同济大学计算机系

计算机组成原理实验报告

学　　号 **2152809**

姓　　名 曾崇然

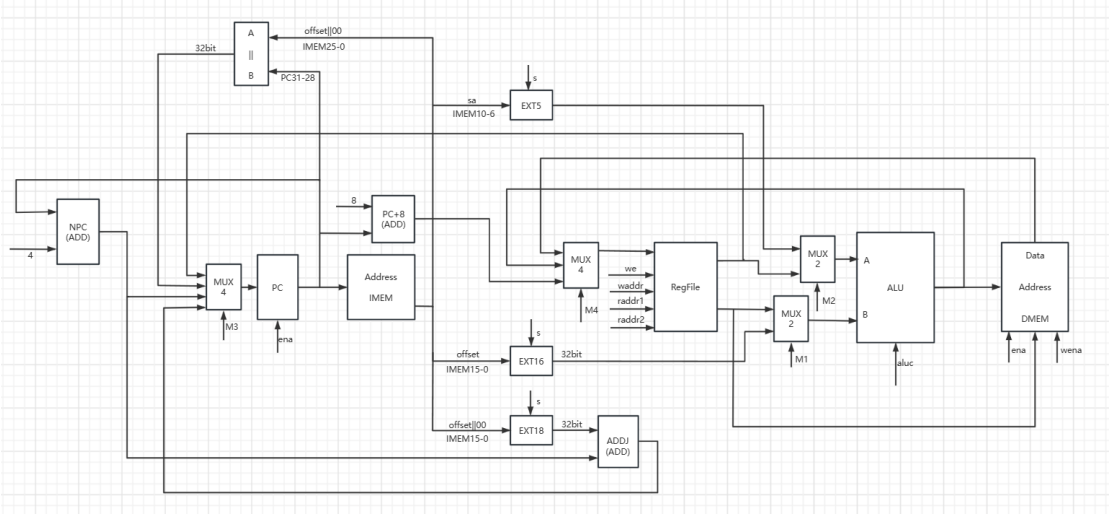
专　　业 计算机科学与技术

授课老师 张冬冬老师

一、实验内容

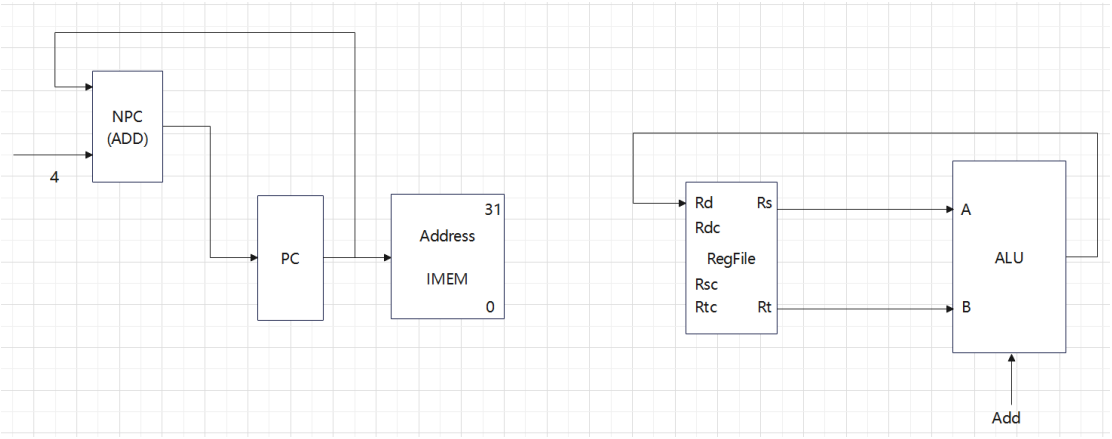
使用 Verilog HDL 语言实现 31 条 MIPS 指令的 CPU 的设计和 仿真

二、数据通路构建

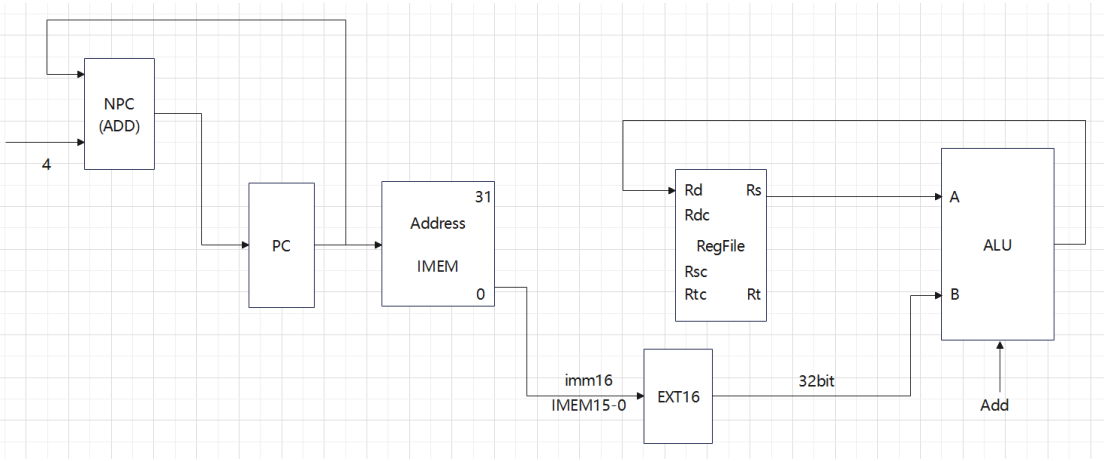


三、数据通路介绍

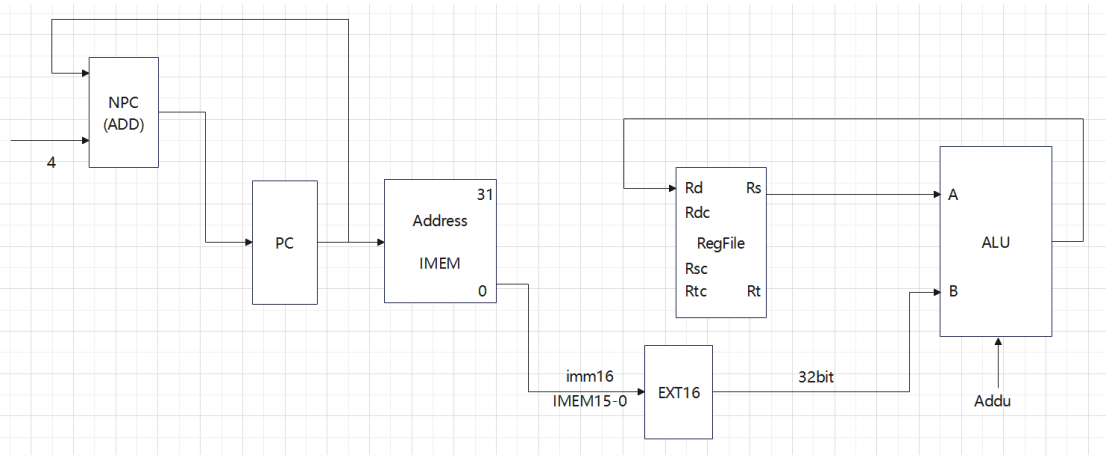
1. add:PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



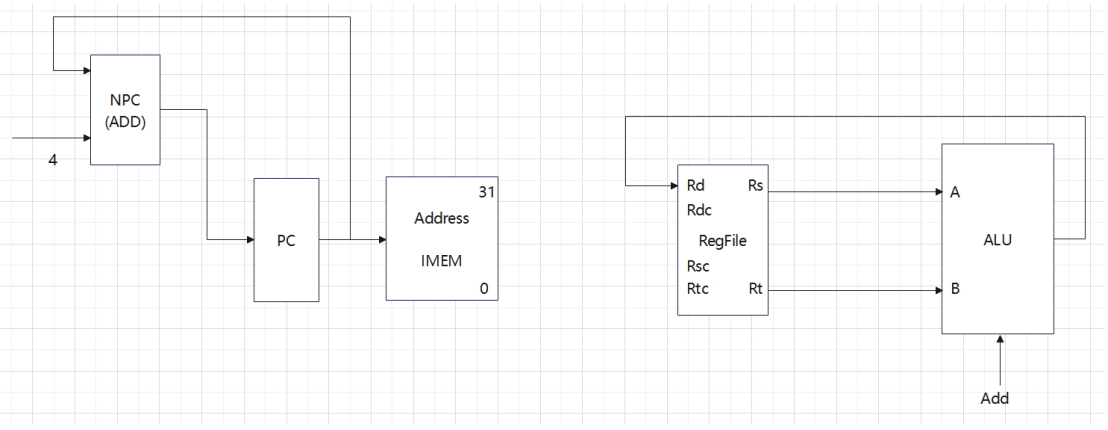
2. addi:PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



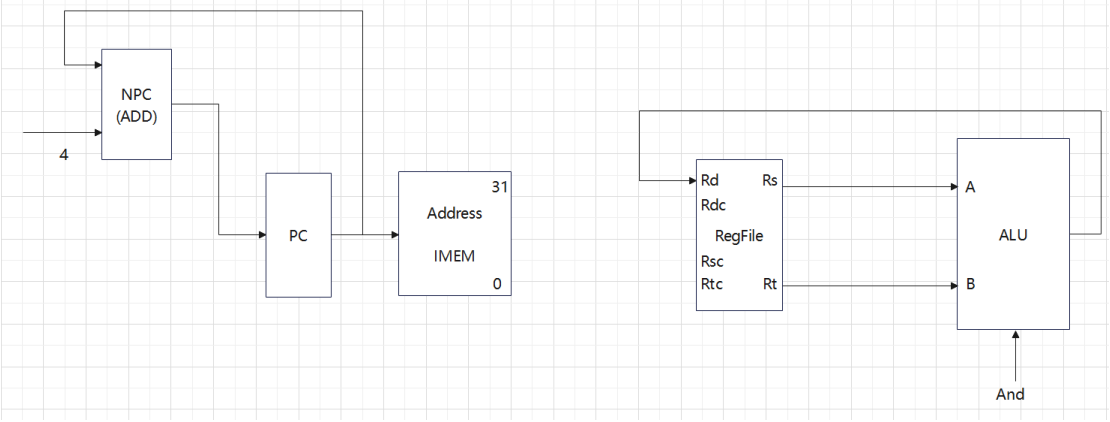
3. addiu: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



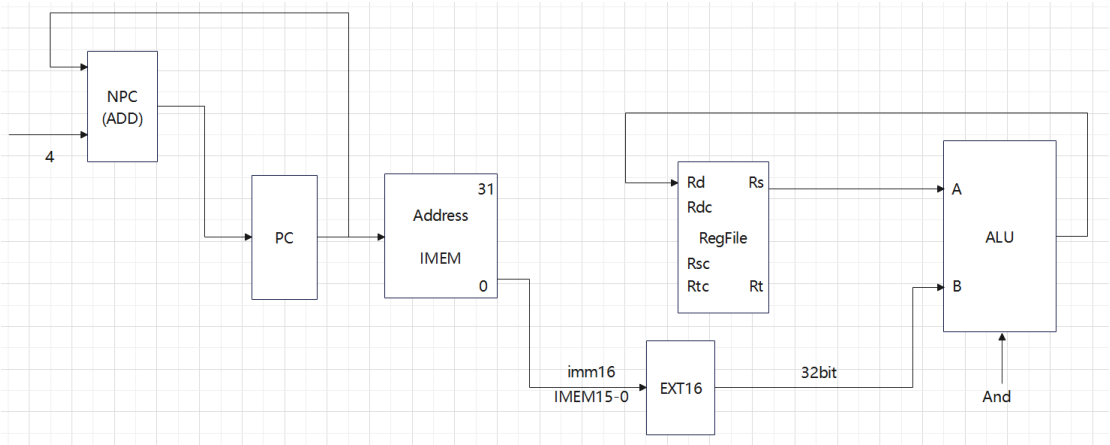
4. addu: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



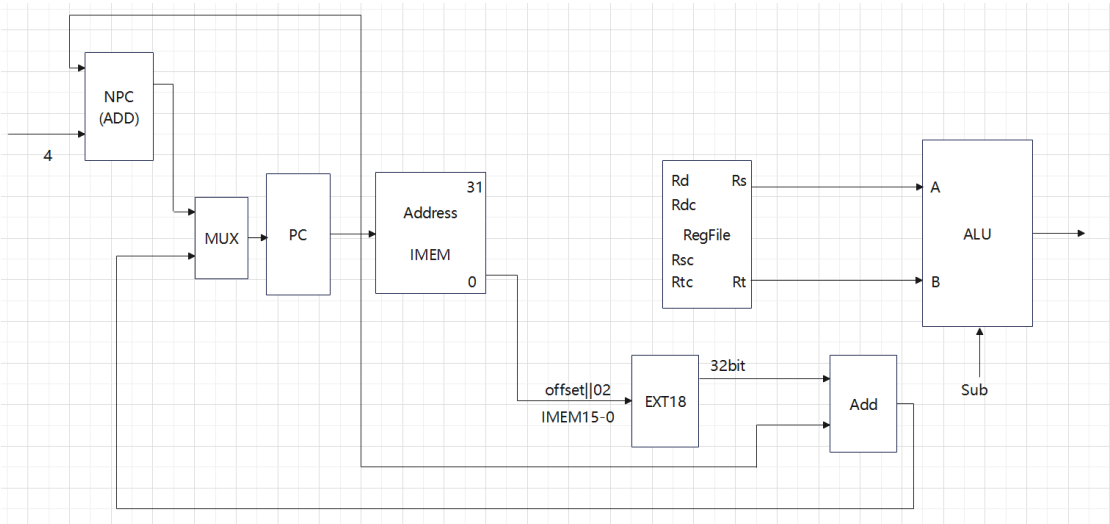
5. and: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



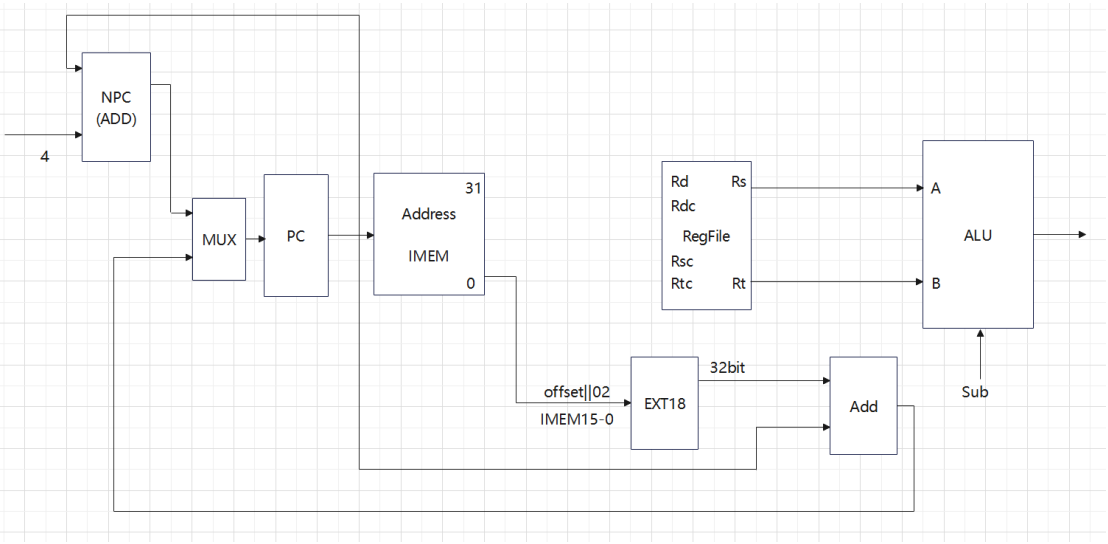
6. andi: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



