



湖南大学  
HUNAN UNIVERSITY

## 课程实验报告

课程名称: 夏季小学期实验

专业班级: 通信工程 1602

姓 名: 吴雨松

学 号: 201608030222

完成时间: 2019 年 9 月 3 日

通信工程系

## 实验名称:

执行 RISC-V 的基本整数指令集 RV32I 的 CPU 设计（单周期实现）

## 实验目标:

设计一个能够执行 RISC-V 基本整数指令集的 CPU

## 实验要求:

采用 VHDL 或 Verilog 语言进行设计

## 实验内容:

CPU 指令集见 <https://riscv.org/specifications/>，我完成的指令是

Load/Store 指令和控制转移指令

具体指令：lb、lbu、lh、lhu、lw、sb、sh、sw、beq、bne、blt、bltu、bge、bgeu、jal、jalr、auipc

## 实验特点:

在这个单周期 cpu 中，Load/Store 指令执行都需要两个周期，而其余的指令执行只需要一个周期

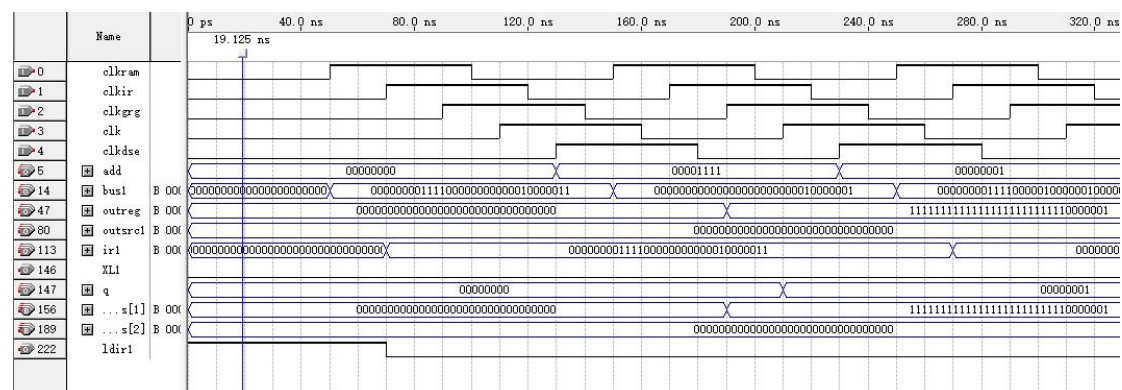
## 测试

### 1、Load/Store 指令

#### (1) lb

测试所用的二进制指令为：00000000111100000000000010000011

功能仿真所得结果为：



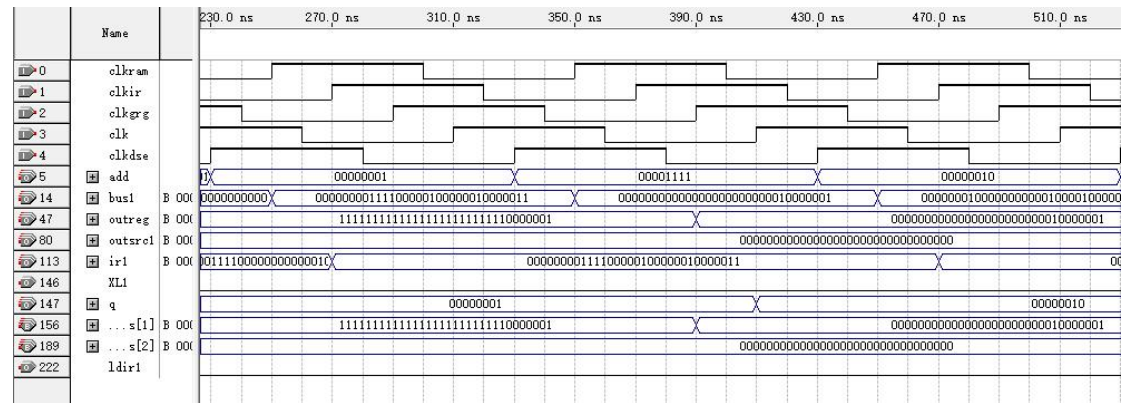
分析：这条指令是将读取存储器 mem[15] 的后 8 位数，再经过有符号数扩展后存储进寄存器 reg[1] 中，由于初始化中 mem[15] 中存储的数值为 00000000000000000000000010000001, 后 8 位的符号位为 1，有符号数扩展后为

11111111111111111111111110000001, 与仿真结果符合。

## (2) lbu

测试所用的二进制指令为: 00000000111100000100000010000011

功能仿真所得结果为:

