湖南大學
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：翟俊璇

学　　号：201608030224

完成时间：2019 年 8 月 30 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/ ，我完成的指令为整数计算指令。

实验数据和分析：

实验代码：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
entity grg is
    port(
    clk: in std_logic;
    load: in std_logic;
    store: in std_logic;
    lui: in std_logic;
    auipc: in std_logic;
    ri: in std_logic;
    rr: in std_logic;
    jal: in std_logic;
    jalr: in std_logic;
    be : in std_logic;
    funct3: in std_logic_vector(2 downto 0);
```

```
    rs1: in std_logic_vector(4 downto 0);
    rs2: in std_logic_vector(4 downto 0);
    rd: in std_logic_vector(4 downto 0);
    mem: in std_logic_vector(31 downto 0);
    imm: in std_logic_vector(11 downto 0);
    imm1: in std_logic_vector(19 downto 0);
    q: in std_logic_vector(7 downto 0);
    src1: out std_logic_vector(31 downto 0);
    src2: out std_logic_vector(31 downto 0);
    outa: out std_logic_vector(31 downto 0);
    wrimem: out std_logic_vector(31 downto 0);
    jum: out std_logic_vector(7 downto 0);
    jud: out std_logic
    );
end grg;
architecture behav of grg is
    type regfile is array(31 downto 0) of std_logic_vector(31 downto 0);
    signal regs:regfile;
    signal lo:std_logic:='0';
    signal srcc1:std_logic_vector(31 downto 0);
    signal srcc2:std_logic_vector(31 downto 0);
    signal tem1:bit_vector(31 downto 0);
    signal count:integer range 0 to 31;
    signal count1:integer range 0 to 31;
    begin
         srcc1<= regs(to_integer(unsigned(rs1)));
        src1 <= srcc1;
        outa <= regs(1);
        srcc2<= regs(to_integer(unsigned(rs2)));
```

```
        src2 <= srcc2;
        tem1 <=to_bitvector(srcc1);
        count<=to_integer(unsigned(imm(4 downto 0)));
        count1 <= to_integer(unsigned(srcc2));
        process(clk)
        variable mem1:std_logic_vector(7 downto 0);
        variable tem:std_logic_vector(31 downto 0);
        variable mem2:std_logic_vector(15 downto 0);
        variable imm2:signed(31 downto 0);
        variable tem2:bit_vector(31 downto 0);
        variable tem3:std_logic_vector(7 downto 0);
        variable tem4:std_logic_vector(32 downto 0);
        variable tem5:std_logic_vector(32 downto 0);
        variable tem6:std_logic_vector(32 downto 0);
        begin
           if(clk'event and clk='1') then
               if(load='1') then
               case lo is
               when '0'=>
               lo<='1';
               when '1'=>
               lo<='0';
               end case;
               if(lo='1') then
               case funct3 is
               when "000"=>--LB
               mem1:=mem(7 downto 0);
               tem:=std_logic_vector(resize(signed(mem1),
tem'LENGTH));
```

```
                regs(to_integer(unsigned(rd)))<=tem;
                when "001"=>--LH
                mem2:=mem(15 downto 0);
                tem:=std_logic_vector(resize(signed(mem2),
tem'LENGTH));
                regs(to_integer(unsigned(rd)))<=tem;
                when "010"=>--LW
                regs(to_integer(unsigned(rd)))<=mem;
                when "100"=>--LBU
                mem1:=mem(7 downto 0);

regs(to_integer(unsigned(rd)))<="00000000000000000000000"&mem1;
                when "101"=>--LHU
                mem2:=mem(15 downto 0);

regs(to_integer(unsigned(rd)))<="0000000000000000"&mem2;
                when others=>
                end case;
                end if;
                elsif(store='1') then
                case lo is
                when '0'=>
                lo<='1';
                when '1'=>
                lo<='0';
                end case;
                if(lo='0') then
                case funct3 is
                when "000"=>--SB
```

```
mem1:=srcc2(7 downto 0);
wrimem<="000000000000000000000000"&mem1;
when "001"=>--SH
mem2:=srcc2(15 downto 0);
wrimem<="0000000000000000"&mem2;
when "010"=>--SW
wrimem<=srcc2;
when others=>
wrimem<=(others=>'Z');
end case;
else
wrimem<=(others=>'Z');
end if;
elsif(lui='1') then--LUI
regs(to_integer(unsigned(rd)))<=imm1&"000000000000";
elsif(auipc='1') then--AUIPC
tem:=imm1&"000000000000";
regs(to_integer(unsigned(rd)))<=imm1&"000000000000";
jum<=tem(7 downto 0);
elsif(ri='1') then
case funct3 is
when "000"=>--ADDI
regs(to_integer(unsigned(rd)))<=(imm+srcc1);
when "010"=>--SLTI
imm2:=resize(signed(imm), imm2'length);
 if(signed(srcc1)<imm2) then
 regs(to_integer(unsigned(rd)))<=(others=>'1');
 else
 regs(to_integer(unsigned(rd)))<=(others=>'0');
```

```
                end if;
                when "011"=>--SLTIU
                if(srcc1<"00000000000000000000"&imm) then
                regs(to_integer(unsigned(rd)))<=(others=>'1');
                else
                regs(to_integer(unsigned(rd)))<=(others=>'0');
                end if;
                when "100"=>--XORI
                imm2:=resize(signed(imm), imm2'length);
                regs(to_integer(unsigned(rd)))<=(srcc1 xor
std_logic_vector(imm2));
                when "110"=>--ORI
                imm2:=resize(signed(imm), imm2'length);
                regs(to_integer(unsigned(rd)))<=(srcc1 or
std_logic_vector(imm2));
                when "111"=>--ANDI
                imm2:=resize(signed(imm), imm2'length);
                regs(to_integer(unsigned(rd)))<=(srcc1 and
std_logic_vector(imm2));
                when "001"=>--SLLI
                tem2:=tem1 sll count;

regs(to_integer(unsigned(rd)))<=to_stdlogicvector(tem2);
                when "101"=>
                if(imm(11 downto 5)="0000000") then--SRLI
                tem2:=tem1 srl count;

regs(to_integer(unsigned(rd)))<=to_stdlogicvector(tem2);
                else--SRAI
```

```
                tem2:=tem1 sra count;

regs(to_integer(unsigned(rd)))<=to_stdlogicvector(tem2);
                end if;
                end case;
                elsif(rr='1') then
               case funct3 is
               when "000"=>
               if(imm(11 downto 5)="0000000") then--ADD
               tem4:=std_logic_vector(resize(signed(srcc1),
tem4'LENGTH));
                tem5:=std_logic_vector(resize(signed(srcc2),
tem5'LENGTH));
               tem6:=std_logic_vector(signed(tem4)+signed(tem5))(32
downto 0);
               if(tem6(32)='1' and tem6(31)='0') then

regs(to_integer(unsigned(rd)))<="10000000000000000000000000000000";
               elsif(tem6(32)='0' and tem6(31)='1') then

regs(to_integer(unsigned(rd)))<="01111111111111111111111111111111";
               else
               regs(to_integer(unsigned(rd)))<=tem6(31 downto 0);
               end if;
               else--SUB
               tem4:=std_logic_vector(resize(signed(srcc1),
tem4'LENGTH));
                tem5:=std_logic_vector(resize(signed(srcc2),
tem5'LENGTH));
```