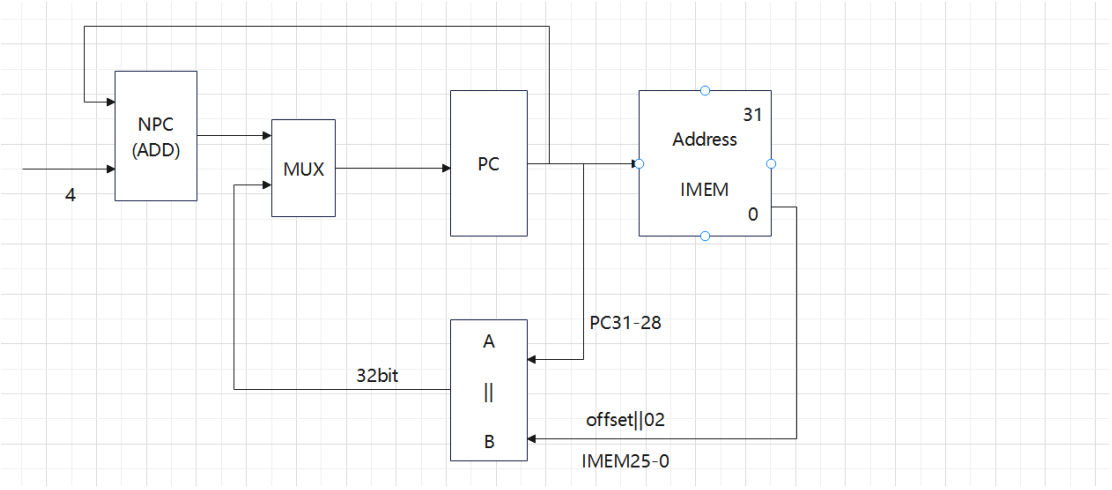
7. beq:PC 的值送到指令存储器，寄存器堆中两个寄存器的内容送到 ALU 运算，根据结果选择将 PC+4 还是指令的 15-0 位扩展结果送到 PC



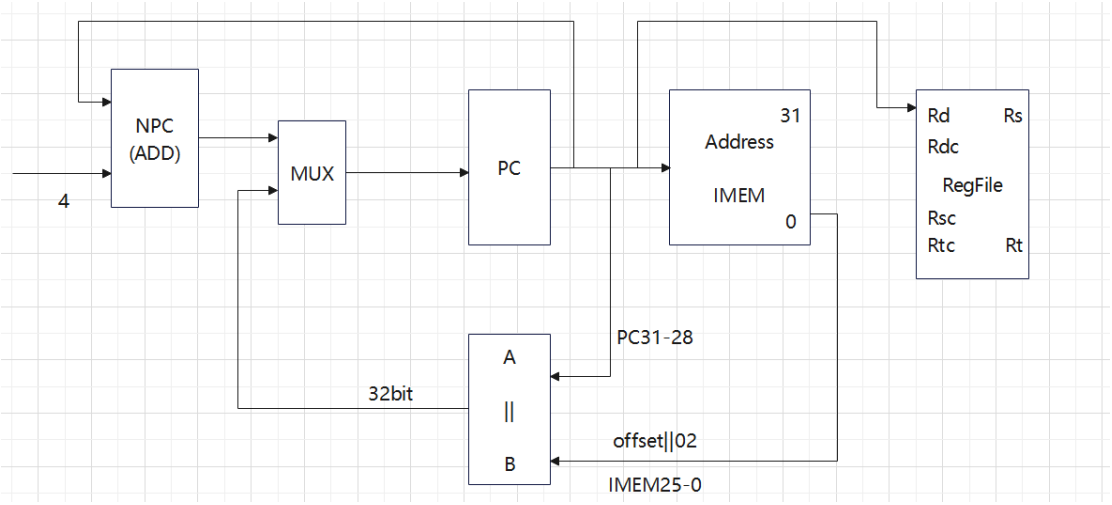
8. bne: PC 的值送到指令存储器，寄存器堆中两个寄存器的内容送到 ALU 运算，根据结果选择将 PC+4 还是指令的 15-0 位扩展结果送到 PC



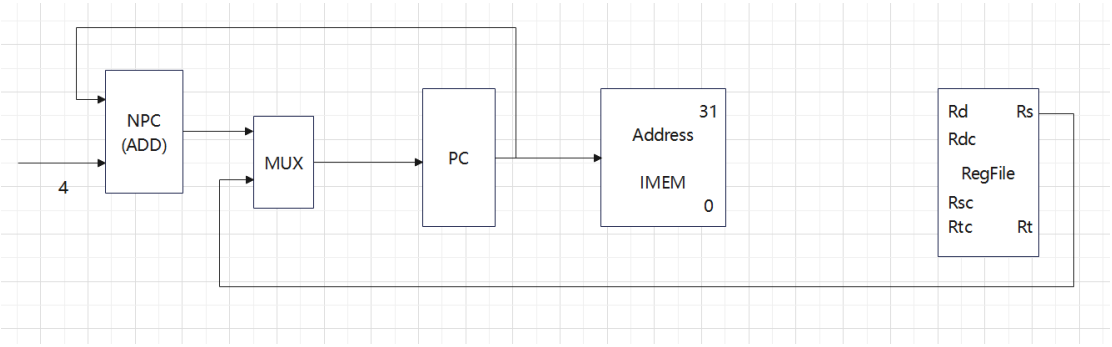
9. j:PC 的值送到指令存储器，指令存储器值的 25-0 位左移两位和 PC 高 4 位拼接送到 PC 寄存器



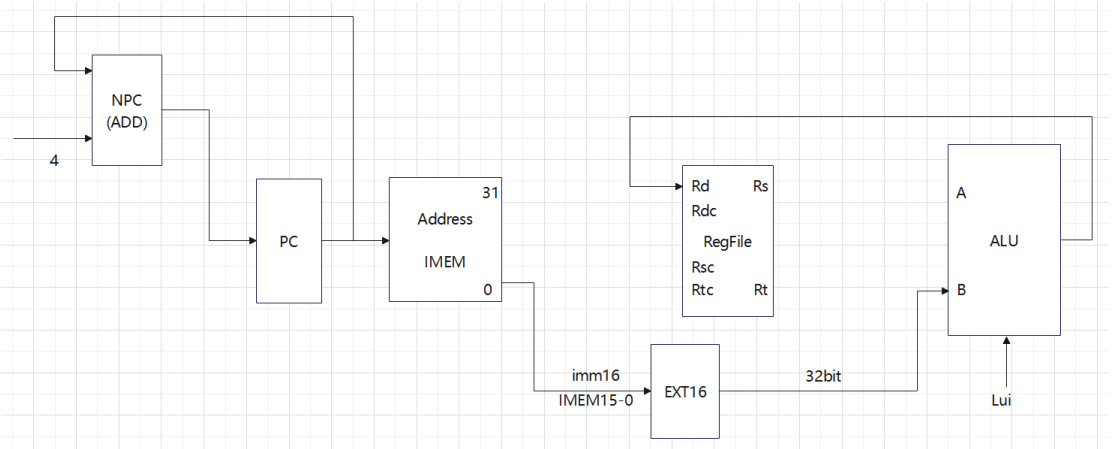
10. jal: PC 的值送到指令存储器和寄存器堆指定寄存器，指令存储器值的 25-0 位左移两位和 PC 高 4 位拼接送到 PC 寄存器



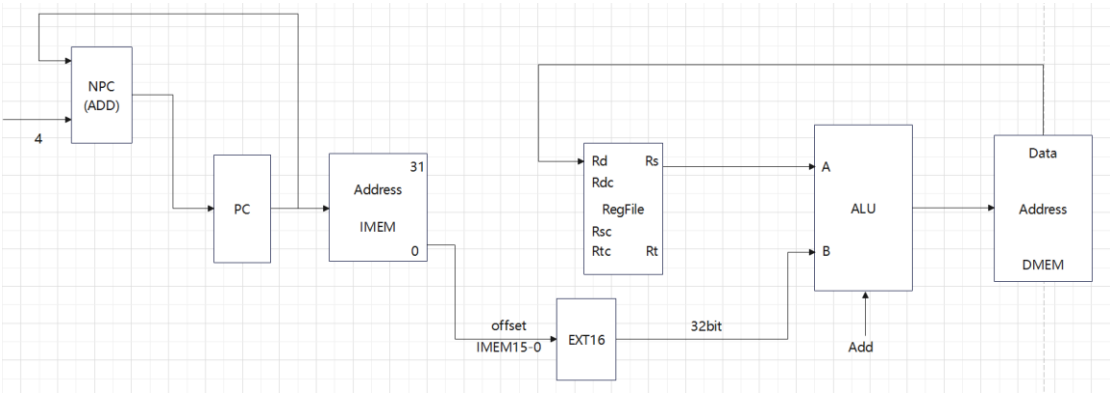
11. jr:PC 的值送到指令存储器，寄存器堆中指定的寄存器送到 PC



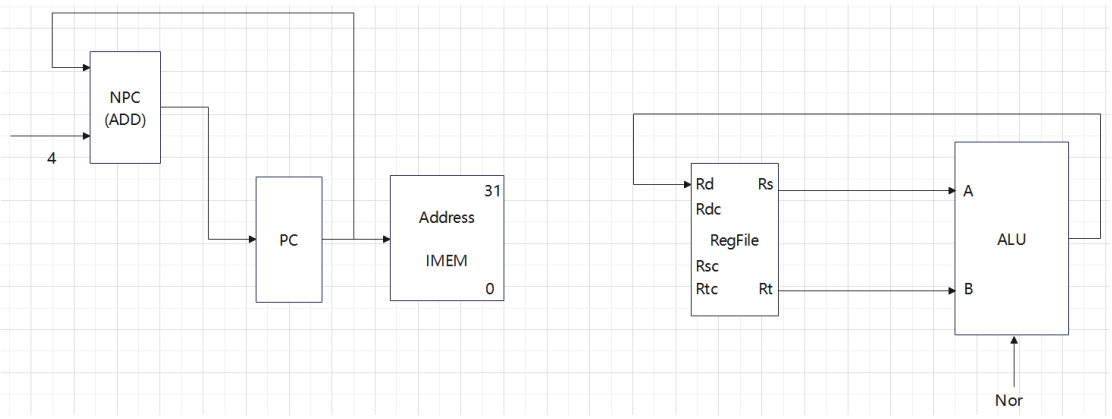
12. lui: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



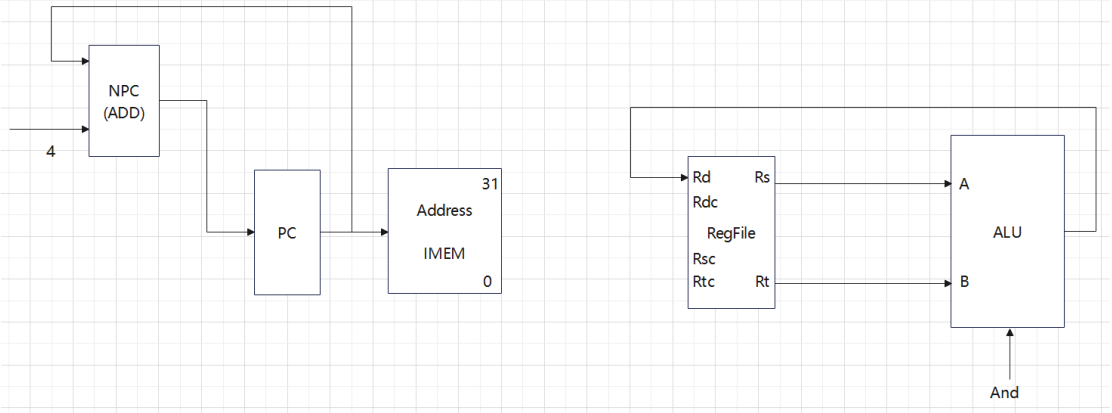
13. lw:PC 的值送指令存储器，PC+4 的值送 PC，指令存储器的值的 15-0 位扩展后送 ALU 一端，寄存器堆中一个寄存器的值送 ALU 另一端，ALU 结果送数据存储器，数据存储器的值送寄存器堆中的指定寄存器



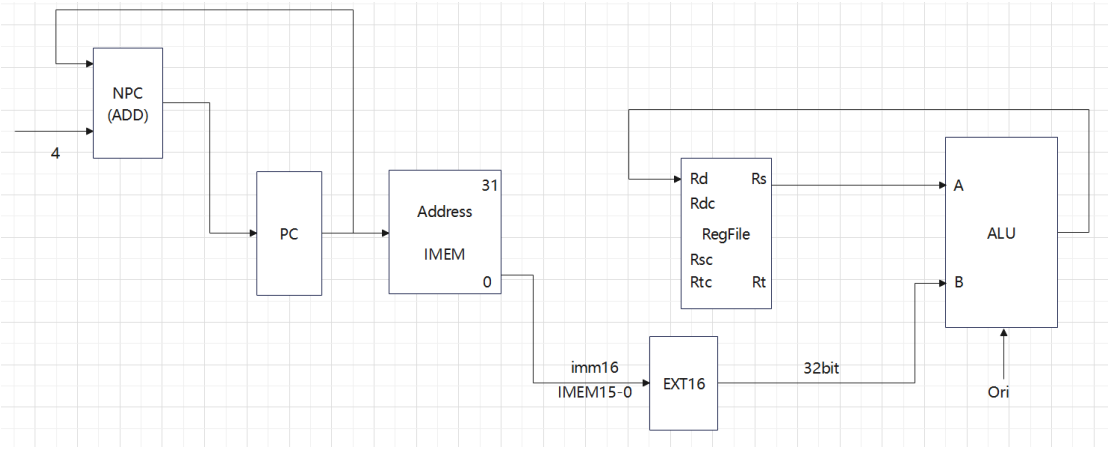
14. nor: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



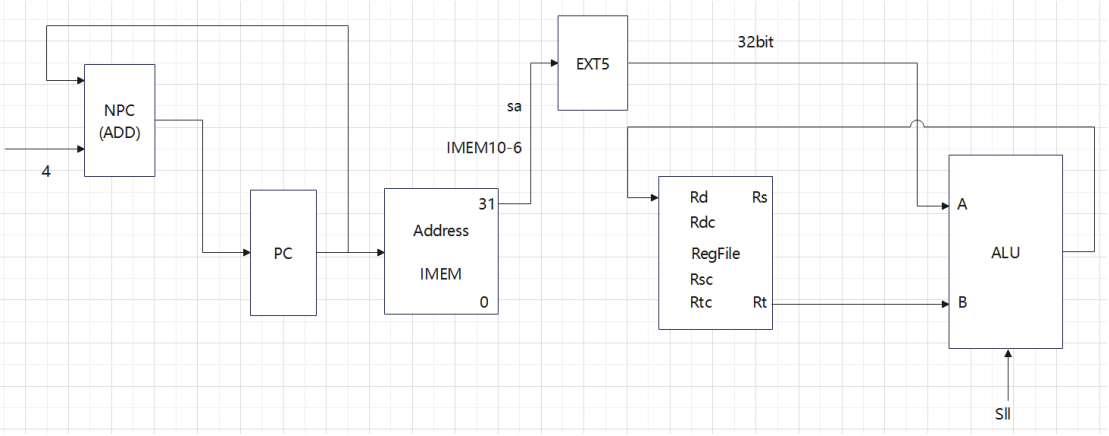
15. or: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



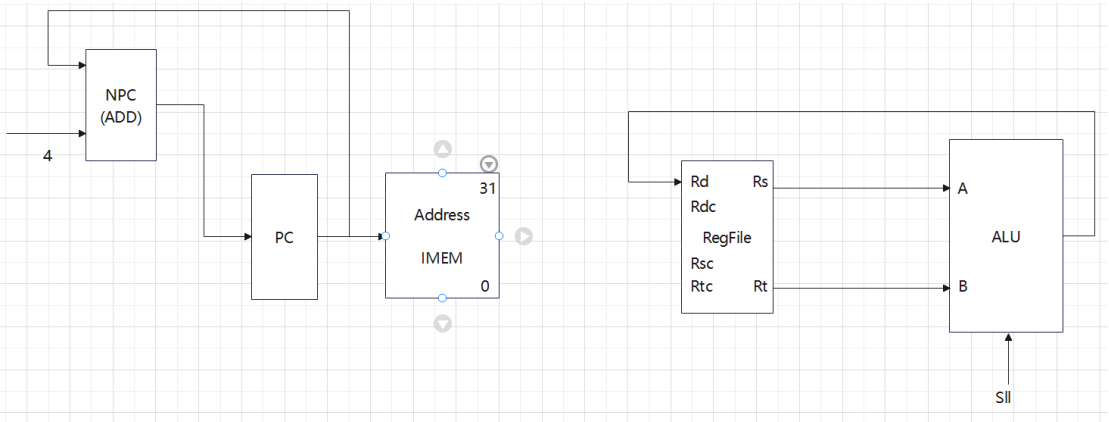
16. ori: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