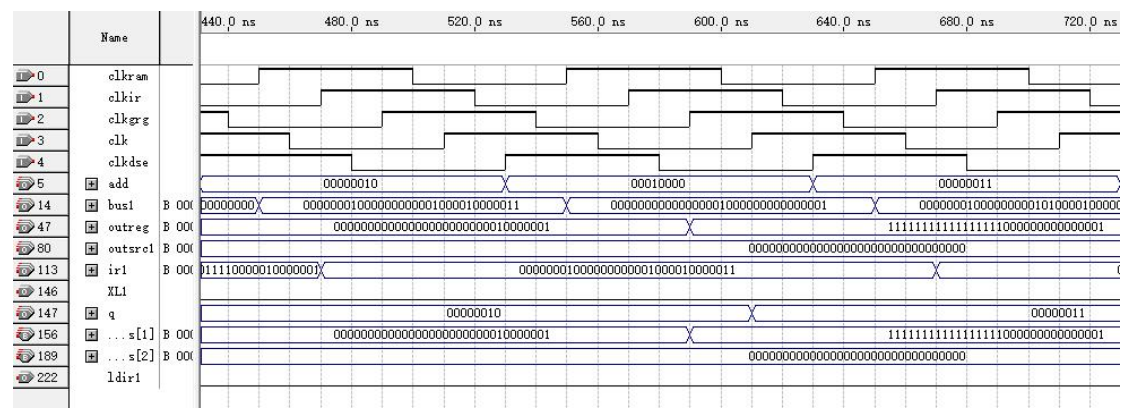


分析: 这条指令是将读取存储器 mem[15] 的后 8 位数, 再经过无符号数扩展后存储进寄存器 reg[1] 中, 由于初始化中 mem[15] 中存储的数值为 00000000000000000000000000000001, 因为无符号数扩展为在左边补零, 结果为 00000000000000000000000000000001, 与仿真结果符合。

## (3) lh

测试所用的二进制指令为: 0000000010000000000010000100000011

功能仿真所得结果为:

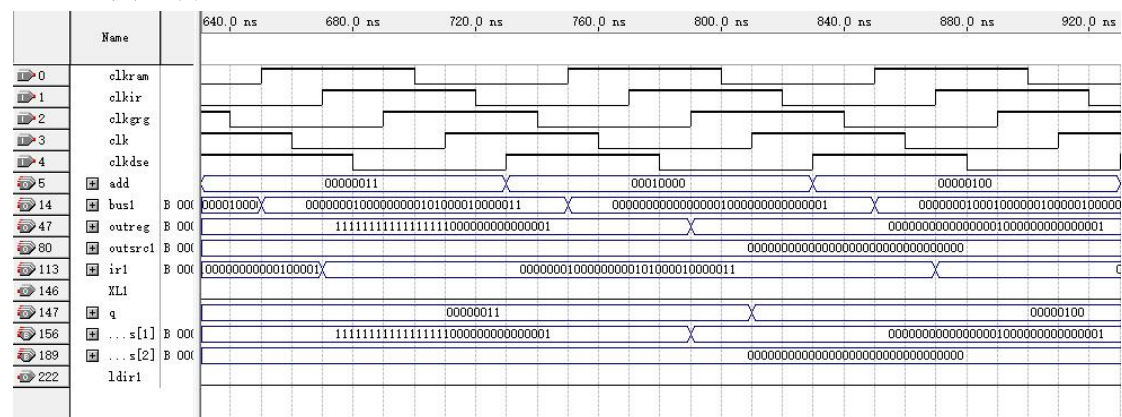


分析: 这条指令是将读取存储器 mem[16] 的后 16 位数, 再经过有符号数扩展后存储进寄存器 reg[1] 中, 由于初始化中 mem[16] 中存储的数值为 00000000000000000000000000000001, 后 16 位的符号位为 1, 有符号数扩展后为 111111111111111111110000000000000001, 与仿真结果符合。

## (4) lhu

测试所用的二进制指令为: 0000000010000000000101000010000011

功能仿真所得结果为：

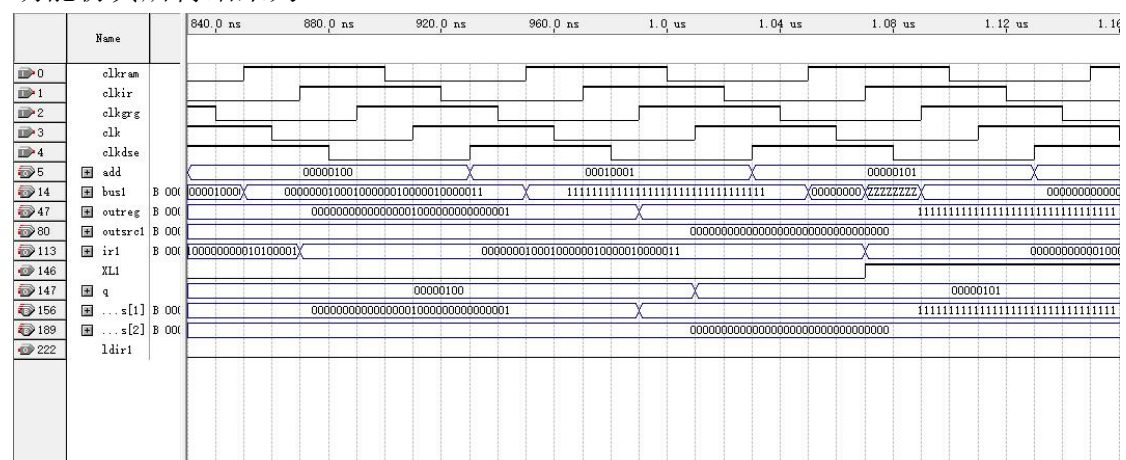


分析：这条指令是将读取存储器 mem[16] 的后 16 位数，再经过无符号数扩展后存储进寄存器 reg[1] 中，由于初始化中 mem[16] 中存储的数值为 00000000000000001000000000000001，因为无符号数扩展为在左边补零，结果为 00000000000000001000000000000001，与仿真结果符合。

