CPU 实验报告

班级：物联 1601

学号：201608010628

姓名：曾彤芳

实验任务

实现单周期 CPU 的设计

实验要求

硬件设计采用 VHDL 或 Verilog 语言，软件设计采用 C/C++或 SystemC 语言，其它语言例如 Chisel、MyHDL 等也可选

实验报告采用 markdown 语言，或者直接上传 PDF 文档

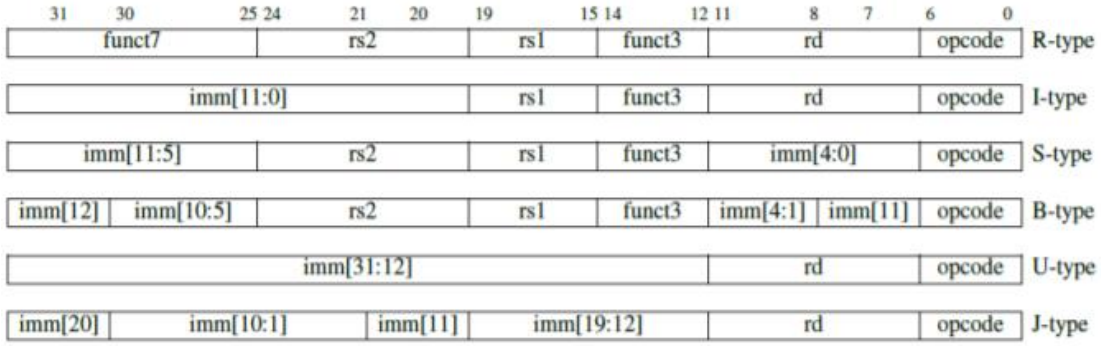
实验最终提交所有代码和文档

实验内容

RISC-V 指令

RV32I 指令集，其包含了六种基本指令格式，分别是：用于寄存器 -寄存器操作的 R 类型指令，用于短立即数和访存 load 操作的 I 型指令，用于访存 store 操 作的 S 型指令，用于条件跳转操作的 B 类型指令，用于长立即数的 U 型指令和用于无条件 跳转的 J 型指令。

RISC-V 指令集编码格式



基本指令格式

四种基础指令格式 R/I/S/U

imm：立即数

rs1：源寄存器 1

rs2：源寄存器 2

rd：目标寄存器

opcode：操作码

整数计算

ADDI：将 12 位有符号立即数和 rs 相加，溢出忽略，直接使用结果的最低 32bit，并存入 rd

SLTI：如果 rs 小于立即数(都是有符号整数),将 rd 置 1,否则置 0

SLTIU：和 SLTI 一致，不过都是无符号数

ANDI/ORI/XORI：rs 与有符号 12 位立即数进行 and,or,xor 操作

ADD/SUB:rs1(+/-)rs2 => rd

SLT/SLTU: 如果 rs1<rs2，rd 写 1; 否则 rd 为 0

AND/OR/XOR: rs1 与 rs2 进行 and,or,xor 操作

SLL/SRL/SRA: 和"寄存器-立即数"指令一致，将 r2 的低 5 位作为立即数即可

NOP 指令:

实际上是 ADDI x0,x0,0

JAL：J 类指令，立即数+pc 为跳转目标，rd 存放 pc+4（返回地址）

跳转范围为 pc(+/-)1MB

JALR：I 类指令，rs+立即数为跳转目标，rd 存放 pc+4（返回地址）

实现远跳转

BEQ/BNE：rs1(==/!=)rs2，分别在相等或者不等时，发生跳转

BLT：rs1 < rs2，跳转

BGE：rs1 >= rs2，跳转

LOAD：rs 作为基地址，加上有符号的偏移，读取到 rd 寄存器

STORE：rs1 作为基地址加上有符号的偏移，作为内存地址，写入内容为 rs2

CSRRW：Atomic Read/Write CSR

读取 CSR 的值存入 rd 寄存器，并将 rs 存入 CSR

CSRRS：Atomic Read and Set Bits in CSR

读取 CSR 的值存入 rd 寄存器，并根据 rs 中高位对 CSR 置 1

CSRRC：Atomic Read and Clear Bits in CSR

读取 CSR 的值存入 rd 寄存器，并根据 rs 中高位对 CSR 置 0

CSRRWI/CSRRSI/CSRRCI

将 CSRRW 类寄存器中的 rs 换成立即数

RDCYCLE：时钟周期计数

RDTIME：时间 tick 数

RDINSTRET：指令数

ECALL

EBREAK

CPU 设计

CPU 在处理指令时，一般需要经过以下几个步骤：

(1) 取指令(IF)：根据程序计数器 PC 中的指令地址，从存储器中取出一条指令，同时，PC 根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入 PC，当然得到的“地址”需要做些变换才送入 PC。