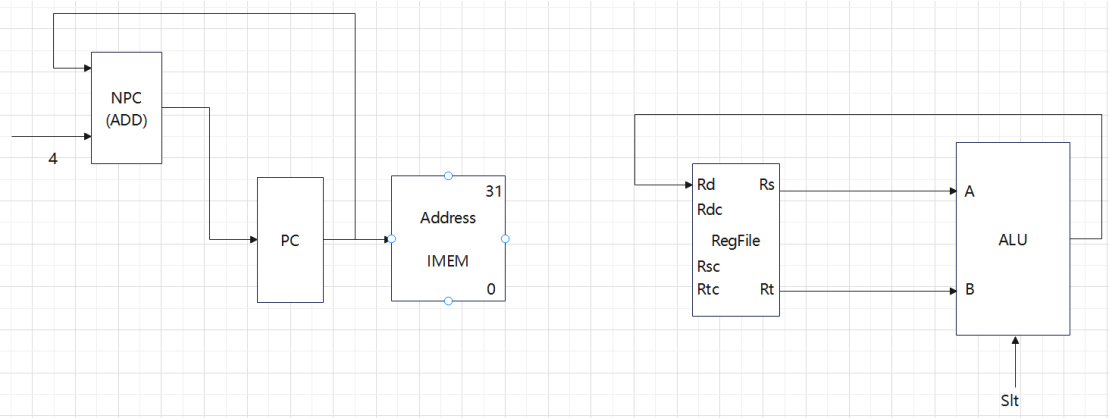
17. sll:PC 的值送指令存储器，PC+4 的值送 PC，指令存储器值的 10-6 位扩展后送 ALU 一端，寄存器堆中一个寄存器的值送 ALU 另一端，ALU 结果送寄存器堆中指定寄存器



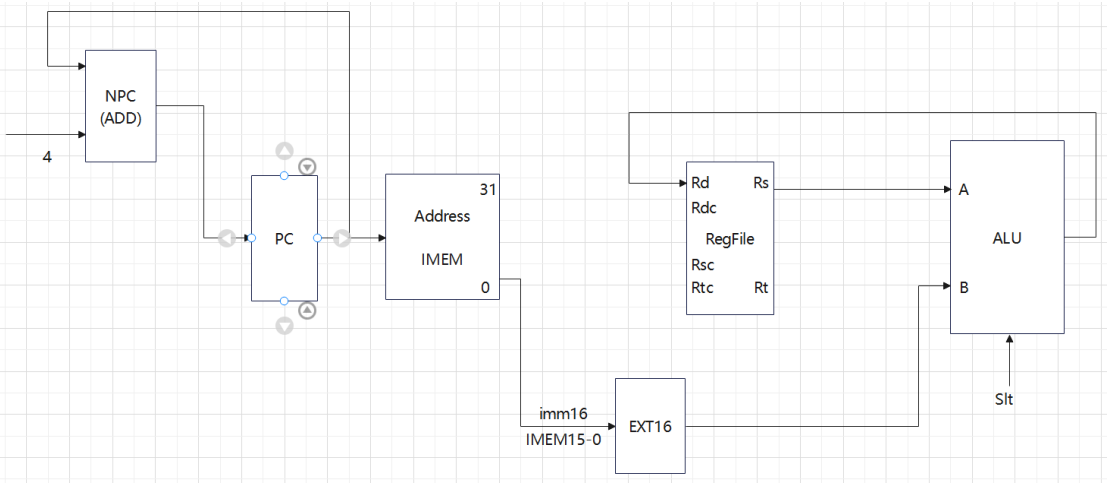
18. sllv: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



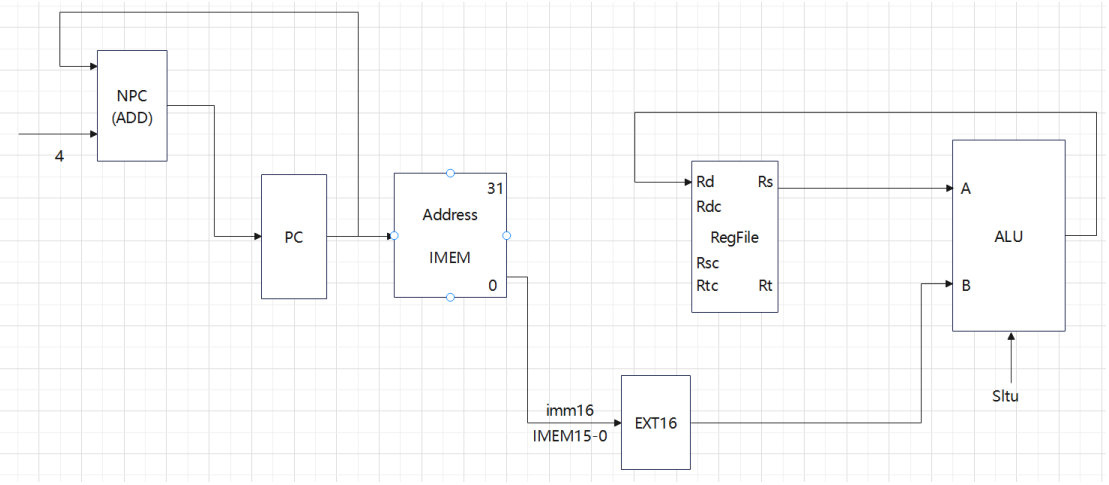
19. slt: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



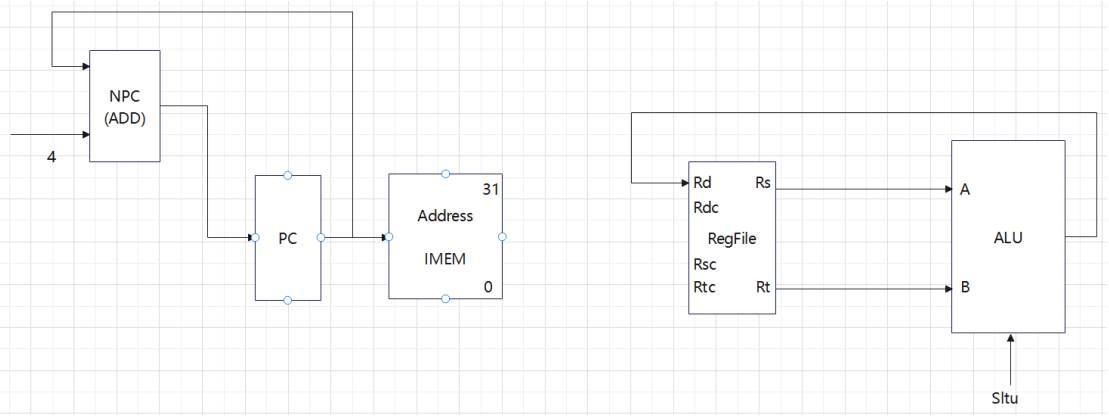
20. slti: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



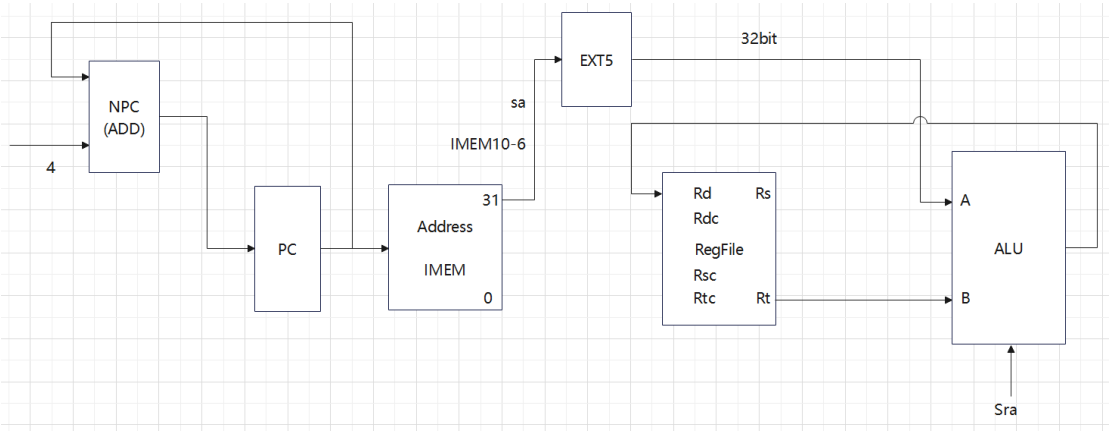
21. sltiu: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



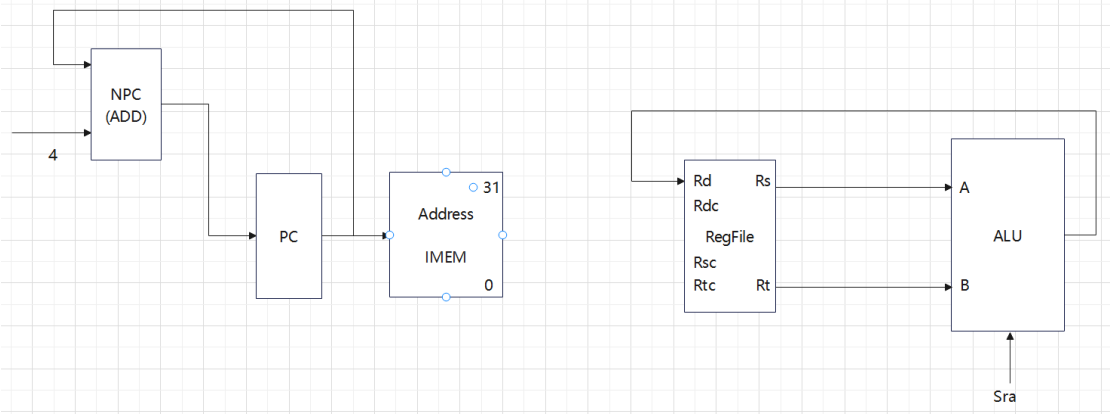
22. sltu PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



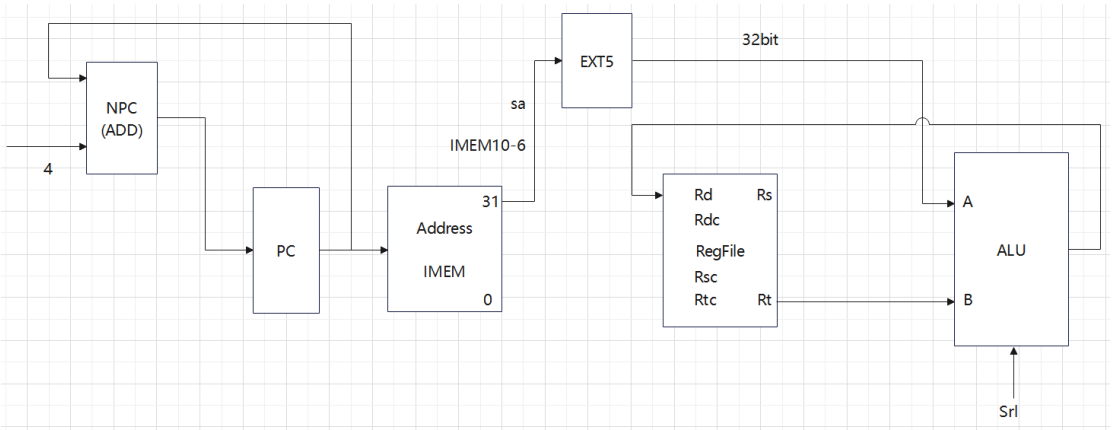
23. sra: PC 的值送指令存储器，PC+4 的值送 PC，指令存储器值的 10-6 位扩展后送 ALU 一端，寄存器堆中一个寄存器的值送 ALU 另一端，ALU 结果送寄存器堆中指定寄存器



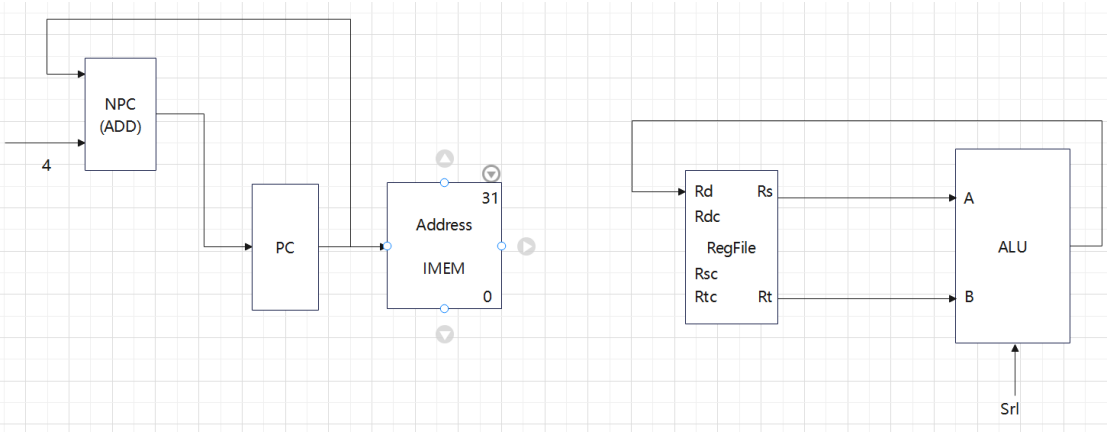
24. srav: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



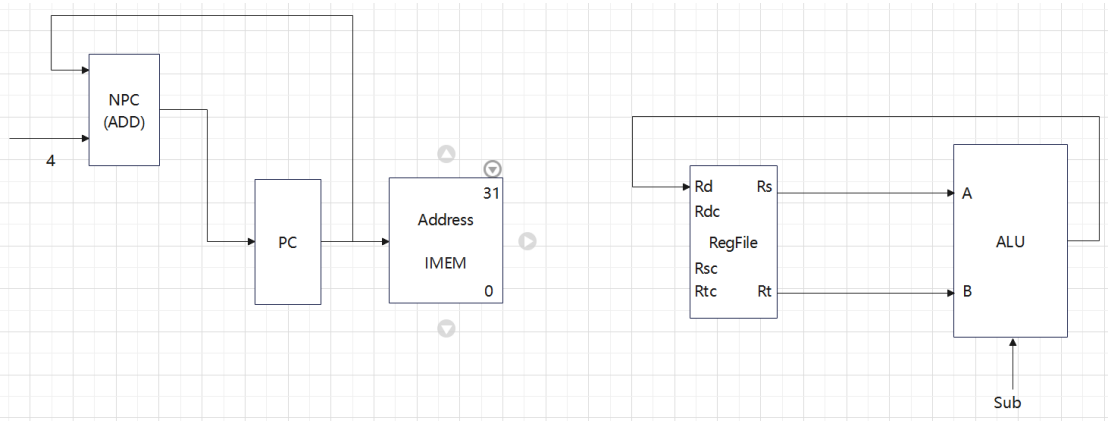
25. srl: PC 的值送指令存储器，PC+4 的值送 PC，指令存储器值的 10-6 位扩展后送 ALU 一端，寄存器堆中一个寄存器的值送 ALU 另一端，ALU 结果送寄存器堆中指定寄存器



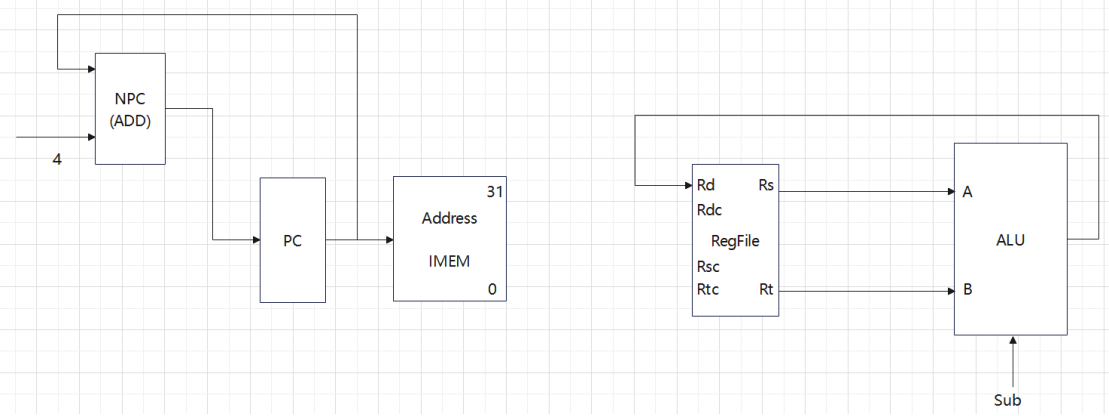
26. srlv: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