#### (5) lw

测试所用的二进制指令为：00000001000100000010000010000011

功能仿真所得结果为：

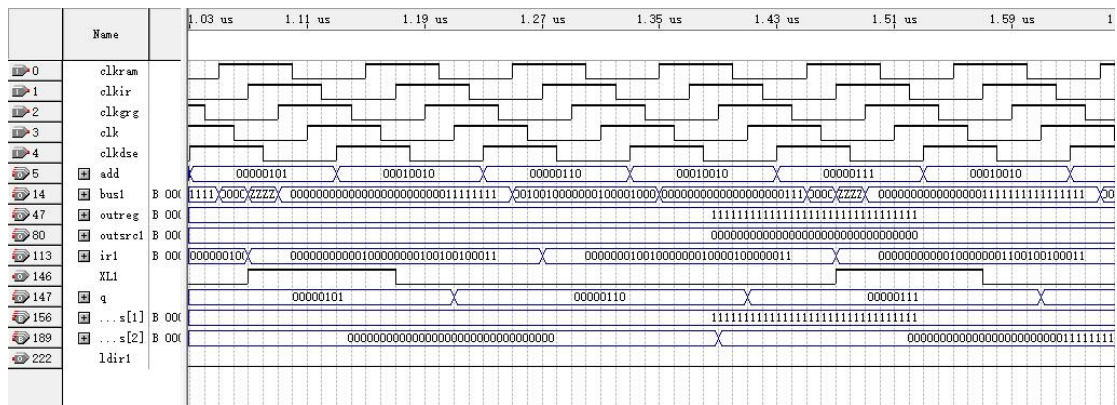


分析：这条指令是将读取存储器 mem[17] 的 32 位数，再存储进寄存器 reg[1] 中，由于初始化中 mem[17] 中存储的数值为 11111111111111111111111111111111，由图知仿真结果为 11111111111111111111111111111111，与仿真结果符合。

#### (6) sb

测试所用的二进制指令为：00000000000100000000100100100011

功能仿真所得结果为：

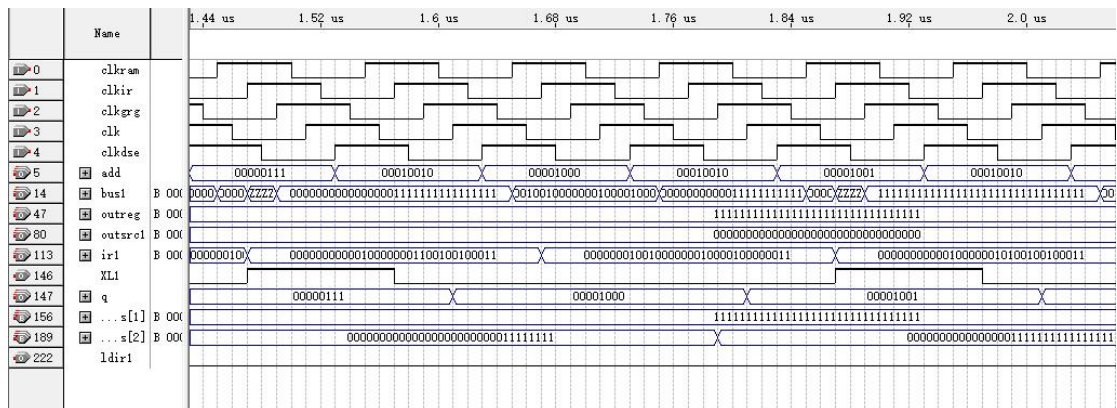


分析：这条指令是将读取寄存器 reg[1] 的 8 位数，经无符号数扩展后再存储进存储器 mem[18] 中，由于在 (5) 中 reg[1] 数值为 11111111111111111111111111111111，将其存入到 mem[18] 中后结果为 000000000000000000000000000000001111111111，接着再执行一条 lw 指令 (00000001001000000010000100000011)，其能读取 mem[18] 的数值并存入到 reg[2] 中，我们由图知指令执行完之后 reg[2] 中的数值为 000000000000000000000000000000001111111111，与理论结果符合。

#### (7) sh

测试所用的二进制指令为：00000000000100000001100100100011

功能仿真所得结果为：



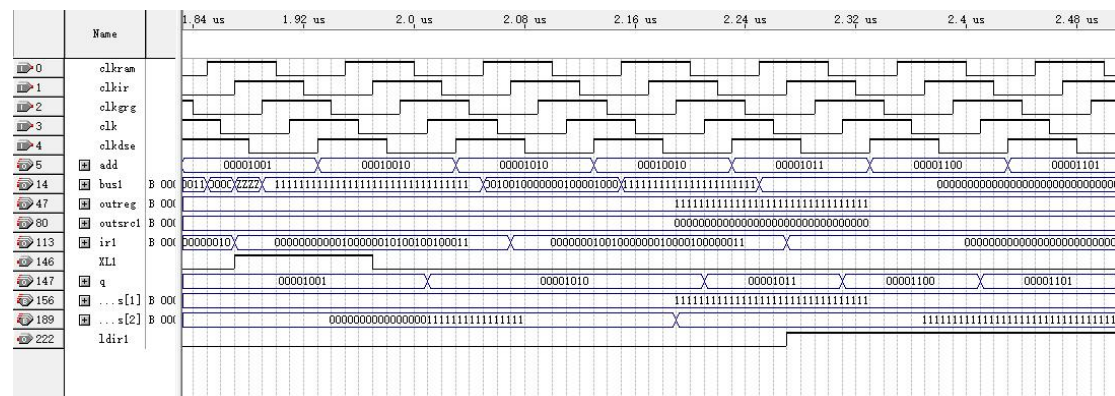
分析：这条指令是将读取寄存器 reg[1] 的 16 位数，经无符号数扩展后再存储进存储器 mem[18] 中，由于在 (5) 中 reg[1] 数值为 11111111111111111111111111111111，将其存入到 mem[18] 中后结果为 000000000000000000000000000000001111111111111111111，接着再执行一条 lw 指令 (00000001001000000010000100000011)，其能读取 mem[18] 的数值并存入到 reg[2] 中，我们由图知指令执行完之后 reg[2] 中的数值为 000000000000000000000000000000001111111111111111111，与理论结果符合。

