同济大学计算机系

计算机组成原理实验报告

学　　号 1953072

姓　　名 肖鹏飞

专　　业 计算机科学与技术

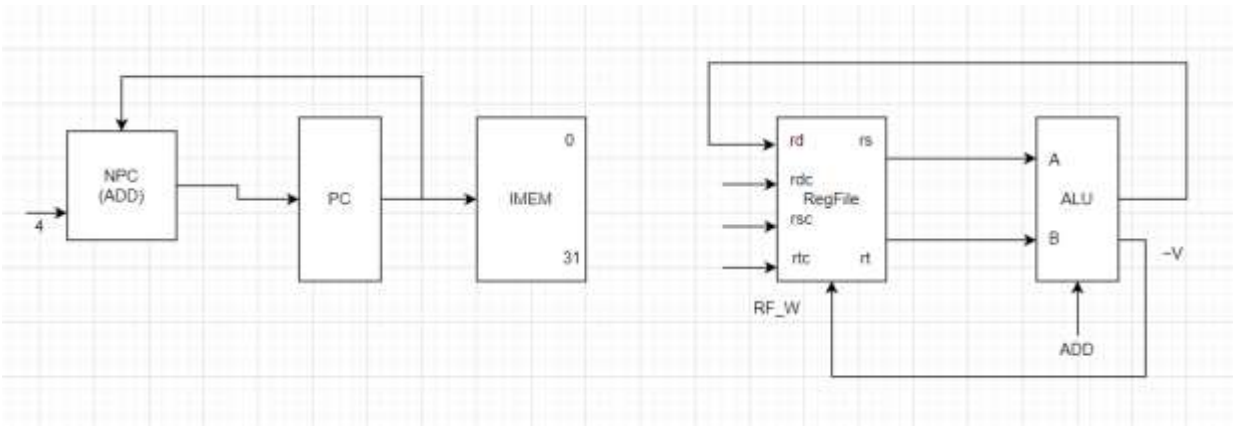
授课老师 张冬冬

一、实验内容

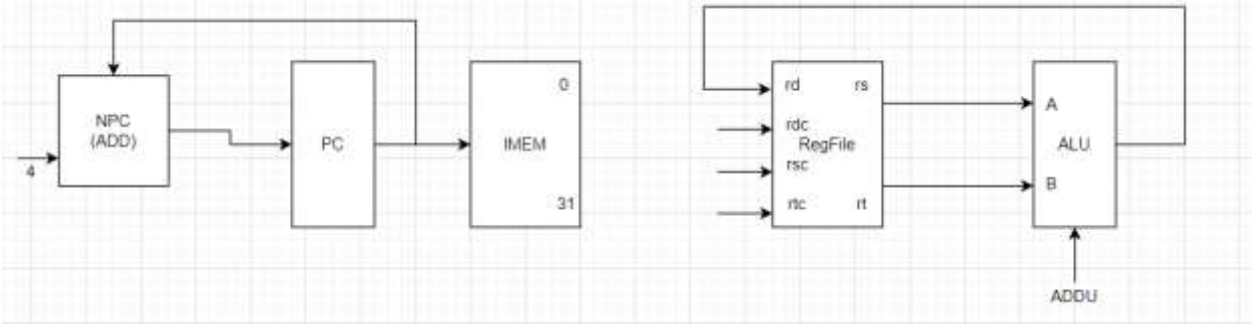
本次实验内容为：实现 MIPS 架构的 31 条指令单周期 CPU 设计。
具体的指令内容如下：
1.ADD
2.ADDU
3.ADDI
4.ADDIU
5.SUB
6.SUBU
7.AND
8.ANDI
9.OR
10.ORI
11.XOR
12.XORI
13.NOR
14.LUI
15.SLL
16.SLLV
17.SRA
18.SRAV
19.SRL
20.SRLV
21.SLT
22.SLTI
23.SLTU
24.SLTIU
25.BEQ
26.BNE
27.J
28.JAL
29.JR
30.LW
31.SW