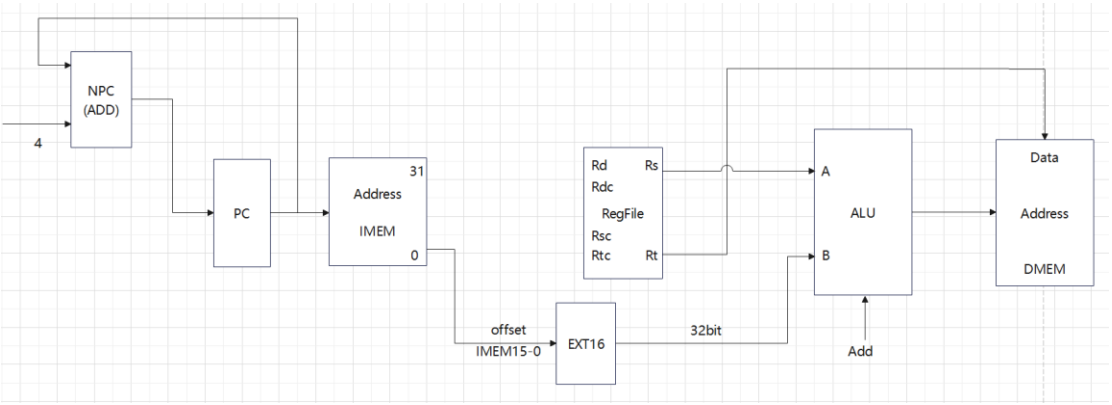
27. sub: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



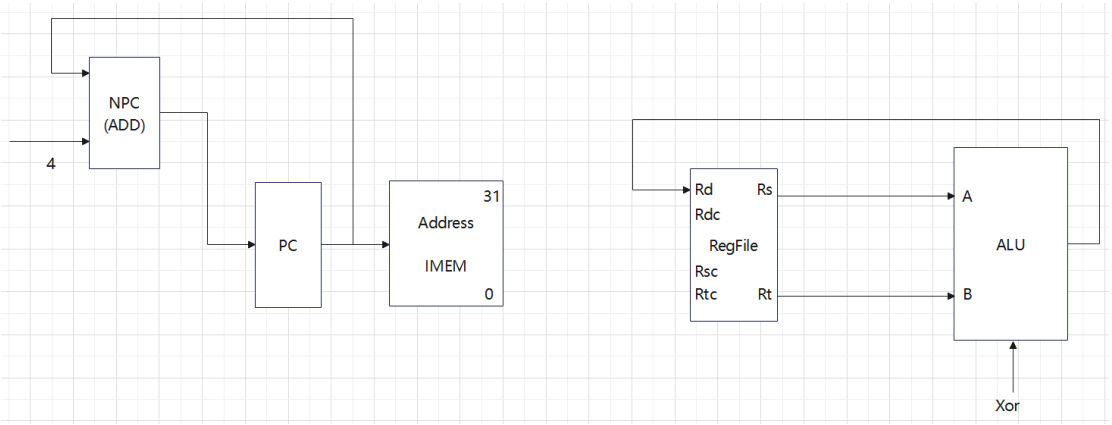
28. subu: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



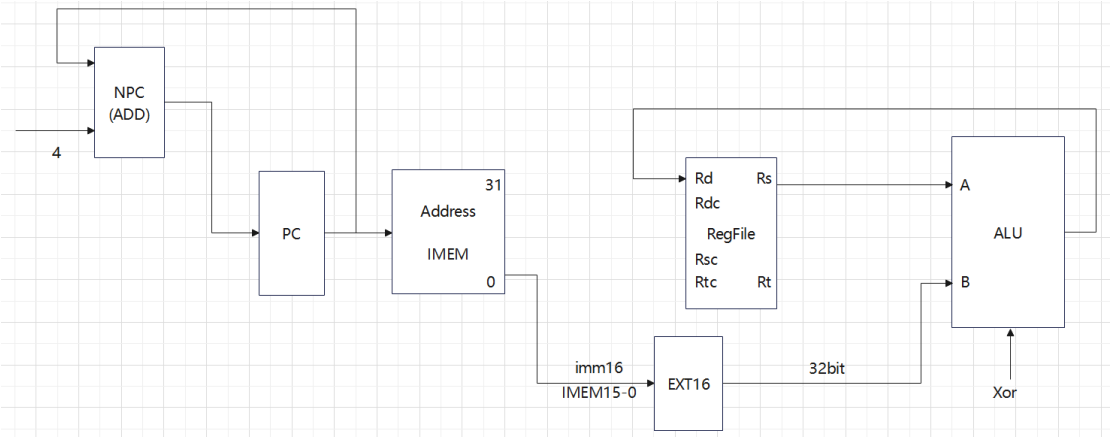
29. sw:PC 寄存器的值送到指令存储器，PC+4 的值送到 PC 寄存器，指令存储器值的 15-0 位扩展后送到 ALU 的一端，寄存器堆中一个寄存器的值送到 ALU 的另一端，ALU 的结果送到数据存储器，寄存器堆中一个寄存器的值送到数据存储器



30. xor: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



31. xori: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



四、控制信号

	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	jr
pcreg_ena	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
d_ram_wena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d_ram_ena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ext5_s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ext16_s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ext18_s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rf_we	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
rf_waddr	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	0
rf_raddr1	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	0	0	0	imem25-21	imem25-21	imem25-21	imem25-21
rf_raddr2	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	0
alu_aluc	4'b0010	4'b0000	4'b0011	4'b0001	4'b0100	4'b0101	4'b0110	4'b0111	4'b1011	4'b1010	4'b1110	4'b1101	4'b1100	4'b1110	4'b1101	4'b1100	0
M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M2	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0
M3	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b00
M4	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	0
指令编号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

	addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu	lui	j	jal
pcreg_ena	1	1	1	1	1	1	1	1	1	1	1	1	1	1
d_ram_wena	0	0	0	0	0	0	1	0	0	0	0	0	0	0
d_ram_ena	0	0	0	0	0	1	1	0	0	0	0	0	0	0
ext5_s	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ext16_s	1	1	0	0	0	1	1	0	0	1	1	0	0	0
ext18_s	0	0	0	0	0	0	0	1	1	0	0	0	0	0
rf_we	1	1	1	1	1	1	0	0	0	1	1	1	0	1
rf_waddr	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	0	0	0	imem20-16	imem20-16	imem20-16	0	5'b11111
rf_raddr1	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	0	0	0
rf_raddr2	0	0	0	0	0	0	imem20-16	imem20-16	imem20-16	imem20-16	0	0	0	0	0
alu_aluc	4'b0010	4'b0000	4'b0100	4'b0101	4'b0110	4'b0000	4'b0000	4'b0011	4'b0011	4'b1011	4'b1010	4'b1000	0	0	0
M1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0
M2	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
M3	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b1{zero}	2'b1{~zero}	2'b10	2'b10	2'b10	2'b01	2'b01	
M4	2'b01	2'b01	2'b01	2'b01	2'b01	2'b00	0	0	0	2'b01	2'b01	2'b01	0	2'b10	
指令编号	17	18	19	20	21	22	23	24	25	26	27	28	29	30	

五、所用到的部件建模

1. **PC 寄存器：**

```
module pcreg(
    input clk,
    input rst,
    input ena,
    input [31 : 0] data_in,
    output reg [31 : 0] data_out
    );
    initial
    begin
        data_out = 32'h00400000;
    end
    always @ (negedge clk)
    begin
        if (rst == 1)
        begin
            data_out <= 32'h00400000;
        end
        else
        begin
            if (ena == 1)
            begin
                data_out <= data_in;
            end
        end
    end
endmodule
```

2. **行为级 ALU：**

```
module alu(
    input [31 : 0] a,
    input [31 : 0] b,
    input [3 : 0] aluc,
    output reg [31 : 0] r,
    output reg zero,
    output reg carry,
    output reg negative,
    output reg overflow
    );
    reg [31 : 0] results [13 : 0];
    reg zeros [13 : 0];
    reg carrys [13 : 0];
    reg negatives [13 : 0];
    reg overflows [13 : 0];
    always @ (*)
    begin
        //无符号加法
        results[0] = a + b;
        if (results[0] == 32'h00000000)
        begin
            zeros[0] = 1;
        end
        else
        begin
            zeros[0] = 0;
        end
        if     ($unsigned(results[0])     <
```

```
$unsigned(a) || $unsigned(results[0]) < $unsigned(b))
            begin
                carrys[0] = 1;
            end
            else
            begin
                carrys[0] = 0;
            end
            if (results[0][31] == 1)
            begin
                negatives[0] = 1;
            end
            else
            begin
                negatives[0] = 0;
            end
            //有符号加法
            results[1] = a + b;
            if (results[1] == 32'h000000)
            begin
                zeros[1] = 1;
            end
            else
            begin
                zeros[1] = 0;
            end
            if (results[1][31] == 1)
            begin
                negatives[1] = 1;
            end
            else
            begin
                negatives[1] = 0;
            end
            if (a[31] == 1 && b[31] == 1 && results[1][31] == 0)
            begin
                overflows[1] = 1;
            end
            else if (a[31] == 0 && b[31] == 0 && results[1][31] ==1)
            begin
                overflows[1] = 1;
            end
            else
            begin
                overflows[1] = 0;
            end
            //无符号减法
            results[2] = a - b;
            if (results[2] == 32'h00000000)
            begin
                zeros[2] = 1;
            end
            else
            begin
                zeros[2] = 0;
            end
            if ($unsigned(a) < $unsigned(b))
            begin
                carrys[2] = 1;
            end
            else
            begin
                carrys[2] = 0;
            end
            if (results[2][31] == 1)
            begin
                negatives[2] = 1;
            end
            else
            begin
                negatives[2] = 0;
            end
            //有符号减法
            results[3] = a - b;
            if (results[3] == 32'h00000000)
            begin
                zeros[3] = 1;
            end
            else
            begin
                zeros[3] = 0;
            end
            if (results[3][31] == 1)
            begin
```

```
            negatives[3] = 1;
        end
        else
        begin
            negatives[3] = 0;
        end
        if (a[31] == 0 && b[31] == 1
&& results[3][31] == 1)
        begin
            overflows[3] = 1;
        end
        else if (a[31] == 1 && b[31] ==
0 && results[3][31] == 0)
        begin
            overflows[3] = 1;
        end
        else
        begin
            overflows[3] = 0;
        end
        //与运算
        results[4] = a & b;
        if (results[4] == 32'h00000000)
        begin
            zeros[4] = 1;
        end
        else
        begin
            zeros[4] = 0;
        end
        if (results[4][31] == 1)
        begin
            negatives[4] = 1;
        end
        else
        begin
            negatives[4] = 0;
        end
        //或运算
        results[5] = a | b;
        if (results[5] == 32'h00000000)
        begin
            zeros[5] = 1;
        end
        else
        begin
            zeros[5] = 0;
        end
        if (results[5][31] == 1)
        begin
            negatives[5] = 1;
        end
        else
        begin
            negatives[5] = 0;
        end
        //异或运算
        results[6] = a ^ b;
        if (results[6] == 32'h00000000)
        begin
            zeros[6] = 1;
        end
        else
        begin
            zeros[6] = 0;
        end
        if (results[6][31] == 1)
        begin
            negatives[6] = 1;
        end
        else
        begin
            negatives[6] = 0;
        end
        //或非运算
        results[7] = ~(a | b);
        if (results[7] == 32'h00000000)
        begin
            zeros[7] = 1;
        end
        else
        begin
            zeros[7] = 0;
        end
        if (results[7][31] == 1)
        begin
            negatives[7] = 1;
        end
```

```
        else
        begin
            negatives[7] = 0;
        end
        //lui 运算
        results[8] = {b[15 : 0], 16'b0};
        if (results[8] == 32'h00000000)
        begin
            zeros[8] = 1;
        end
        else
        begin
            zeros[8] = 0;
        end
        carrys[8] = b[16];
        if (results[8][31] == 1)
        begin
            negatives[8] = 1;
        end
        else
        begin
            negatives[8] = 0;
        end
        //有符号比较
        results[9] = ($signed(a) < $signed(b)) ? 1 : 0;
        if (results[9] == 32'h00000000)
        begin
            zeros[9] = 1;
        end
        else
        begin
            zeros[9] = 0;
        end
        negatives[9] = 0;
        //无符号比较
        if ($unsigned(a) < $unsigned(b))
        begin
            results[10] = 1;
        end
        else
        begin
            results[10] = 0;
        end
        if (results[10] == 32'h00000000)
        begin
            zeros[10] = 1;
        end
        else
        begin
            zeros[10] = 0;
        end
        negatives[10] = 0;
        //算术右移
        results[11] = $signed(b) >>> a;
        if (results[11] == 0)
        begin
            zeros[11] = 1;
        end
        else
        begin
            zeros[11] = 0;
        end
        if (a != 0)
        begin
            if ($signed(a) <= 32)
            begin
                carrys[11] = b[a - 1];
            end
            else
            begin
                carrys[11] = b[31];
            end
        end
        if (results[11][31] == 1)
        begin
            negatives[11] = 1;
        end
        else
        begin
            negatives[11] = 0;
        end
        //逻辑左移/算术左移
        results[12] = b << a;
        if (results[12] == 32'h00000000)
```

```
            begin
                zeros[12] = 1;
            end
            else
            begin
                zeros[12] = 0;
            end
            if (a != 0)
            begin
                if ($signed(a) <= 32)
                begin
                    carrys[12] = b[32 - a];
                end
                else
                begin
                    carrys[12] = 0;
                end
            end
            if (results[12][31] == 1)
            begin
                negatives[12] = 1;
            end
            else
            begin
                negatives[12] = 0;
            end
            //逻辑右移
            results[13] = b >> a;
            if (results[13] == 32'h00000000)
            begin
                zeros[13] = 1;
            end
            else
            begin
                zeros[13] = 0;
            end
            if (a != 0)
            begin
                if ($signed(a) <= 32)
                begin
                    carrys[13] = b[a - 1];
                end
                else
                begin
                    carrys[13] = 0;
                end
            end
            if (results[13][31] == 1)
            begin
                negatives[13] = 1;
            end
            else
            begin
                negatives[13] = 0;
            end
            if (aluc == 4'b0000)//无符号加
            begin
                r = results[0];
                zero = zeros[0];
                carry = carrys[0];
                negative = negatives[0];
                overflow = overflows[0];
            end
            else if (aluc == 4'b0010)//有符号加
            begin
                r = results[1];
                zero = zeros[1];
                carry = carrys[1];
                negative = negatives[1];
                overflow = overflows[1];
            end
            else if (aluc == 4'b0001)//无符号减
            begin
                r = results[2];
                zero = zeros[2];
                carry = carrys[2];
                negative = negatives[2];
                overflow = overflows[2];
            end
            else if (aluc == 4'b0011)//有符号减
            begin
                r = results[3];
                zero = zeros[3];
                carry = carrys[3];
```

```
            negative = negatives[3];
            overflow = overflows[3];
        end
        else if (aluc == 4'b0100)//and
        begin
            r = results[4];
            zero = zeros[4];
            carry = carrys[4];
            negative = negatives[4];
            overflow = overflows[4];
        end
        else if (aluc == 4'b0101)//or
        begin
            r = results[5];
            zero = zeros[5];
            carry = carrys[5];
            negative = negatives[5];
            overflow = overflows[5];
        end
        else if (aluc == 4'b0110)//xor
        begin
            r = results[6];
            zero = zeros[6];
            carry = carrys[6];
            negative = negatives[6];
            overflow = overflows[6];
        end
        else if (aluc == 4'b0111)//nor
        begin
            r = results[7];
            zero = zeros[7];
            carry = carrys[7];
            negative = negatives[7];
            overflow = overflows[7];
        end
        else if (aluc == 4'b1000 || aluc
== 4'b1001)//lui
        begin
            r = results[8];
            zero = zeros[8];
            carry = carrys[8];
            negative = negatives[8];
            overflow = overflows[8];
        end
        else if (aluc == 4'b1011)//slt
        begin
            r = results[9];
            zero = zeros[9];
            carry = carrys[9];
            negative = negatives[9];
            overflow = overflows[9];
        end
        else if (aluc == 4'b1010)//sltu
        begin
            r = results[10];
            zero = zeros[10];
            carry = carrys[10];
            negative = negatives[10];
            overflow = overflows[10];
        end
        else if (aluc == 4'b1100)//sra
        begin
            r = results[11];
            zero = zeros[11];
            carry = carrys[11];
            negative = negatives[11];
            overflow = overflows[11];
        end
        else if (aluc == 4'b1110 || aluc
== 4'b1111)//sll
        begin
            r = results[12];
            zero = zeros[12];
            carry = carrys[12];
            negative = negatives[12];
            overflow = overflows[12];
        end
        else//srl
        begin
            r = results[13];
            zero = zeros[13];
            carry = carrys[13];
            negative = negatives[13];
            overflow = overflows[13];
        end
    end
endmodule
```