#### (8) sw



测试所用的二进制指令为：00000000000100000010100100100011

功能仿真所得结果为：



分析：这条指令是将读取寄存器 reg[1] 的 32 位数，再存储进存储器 mem[18] 中，由于在 (5) 中 reg[1] 数值为 11111111111111111111111111111111，将其存入到 mem[18] 中后结果为 11111111111111111111111111111111，接着再执行一条 lw 指令(00000001001000000010000100000011)，其能读取 mem[18] 的数值并存入到 reg[2] 中，我们由图知指令执行完之后 reg[2] 中的数值为 11111111111111111111111111111111，与理论结果符合。

## 1、控制转移指令指令

在进行下列测试指令前，先执行 3 条 lw 指令，分别将 mem[17]、mem[18]、mem[19] 的数据存储到 reg[1]、reg[2]、reg[3] 中。而这三个数值如下

mem[17]:10000000000000000000000000000010

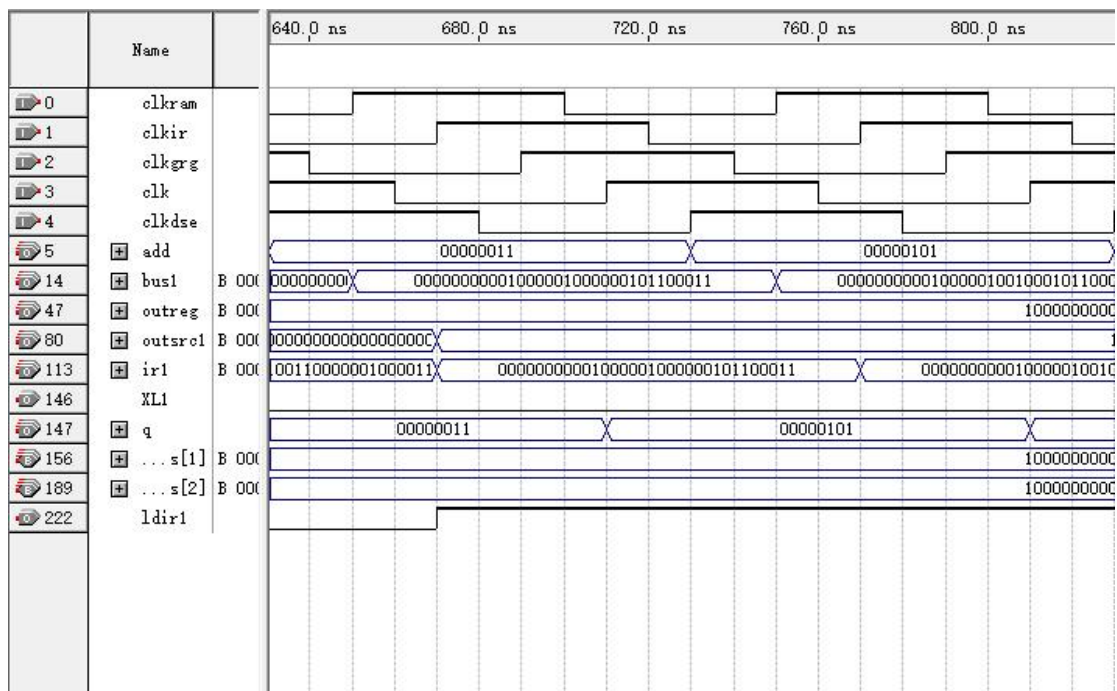
mem[18]:10000000000000000000000000000010