二、硬件逻辑图

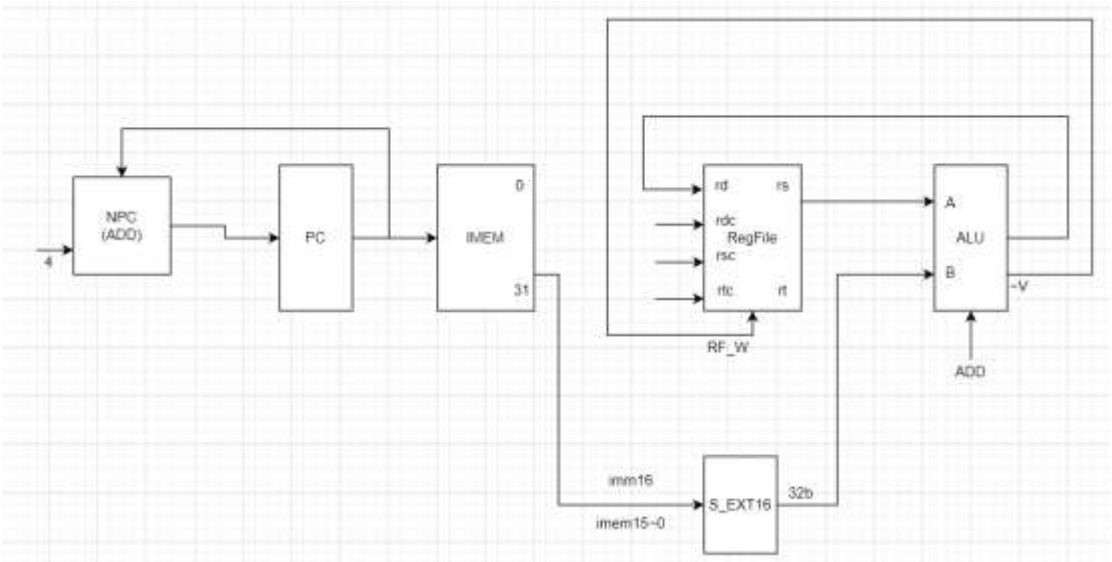
1.ADD



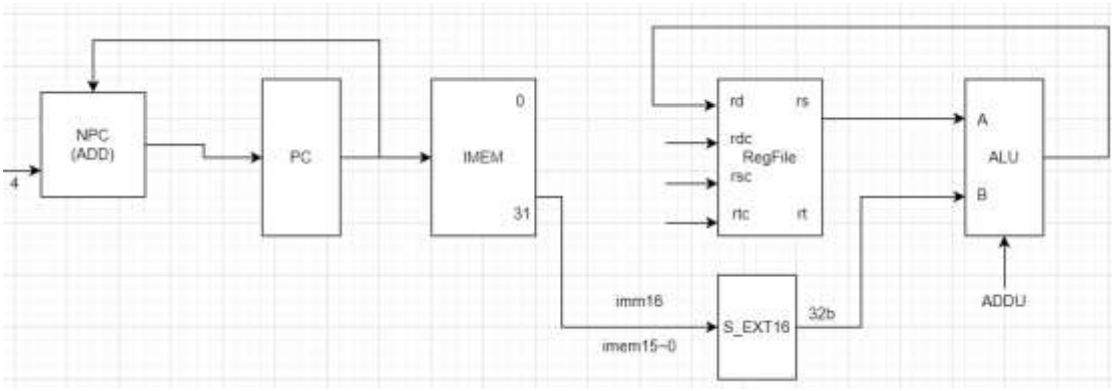
2.ADDU



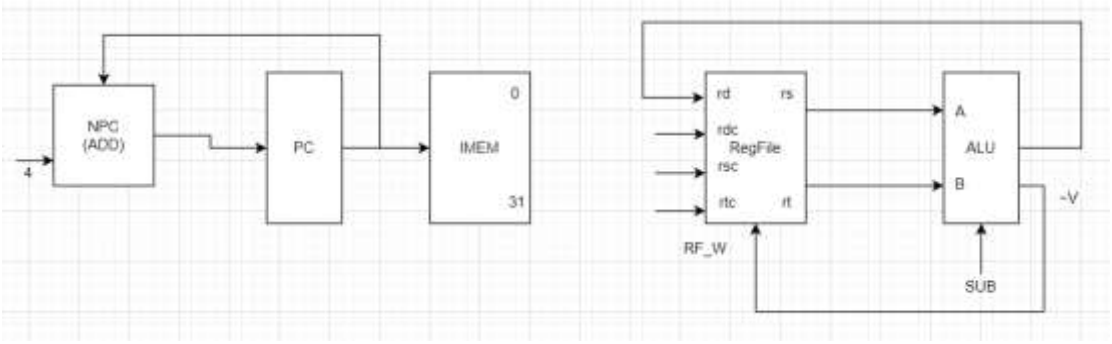
3.ADDI



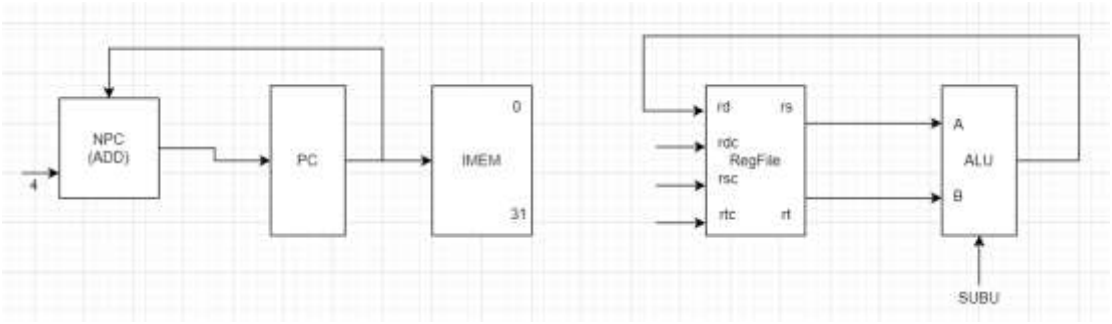
4.ADDIU



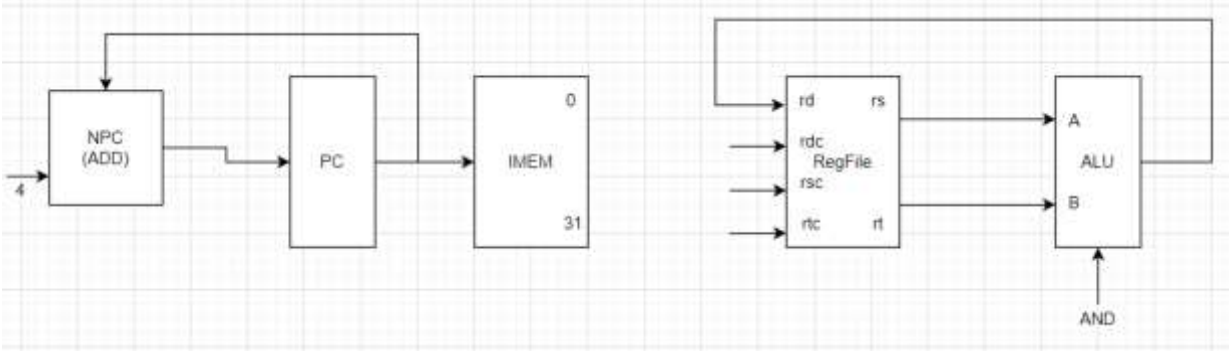
5.SUB



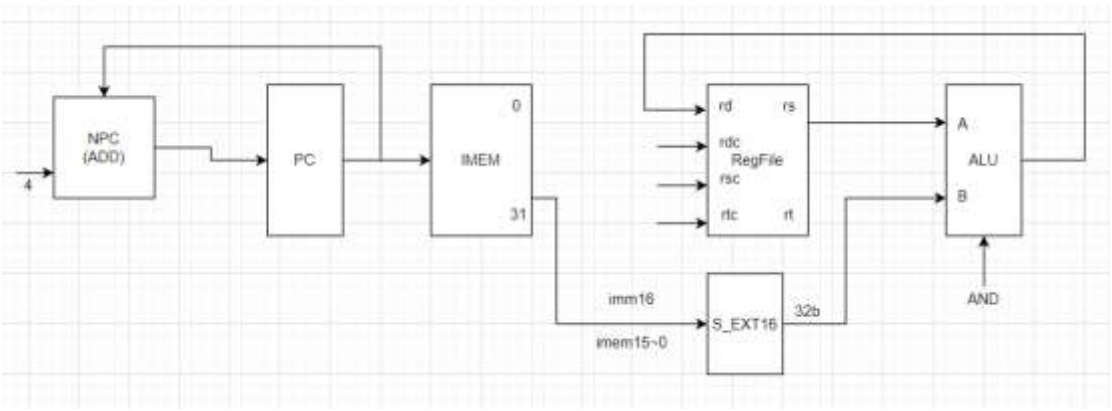
6.SUBU



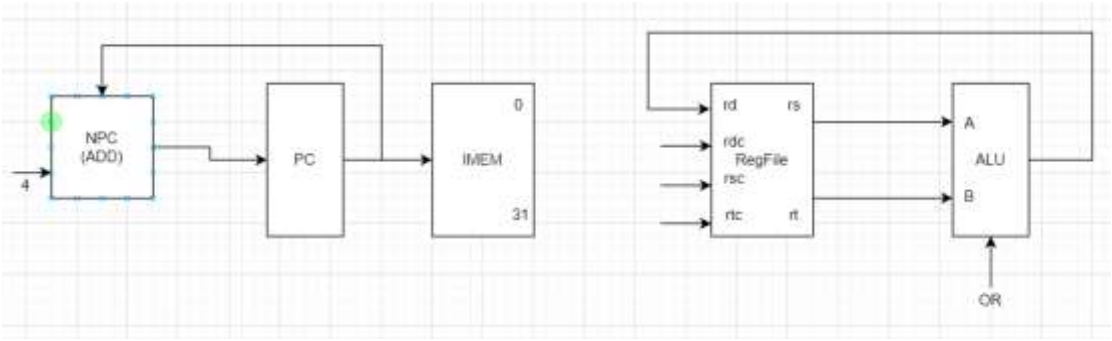
7.AND



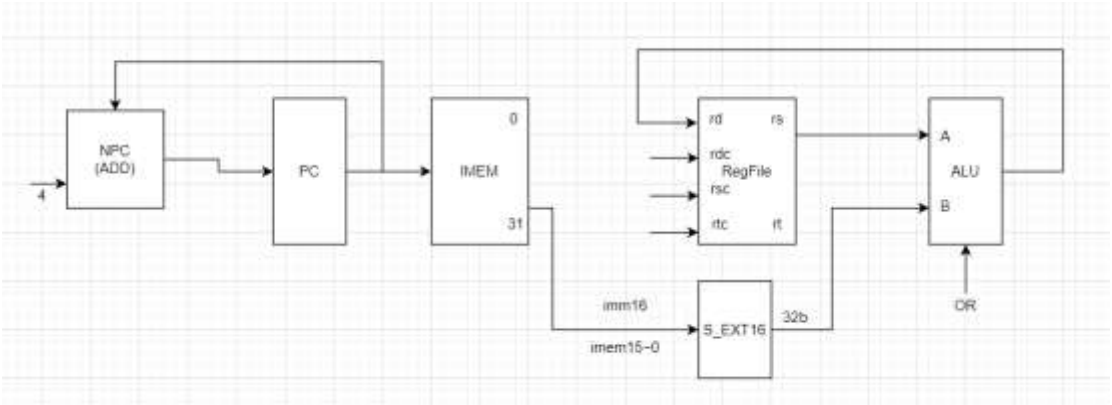
8.ANDI



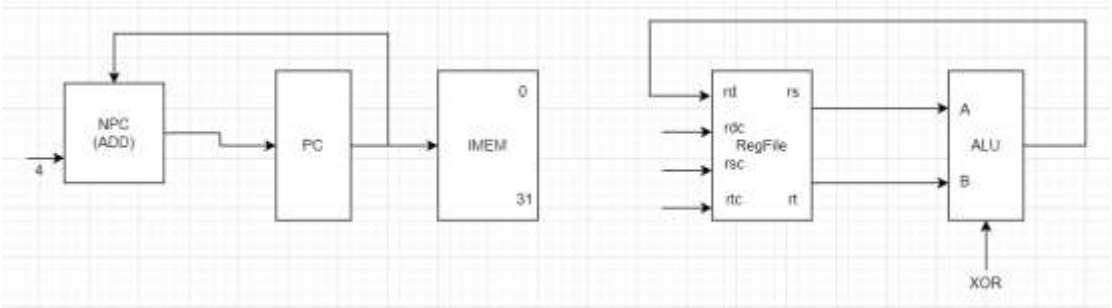
9.OR



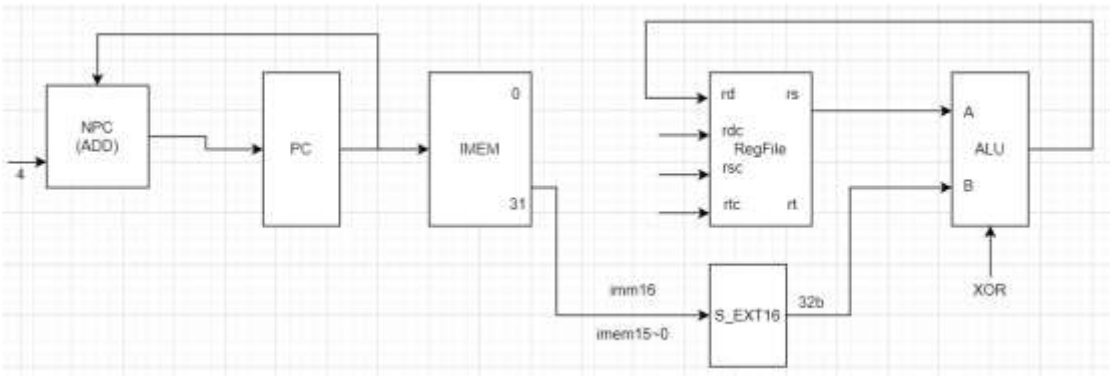
10.ORI



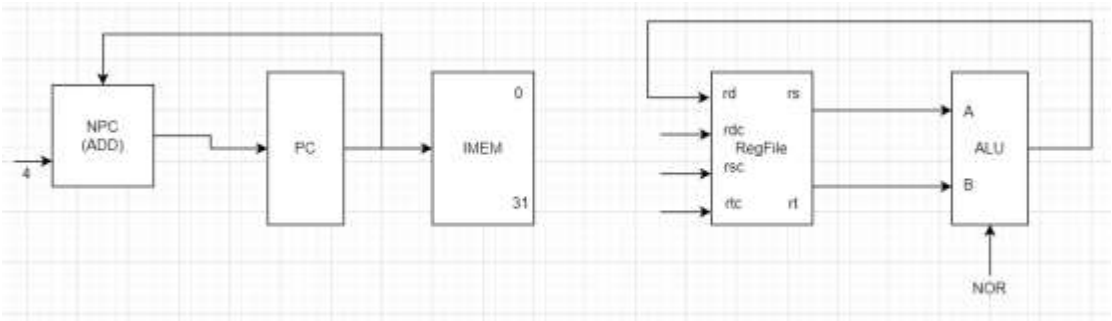
11.XOR



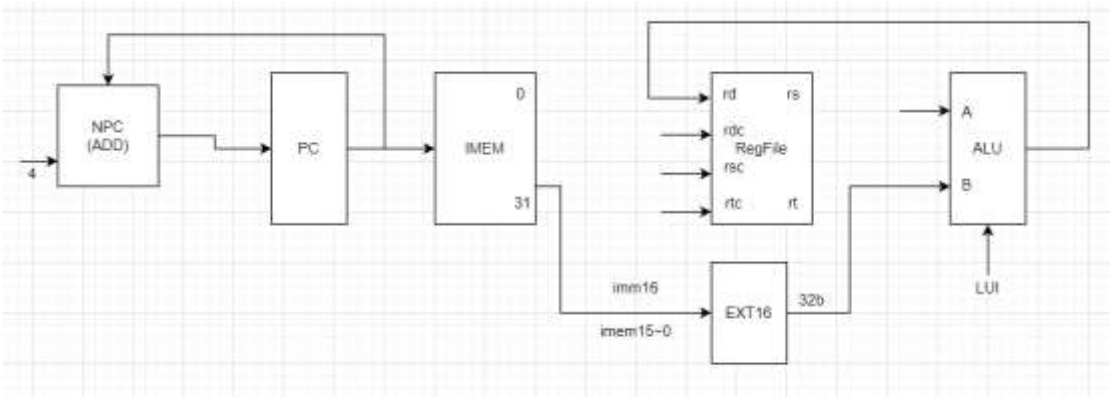
12.XORI



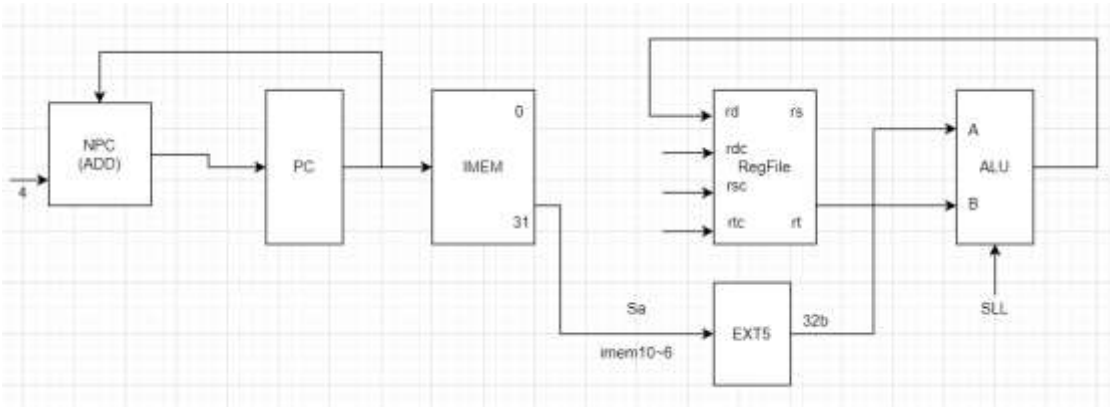
13.NOR



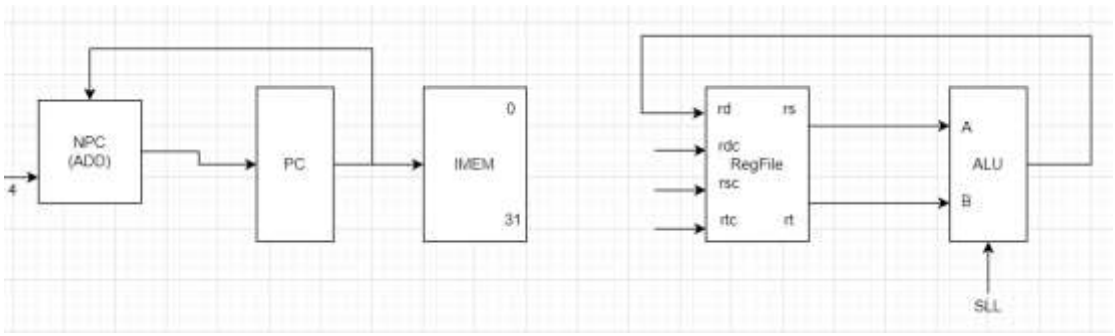
14.LUI



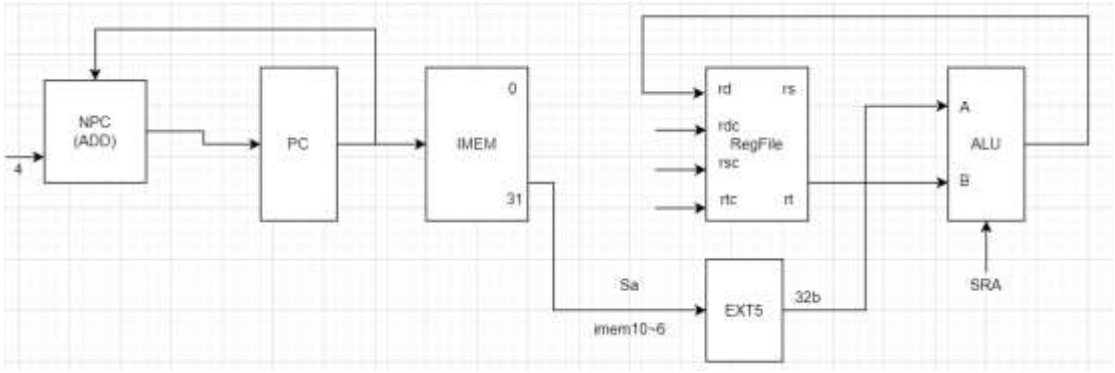
15.SLL



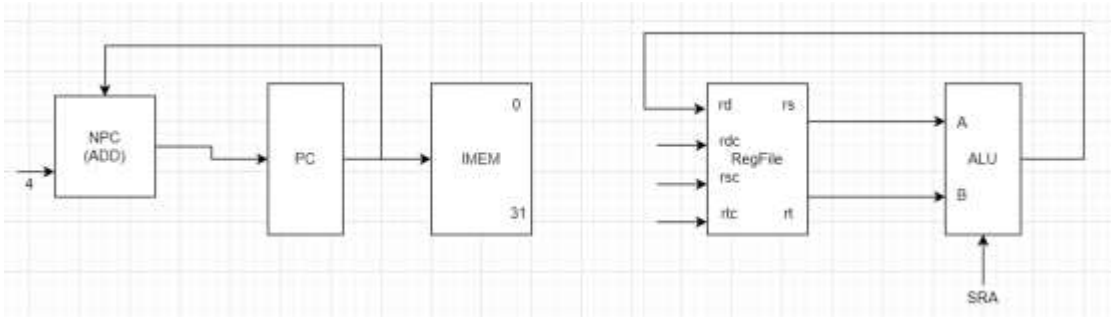
16.SLLV



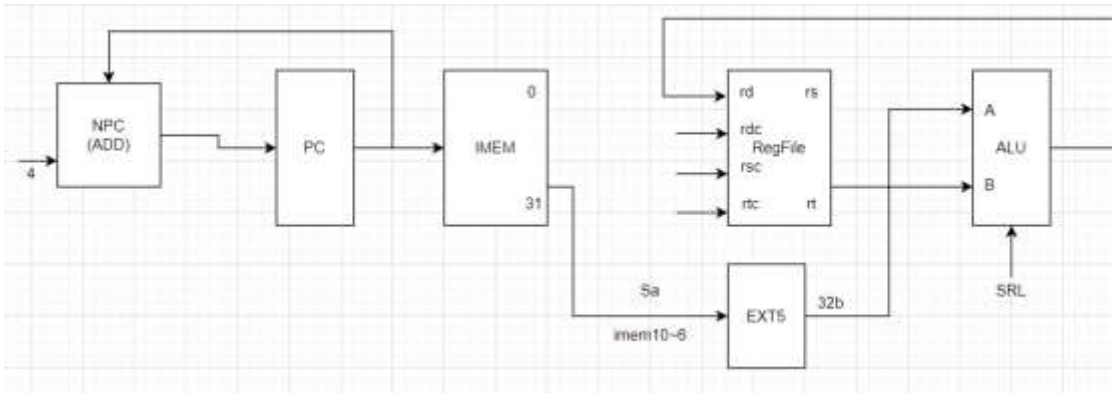
17.SRA



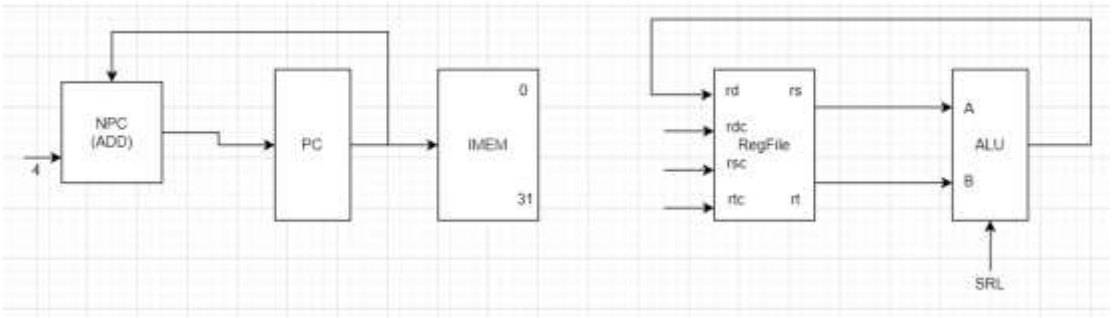
18.SRAV



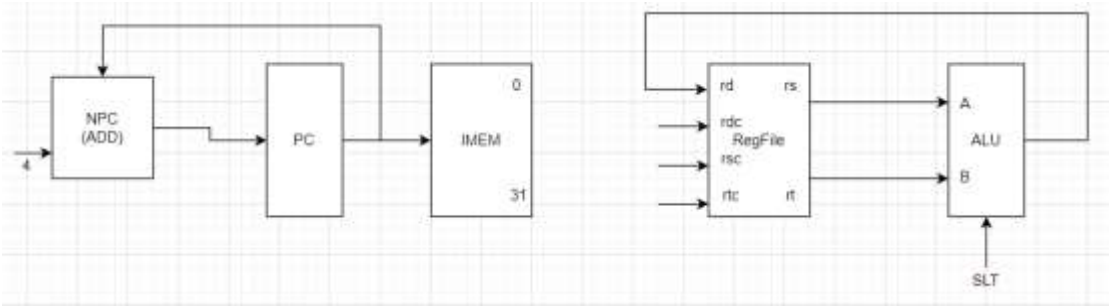
19.SRL



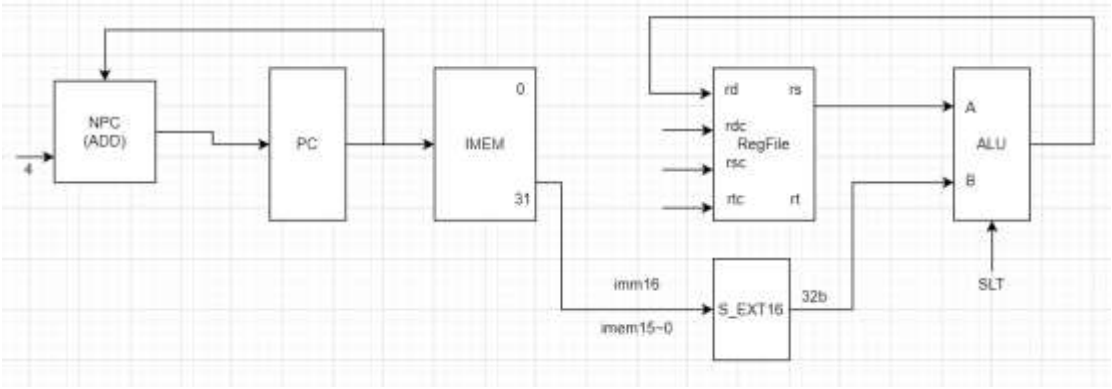
20.SRLV



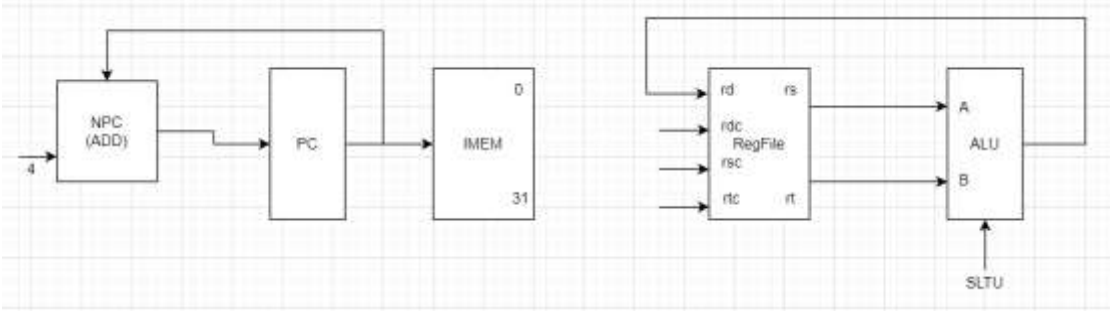
21.SLT