```
                tem6:=std_logic_vector(signed(tem4)-signed(tem5))(32
downto 0);
                if(tem6(32)='1' and tem6(31)='0') then

regs(to_integer(unsigned(rd)))<="10000000000000000000000000000000";
                elsif(tem6(32)='0' and tem6(31)='1') then

regs(to_integer(unsigned(rd)))<="01111111111111111111111111111111";
                else
                regs(to_integer(unsigned(rd)))<=tem6(31 downto 0);
                end if;
                end if;
                when "001"=>--SLL
                tem2:=tem1 sll count1;

regs(to_integer(unsigned(rd)))<=to_stdlogicvector(tem2);
                 when "010"=>--SLT
                 if(signed(srcc1)<signed(srcc2)) then
                 regs(to_integer(unsigned(rd)))<=(others=>'1');
                 else
                 regs(to_integer(unsigned(rd)))<=(others=>'0');
                 end if;
                 when "011"=>--SLTU
                 if(srcc1<srcc2) then
                 regs(to_integer(unsigned(rd)))<=(others=>'1');
                 else
                 regs(to_integer(unsigned(rd)))<=(others=>'0');
                 end if;
                 when "100"=>--XOR
```

```
                regs(to_integer(unsigned(rd)))<=(srcc1 xor srcc2);
                when "101"=>
                if(imm(11 downto 5)="0000000") then--SRL
                tem2:=tem1 srl count1;

regs(to_integer(unsigned(rd)))<=to_stdlogicvector(tem2);
              else--SRA
              tem2:=tem1 sra count1;

regs(to_integer(unsigned(rd)))<=to_stdlogicvector(tem2);
              end if;
              when "110"=>--OR
              regs(to_integer(unsigned(rd)))<=(srcc1 or srcc2);
              when "111"=>--AND
                regs(to_integer(unsigned(rd)))<=(srcc1 and srcc2);
                end case;
                elsif(jal='1') then--JAL
              imm2:=resize(signed(imm1), imm2'length);
              tem3:=std_logic_vector(imm2*2)(7 downto 0);
              jum<=tem3;

regs(to_integer(unsigned(rd)))<="00000000000000000000000"&tem3;
              elsif(jalr='1') then--JALR
              imm2:=resize(signed(imm), imm2'length);
              tem3:=std_logic_vector(signed(imm2)+signed(srcc1))(7
downto 0);
              jum<=tem3;

regs(to_integer(unsigned(rd)))<="00000000000000000000000"&tem3;
```

```
elsif(be='1') then
case funct3 is
when "000"=>--BEQ
if(srcc1=srcc2) then
imm2:=resize(signed(imm), imm2'length);
jum<=std_logic_vector(imm2*2)(7 downto 0);
jud<='1';
else
jud<='0';
end if;
when "001"=>--BNE
if(srcc1/=srcc2) then
imm2:=resize(signed(imm), imm2'length);
jum<=std_logic_vector(imm2*2)(7 downto 0);
jud<='1';
else
jud<='0';
end if;
when "100"=>--BLT
if(signed(srcc1)<signed(srcc2)) then
imm2:=resize(signed(imm), imm2'length);
jum<=std_logic_vector(imm2*2)(7 downto 0);
jud<='1';
else
jud<='0';
end if;
when "101"=>--BGE
if(signed(srcc1)>signed(srcc2)) then
imm2:=resize(signed(imm), imm2'length);
```