mem[19]:00000000000000000000000000000001

### (1) beq

测试所用的二进制指令为：000000000001000001000000101100011

功能仿真所得结果为：

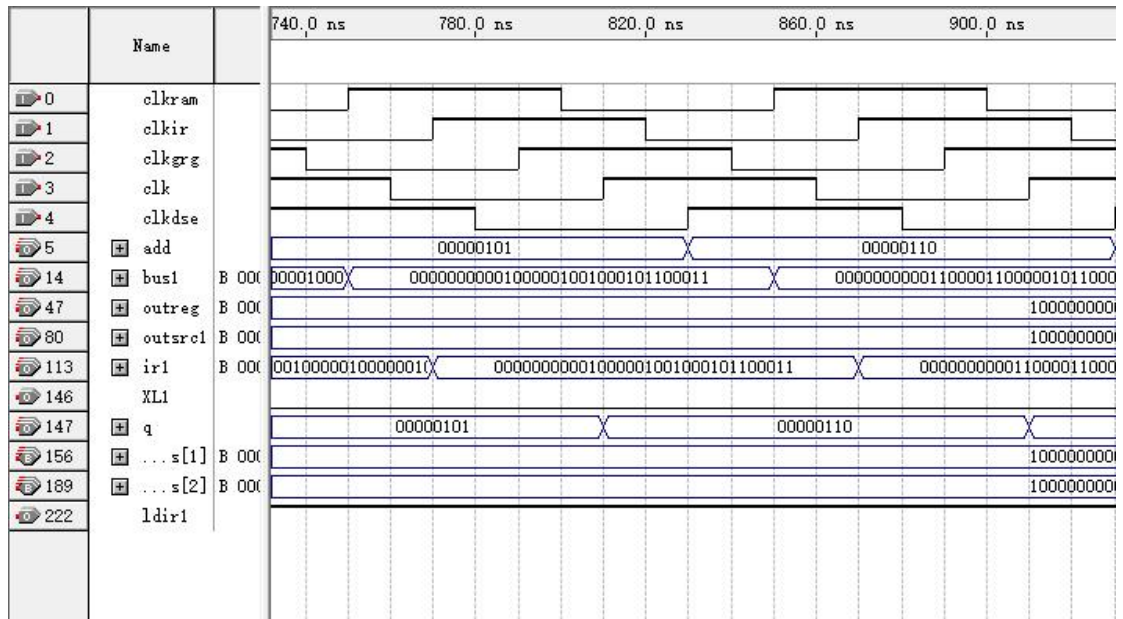


分析：这条指令是比较 reg[1]和 reg[2]的大小，当相等时跳转到指定位置，因为此时 reg[1]和 reg[2]的大小相等，所以跳转到地址为 5 的位置，由图知跳转到 00000101 处，与理论结果符合。

## (2) bne

测试所用的二进制指令为：00000000001000001001000101100011

功能仿真所得结果为：

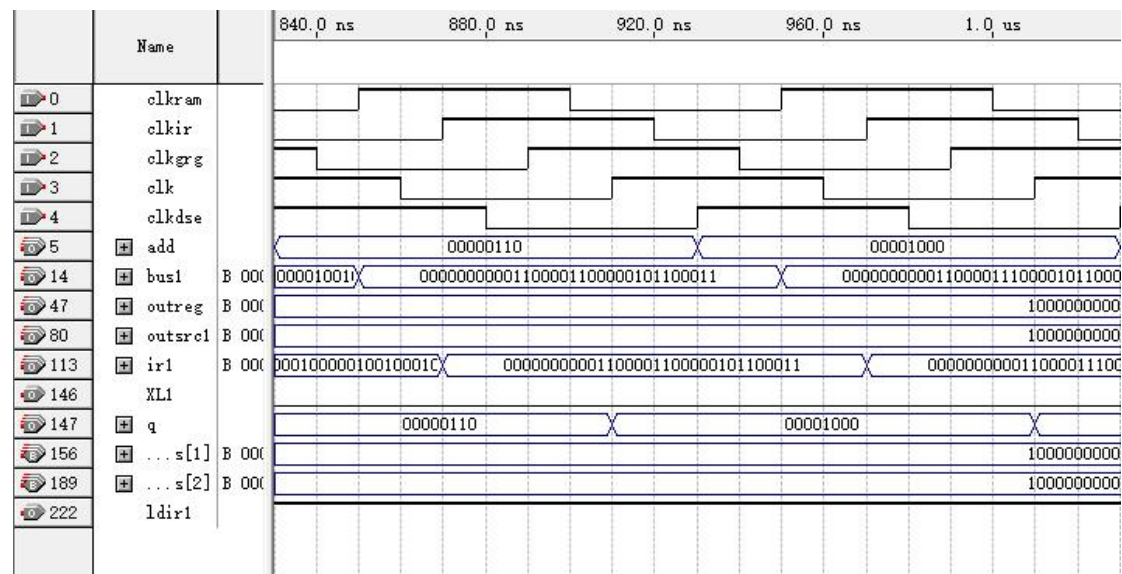


分析：这条指令是比较 reg[1]和 reg[2]的大小，当不等时跳转到指定位置，因为此时 reg[1]和 reg[2]的大小相等，所以不执行跳转，由图知下一个地址为 pc+1，与理论结果符合。