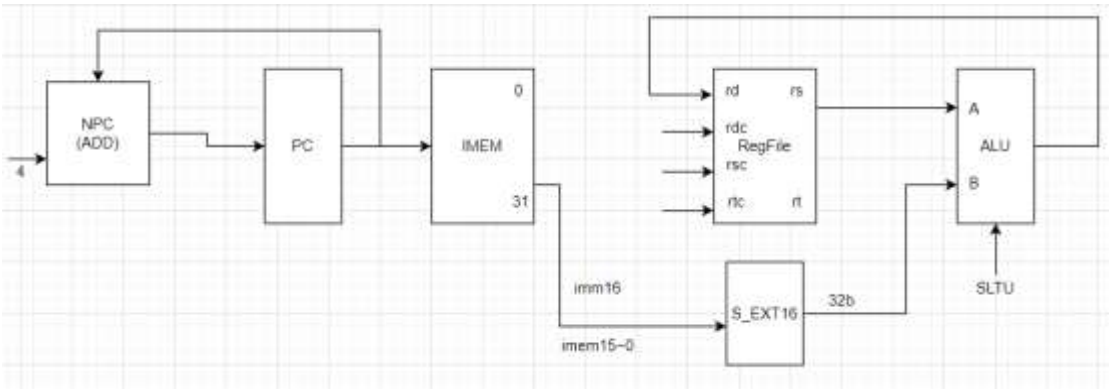
22.SLTI



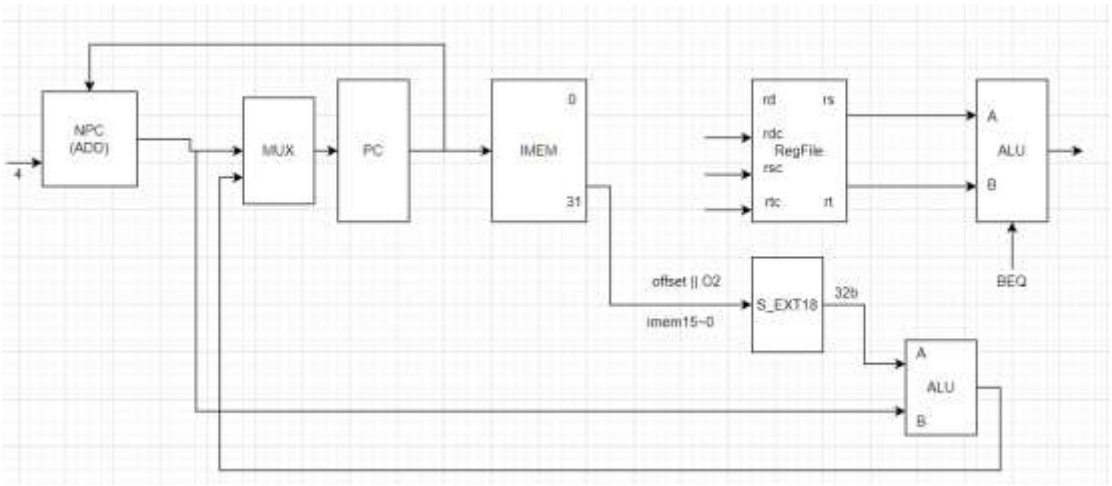
23.SLTU



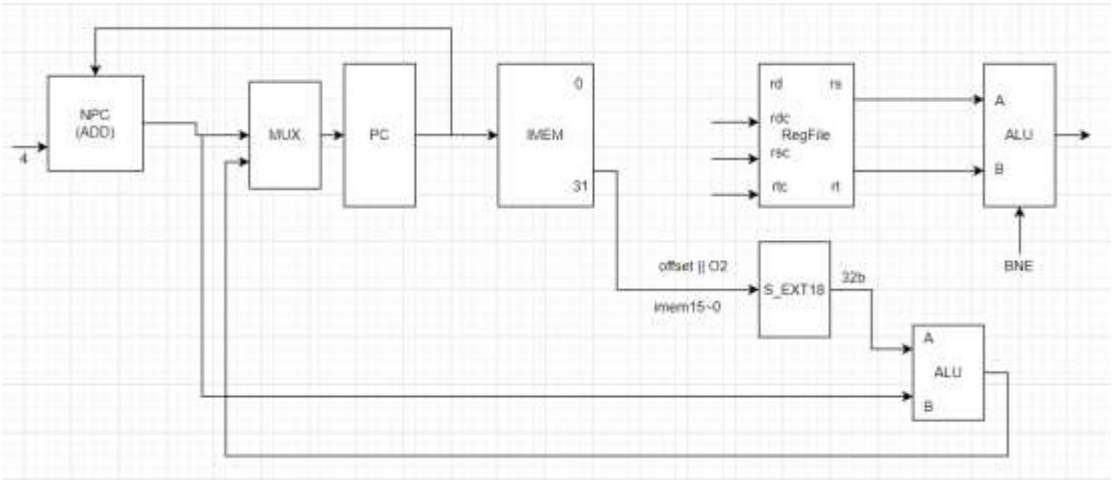
24.SLTIU



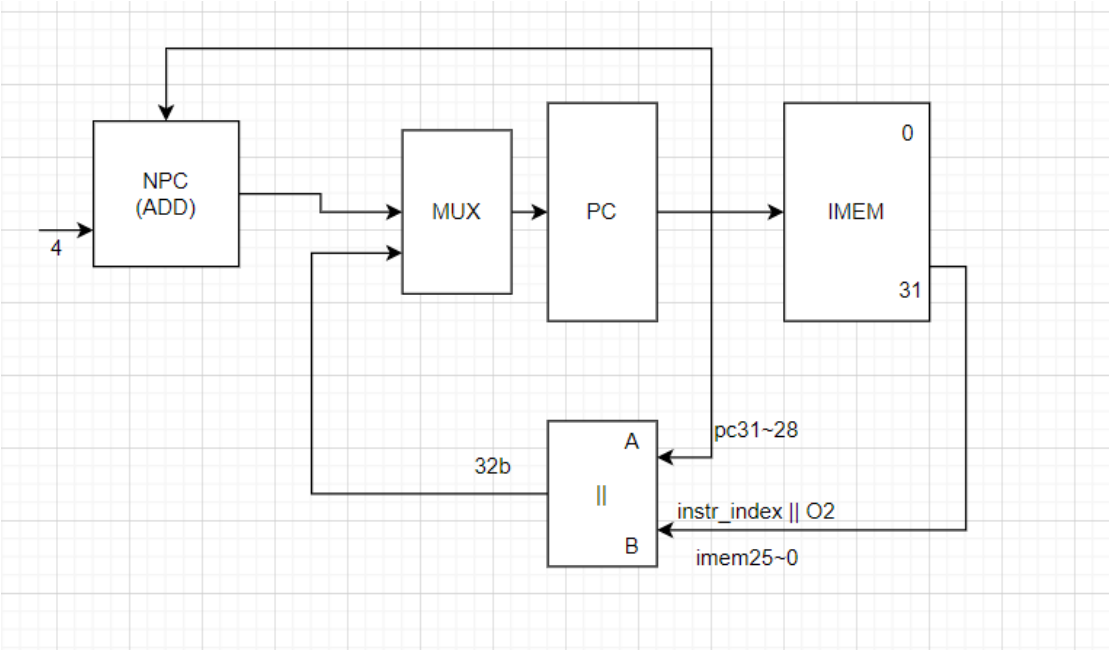
25.BEQ



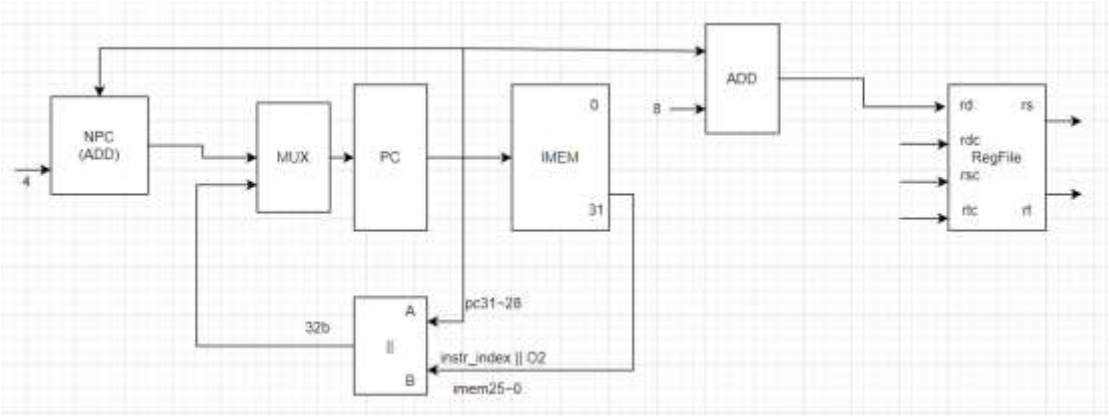
26.BNE



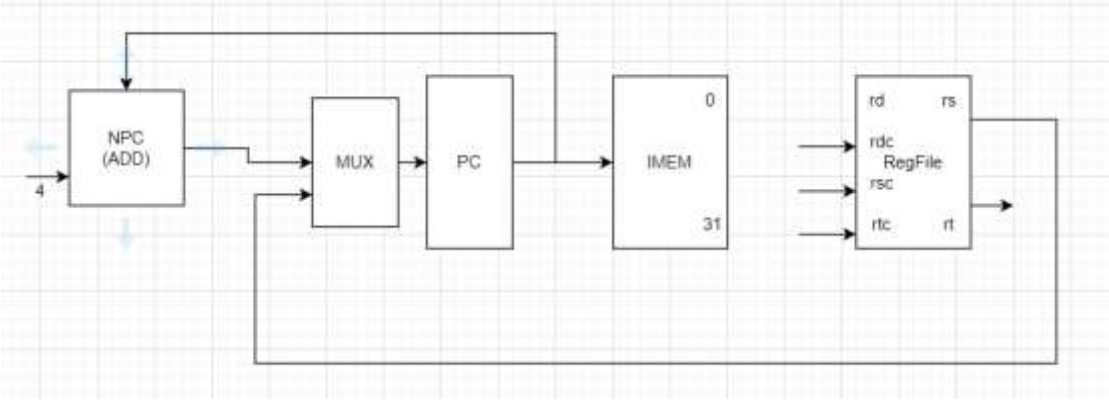
27.J



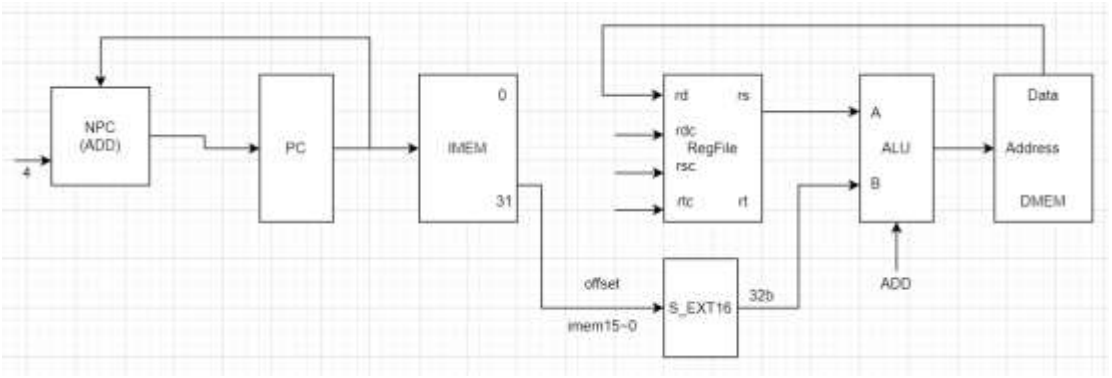
28.JAL



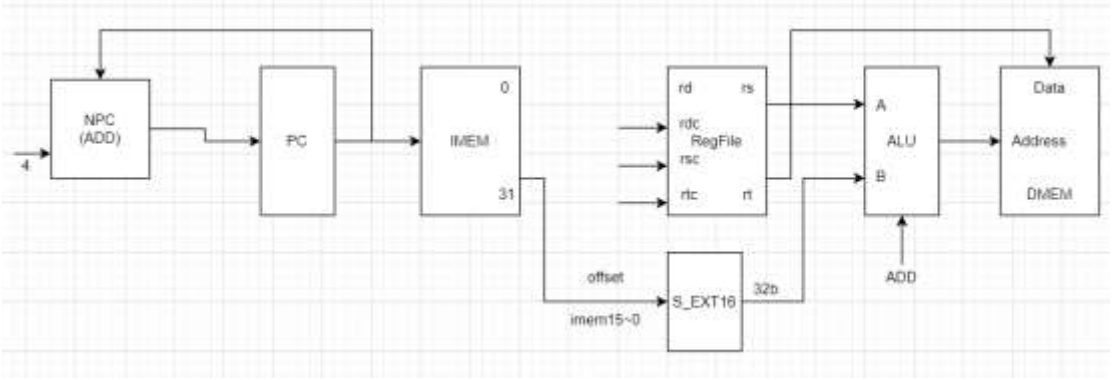
29.JR



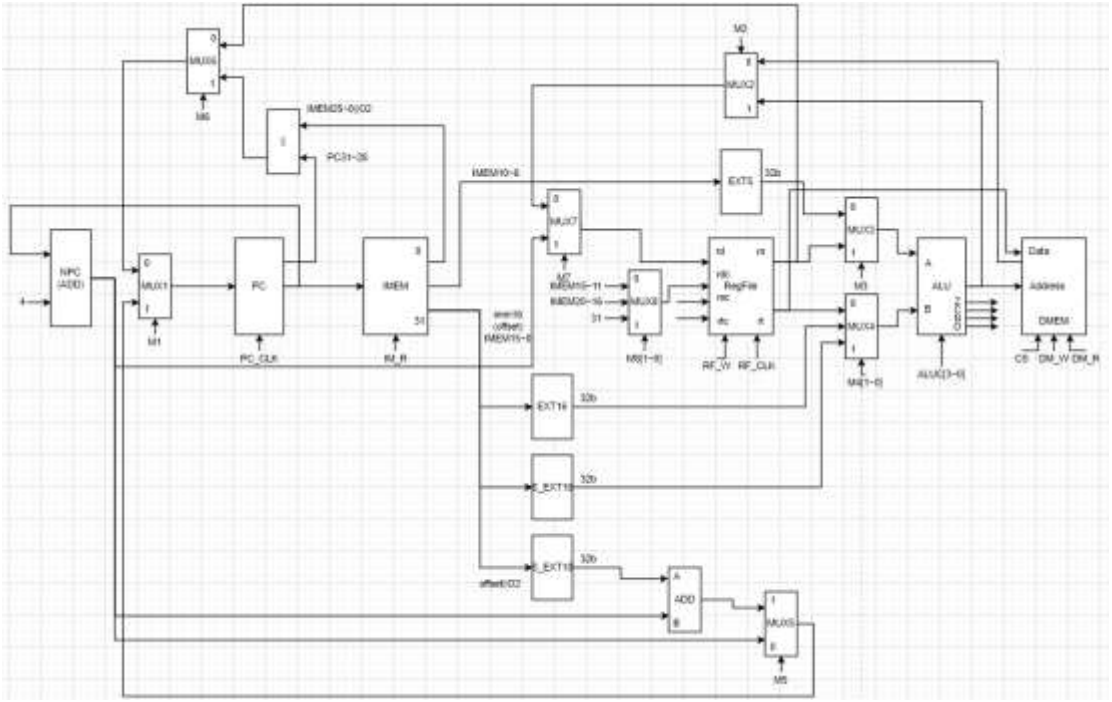
30.LW



31.SW



31 条指令单周期 CPU 设计:



组成部件：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	指令	PC	NPC	IMEM	Regfile		ALU			Ext5	Ext16	DMEM		S_Ext16	S_Ext18	ADD		||		MUX1-8
2					rd	RF_W	A	B	ALUC			Addr	Data			A	B	A	B	
3	ADD	npc	pc	pc	ALU	~V	rs	rt	ADD											
4	ADDU	npc	pc	pc	ALU		rs	rt	ADDU											
5	ADDI	npc	pc	pc	ALU	~V	rs	S_Ext16	ADD					imm16						
6	ADDIU	npc	pc	pc	ALU		rs	S_Ext16	ADDU					imm16						
7	SUB	npc	pc	pc	ALU	~V	rs	rt	SUB											
8	SUBU	npc	pc	pc	ALU		rs	rt	SUBU											
9	AND	npc	pc	pc	ALU		rs	rt	AND											
10	ANDI	npc	pc	pc	ALU		rs	Ext16	AND		imm16									
11	OR	npc	pc	pc	ALU		rs	rt	OR											