(2) 指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3) 指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4) 存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5) 结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

单周期 CPU，是在一个时钟周期内完成这五个阶段的处理。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
entity mycpu 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 mycpu;

architecture behav of mycpu 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<="00000000000000000000000"&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);
                    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

测试平台

模块	配置	备注
CPU	Core i5-6700U	
操作系统	Windows10	

测试结果

部分测试指令：

00000000001000010000000010110011

00000000000100010000000010110011

00000010000100010000000001001001

01000000001000010000000010110011

00000000001100001000000010010011

00000000000010001000000100010011

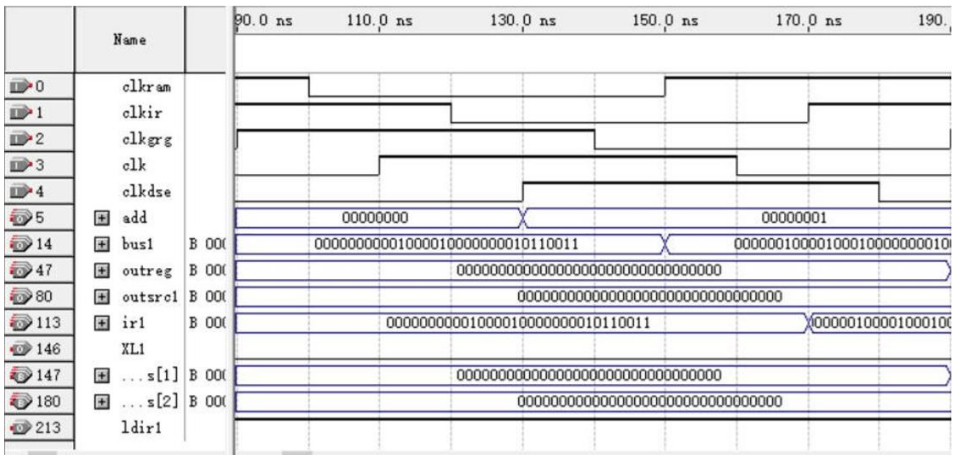
01000000000100010000000010110011

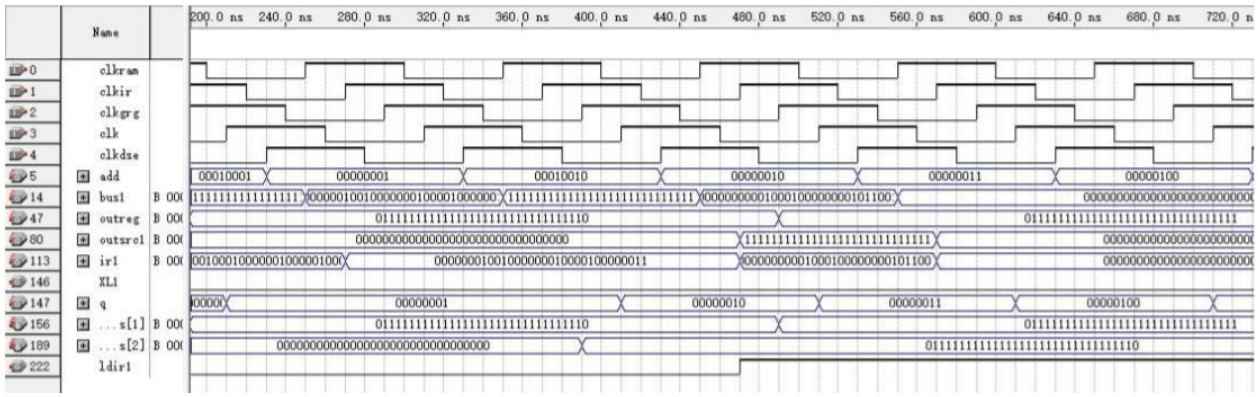
00011000100000100000000010110111

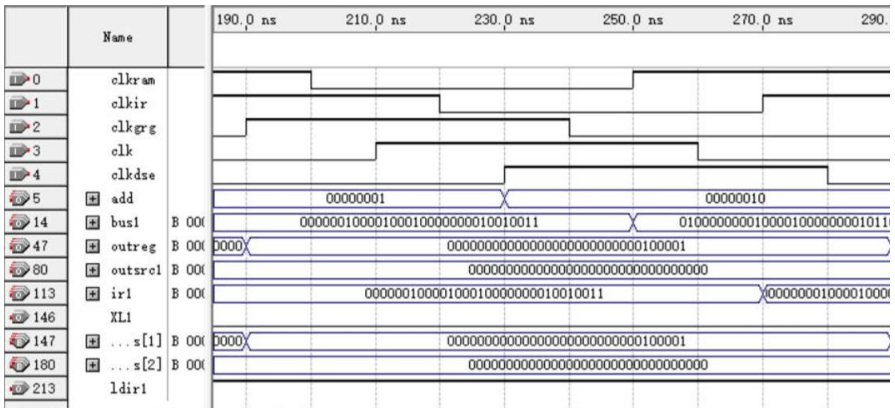
00000000000100000110000010110011

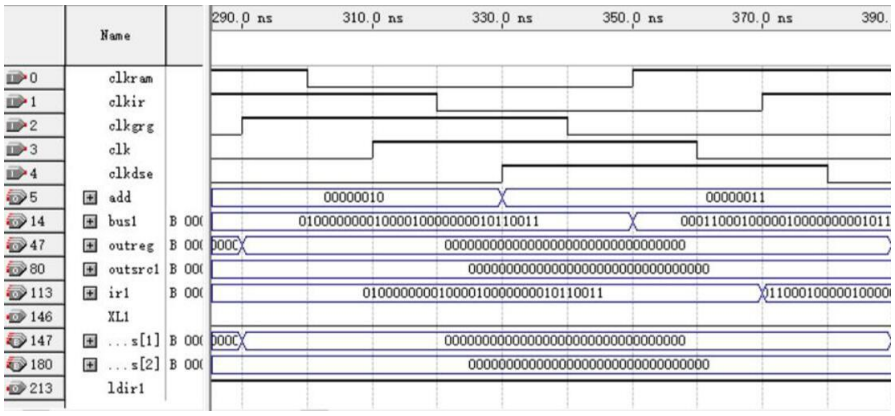
00000010000100010100000010010011

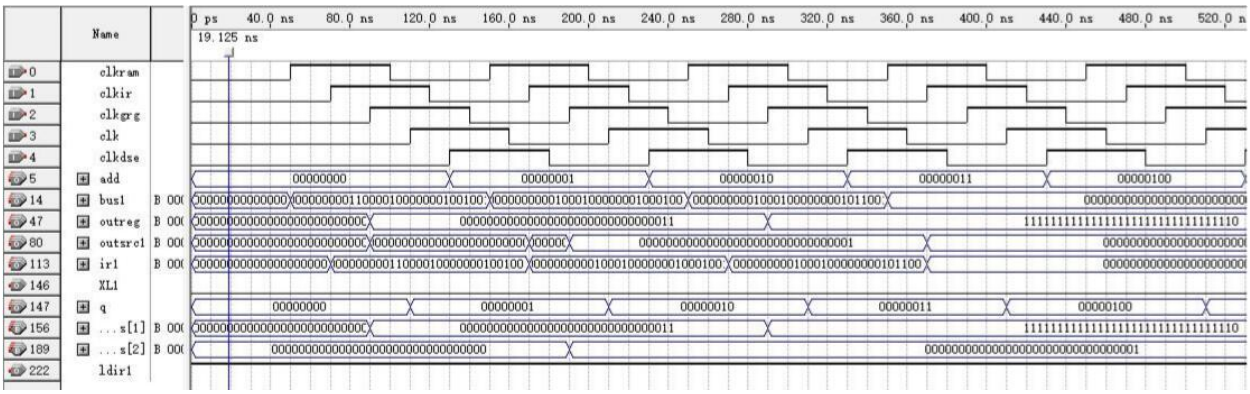
部分测试结果：

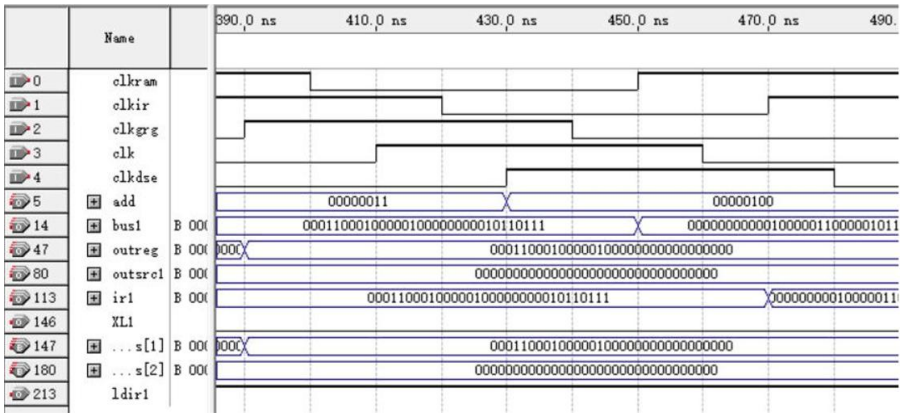


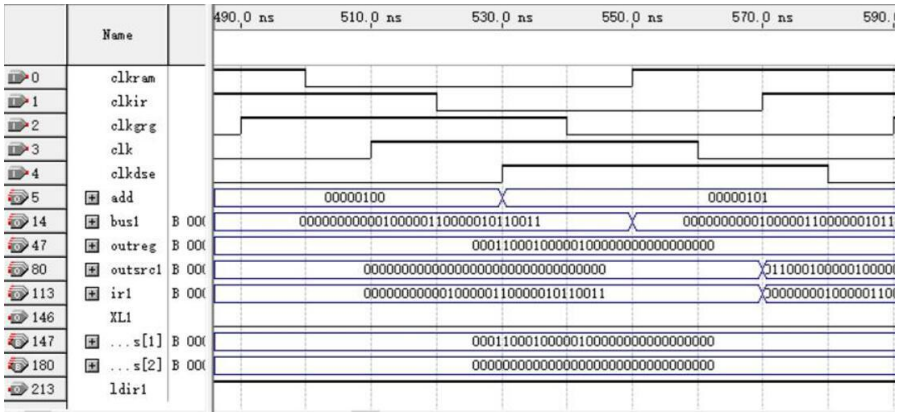


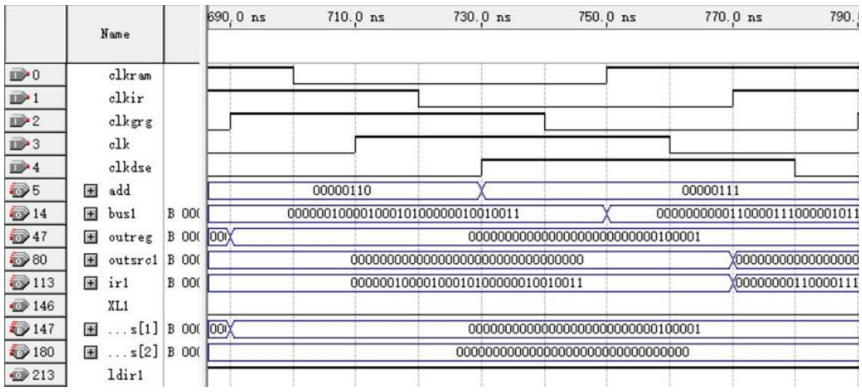



Name
290.0 ns
310.0 ns
330.0 ns
350.0 ns
370.0 ns
390.
0 clkram
1 clkir