### (3) blt

测试所用的二进制指令为：00000000001100001100000101100011

功能仿真所得结果为：

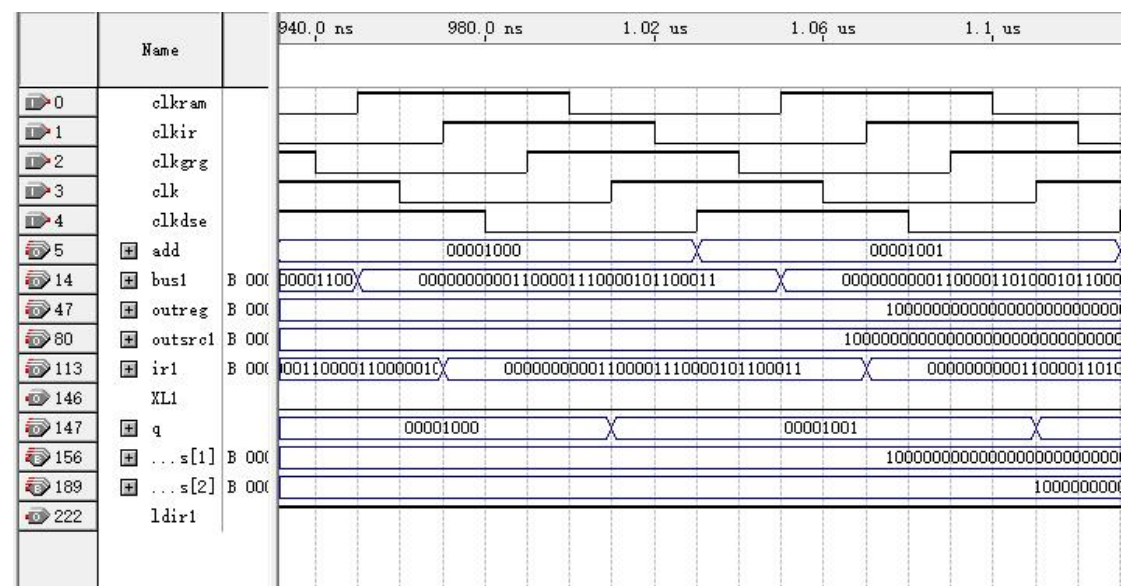


分析：这条指令是比较 reg[1] 和 reg[3] 有符号数的大小，当 reg[1] < reg[3] 时跳转到指定位置，因为此时 reg[1] 的符号位为 1，reg[3] 的符号位为 0，所符合条件，跳转到地址为 8 的位置，由图知下一个地址为 00001000，与理论结果符合。

### (4) bltu

测试所用的二进制指令为：00000000001100001110000101100011

功能仿真所得结果为：



分析：这条指令是比较 reg[1] 和 reg[3] 有符号数的大小，当 reg[1] < reg[3] 时跳转到指定位置，因为此时 reg[1] 的符号位为 1，reg[3] 的符号位为 0，所符合