3. **寄存器堆：**

```
module regfile(
    input clk,
    input rst,
    input we,
    input [4:0] raddr1,
    input [4:0] raddr2,
    input [4:0] waddr,
    input [31:0] wdata,
    output [31:0] rdata1,
    output [31:0] rdata2
    );
    reg [31 : 0] array_reg [31 : 0];
    assign rdata1 = array_reg[raddr1];
    assign rdata2 = array_reg[raddr2];
    initial
    begin
        array_reg[0] = 0;
        array_reg[1] = 0;
        array_reg[2] = 0;
        array_reg[3] = 0;
        array_reg[4] = 0;
        array_reg[5] = 0;
        array_reg[6] = 0;
        array_reg[7] = 0;
        array_reg[8] = 0;
        array_reg[9] = 0;
        array_reg[10] = 0;
        array_reg[11] = 0;
        array_reg[12] = 0;
        array_reg[13] = 0;
        array_reg[14] = 0;
        array_reg[15] = 0;
        array_reg[16] = 0;
        array_reg[17] = 0;
        array_reg[18] = 0;
        array_reg[19] = 0;
        array_reg[20] = 0;
        array_reg[21] = 0;
        array_reg[22] = 0;
        array_reg[23] = 0;
        array_reg[24] = 0;
        array_reg[25] = 0;
        array_reg[26] = 0;
        array_reg[27] = 0;
        array_reg[28] = 0;
        array_reg[29] = 0;
        array_reg[30] = 0;
        array_reg[31] = 0;
    end
    always @ (posedge clk)
    begin
        if (rst == 1)
        begin
            array_reg[0] = 0;
            array_reg[1] = 0;
            array_reg[2] = 0;
            array_reg[3] = 0;
            array_reg[4] = 0;
            array_reg[5] = 0;
            array_reg[6] = 0;
            array_reg[7] = 0;
            array_reg[8] = 0;
            array_reg[9] = 0;
            array_reg[10] = 0;
            array_reg[11] = 0;
            array_reg[12] = 0;
            array_reg[13] = 0;
            array_reg[14] = 0;
            array_reg[15] = 0;
            array_reg[16] = 0;
            array_reg[17] = 0;
            array_reg[18] = 0;
            array_reg[19] = 0;
            array_reg[20] = 0;
            array_reg[21] = 0;
            array_reg[22] = 0;
            array_reg[23] = 0;
            array_reg[24] = 0;
            array_reg[25] = 0;
            array_reg[26] = 0;
            array_reg[27] = 0;
            array_reg[28] = 0;
            array_reg[29] = 0;
            array_reg[30] = 0;
            array_reg[31] = 0;
        end
```

```
        else                                    wdata;
        begin                                           end
            if (we == 1 && waddr !=                     end
0)                                              end
            begin                               endmodule
                array_reg[waddr]    =
```

4. **将几位数拼接为位数更多的部件：**

```
module concatenator(
    input [25 : 0] offset,
    input [3 : 0] pc,
    output [31 : 0] r
    );
    assign r = {{pc}, {offset}, {2'b00}};
endmodule
```

5. **加法器：**

```
module add(
    input [31 : 0] a,
    input [31 : 0] b,
    output [31 : 0] r
    );
    assign r = a + b;
endmodule
```

6. **位数扩展器：**

```
module ext #(parameter WIDTH = 16)(
    input [WIDTH - 1 : 0] a,
    input sext,
    output [31 : 0] b
    );
    assign b = {sext ? {32 - WIDTH{a[WIDTH - 1]}} : {32 - WIDTH{1'b0}}, a[WIDTH -
1 : 0]};
endmodule
```

7. **二选一数据选择器：**

```
module mux2(
    input [31 : 0] a,
    input [31 : 0] b,
    input s,
    output [31 : 0] r
    );
    assign r = s ? b : a;
endmodule
```

8. **四选一数据选择器：**

```
module mux4(                                    input [31 : 0] b,
    input [31 : 0] a,                           input [31 : 0] c,
```

```
    input [31 : 0] d,
    input [1 : 0] s,
    output reg [31:0] r
    );
    always @ (*)
    begin
        if (s == 0)
        begin
            r = a;
        end
        else if (s == 1)
        begin
            r = b;
        end
        else if (s == 2)
        begin
            r = c;
        end
        else
        begin
            r = d;
        end
    end
endmodule
```

9. ram：

```
module ram(
    input clk,
    input ram_ena,
    input wena,
    input [10 : 0] addr,
    input [31 : 0] data_in,
    output [31 : 0] data_out
    );
    reg [31 : 0] data [2047 : 0];

    assign data_out = ram_ena ? data[addr] : 32'bzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz;

    always @ (posedge clk)
    begin
        if (ram_ena == 1 && wena == 1)
        begin
            data[addr] = data_in;
        end
    end
endmodule
```

六、控制模块建模以及 CPU 建模

1. **控制模块建模：**

```
module cpu31controller(
    input [31 : 0] instruction,
    input zero,
    output reg pcreg_ena,
    output reg d_ram_wena,
    output reg d_ram_ena,
    output reg ext5_s,
```

```
        output reg ext16_s,
        output reg ext18_s,
        output reg rf_we,
        output reg [4 : 0] rf_waddr,
        output reg [4 : 0] rf_raddr1,
        output reg [4 : 0] rf_raddr2,
        output reg [3 : 0] aluc,
        output reg M1,
        output reg M2,
        output [1 : 0] M3,
        output reg [1 : 0] M4
        );
        reg [30 : 0] op;
        assign M3 = op[16] ? 2'b00 : ((op[29] || op[30]) ? 2'b01 : (op[24] ? {{1'b1}, {zero}} :
(op[25] ? {{1'b1}, {~zero}} : 2'b10)));
        always @ (instruction)
        begin
            if (instruction[31 : 26] == 6'b000000)
            begin
                if (instruction[5 : 0] == 6'b100000)
                begin
                    op = 31'b0000000000000000000000000000001;
                    aluc = 4'b0010;
                end
                else if (instruction[5 : 0] == 6'b100001)
                begin
                    op = 31'b0000000000000000000000000000010;
                    aluc = 4'b0000;
                end
                else if (instruction[5 : 0] == 6'b100010)
                begin
                    op = 31'b0000000000000000000000000000100;
                    aluc = 4'b0011;
                end
                else if (instruction[5 : 0] == 6'b100011)
                begin
                    op = 31'b0000000000000000000000000001000;
                    aluc = 4'b0001;
                end
                else if (instruction[5 : 0] == 6'b100100)
                begin
                    op = 31'b0000000000000000000000000010000;
                    aluc = 4'b0100;
                end
```

```
else if (instruction[5 : 0] == 6'b100101)
begin
	op = 31'b00000000000000000000000000100000;
	aluc = 4'b0101;
end
else if (instruction[5 : 0] == 6'b100110)
begin
	op = 31'b00000000000000000000000001000000;
	aluc = 4'b0110;
end
else if (instruction[5 : 0] == 6'b100111)
begin
	op = 31'b00000000000000000000000010000000;
	aluc = 4'b0111;
end
else if (instruction[5 : 0] == 6'b101010)
begin
	op = 31'b00000000000000000000000100000000;
	aluc = 4'b1011;
end
else if (instruction[5 : 0] == 6'b101011)
begin
	op = 31'b00000000000000000000001000000000;
	aluc = 4'b1010;
end
else if (instruction[5 : 0] == 6'b000000)
begin
	op = 31'b00000000000000000000010000000000;
	aluc = 4'b1110;
end
else if (instruction[5 : 0] == 6'b000010)
begin
	op = 31'b00000000000000000000100000000000;
	aluc = 4'b1101;
end
else if (instruction[5 : 0] == 6'b000011)
begin
	op = 31'b00000000000000000001000000000000;
	aluc = 4'b1100;
end
else if (instruction[5 : 0] == 6'b000100)
begin
	op = 31'b00000000000000000010000000000000;
	aluc = 4'b1110;
```

```
        end
        else if (instruction[5 : 0] == 6'b000110)
        begin
            op = 31'b0000000000000000100000000000000;
            aluc = 4'b1101;
        end
        else if (instruction[5 : 0] == 6'b000111)
        begin
            op = 31'b0000000000000001000000000000000;
            aluc = 4'b1100;
        end
        else if (instruction[5 : 0] == 6'b001000)
        begin
            op = 31'b0000000000000010000000000000000;
            aluc = 0;
        end
    end
    else if (instruction[31 : 26] == 6'b000010)
    begin
        op = 31'b0100000000000000000000000000000;
        aluc = 0;
    end
    else if (instruction[31 : 26] == 6'b000011)
    begin
        op = 31'b1000000000000000000000000000000;
        aluc = 0;
    end
    else if (instruction[31 : 26] == 6'b001000)
    begin
        op = 31'b0000000000000100000000000000000;
        aluc = 4'b0010;
    end
    else if (instruction[31 : 26] == 6'b001001)
    begin
        op = 31'b0000000000001000000000000000000;
        aluc = 4'b0000;
    end
    else if (instruction[31 : 26] == 6'b001100)
    begin
        op = 31'b0000000000010000000000000000000;
        aluc = 4'b0100;
    end
    else if (instruction[31 : 26] == 6'b001101)
    begin
```

```
        op = 31'b00000000001000000000000000000000;
        aluc = 4'b0101;
end
else if (instruction[31 : 26] == 6'b001110)
begin
        op = 31'b00000000010000000000000000000000;
        aluc = 4'b0110;
end
else if (instruction[31 : 26] == 6'b100011)
begin
        op = 31'b00000000100000000000000000000000;
        aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b101011)
begin
        op = 31'b00000001000000000000000000000000;
        aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b000100)
begin
        op = 31'b00000010000000000000000000000000;
        aluc = 4'b0011;
end
else if (instruction[31 : 26] == 6'b000101)
begin
        op = 31'b00000100000000000000000000000000;
        aluc = 4'b0011;
end
else if (instruction[31 : 26] == 6'b001010)
begin
        op = 31'b00001000000000000000000000000000;
        aluc = 4'b1011;
end
else if (instruction[31 : 26] == 6'b001011)
begin
        op = 31'b00010000000000000000000000000000;
        aluc = 4'b1010;
end
else
begin
        op = 31'b00100000000000000000000000000000;
        aluc = 4'b1000;
end
pcreg_ena = 1;
```

```
        d_ram_wena = op[23];
        d_ram_ena = op[22] || op[23];
        ext5_s = 0;
        ext16_s = op[17] || op[18] || op[22] || op[23] || op[26] || op[27];
        ext18_s = op[24] || op[25];
        rf_we = ~(op[16] || op[23] || op[24] || op[25] || op[29]);
        if (op[17] || op[18] || op[19] || op[20] || op[21] || op[22] || op[26] || op[27] || op[28])
        begin
            rf_waddr = instruction[20 : 16];
        end
        else if (op[16] || op[23] || op[24] || op[25])
        begin
            rf_waddr = 0;
        end
        else if (op[30])
        begin
            rf_waddr = 5'b11111;
        end
        else
        begin
            rf_waddr = instruction[15 : 11];
        end
        if (op[10] || op[11] || op[12] || op[28] || op[29] || op[30])
        begin
            rf_raddr1 = 0;
        end
        else
        begin
            rf_raddr1 = instruction[25 : 21];
        end
        if (op[16] || op[17] || op[18] || op[19] || op[20] || op[21] || op[22] || op[26] || op[27] ||
op[28] || op[29] || op[30])
        begin
            rf_raddr2 = 0;
        end
        else
        begin
            rf_raddr2 = instruction[20 : 16];
        end
        M1 = op[17] || op[18] || op[19] || op[20] || op[21] || op[22] || op[23] || op[26] || op[27]
|| op[28];
        M2 = ~(op[10] || op[11] || op[12] || op[16] || op[28] || op[29] || op[30]);
        if (op[16] || op[22] || op[23] || op[24] || op[25] || op[29])
        begin
```

```
            M4 = 2'b00;
        end
        else if (op[30])
        begin
            M4 = 2'b10;
        end
        else
        begin
            M4 = 2'b01;
        end
        if (instruction == 32'hffffffff)
        begin
            pcreg_ena = 0;
            rf_we = 0;
        end
    end
endmodule
```

2. cpu 建模

```
module cpu(
    input clk,
    input reset,
    input [31 : 0] inst,
    input [31 : 0] dmem_out,
    output [31 : 0] pc,
    output dmem_ena,
    output dmem_wena,
    output [31 : 0] dmem_addr,
    output [31 : 0] dmem_in
    );
    wire [31 : 0] vpc;
    wire [31 : 0] valu;
    wire zero;
    wire carry;
    wire negative;
    wire overflow;
    wire [31 : 0] vreg1;
    wire [31 : 0] vreg2;

    wire [31 : 0] vconcat;
    wire [31 : 0] vaddj;
    wire [31 : 0] vnpc;
    wire [31 : 0] vpc_8;
    wire [31 : 0] vext18;
    wire [31 : 0] vext16;
```

```
    wire [31 : 0] vext5;

    wire [31 : 0] vmux_1;
    wire [31 : 0] vmux_2;
    wire [31 : 0] vmux_3;
    wire [31 : 0] vmux_4;

    wire pcreg_ena;
    wire ext5_s;
    wire ext16_s;
    wire ext18_s;
    wire reg_we;
    wire [4 : 0] reg_waddr;
    wire [4 : 0] reg_raddr1;
    wire [4 : 0] reg_raddr2;
    wire [3 : 0] aluc;
    wire M1;
    wire M2;
    wire [1 : 0] M3;
    wire [1 : 0] M4;

    assign pc = vpc;
    assign dmem_addr = valu;
    assign dmem_in = vreg2;

    cpu31controller cpu31controller_inst(inst, zero, pcreg_ena, dmem_wena, dmem_ena, ext5_s, ext16_s, ext18_s, reg_we, reg_waddr, reg_raddr1, reg_raddr2, aluc, M1, M2, M3, M4);

    pcreg pcreg_inst(clk, reset, pcreg_ena, vmux_3, vpc);
    alu alu_inst(vmux_2, vmux_1, aluc, valu, zero, carry, negative, overflow);
    regfile cpu_ref(clk, reset, reg_we, reg_raddr1, reg_raddr2, reg_waddr, vmux_4, vreg1, vreg2);

    concatenator concat(inst[25 : 0], vpc[31 : 28], vconcat);
    add addj(vnpc, vext18, vaddj);
    add npc(vpc, 4, vnpc);
    add pc_8(vpc, 4, vpc_8);
    ext #(.WIDTH(5)) ext5(inst[10 : 6], ext5_s, vext5);
    ext #(.WIDTH(16)) ext16(inst[15 : 0], ext16_s, vext16);
    ext #(.WIDTH(18)) ext18({{inst[15 : 0]}, {2'b00}}, ext18_s, vext18);

    mux2 mux_1(vreg2, vext16, M1, vmux_1);
    mux2 mux_2(vext5, vreg1, M2, vmux_2);
```

```
        mux4 mux_3(vreg1, vconcat, vnpc, vaddj, M3, vmux_3);
        mux4 mux_4(dmem_out, valu, vpc_8, 0, M4, vmux_4);
    endmodule
```

七、顶层模块设计

```
module sccomp_dataflow(
    input clk_in,
    input reset,
    output [31:0] inst,
    output [31:0] pc
    );
    wire [31 : 0] dmem_out;
    wire dmem_ena;
    wire dmem_wena;
    wire [31 : 0] dmem_addr;
    wire [31 : 0] dmem_in;
    wire [31 : 0] instruction;
    reg [25:0] cnt;

    assign inst = instruction;

    cpu sccpu(clk_in, reset, instruction, dmem_out, pc, dmem_ena, dmem_wena, dmem_addr, dmem_in);
    imem im(pc[12 : 2], instruction);
    ram dmem(clk_in, dmem_ena, dmem_wena, dmem_addr[12 : 0], dmem_in, dmem_out);
endmodule
```