12	ORI	npc	pc	pc	ALU		rs	Ext16	OR		imm16									
13	XOR	npc	pc	pc	ALU		rs	rt	XOR											
14	XORI	npc	pc	pc	ALU		rs	Ext16	XOR		imm16									
15	NOR	npc	pc	pc	ALU		rs	rt	NOR											
16	LUI	npc	pc	pc	ALU			Ext16	LUI		imm16									
17	SLL	npc	pc	pc	ALU		Ext5	rt	SLL	Sa										
18	SLLV	npc	pc	pc	ALU		rs	rt	SLL											
19	SRA	npc	pc	pc	ALU		Ext5	rt	SRA	Sa										
20	SRAV	npc	pc	pc	ALU		rs	rt	SRA											
21	SRL	npc	pc	pc	ALU		Ext5	rt	SRL	Sa										
22	SRLV	npc	pc	pc	ALU		rs	rt	SRL											
23	SLT	npc	pc	pc	ALU		rs	rt	SLT											
24	SLTI	npc	pc	pc	ALU		rs	S_Ext16	SLT					imm16						
25	SLTU	npc	pc	pc	ALU		rs	rt	SLTU											
26	SLTIU	npc	pc	pc	ALU		rs	S_Ext16	SLTU					imm16						
27																				
28	BEQ	add	pc	pc			rs	rt	SUBU						offset@O2	NPC	S_Ext18			
29	BNE	add	pc	pc			rs	rt	SUBU						offset@O2	NPC	S_Ext18			
30	J	||	pc	pc														pc31~28	imem25~0	
31	JAL	||	pc	pc	pc+4													pc31~28	imem25~0	
32	JR	rs	pc	pc																
33																				
34	LW	npc	pc	pc	Data		rs	S_Ext16	ADD			ALU		offset						