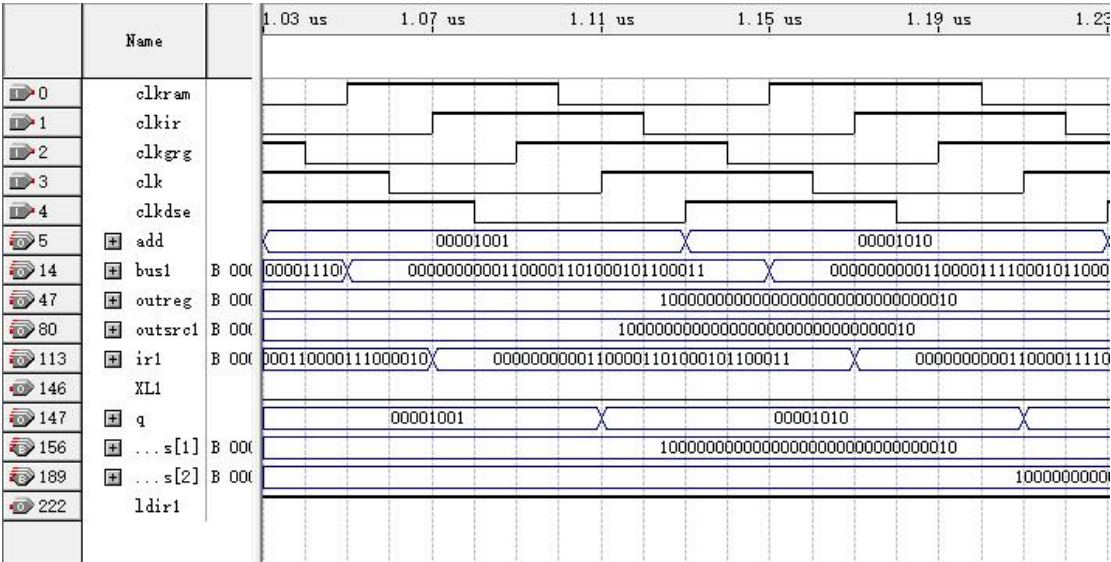


条件，跳转到地址为 8 的位置，由图知下一个地址为 00001000，与理论结果符合。

(5) bge

测试所用的二进制指令为：00000000001100001101000101100011

功能仿真所得结果为：

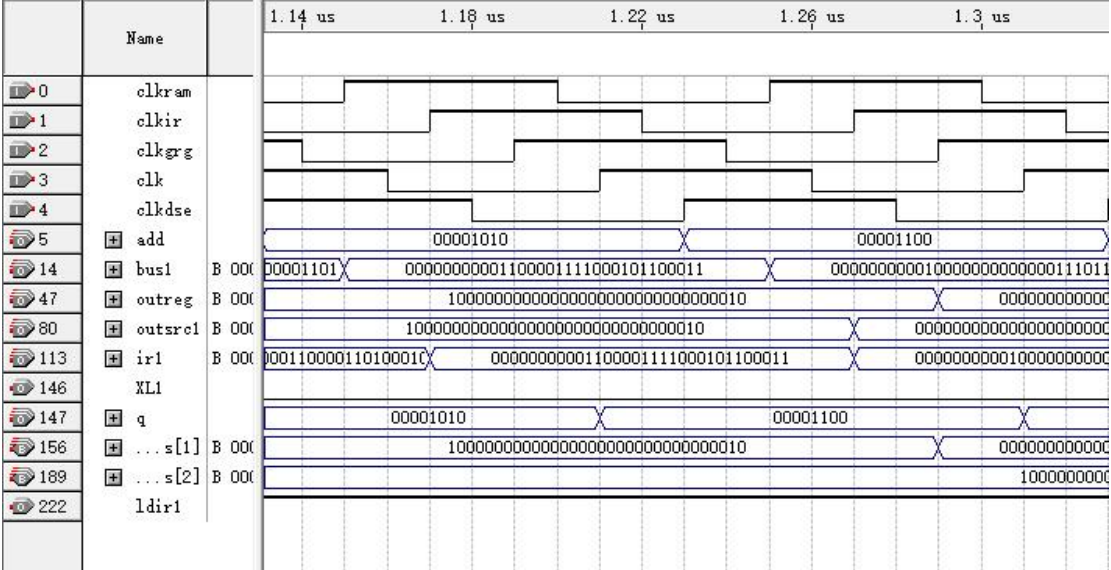


分析：这条指令是比较 reg[1]和 reg[3]有符号数的大小，当 reg[1]>reg[3]时跳转到指定位置，因为此时 reg[1]的符号位为 1，reg[3]的符号位为 0，不符合条件，不执行跳转，由图知下一个地址为 00001000，与理论结果符合。

(6) bgeu

测试所用的二进制指令为：00000000001100001111000101100011

功能仿真所得结果为：



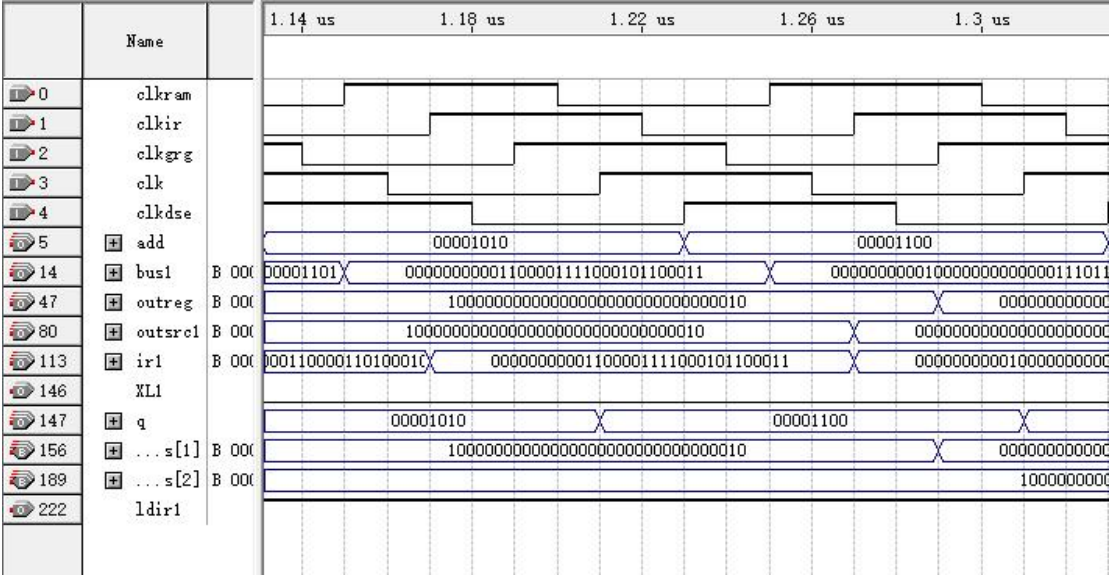
分析：这条指令是比较 reg[1]和 reg[3]无符号数的大小，当 reg[1]>reg[3]时

跳转到指定位置，因为此时  $\text{reg}[1] > \text{reg}[3]$ ，符合条件，跳转到为 12 的位置，由图知下一个地址为 00001100，与理论结果符合。