```
                jum<=std_logic_vector(imm2*2)(7 downto 0);
                jud<='1';
                else
                jud<='0';
                end if;
                when "110"=>--BLTU
                if(srcc1<srcc2) then
                imm2:=resize(signed(imm), imm2'length);
                jum<=std_logic_vector(imm2*2)(7 downto 0);
                jud<='1';
                else
                jud<='0';
                end if;
                when "111"=>--BGEU
                if(srcc1>srcc2) then
                imm2:=resize(signed(imm), imm2'length);
                jum<=std_logic_vector(imm2*2)(7 downto 0);
                jud<='1';
                else
                jud<='0';
                end if;
                when others=>
                end case;
                end if;
            end if;
        end process;
end behav;
```

测试环境：

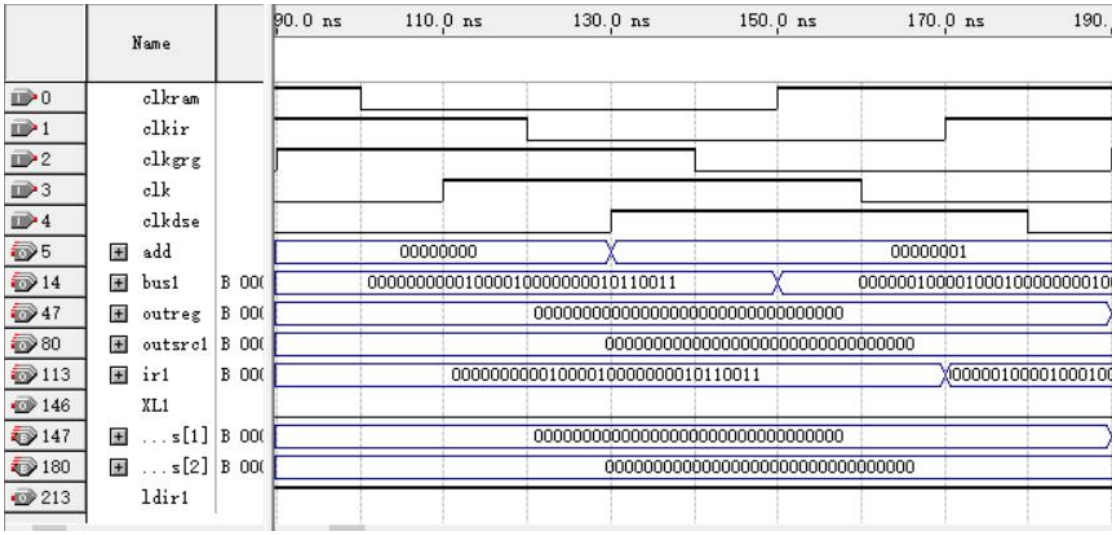