八、前仿真测试

我在控制器中添加了当指令读取到 ffffffff 时关闭 PC 寄存器和寄存器堆的写入信号，并在 coe 文件的最后添加了一条 ffffffff 指令，因此 PC 寄存器和寄存器堆里的值能维持在最后一条指令执行后的状态，只需观察波形的最后即可判断前仿真结果（我也将每个周期的结果都打印出来进行比对，但是不便于放置在报告中，所以就只能展示几条代表性指令的波形）

测试文件如下：

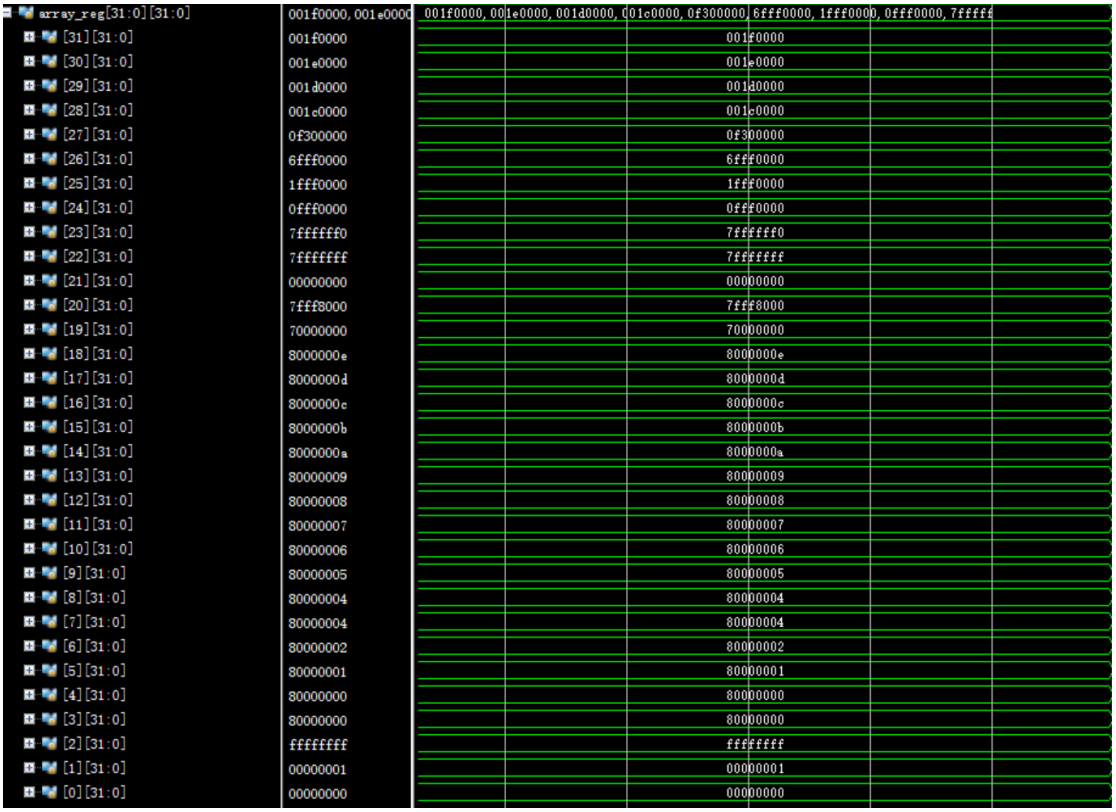
```
module scc_tb();
    integer result;
    reg clk_in;
    reg reset;
    wire [31:0] inst;
    wire [31:0] pc;
    reg [10 : 0] count;
    sccomp_dataflow sccomp_dataflow_inst(clk_in, reset, inst, pc);
    initial
```

```
begin
    result = $fopen("result.txt", "w");
    clk_in = 0;
    reset = 0;
    count = 0;
end
always
begin
    #5 clk_in = ~clk_in;
    if (clk_in == 1)
    begin
        if (count == 1050)
        begin
            $fclose(result);
        end
        count = count + 1;
        $fdisplay(result, "count: %d", count);
        $fdisplay(result, "pc: %h", pc);
        $fdisplay(result, "instr: %h", inst);
        $fdisplay(result, "reg0: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[0]);
        $fdisplay(result, "reg1: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[1]);
        $fdisplay(result, "reg2: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[2]);
        $fdisplay(result, "reg3: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[3]);
        $fdisplay(result, "reg4: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[4]);
        $fdisplay(result, "reg5: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[5]);
        $fdisplay(result, "reg6: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[6]);
        $fdisplay(result, "reg7: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[7]);
        $fdisplay(result, "reg8: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[8]);
        $fdisplay(result, "reg9: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[9]);
        $fdisplay(result, "reg10: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[10]);
        $fdisplay(result, "reg11: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[11]);
        $fdisplay(result, "reg12: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[12]);
        $fdisplay(result, "reg13: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[13]);
        $fdisplay(result, "reg14: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[14]);
        $fdisplay(result, "reg15: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[15]);
        $fdisplay(result, "reg16: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[16]);
        $fdisplay(result, "reg17: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[17]);
        $fdisplay(result, "reg18: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[18]);
        $fdisplay(result, "reg19: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[19]);
        $fdisplay(result, "reg20: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[20]);
        $fdisplay(result, "reg21: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[21]);
        $fdisplay(result, "reg22: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[22]);
        $fdisplay(result, "reg23: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[23]);
        $fdisplay(result, "reg24: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[24]);
```

```
            $fdisplay(result, "reg25: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[25]);
            $fdisplay(result, "reg26: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[26]);
            $fdisplay(result, "reg27: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[27]);
            $fdisplay(result, "reg28: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[28]);
            $fdisplay(result, "reg29: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[29]);
            $fdisplay(result, "reg30: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[30]);
            $fdisplay(result, "reg31: %h", sccomp_dataflow_inst.sccpu.cpu_ref.array_reg[31]);
            $fdisplay(result, "********");
        end
    end
endmodule
```

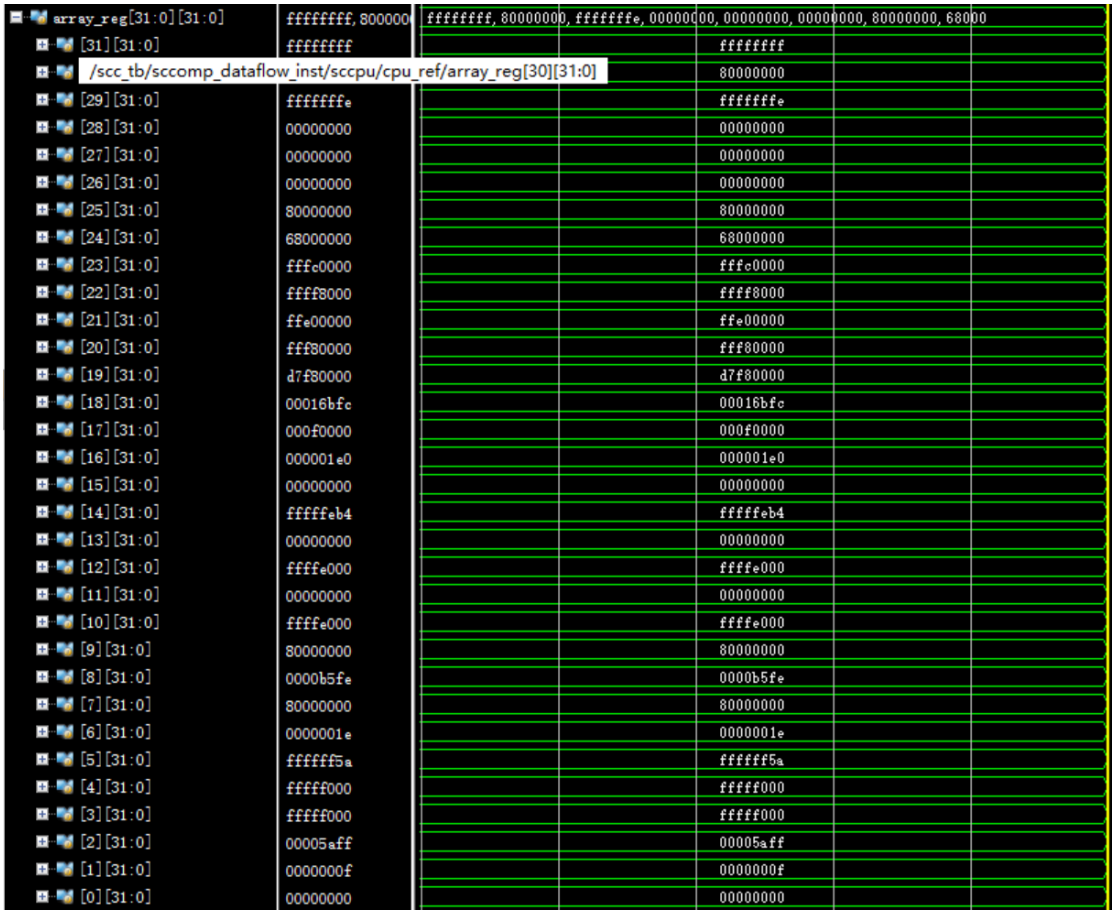
1. **add:**

   结果：



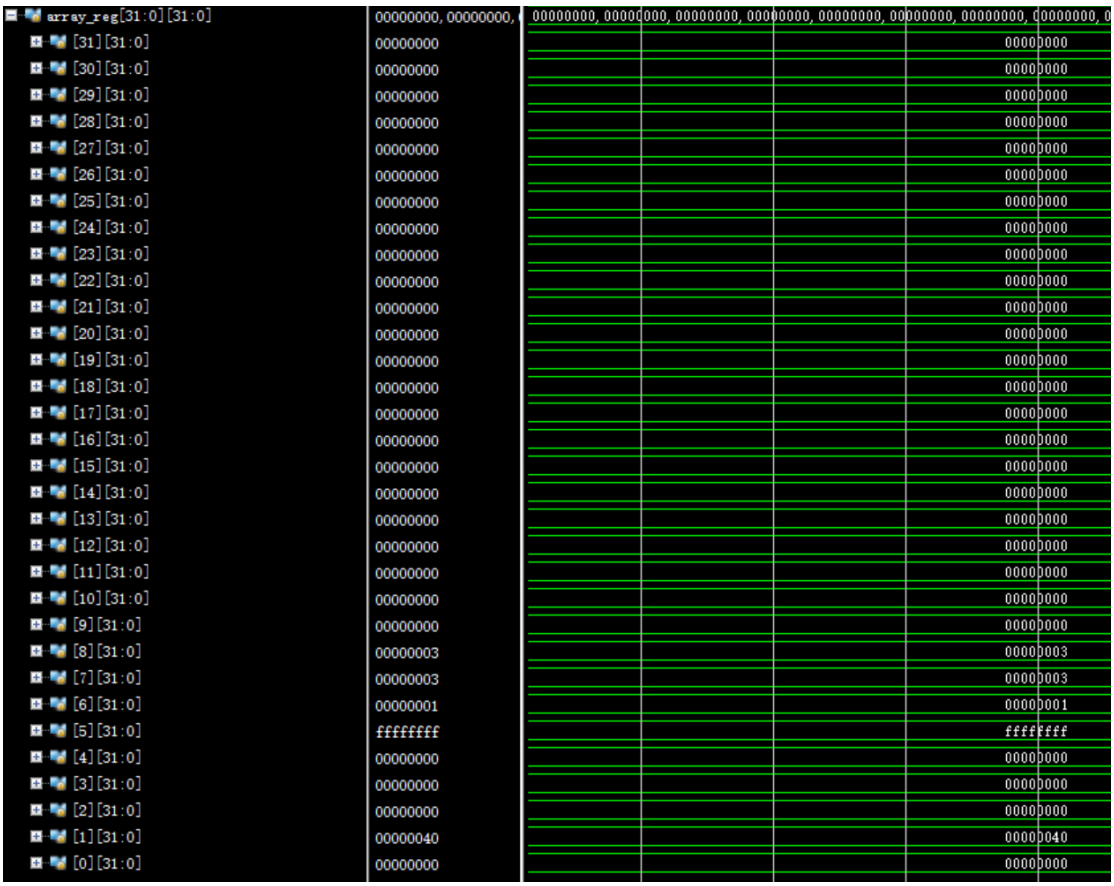
2. **sll:**

   结果：


array_reg[31:0][31:0]
ffffffff, 800000
ffffffff, 80000000, fffffffe, 00000000, 00000000, 00000000, 80000000, 68000
[31][31:0]
ffffffff
ffffffff
/scc_tb/sccomp_dataflow_inst/sccpu/cpu_ref/array_reg[30][31:0]
[29][31:0]
fffffffe
fffffffe
[28][31:0]
00000000
00000000
[27][31:0]
00000000
00000000
[26][31:0]
00000000
00000000
[25][31:0]
80000000
80000000
[24][31:0]
68000000
68000000
[23][31:0]
fffc0000
fffc0000
[22][31:0]
ffff8000
ffff8000
[21][31:0]
ffe00000
ffe00000
[20][31:0]
fff80000
fff80000
[19][31:0]
d7f80000
d7f80000
[18][31:0]
00016bfc
00016bfc
[17][31:0]
000f0000
000f0000
[16][31:0]
000001e0
000001e0
[15][31:0]
00000000
00000000
[14][31:0]
fffffeb4
fffffeb4
[13][31:0]
00000000
00000000
[12][31:0]
ffffe000
ffffe000
[11][31:0]
00000000
00000000
[10][31:0]
ffffe000
ffffe000
[9][31:0]
80000000
80000000
[8][31:0]
0000b5fe
0000b5fe
[7][31:0]
80000000
80000000
[6][31:0]
0000001e
0000001e
[5][31:0]
ffffff5a
ffffff5a
[4][31:0]
fffff000
fffff000
[3][31:0]
fffff000
fffff000
[2][31:0]
00005aff
00005aff
[1][31:0]
0000000f
0000000f
[0][31:0]
00000000
00000000