35	SW	npc	pc	pc			rs	S_Ext16	ADD			ALU	rt	offset						

Sheet1

控制信号：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1		ADD	ADDU	ADDI	ADDIU	SUB	SUBU	AND	ANDI	OR	ORI	XOR	XORI	NOR	LUI	SLL	SLLV	SRA	SRAV	SRL	SRLV	SLT	SLTI	SLTU	SLTIU
2																									
3	PC_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	M_R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5																									
6	rs4~0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21			IM25-21		IM25-21		IM25-21	IM25-21	IM25-21	IM25-21	IM25-21
7	rt4~0	IM20-16	IM20-16			IM20-16	IM20-16	IM20-16		IM20-16		IM20-16		IM20-16		IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16		IM20-16	
8	rdst~0	IM15-11	IM15-11	IM20-16	IM20-16	IM15-11	IM15-11	IM15-11	IM20-16	IM15-11	IM20-16	IM15-11	IM20-16	IM15-11	IM20-16	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM20-16	IM15-11	IM20-16
9	RF_W	~V	1	~V	1	~V	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	RF_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11																									
12	ALUC3~0	ADD	ADDU	ADD	ADDU	SUB	SUBU	AND	AND	OR	OR	XOR	XOR	NOR	LUI	SLL	SLL	SRA	SRA	SRL	SRL	SLT	SLT	SLTU	SLTU
13																									
14	M1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
15	M2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	M3																								
17	M41~0	0-0	0-0	1-0	1-0	0-0	0-0	0-0	0-1	0-0	0-1	0-0	0-1	0-0	0-1	0-0	0-0	0-0	0-0	0-0	0-0	0-0	1-0	0-0	1-0
18	M5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	M6																								
20	M7																								
21	M81~0	0	0	1	1	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	
22																									
23	CS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	DM_R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	DM_W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

AA	AB	AC	AD	AE	AF	AG	AH
BEQ	BNE	J	JAL	JR		LW	SW
1	1	1	1	1		1	1
1	1	1	1	1		1	1
IM25-21	IM25-21					IM25-21	IM25-21
IM20-16	IM20-16					IM20-16	IM20-16
0	0	0	1	0		1	0
0	0	0	1	0		1	0
SUBU	SUBU					ADD	ADD
1	1	0	0	0		1	1
						0	
0-0	0-0	0-0	0-0	0-0		1-0	1-0
1	1					0	0
		1	1	0			
			1				
1	1		10			1	1
0	0	0	0	0		1	1
0	0	0	0	0		1	0
0	0	0	0	0		0	1