部件	配置	备注
CPU	i5-6300HQ	
内存	8GB	
操作系统	Windows 10 1903	中文版
仿真软件	quartus 9.0	

测试结果：

（1）add

①测试所用的二进制指令为：00000000001000010000000010110011

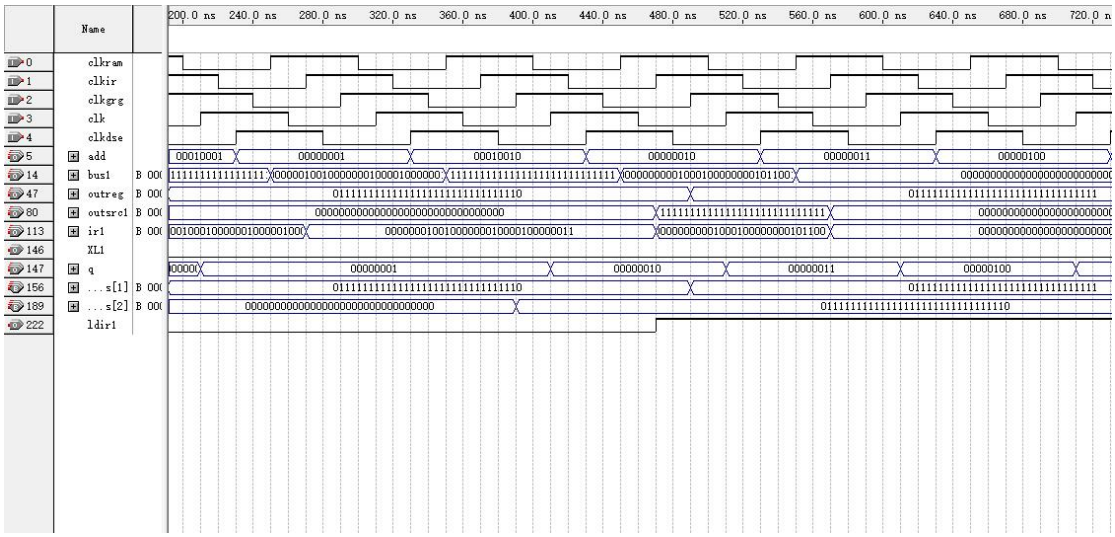
功能仿真所得结果为：



分析：这条指令是将寄存器 2 的数与自身相加存入寄存器 1 中，由于寄存器 2 中未写入数值，相加结果为 0，寄存器 1 显示为 0。

②测试用二进制指令为：00000000000100010000000010110011

功能仿真结果为：

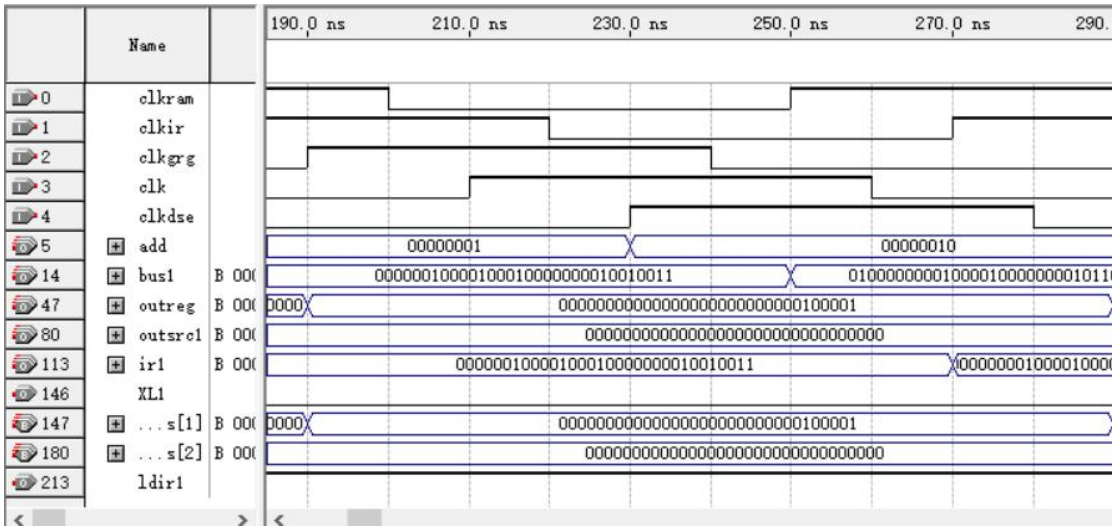


分析：首先在寄存器 1 与 2 中都写入 01111111111111111111111111111110，故当寄存器 1 的值与寄存器 2 的值相加时，加法溢出。结果显示此时加法已溢出，并摒弃了最高位，该指令正确执行。

（2）addi

测试所用的二进制指令为：00000010000100010000000010010011

功能仿真所得结果为：

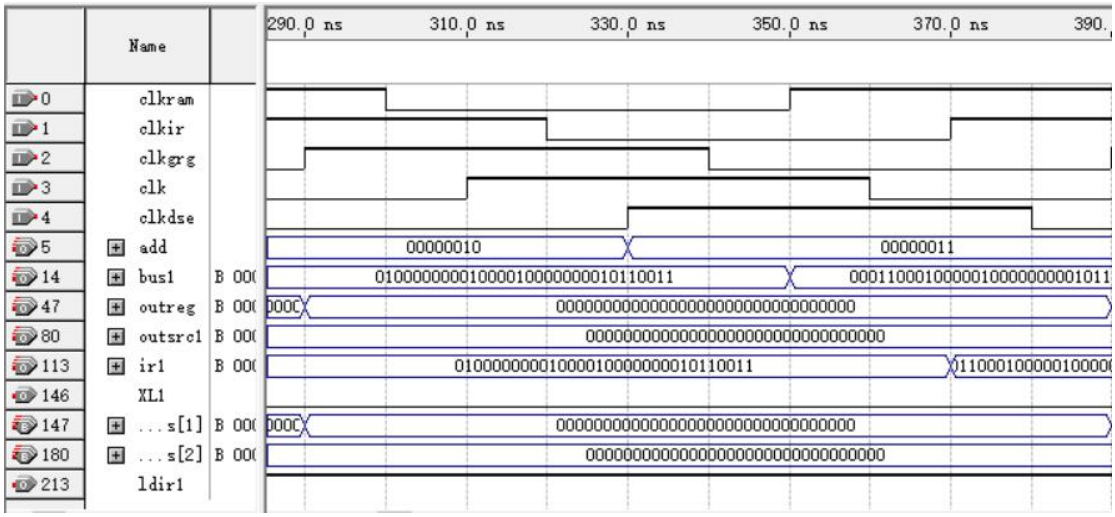


分析：该指令的意思是将指令的前 12 位（也就是立即数）与寄存器 2 中存储的数值相加，所得结果存入寄存器 1 中。由于寄存器 2 中未写入数值，故寄存器 1 中显示的数值为该立即数的数值，该指令正确执行。

（3）sub

①测试所用的二进制指令为：01000000001000010000000010110011

功能仿真结果为：



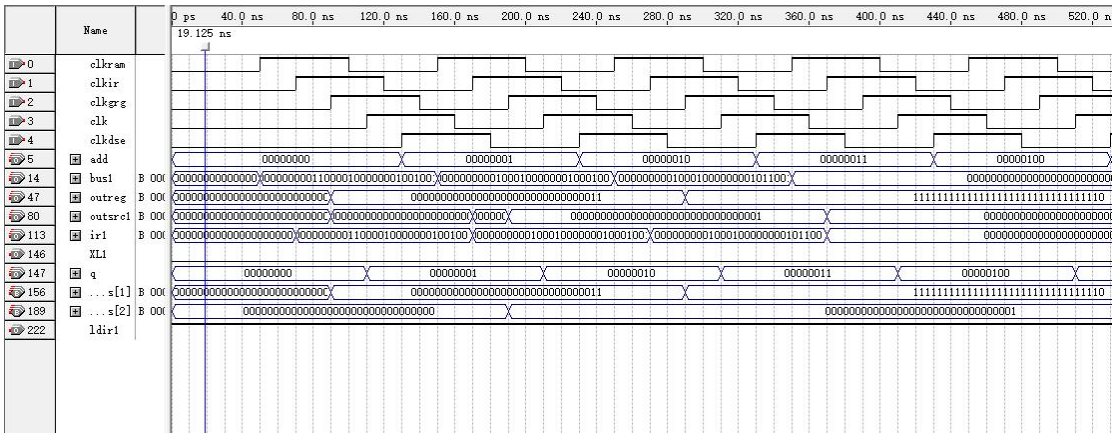
分析：该指令将寄存器 1 的值与寄存器 2 的值相减，所得结果存入寄存器 1 中。由于两个寄存器中的值均为 0，故寄存器 1 显示的值为 0。

②测试用二进制指令为：00000000001100001000000010010011