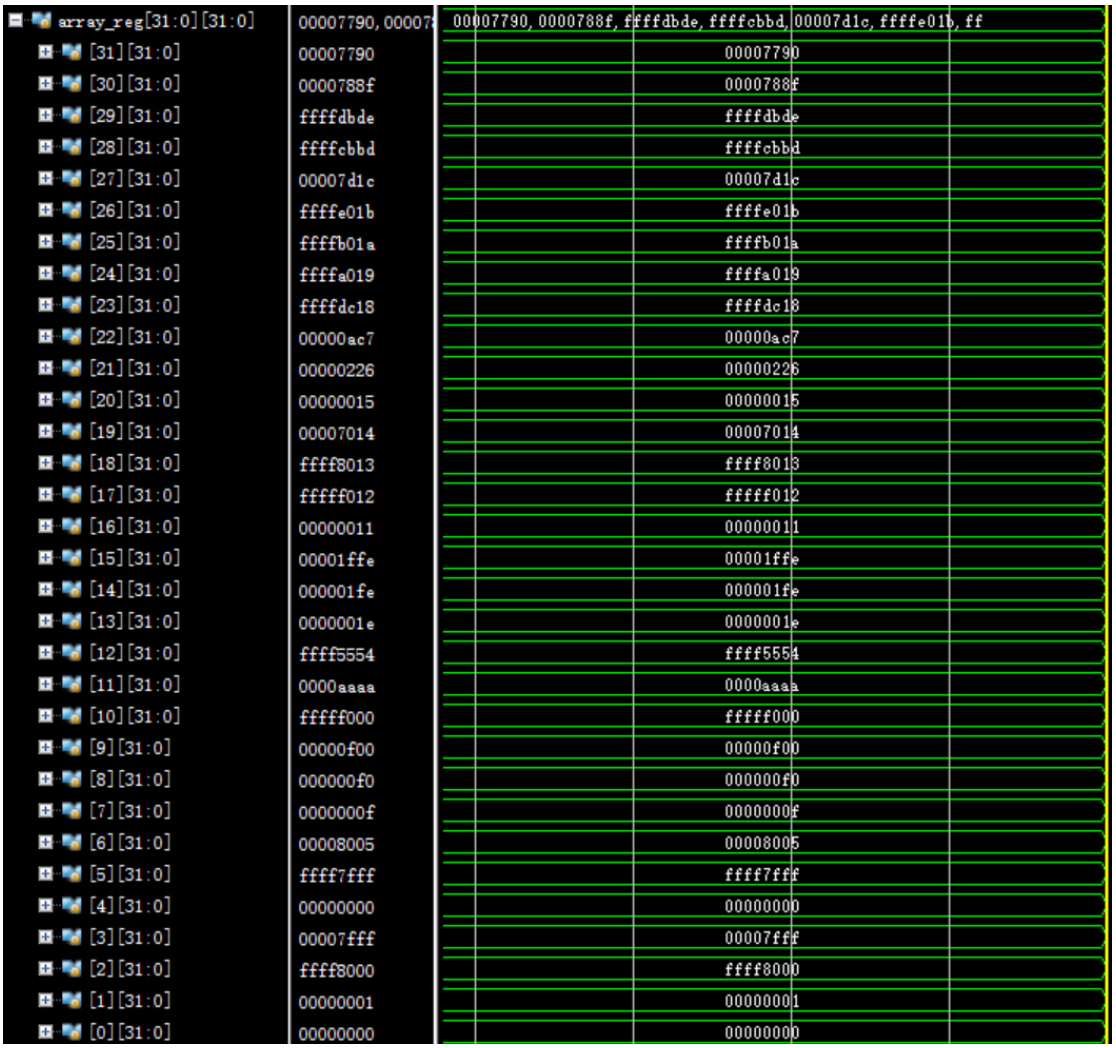
3. **jr:**

结果：


array_reg[31:0][31:0]
00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,
[31][31:0] 00000000 00000000
[30][31:0] 00000000 00000000
[29][31:0] 00000000 00000000
[28][31:0] 00000000 00000000
[27][31:0] 00000000 00000000
[26][31:0] 00000000 00000000
[25][31:0] 00000000 00000000
[24][31:0] 00000000 00000000
[23][31:0] 00000000 00000000
[22][31:0] 00000000 00000000
[21][31:0] 00000000 00000000
[20][31:0] 00000000 00000000
[19][31:0] 00000000 00000000
[18][31:0] 00000000 00000000
[17][31:0] 00000000 00000000
[16][31:0] 00000000 00000000
[15][31:0] 00000000 00000000
[14][31:0] 00000000 00000000
[13][31:0] 00000000 00000000
[12][31:0] 00000000 00000000
[11][31:0] 00000000 00000000
[10][31:0] 00000000 00000000
[9][31:0] 00000000 00000000
[8][31:0] 00000003 00000003
[7][31:0] 00000003 00000003
[6][31:0] 00000001 00000001
[5][31:0] ffffffff ffffffff
[4][31:0] 00000000 00000000
[3][31:0] 00000000 00000000
[2][31:0] 00000000 00000000
[1][31:0] 00000040 00000040
[0][31:0] 00000000 00000000

4. **addi:**

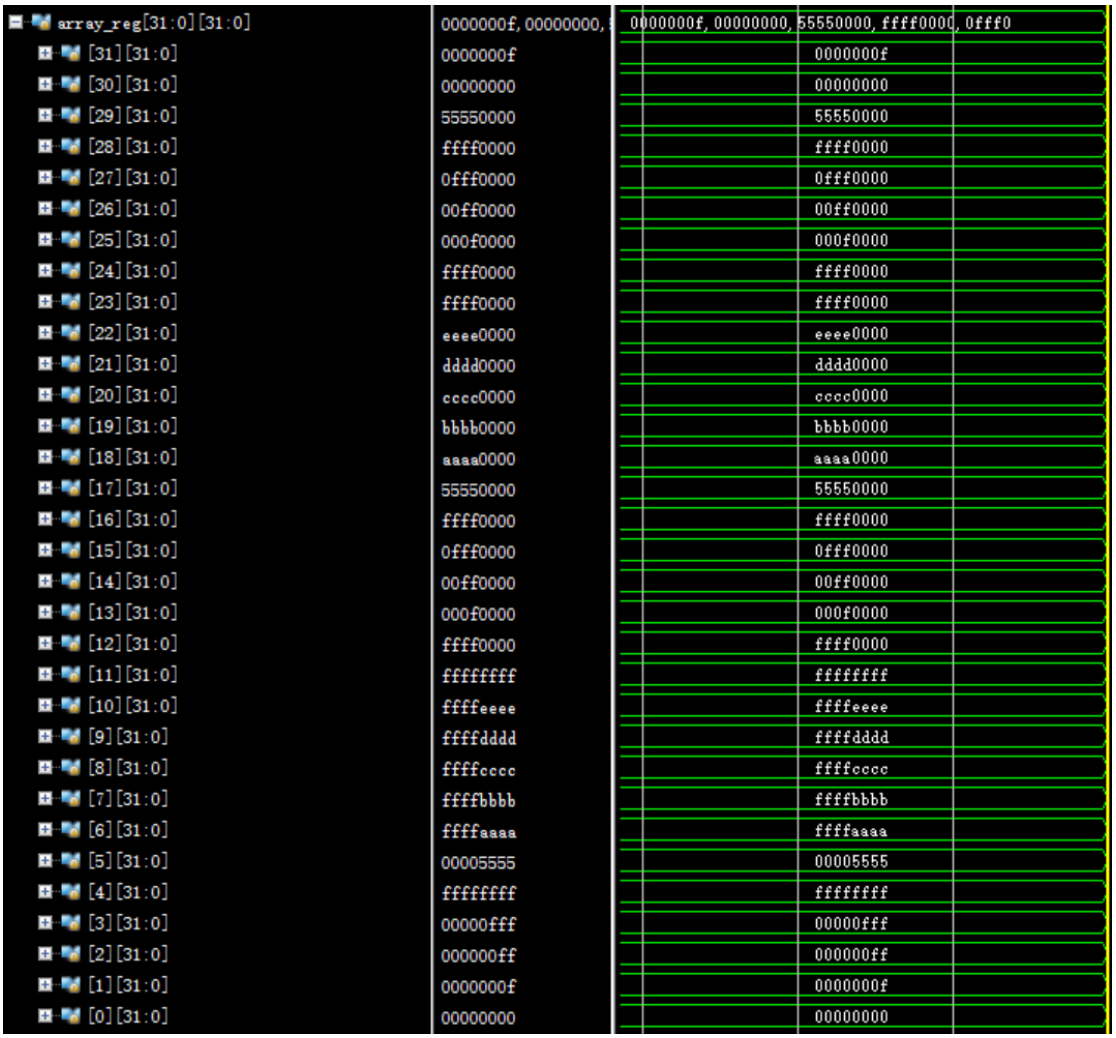
结果：



array_reg[31:0][31:0]	00007790, 00007	00007790, 0000788f, ffffdbde, ffffcbbd, 00007d1c, ffffe01b, ff
[31][31:0]	00007790	00007790
[30][31:0]	0000788f	0000788f
[29][31:0]	ffffdbde	ffffdbde
[28][31:0]	ffffcbbd	ffffcbbd
[27][31:0]	00007d1c	00007d1c
[26][31:0]	ffffe01b	ffffe01b
[25][31:0]	ffffb01a	ffffb01a
[24][31:0]	ffffa019	ffffa019
[23][31:0]	ffffdc18	ffffdc18
[22][31:0]	00000ac7	00000ac7
[21][31:0]	00000226	00000226
[20][31:0]	00000015	00000015
[19][31:0]	00007014	00007014
[18][31:0]	ffff8013	ffff8013
[17][31:0]	fffff012	fffff012
[16][31:0]	00000011	00000011
[15][31:0]	00001ffe	00001ffe
[14][31:0]	000001fe	000001fe
[13][31:0]	0000001e	0000001e
[12][31:0]	ffff5554	ffff5554
[11][31:0]	0000aaaa	0000aaaa
[10][31:0]	fffff000	fffff000
[9][31:0]	00000f00	00000f00
[8][31:0]	000000f0	000000f0
[7][31:0]	0000000f	0000000f
[6][31:0]	00008005	00008005
[5][31:0]	ffff7fff	ffff7fff
[4][31:0]	00000000	00000000
[3][31:0]	00007fff	00007fff
[2][31:0]	ffff8000	ffff8000
[1][31:0]	00000001	00000001
[0][31:0]	00000000	00000000

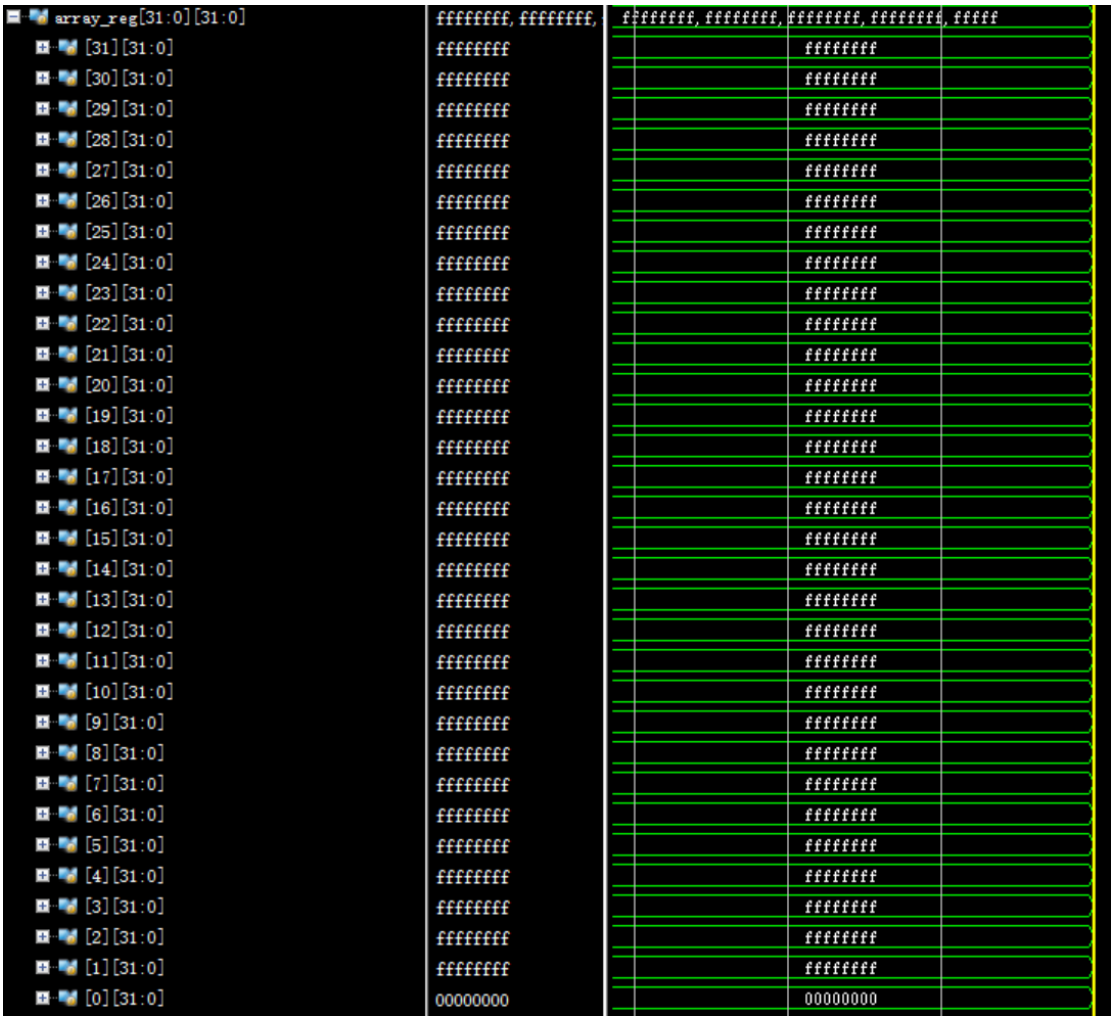
5. **lw&sw:**

   结果：


array_reg[31:0][31:0]
0000000f, 00000000,
0000000f, 00000000, 55550000, ffff0000, 0fff0
[31][31:0] 0000000f 0000000f
[30][31:0] 00000000 00000000
[29][31:0] 55550000 55550000
[28][31:0] ffff0000 ffff0000
[27][31:0] 0fff0000 0fff0000
[26][31:0] 00ff0000 00ff0000
[25][31:0] 000f0000 000f0000
[24][31:0] ffff0000 ffff0000
[23][31:0] ffff0000 ffff0000
[22][31:0] eeee0000 eeee0000
[21][31:0] dddd0000 dddd0000
[20][31:0] cccc0000 cccc0000
[19][31:0] bbbb0000 bbbb0000
[18][31:0] aaaa0000 aaaa0000
[17][31:0] 55550000 55550000
[16][31:0] ffff0000 ffff0000
[15][31:0] 0fff0000 0fff0000
[14][31:0] 00ff0000 00ff0000
[13][31:0] 000f0000 000f0000
[12][31:0] ffff0000 ffff0000
[11][31:0] ffffffff ffffffff
[10][31:0] ffffeeee ffffeeee
[9][31:0] ffffdddd ffffdddd
[8][31:0] ffffcccc ffffcccc
[7][31:0] ffffbbbb ffffbbbb
[6][31:0] ffffaaaa ffffaaaa
[5][31:0] 00005555 00005555
[4][31:0] ffffffff ffffffff
[3][31:0] 00000fff 00000fff
[2][31:0] 000000ff 000000ff
[1][31:0] 0000000f 0000000f
[0][31:0] 00000000 00000000

6. **beq:**

结果：



经过比较仿真结果、寄存器内容的变化过程和 Mars 的执行结果与过程，31 条指令均通过前仿真。

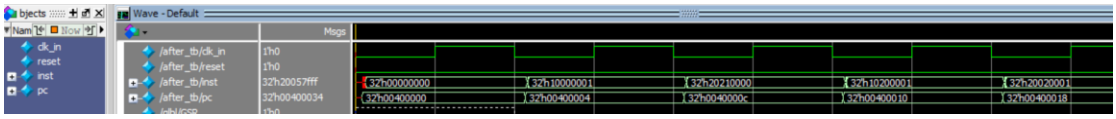
九、后仿真测试

由于后仿真不能打印结果，所以我重新编写了一个测试文件：

```
module after_tb();
    reg clk_in;
    reg reset;
    wire [31:0] inst;
    wire [31:0] pc;
    initial
    begin
        clk_in = 0;
        reset = 0;
    end
    always
        #50 clk_in = ~clk_in;
    sccomp_dataflow sccomp_dataflow_inst(clk_in, reset, inst, pc);
```