(7) bgeu

测试所用的二进制指令为：00000000001100001111000101100011

功能仿真所得结果为：

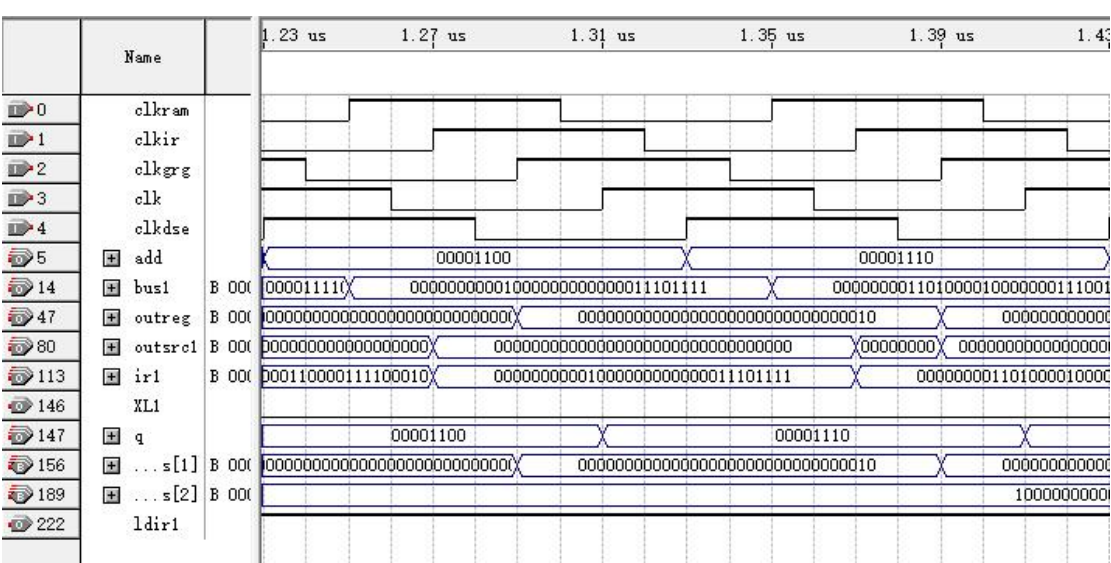


分析：这条指令是比较  $\text{reg}[1]$  和  $\text{reg}[3]$  无符号数的大小，当  $\text{reg}[1] > \text{reg}[3]$  时跳转到指定位置，因为此时  $\text{reg}[1] > \text{reg}[3]$ ，符合条件，跳转到为 12 的位置，由图知下一个地址为 00001100，与理论结果符合。

(8) jal

测试所用的二进制指令为：00000000001000000000000011101111

功能仿真所得结果为：



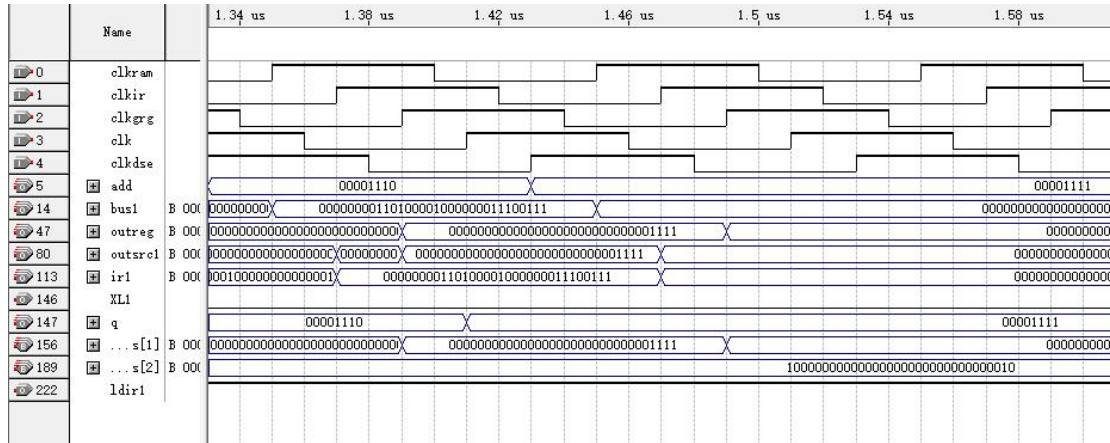
分析：这条指令跳转到 14( $\text{pc}+2$ ) 的位置，由图知下一个地址为 00001110，同时

也把 2 的等价 32 位二进制数存入 reg[1] 中，与理论结果符合。

(9) jalr

测试所用的二进制指令为：00000000110100001000000011100111

功能仿真所得结果为:

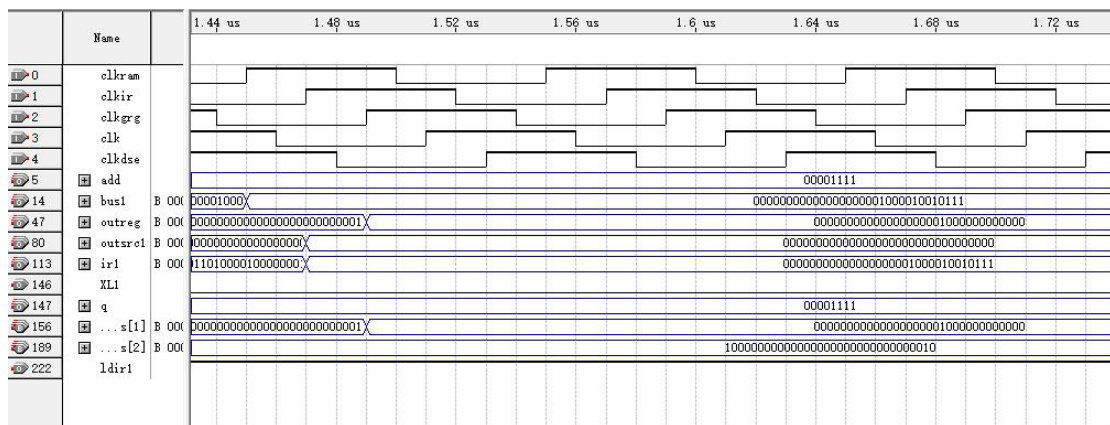


分析：这条指令跳转到  $15(\text{reg}[1]+13)$  的位置，由图知下一个地址为 00001111，与理论结果符合。

(10) auipc

测试所用的二进制指令为：000000000000000000001000010010111

功能仿真所得结果为:



分析：这条指令跳转到 15 (pc+0) 的位置，由图知以后的地址一直为 00001111，与理论结果符合。

### 实验总结:

通过上述的测试, Load/Store 指令和控制转移指令的仿真结果都是正确的。