00000000000100010000000100010011

01000000000100010000000010110011

功能仿真结果为：

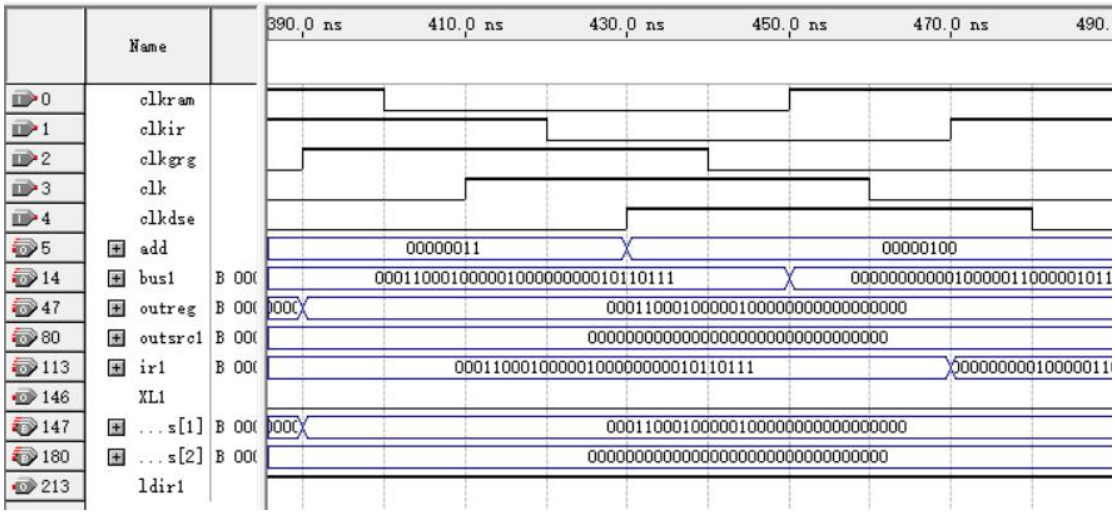


分析：首先给寄存器 1 赋值 3，给寄存器 2 赋值 1，再使用 sub 指令使寄存器 2 的值减寄存器 1 的值的结果存入寄存器 1 中，因此减法计算是溢出的，高位为 1，所得结果为-2，故该指令正确执行。

（4）lui

测试用二进制指令为：00011000100000100000000010110111

功能仿真结果为：

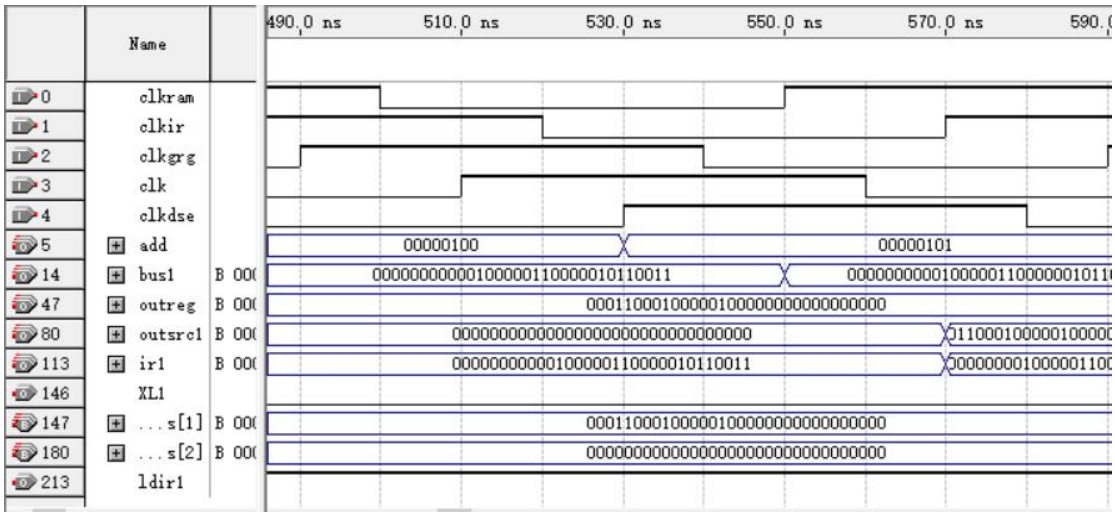


分析：这条指令是将 20 位立即数移至寄存器 1 的前 20 位，而寄存器后 12 位置 0。从结果可知，此时立即数已向前移位 12 位，后 12 位置 0，该指令正确执行。

（5）xor

测试用二进制指令为：00000000000100000110000010110011

功能仿真结果为：

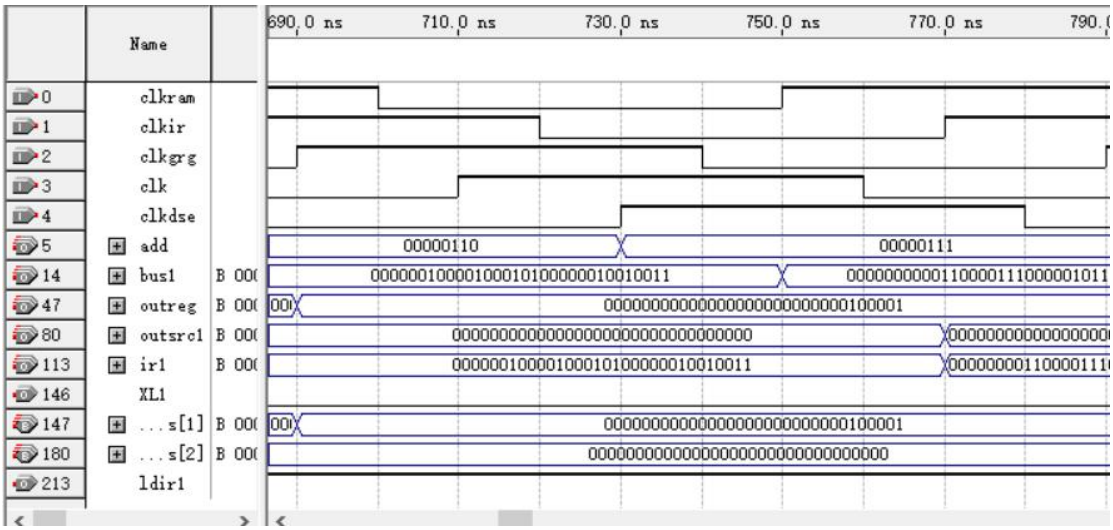


分析：这条指令的意思是将寄存器 1 中的值与寄存器 2 的值进行异或运算，所得结果存入寄存器 1 中。由于寄存器 2 中未写入值，故异或结果为寄存器 1 中的值，即 lui 指令执行之后的结果，该指令正确执行。

（6）xori

测试用二进制指令为：00000010000100010100000010010011

功能仿真结果为：

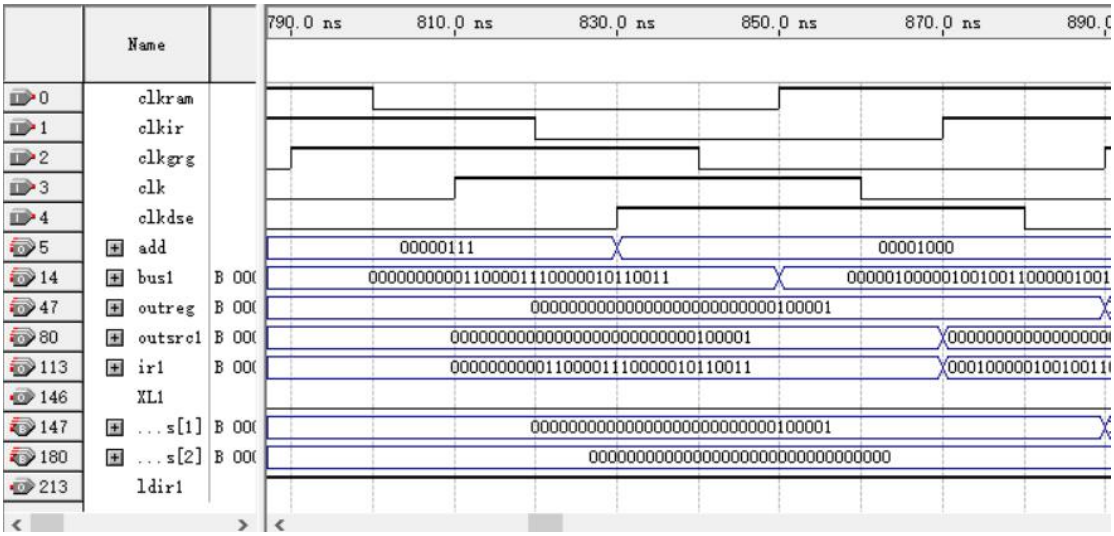


分析：该指令是将寄存器 2 与立即数进行异或运算，并将结果存入寄存器 1 中。由于寄存器 2 的值为 0，故结果为立即数的值，该指令正确执行。

（7）or

测试用二进制指令为：00000000001100001110000010110011

功能仿真结果为：