endmodule

我测试了下发的 mips_31_mars_simulate.coe 文件，结果如下：



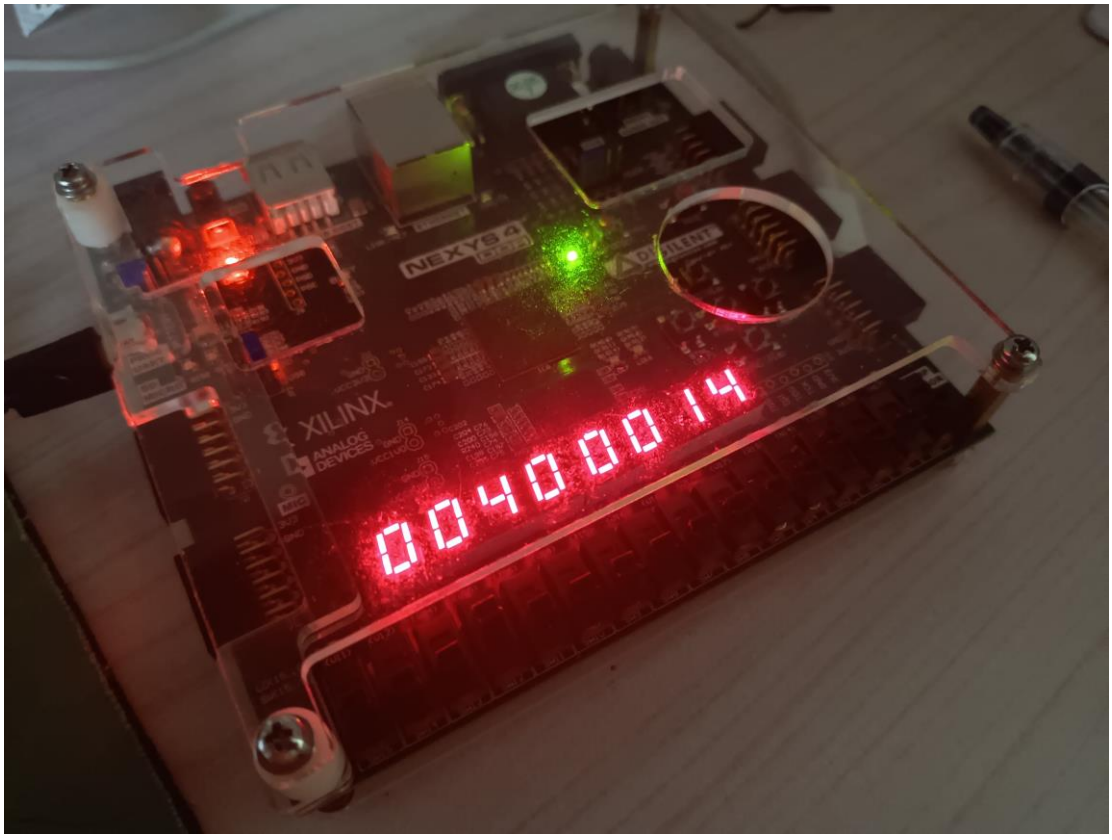
十、下板结果

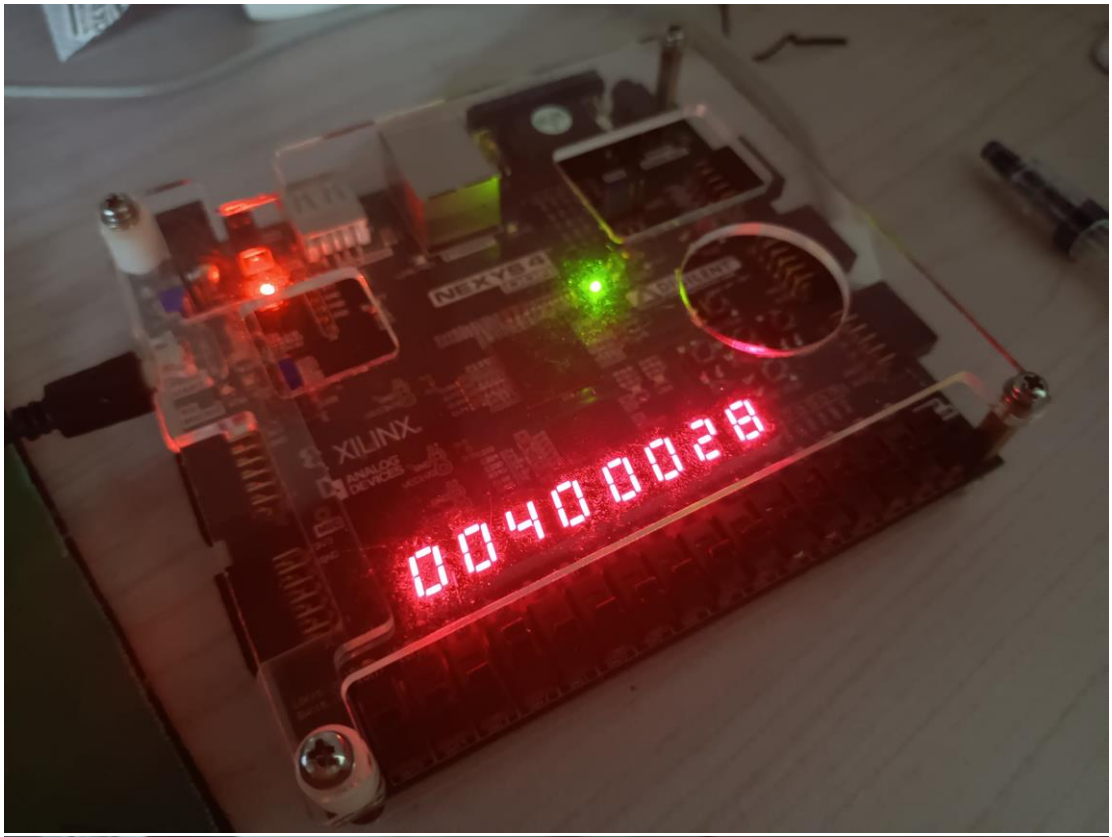
由于下板需要在七段数码管显示结果，为了便于观察，我对 cpu 进行了分频，并添加了顶层模块，如下：

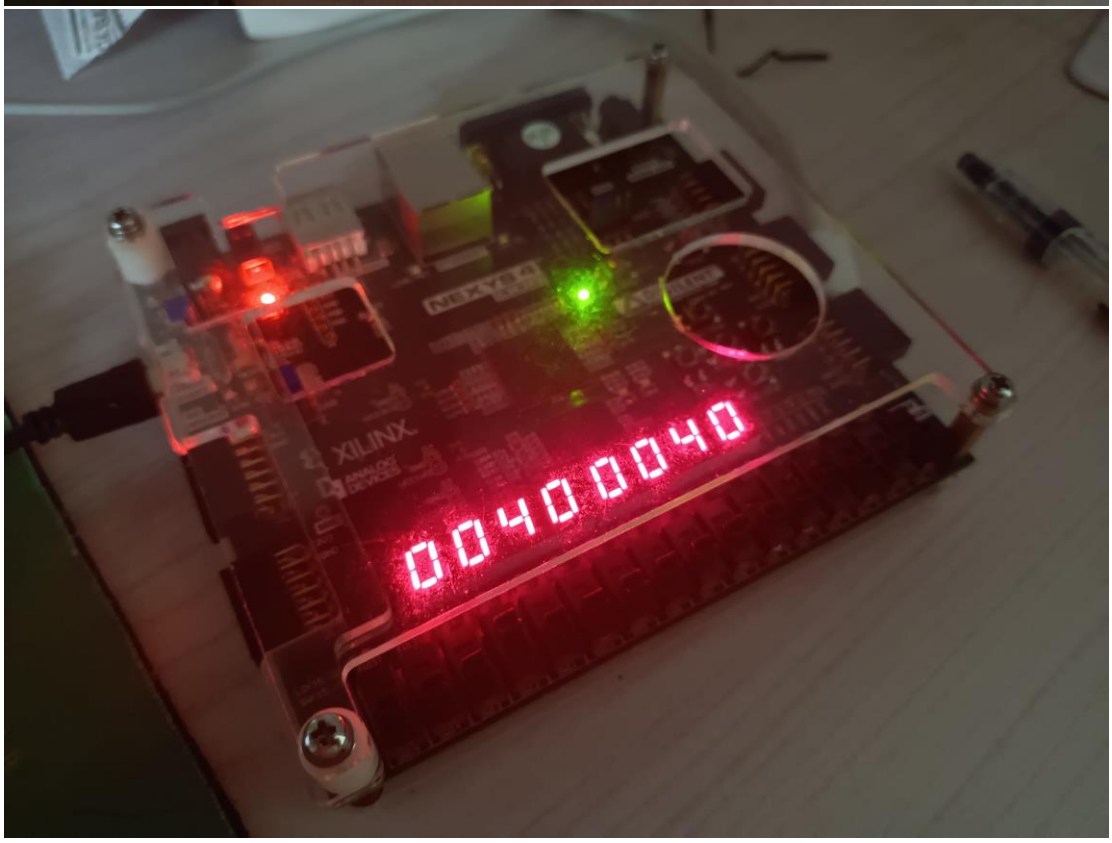
```
module cpu31top(
    input clk_in,
    input reset,
    output [7 : 0] o_seg,
    output [7 : 0] o_sel
    );
    wire [31 : 0] inst;
    wire [31 : 0] pc;
    sccomp_dataflow sccomp_dataflow_inst(clk_in, reset, inst, pc);
    seg7x16(clk_in, reset, 1, inst, o_seg, o_sel);
endmodule
```

下板结果如图所示：

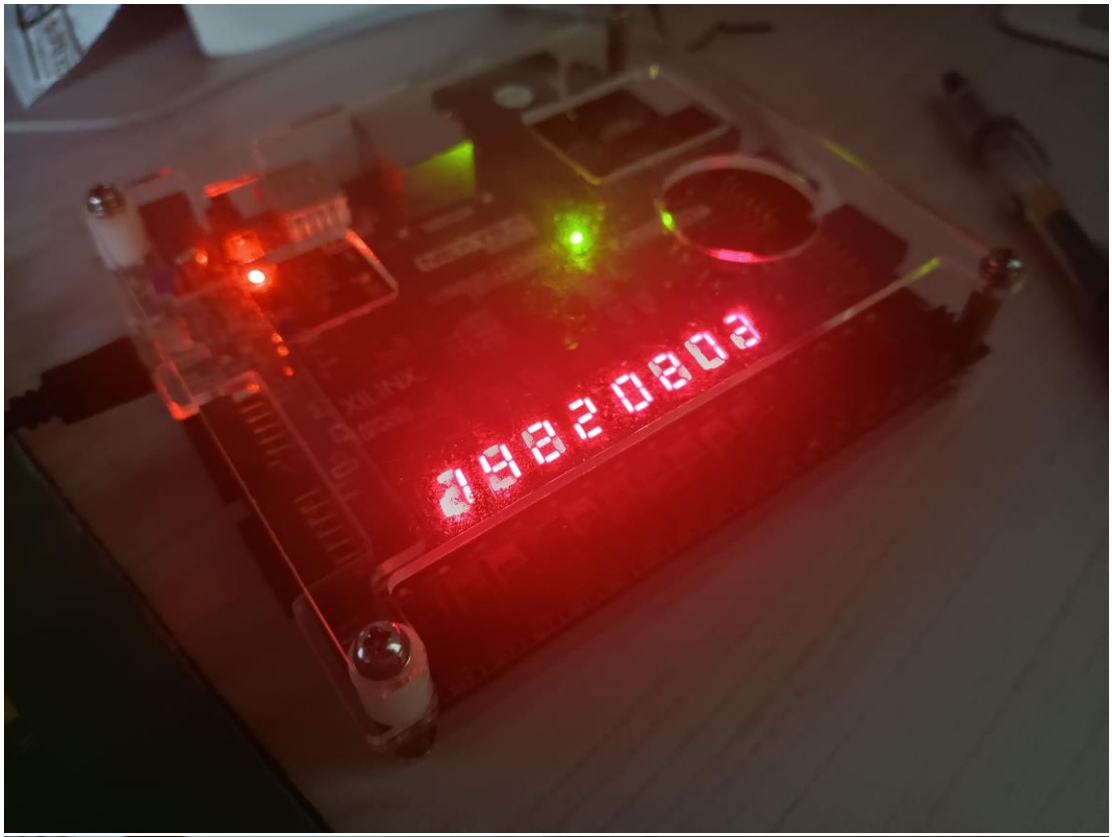
PC 寄存器的值显示：

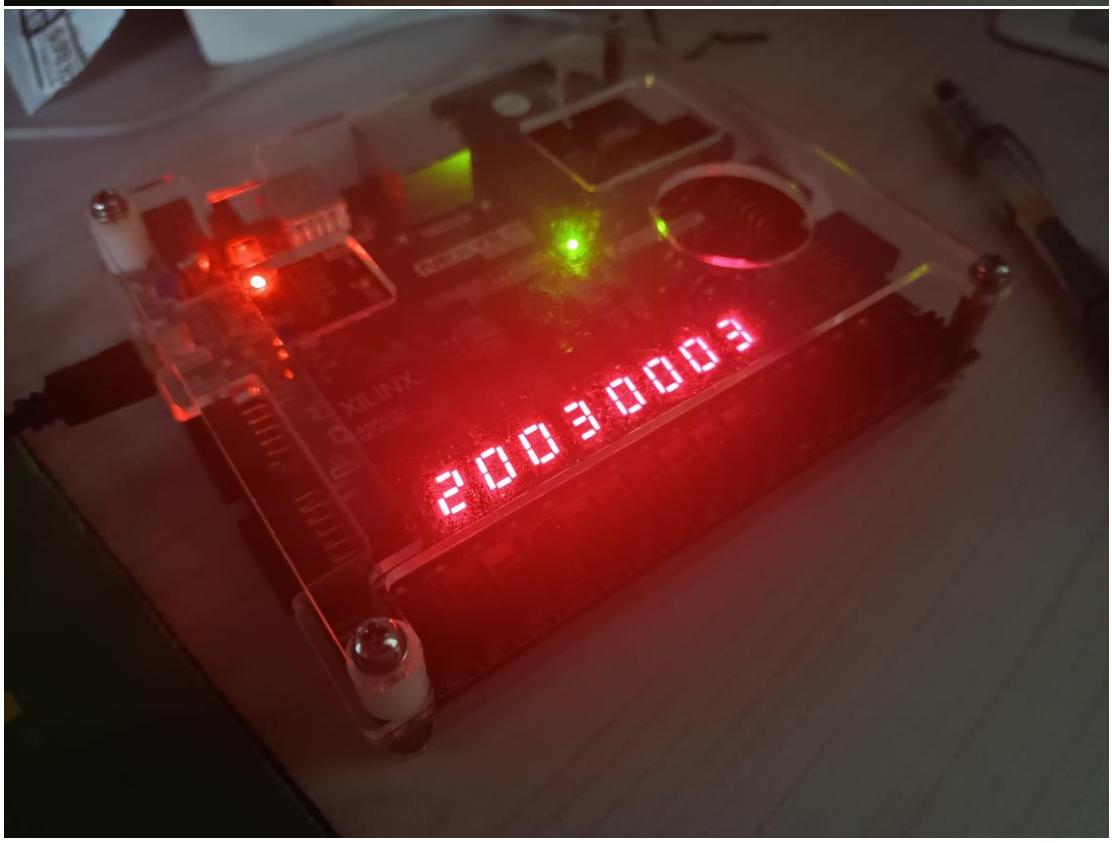


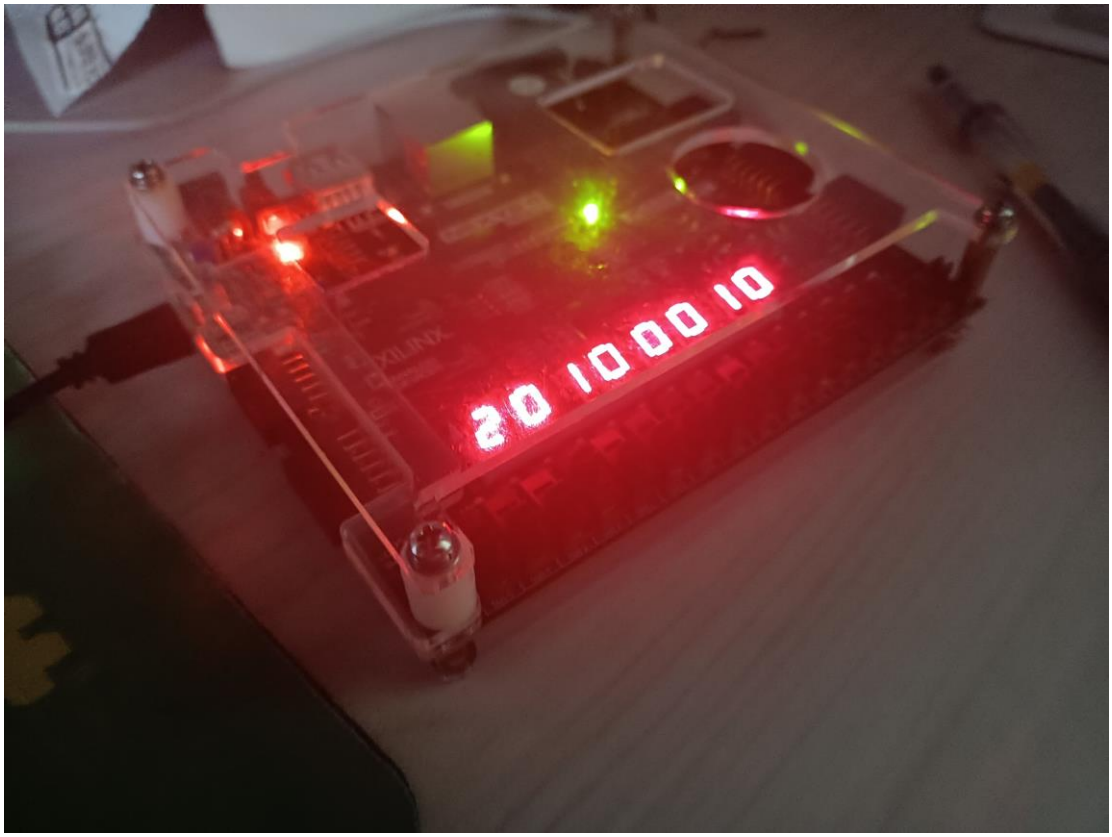




当前指令的值显示：


20220803


20030003


20 10 00 10