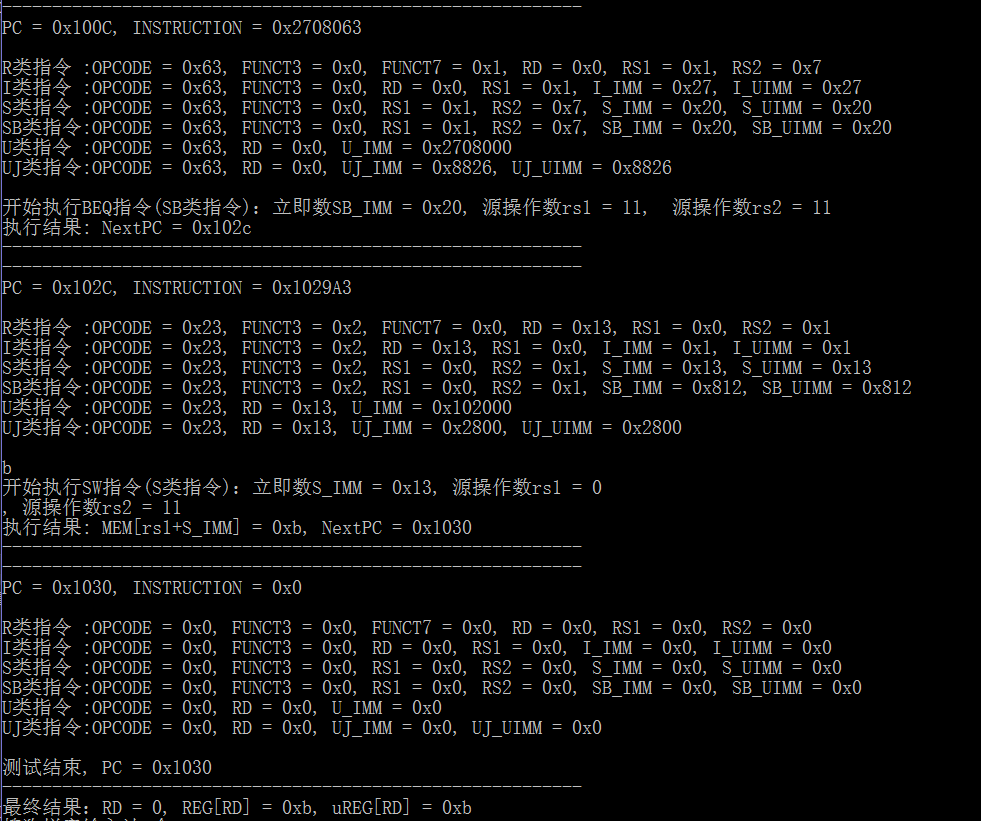
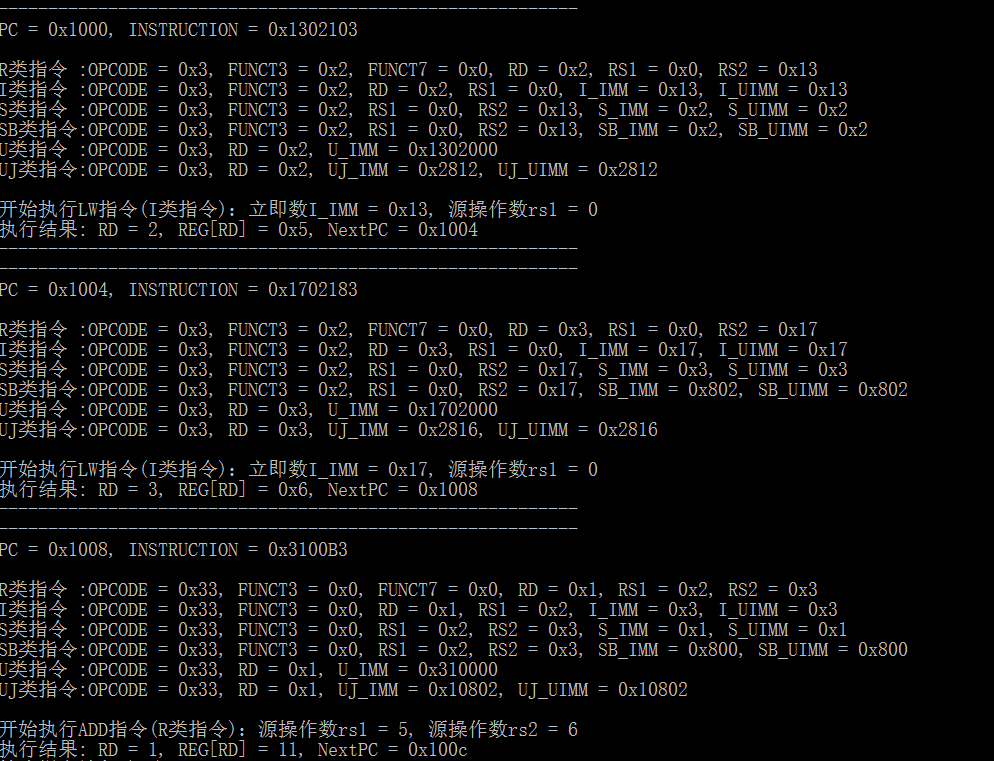
**多指令模式：**

这个模式就是指一次性向内存中存储多个指令，然后依序运行，最后输出结果。

我们这里仅构造了一个简单的多指令。仅供参考：



测试结果如下：



我们可以看出，最后MEM中存储的是正确结果。

**三.自定义测试指令**

1.首先需要根据RISCV32基本指令集构建出指令集的insn

2.将需要的数据使用put\_int或者write\_mem写入寄存器、内存

3.将指令insn写入PC所指的内存中

4.编译运行，结果可以在console窗口内显示出来