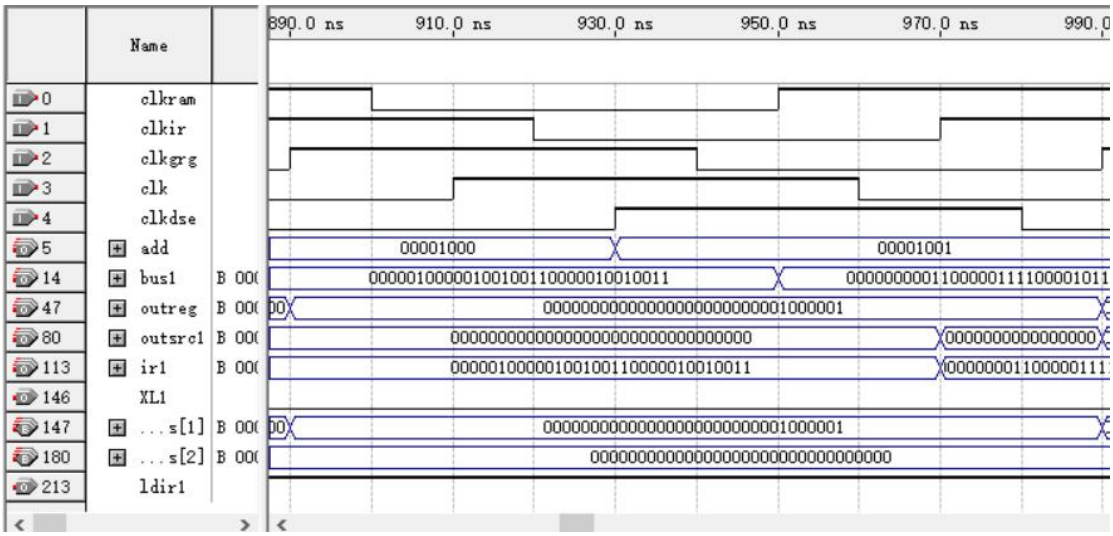


分析：该指令是将寄存器 1 的值与寄存器 3 的值进行或运算，所得结果存入寄存器 1 中。由于寄存器 3 中的值为 0，故显示结果为寄存器 1 中的值，即 xori 指令执行结果，该指令正确执行。

（8）ori

测试用二进制指令为：00000100000100100110000010010011

功能仿真结果为：

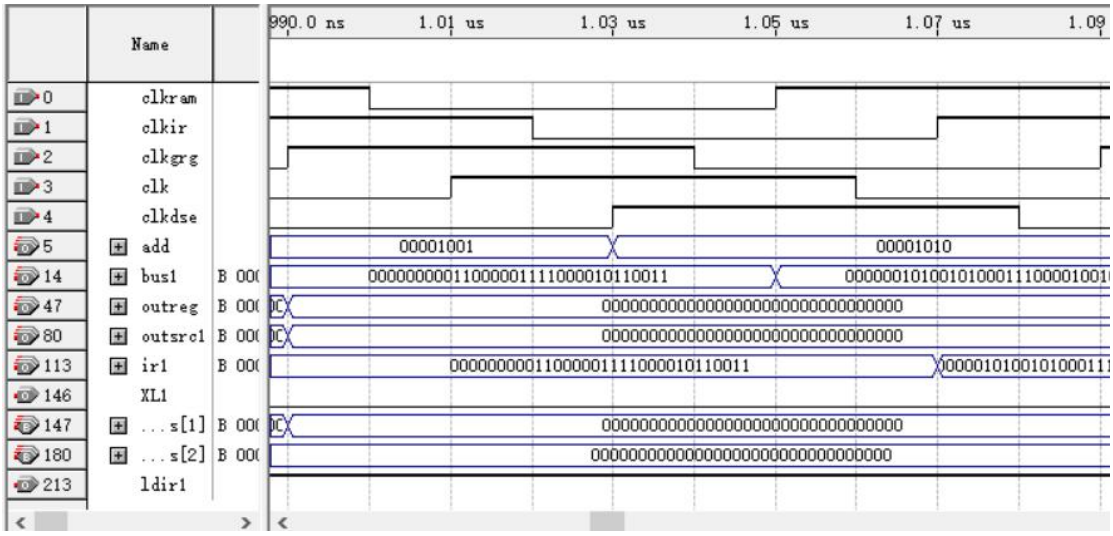


分析：该指令是将寄存器 4 中的值与立即数进行 or 运算，所得结果存入寄存器 1 中。由于寄存器 4 中未写入值，故所得结果为立即数的值，该指令正确执行。

（9）and

测试用二进制指令为：00000000011000001111000010110011

功能仿真结果为：

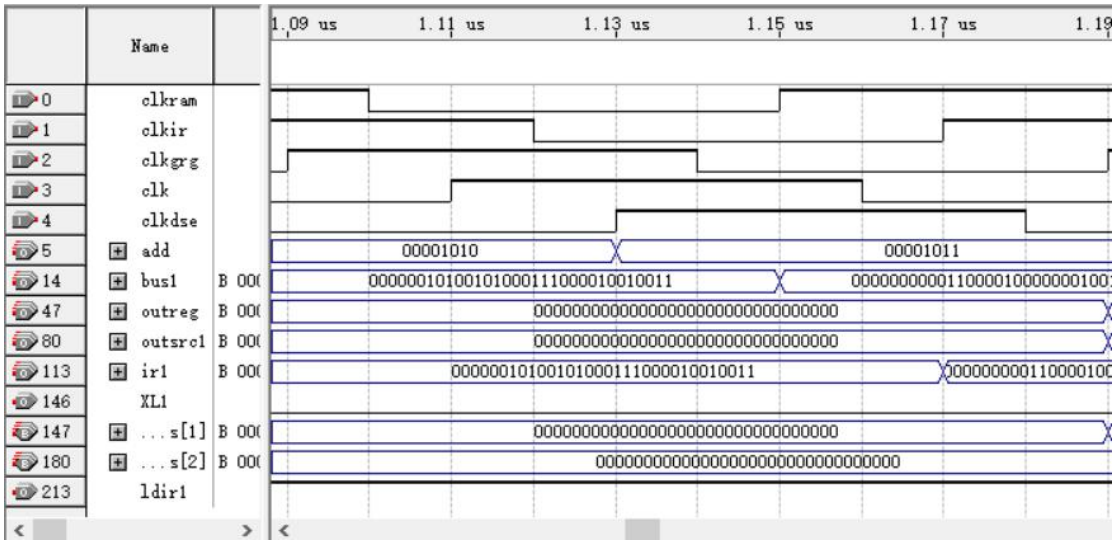


分析：该指令是将寄存器 6 与寄存器 15 的值进行 and 运算，由于这两个寄存器中未写入数值，故 and 结果为 0，该指令正确执行。

（10）andi

测试用二进制指令为：00000010100101000111000010010011

功能仿真结果为：



分析：该指令是将寄存器 8 的值与立即数进行 and 预算，所得结果存入寄存器 1 中。由于寄存器 8 中值为 0，故进行 and 运算后所得结果为 0，该指令正确执行。

（11）sll

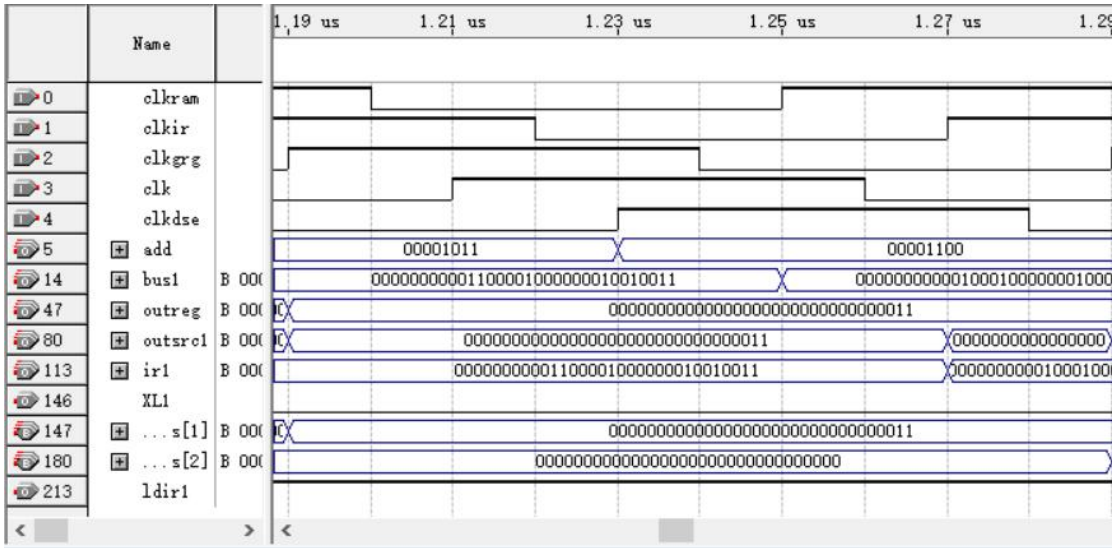
测试用二进制指令为：00000000001100001000000010010011

00000000000100010000000100010011

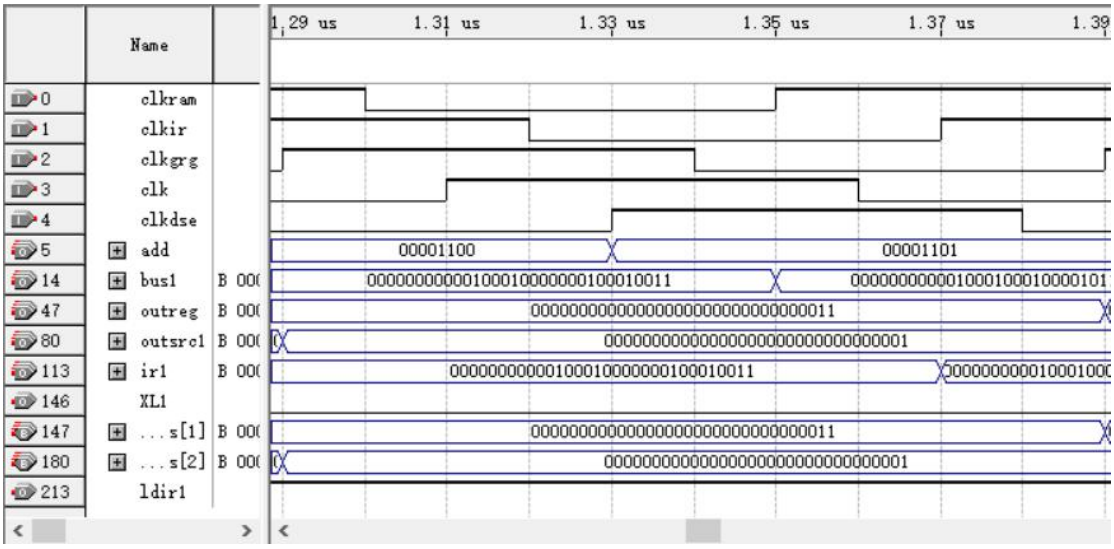
00000000000100010001000010110011

功能仿真结果为：

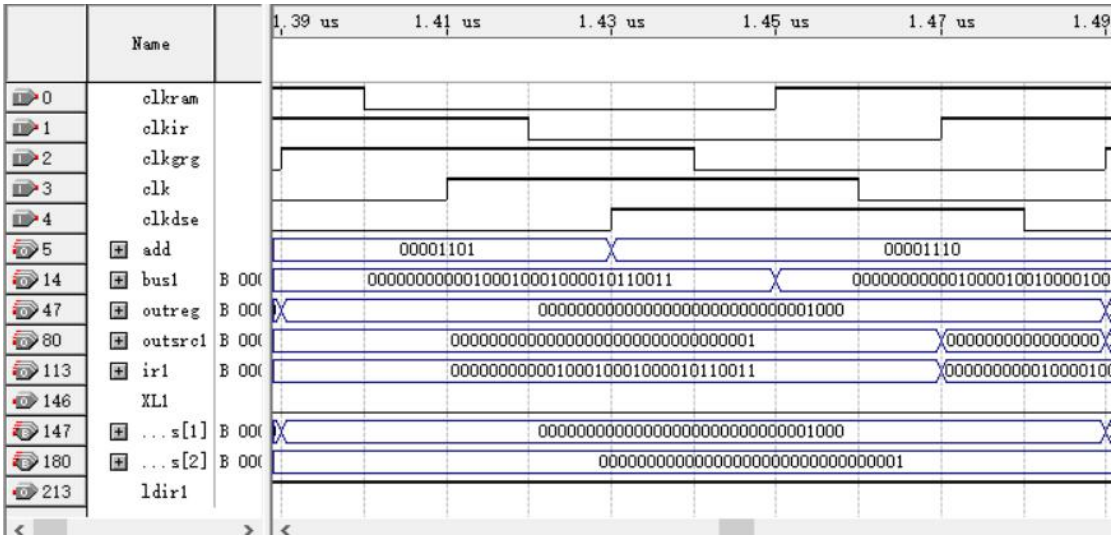
寄存器 1 赋值：