三、模块建模

（该部分要求对实验中建模的所有模块进行功能描述，并列出各模块建模的 verilog 代码）

顶层模块：sccomp_dataflow

```
module sccomp_dataflow(
input clk_in,
input reset,
output [31:0] inst,
output [31:0] pc
    );

                        //控制信号
/*------------------------*/
wire Z;
wire C;
wire N;
wire O;
```

```
// wire    PC_CLK;
//    wire    IM_R;
wire    RF_W;
// wire    RF_CLK;
wire    [3:0]ALUC;

wire    M1;
wire    M2;
wire    M3;
wire    [1:0]M4;
wire    M5;
wire    M6;
wire    M7;
wire    [1:0]M8;

wire    CS;
wire    DM_R;
wire    DM_W;
/*------------------------*/
//wire [31:0] imem_out;
wire [31:0] dmem_out;
wire [31:0] rt_out;
wire [31:0] alu_out;

wire [31:0] im_addr;
wire [31:0] dm_addr;


        assign im_addr =    pc- 32'h00400000;
        assign dm_addr =    alu_out- 32'h10010000;

        IMEM imemory(pc[12:2],inst);
        DMEM dmemory(clk_in,1'b1,DM_W,dm_addr[12:2],rt_out,dmem_out);
        cpu sccpu(clk_in,reset,pc,inst,dmem_out,rt_out,alu_out,
        Z,C,N,O,
        RF_W,ALUC,
        M1,M2,M3,M4,M5,M6,M7,M8,
        CS,DM_R,DM_W);

        instruction_to_control itc(
        inst,
        Z,C,N,O,
        RF_W,ALUC,
        M1,M2,M3,M4,M5,M6,M7,M8,
```

```
        CS,DM_R,DM_W);


endmodule


module Regfiles(
input clk,
input rst,
input rf_w,

input [4:0]raddr1,
input [4:0]raddr2,
input [4:0]waddr,
input [31:0]wdata,
output reg [31:0]rdata1,
output reg [31:0]rdata2
        );

          reg [31:0] array_reg [31:0];//reg 数组,前面为位数，后面为数组元素个数

          always @(posedge clk or posedge rst)
          begin
          if(rst==1'b1)
          begin
          array_reg[0]=32'b0;
          array_reg[1]=32'b0;
          array_reg[2]=32'b0;
          array_reg[3]=32'b0;
          array_reg[4]=32'b0;
          array_reg[5]=32'b0;
          array_reg[6]=32'b0;
          array_reg[7]=32'b0;
          array_reg[8]=32'b0;
          array_reg[9]=32'b0;
          array_reg[10]=32'b0;
          array_reg[11]=32'b0;
          array_reg[12]=32'b0;
          array_reg[13]=32'b0;
          array_reg[14]=32'b0;
          array_reg[15]=32'b0;
          array_reg[16]=32'b0;
          array_reg[17]=32'b0;
          array_reg[18]=32'b0;
```

```
array_reg[19]=32'b0;
array_reg[20]=32'b0;
array_reg[21]=32'b0;
array_reg[22]=32'b0;
array_reg[23]=32'b0;
array_reg[24]=32'b0;
array_reg[25]=32'b0;
array_reg[26]=32'b0;
array_reg[27]=32'b0;
array_reg[28]=32'b0;
array_reg[29]=32'b0;
array_reg[30]=32'b0;
array_reg[31]=32'b0;


end
else
begin
    if(rf_w==1)
    begin
    if (waddr == 0)
    array_reg[0] = 0;
    else if (waddr == 1)
    array_reg[1] = wdata;
    else if (waddr == 2)
    array_reg[2] = wdata;
    else if (waddr == 3)
    array_reg[3] = wdata;
    else if (waddr == 4)
    array_reg[4] = wdata;
    else if (waddr == 5)
    array_reg[5] = wdata;
    else if (waddr == 6)
    array_reg[6] = wdata;
    else if (waddr == 7)
    array_reg[7] = wdata;
    else if (waddr == 8)
    array_reg[8] = wdata;
    else if (waddr == 9)
    array_reg[9] = wdata;
    else if (waddr == 10)
    array_reg[10] = wdata;
    else if (waddr == 11)
    array_reg[11] = wdata;
```

```
else if (waddr == 12)
array_reg[12] = wdata;
else if (waddr == 13)
array_reg[13] = wdata;
else if (waddr == 14)
array_reg[14] = wdata;
else if (waddr == 15)
array_reg[15] = wdata;
else if (waddr == 16)
array_reg[16] = wdata;
else if (waddr == 17)
array_reg[17] = wdata;
else if (waddr == 18)
array_reg[18] = wdata;
else if (waddr == 19)
array_reg[19] = wdata;
else if (waddr == 20)
array_reg[20] = wdata;
else if (waddr == 21)
array_reg[21] = wdata;
else if (waddr == 22)
array_reg[22] = wdata;
else if (waddr == 23)
array_reg[23] = wdata;
else if (waddr == 24)
array_reg[24] = wdata;
else if (waddr == 25)
array_reg[25] = wdata;
else if (waddr == 26)
array_reg[26] = wdata;
else if (waddr == 27)
array_reg[27] = wdata;
else if (waddr == 28)
array_reg[28] = wdata;
else if (waddr == 29)
array_reg[29] = wdata;
else if (waddr == 30)
array_reg[30] = wdata;
else if (waddr == 31)
array_reg[31] = wdata;
else
begin

end
```

```
    end
    else//rf_w==0
    begin

    end




end
end

always @(*)
begin
if(raddr1==0)
rdata1=array_reg[0];
else if(raddr1==1)
rdata1=array_reg[1];
else if(raddr1==2)
rdata1=array_reg[2];
else if(raddr1==3)
rdata1=array_reg[3];
else if(raddr1==4)
rdata1=array_reg[4];
else if(raddr1==5)
rdata1=array_reg[5];
else if(raddr1==6)
rdata1=array_reg[6];
else if(raddr1==7)
rdata1=array_reg[7];
else if(raddr1==8)
rdata1=array_reg[8];
else if(raddr1==9)
rdata1=array_reg[9];
else if(raddr1==10)
rdata1=array_reg[10];
else if(raddr1==11)
rdata1=array_reg[11];
else if(raddr1==12)
rdata1=array_reg[12];
else if(raddr1==13)
rdata1=array_reg[13];
```

```
else if(raddr1==14)
rdata1=array_reg[14];
else if(raddr1==15)
rdata1=array_reg[15];
else if(raddr1==16)
rdata1=array_reg[16];
else if(raddr1==17)
rdata1=array_reg[17];
else if(raddr1==18)
rdata1=array_reg[18];
else if(raddr1==19)
rdata1=array_reg[19];
else if(raddr1==20)
rdata1=array_reg[20];
else if(raddr1==21)
rdata1=array_reg[21];
else if(raddr1==22)
rdata1=array_reg[22];
else if(raddr1==23)
rdata1=array_reg[23];
else if(raddr1==24)
rdata1=array_reg[24];
else if(raddr1==25)
rdata1=array_reg[25];
else if(raddr1==26)
rdata1=array_reg[26];
else if(raddr1==27)
rdata1=array_reg[27];
else if(raddr1==28)
rdata1=array_reg[28];
else if(raddr1==29)
rdata1=array_reg[29];
else if(raddr1==30)
rdata1=array_reg[30];
else if(raddr1==31)
rdata1=array_reg[31];
else
begin

end

if(raddr2==0)
rdata2=array_reg[0];
else if(raddr2==1)
```

```
rdata2=array_reg[1];
else if(raddr2==2)
rdata2=array_reg[2];
else if(raddr2==3)
rdata2=array_reg[3];
else if(raddr2==4)
rdata2=array_reg[4];
else if(raddr2==5)
rdata2=array_reg[5];
else if(raddr2==6)
rdata2=array_reg[6];
else if(raddr2==7)
rdata2=array_reg[7];
else if(raddr2==8)
rdata2=array_reg[8];
else if(raddr2==9)
rdata2=array_reg[9];
else if(raddr2==10)
rdata2=array_reg[10];
else if(raddr2==11)
rdata2=array_reg[11];
else if(raddr2==12)
rdata2=array_reg[12];
else if(raddr2==13)
rdata2=array_reg[13];
else if(raddr2==14)
rdata2=array_reg[14];
else if(raddr2==15)
rdata2=array_reg[15];
else if(raddr2==16)
rdata2=array_reg[16];
else if(raddr2==17)
rdata2=array_reg[17];
else if(raddr2==18)
rdata2=array_reg[18];
else if(raddr2==19)
rdata2=array_reg[19];
else if(raddr2==20)
rdata2=array_reg[20];
else if(raddr2==21)
rdata2=array_reg[21];
else if(raddr2==22)
rdata2=array_reg[22];
else if(raddr2==23)
```

```
        rdata2=array_reg[23];
        else if(raddr2==24)
        rdata2=array_reg[24];
        else if(raddr2==25)
        rdata2=array_reg[25];
        else if(raddr2==26)
        rdata2=array_reg[26];
        else if(raddr2==27)
        rdata2=array_reg[27];
        else if(raddr2==28)
        rdata2=array_reg[28];
        else if(raddr2==29)
        rdata2=array_reg[29];
        else if(raddr2==30)
        rdata2=array_reg[30];
        else if(raddr2==31)
        rdata2=array_reg[31];
        else
        begin

        end

        end

endmodule


module pc(
input PC_CLK,
input reset,
input [31:0] pc_in,
output reg [31:0] pc_out
    );

    always @(negedge PC_CLK or posedge reset)//上升沿  or  下降沿？
    begin
        if(reset==1'b1)
            pc_out<=32'h00400000;
        else
            pc_out<=pc_in;
    end

endmodule
```

```
module selector41(
input [31:0] iC0,
input [31:0] iC1,
input [31:0] iC2,
input [31:0] iC3,
input [1:0]iS,
output reg [31:0] oZ