2 clkgrg
3 clk
4 clkdse
5 add
14 bus1
47 outreg
80 outsrc1
113 ir1
146 XL1
147 ...s[1]
180 ...s[2]
213 ldir1
00000010
00000011


Name
0 ps
40.0 ns
80.0 ns
120.0 ns
160.0 ns
200.0 ns
240.0 ns
280.0 ns
320.0 ns
360.0 ns
400.0 ns
440.0 ns
480.0 ns
19.125 ns
0 clkram
1 clkir
2 clkgrg
3 clk
4 clkdse
5 add
14 bus1
47 outreg
80 outsrc1
113 ir1
146 XL1
147 q
156 ...s[1]
189 ...s[2]
222 ldir1
00000000
00000001
00000010
00000011
00000100


Name
390.0 ns
410.0 ns
430.0 ns
450.0 ns
470.0 ns
490.
0 clkram
1 clkir
2 clkgrg
3 clk
4 clkdse
5 add
14 bus1
47 outreg
80 outsrc1
113 ir1
146 XL1
147 ...s[1]
180 ...s[2]
213 ldir1
00000011
00000100





结果分析

测试了 ADD，ADDI，SUB，LUI，XOR，XORI 等指令，显示结果均正确，所有指令正确执行，实现了单周期 CPU 的设计，达到了实验目的，本次实验收获很多。