寄存器 2 赋值：



左移：

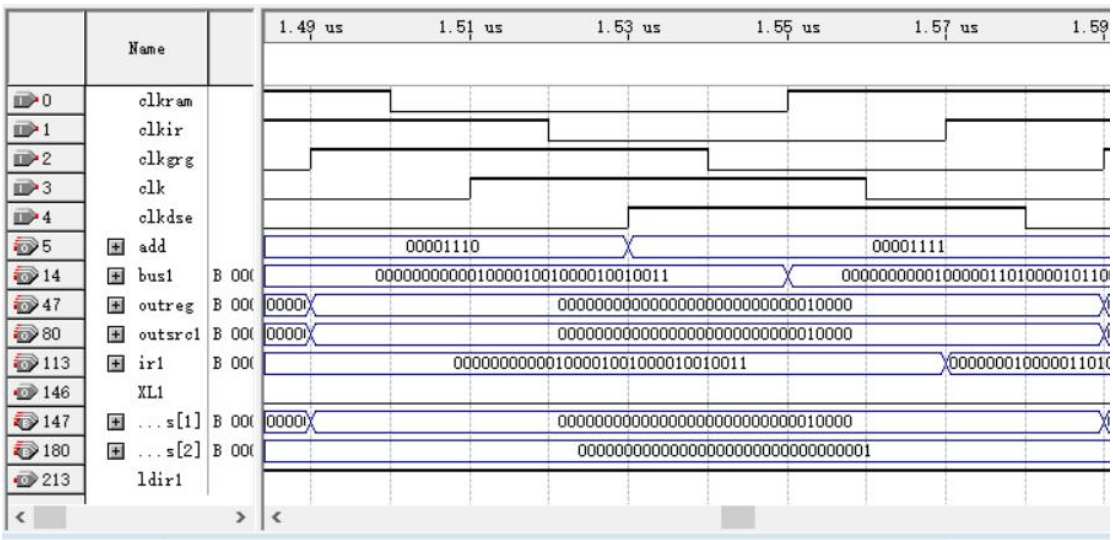


分析：这三条指令首先对寄存器 1 与寄存器 2 进行赋值，然后在将寄存器 2 的值左移寄存器 1 中的值，并将结果存入寄存器 1 中。寄存器 2 中的值为 1，左移三位之后为 8，该指令正确执行。

（12）slli

测试用二进制指令为：00000000000100001001000010110011

功能仿真结果为：

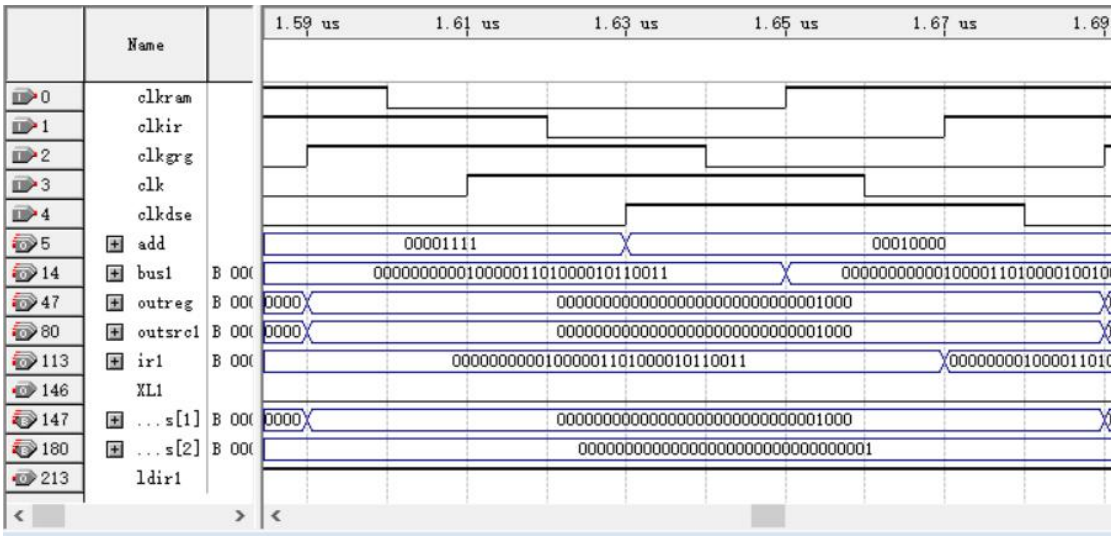


分析：该指令是将寄存器 1 中的值左移 1（立即数）位。寄存器 1 中的值为 8，左移 1 位之后为 16，故该指令正确执行。

（13）srl

测试用二进制指令为：00000000001000001101000010110011

功能仿真结果为：

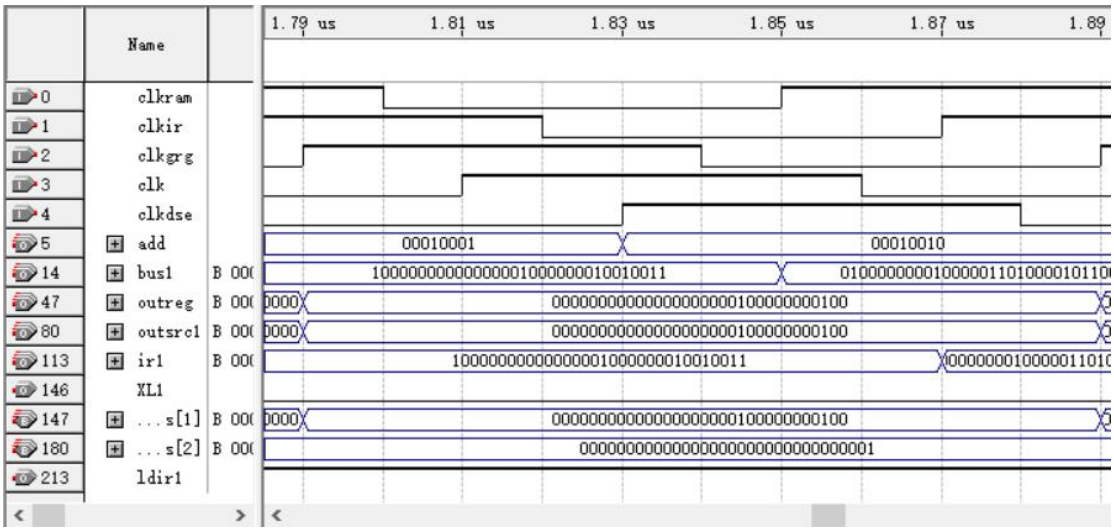


分析：该指令是将寄存器 1 中的值右移寄存器 2 中的值。由于寄存器 1 中的值为 16，寄存器 2 中的值为 1，故右移之后寄存器 1 中的值为 8，该指令正确执行。

（14）srli

测试用二进制指令为：00000000000100001101000010010011

功能仿真结果为：



分析：该指令是将寄存器 1 中的值右移 1（立即数）位。由于寄存器 1 中的值为 8，故右移之后为 4，该指令正确执行。

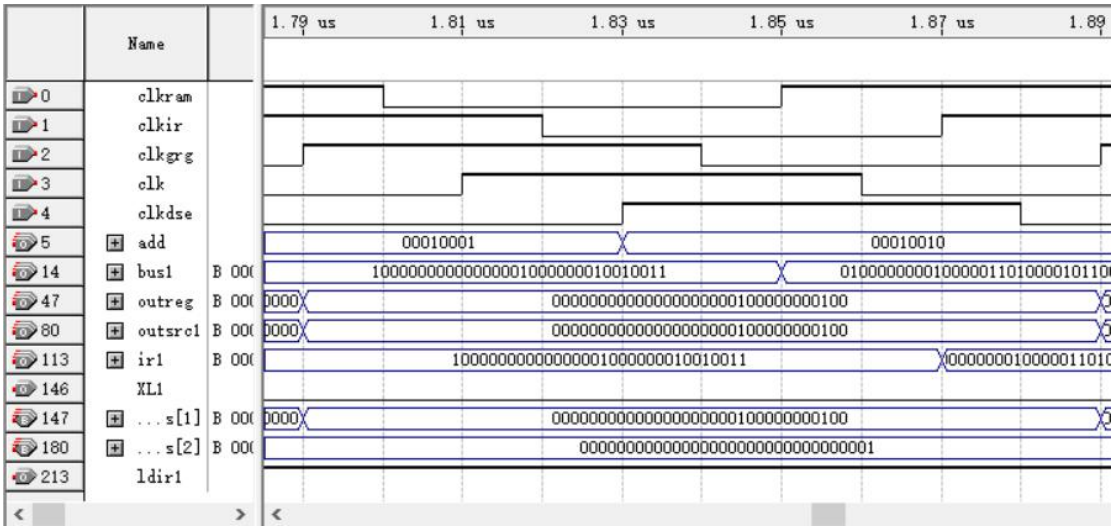
（15）sra

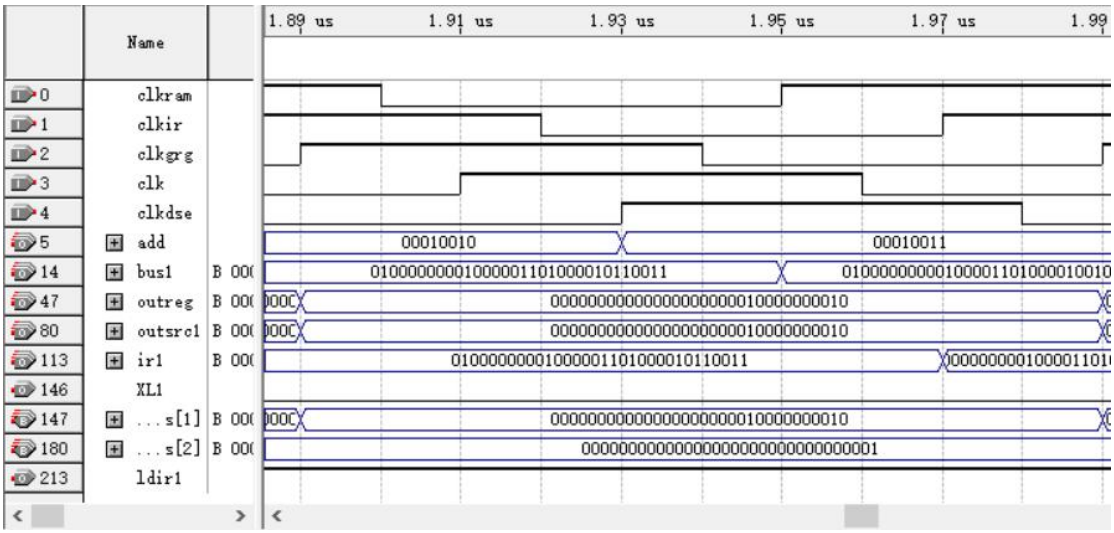
测试用二进制指令为：10000000000000001000000010010011

01000000001000001101000010110011

功能仿真结果为：

增大寄存器 1 中的数：



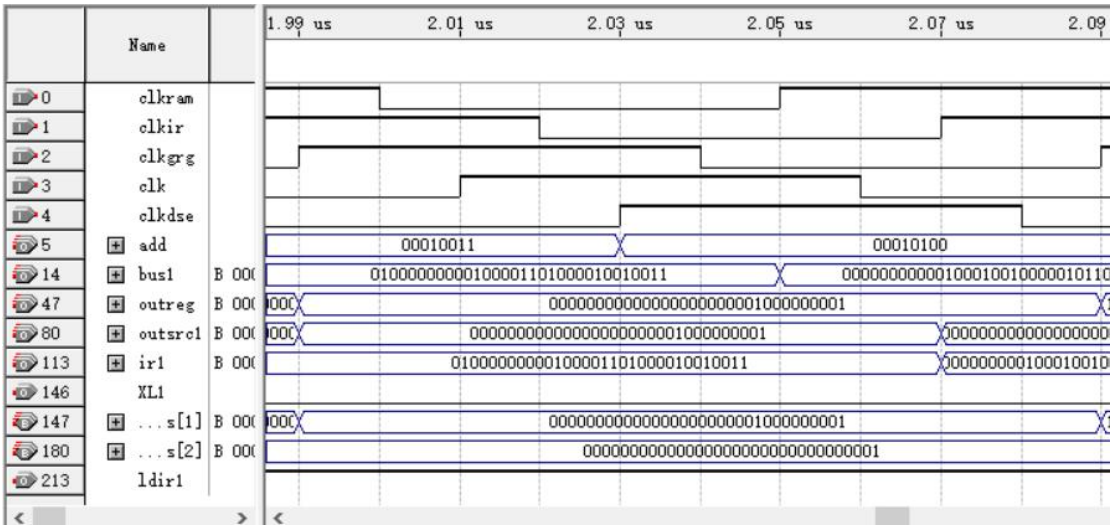


分析：这两条指令首先增大寄存器 1 的数值，然后将寄存器 1 的值右移 1 位（即寄存器 2 的值）。由于寄存器 1 的最高位为 0，故右移之后以 0 填充，该指令正确执行。

（16）srai

测试用二进制指令为：01000000000100001101000010010011

功能仿真结果为：

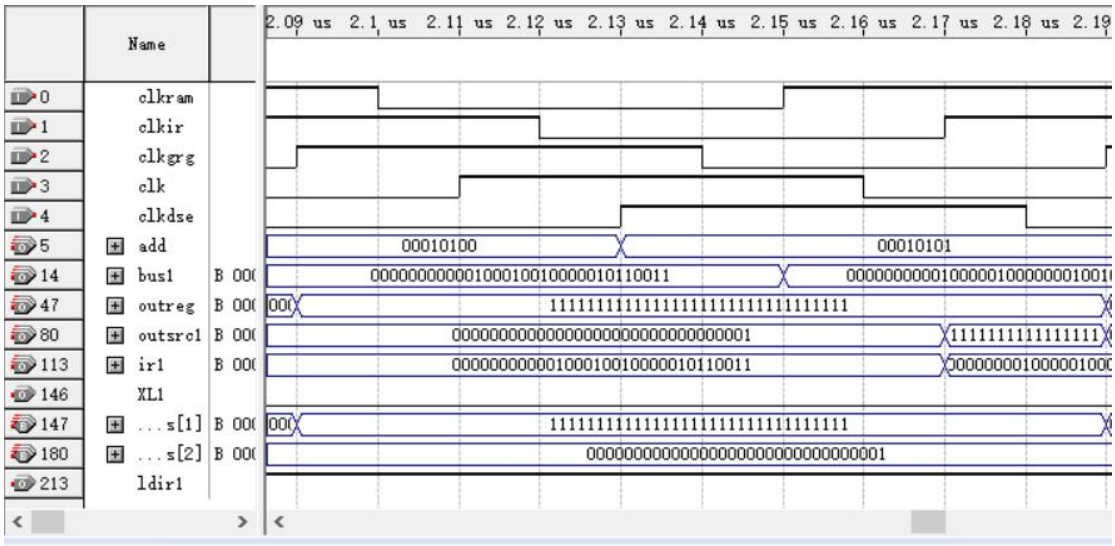


分析：这条指令将寄存器 1 中的值右移 1（即立即数）位，由于最高位为 0，故以 0 填充，该指令正确执行。

（17）slt

测试用二进制指令为：00000000000100010010000010110011

功能仿真结果为：



分析：这条指令是将寄存器 1 的值与寄存器 2 的值进行比较，如果寄存器 2 的值比寄存器 1 的值小，则向寄存器 1 存入 1，反之，则存入 0。而此时寄存器 1 的值比寄存器 2 的值大，故将寄存器 1 的所有位均刷新为 1，该指令正确执行。

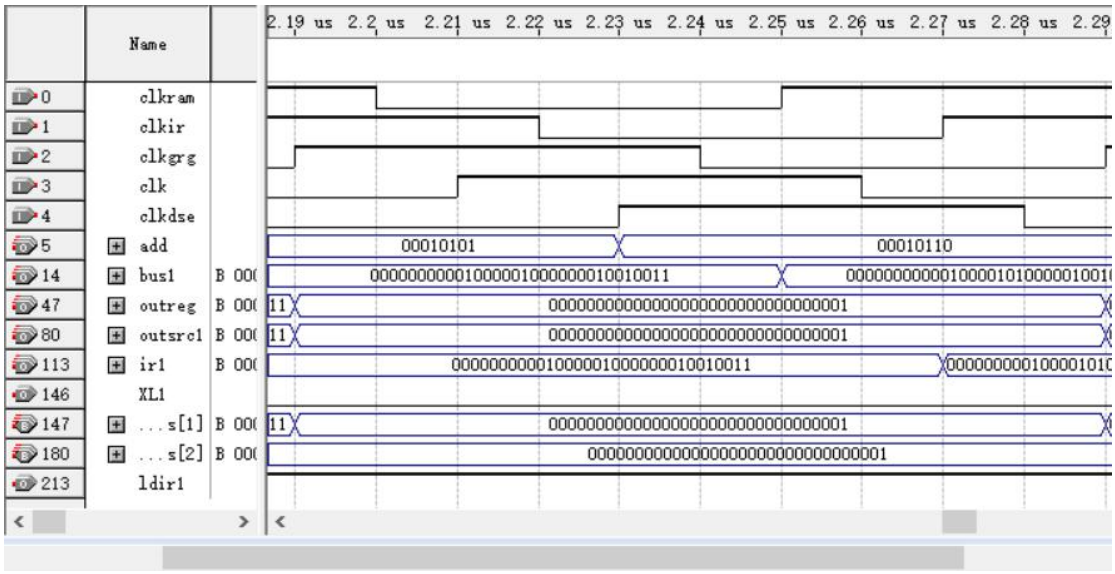
（18）slti

测试用二进制指令为：00000000001000001000000010010011

00000000000100001010000010010011

功能仿真结果为：

寄存器 1 的数化小：



分析：这两条指令首先将寄存器 1 的值化小，即利用立即数加法将前 31 位刷新为 0（立即数加法无视溢出）。然后将寄存器 1 的值与立即数进行比较，由于此时寄存器 1 的值与立即数相等，故向寄存器 1 中写入 0，该指令正确执行。

实验总结：

在本次实验中，虽然我负责的是整数计算指令，但是我对 CPU 的理解加深了。相比起我们在数字逻辑中编写的简易 CPU，RISC-V 指令集的指令可以将指令划分为不同的区域块进行识别，执行指令的效率提高，也更容易掌握这条指令的执行目的。

同时，这个实验也让我对 CPU 的各模块有了更为清楚的认识，在和小组内同学合作的时候也从他们身上学到了很多，老师的提示也给我们很大的启发，可以说这个小学期我收获颇丰。