	);
	always @(*)
	begin

	if(iS==0)
	oZ=iC0;
	else if(iS==1)
	oZ=iC1;
	else if(iS==2)
	oZ=iC2;
	else
	oZ=iC3;

	end

endmodule

module selector41_5(
input [4:0] iC0,
input [4:0] iC1,
input [4:0] iC2,
input [4:0] iC3,
input [1:0]iS,
output reg [4:0] oZ
	);
	always @(*)
	begin

	if(iS==0)
	oZ=iC0;
	else if(iS==1)
	oZ=iC1;
	else if(iS==2)
	oZ=iC2;
	else
	oZ=iC3;
```

```
        end

endmodule

module selector21(
input [31:0] iC0,
input [31:0] iC1,
input iS,
output reg [31:0] oZ
        );
        always @(*)
        begin

        if(iS==0)
        oZ=iC0;
        else
        oZ=iC1;

        end

endmodule


module joint_pc_im(
input [27:0] imem_in,
input [3:0] pc_in,
output [31:0] joint_out
        );

        assign joint_out={pc_in,imem_in};

endmodule


module IMEM(
input [10:0]addr,
output [31:0]instr
        );

        imem im(.a(addr),.spo(instr));
endmodule
```

```
module i_to_controller(
input [5:0]op,
input [5:0]func,
input Z,
input C,
input N,
input O,
//output    PC_CLK,
//output    IM_R,
//output    [4:0]rsc,
//output    [4:0]rtc,
//output    [4:0]rdc,
output    RF_W,
//output    RF_CLK,
output    [3:0]ALUC,

output    M1,
output    M2,
output    M3,
output    [1:0]M4,
output    M5,
output    M6,
output    M7,
output    [1:0]M8,

output    CS,
output    DM_R,
output    DM_W
    );

    //指令种类，组合逻辑构建
    wire i_add,i_addi,i_addu,i_addiu,i_sub,i_subu;
    wire i_and,i_andi,i_or,i_ori,i_xor,i_xori,i_nor;
    wire i_lui;
    wire i_sll,i_sllv,i_sra,i_srav,i_srl,i_srlv;
    wire i_slt,i_slti,i_sltu,i_sltiu;
    wire i_beq,i_bne;
    wire i_j,i_jal,i_jr;
    wire i_lw,i_sw;

    //R-TYPE
    assign
i_add=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~fun
c[3])&(~func[2])&(~func[1])&(~func[0]);
```

```
    assign
i_addu=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~fu
nc[3])&(~func[2])&(~func[1])&(func[0]);
    assign
i_sub=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~fun
c[3])&(~func[2])&(func[1])&(~func[0]);
    assign
i_subu=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~fu
nc[3])&(~func[2])&(func[1])&(func[0]);
    assign
i_and=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~fun
c[3])&(func[2])&(~func[1])&(~func[0]);
    assign
i_or=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[
3])&(func[2])&(~func[1])&(func[0]);
    assign
i_xor=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func
[3])&(func[2])&(func[1])&(~func[0]);
    assign
i_nor=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func
[3])&(func[2])&(func[1])&(func[0]);
    assign
i_slt=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(func[3
])&(~func[2])&(func[1])&(~func[0]);
    assign
i_sltu=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(func[
3])&(~func[2])&(func[1])&(func[0]);

    assign
i_sll=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(~fun
c[3])&(~func[2])&(~func[1])&(~func[0]);
    assign
i_srl=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(~fun
c[3])&(~func[2])&(func[1])&(~func[0]);
    assign
i_sra=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(~fun
c[3])&(~func[2])&(func[1])&(func[0]);
    assign
i_sllv=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(~fu
nc[3])&(func[2])&(~func[1])&(~func[0]);
    assign
i_srlv=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(~fu
nc[3])&(func[2])&(func[1])&(~func[0]);
    assign
```

```
i_srav=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(~func[3])&(func[2])&(func[1])&(func[0]);
    assign i_jr=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(func[3])&(~func[2])&(~func[1])&(~func[0]);

    //I-TYPE
    assign i_addi=(~op[5])&(~op[4])&(op[3])&(~op[2])&(~op[1])&(~op[0]);
    assign i_addiu=(~op[5])&(~op[4])&(op[3])&(~op[2])&(~op[1])&(op[0]);
    assign i_andi=(~op[5])&(~op[4])&(op[3])&(op[2])&(~op[1])&(~op[0]);
    assign i_ori=(~op[5])&(~op[4])&(op[3])&(op[2])&(~op[1])&(op[0]);
    assign i_xori=(~op[5])&(~op[4])&(op[3])&(op[2])&(op[1])&(~op[0]);
    assign i_lui=(~op[5])&(~op[4])&(op[3])&(op[2])&(op[1])&(op[0]);
    assign i_lw=(op[5])&(~op[4])&(~op[3])&(~op[2])&(op[1])&(op[0]);
    assign i_sw=(op[5])&(~op[4])&(op[3])&(~op[2])&(op[1])&(op[0]);
    assign i_beq=(~op[5])&(~op[4])&(~op[3])&(op[2])&(~op[1])&(~op[0]);
    assign i_bne=(~op[5])&(~op[4])&(~op[3])&(op[2])&(~op[1])&(op[0]);
    assign i_slti=(~op[5])&(~op[4])&(op[3])&(~op[2])&(op[1])&(~op[0]);
    assign i_sltiu=(~op[5])&(~op[4])&(op[3])&(~op[2])&(op[1])&(op[0]);

    //J-TYPE
    assign i_j=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(op[1])&(~op[0]);
    assign i_jal=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(op[1])&(op[0]);

    //实例化，产生控制信号
    controller con(i_add, i_addu, i_addi, i_addiu, i_sub, i_subu,i_and, i_andi, i_or, i_ori, i_xor, i_xori, i_nor,
            i_lui,i_sll, i_sllv, i_sra, i_srav, i_srl, i_srlv,i_slt, i_slti, i_sltu, i_sltiu,i_beq, i_bne,
            i_j, i_jal, i_jr,i_lw, i_sw,
            Z,C,N,O,
            RF_W,ALUC,
            M1,M2,M3,M4,M5,M6,M7,M8,
            CS,DM_R,DM_W);

endmodule


module EXT_5(
input [4:0]a,
output [31:0]b
);
assign b={{27{1'b0}},a};
endmodule
```

```
module EXT_16(
input [15:0]a,
output [31:0]b
);
assign b={{16{1'b0}},a};
endmodule

module S_EXT_16(
input [15:0]a,
output [31:0]b
);
assign b={{16{a[15]}},a};
endmodule

module S_EXT_18(
input [17:0]a,
output [31:0]b
);
assign b={{14{a[17]}},a};
endmodule


module instruction_to_control(
input [31:0]instru,
input Z,
input C,
input N,
input O,
//output    PC_CLK,
//output    IM_R,
output    RF_W,
//output    RF_CLK,
output    [3:0]ALUC,

output    M1,
output    M2,
output    M3,
output    [1:0]M4,
output    M5,
output    M6,
output    M7,
output    [1:0]M8,

output    CS,
```

```
output    DM_R,
output    DM_W
      );

            i_to_controller i_to_con(instru[31:26],instru[5:0],
              Z,C,N,O,
              RF_W,ALUC,
              M1,M2,M3,M4,M5,M6,M7,M8,
              CS,DM_R,DM_W);

endmodule


module DMEM(
input clk,
input ena,
input wena,//高电平写入有效，低电平读有效
input [10:0] addr,
input [31:0] data_in,
output reg [31:0] data_out
      );

      reg [31:0] memory [0:31];//reg 数组,前面为位数，后面为数组元素个数

      always @(*)
      begin
        if(ena==0)
          data_out=32'hzzzzzzzz;
          else
          begin

          if(wena==1)//写有效
          begin
         // memory[addr]=data_in;
          end
          else//读有效
          begin
          data_out=memory[addr];
          end

          end
```

```
        end

        always @(posedge clk)
        begin
        if(ena==0)
        begin
        end
      // data_out=32'hzzzzzzzz;
        else
        begin

        if(wena==1)//写有效
        begin
        memory[addr]=data_in;
        end
        else//读有效
        begin
    //    data_out=memory[addr];
        end

        end

        end



endmodule


module cpu(
input clk_in,
input reset,
output [31:0] pc_out,
input [31:0] imem_out,
input [31:0] dmem_out,
output [31:0] rt_out,
output [31:0] alu_out,

output Z,
output C,
output N,
output O,
//input    PC_CLK,
//input    IM_R,
```

```
input    RF_W,
//input    RF_CLK,
input    [3:0]ALUC,

input    M1,
input    M2,
input    M3,
input    [1:0]M4,
input    M5,
input    M6,
input    M7,
input    [1:0]M8,

input    CS,
input    DM_R,
input    DM_W

      );



      //各个部件的输出
      /*-----------------------*/
      wire [31:0] mux1_out;
      wire [31:0] mux2_out;
      wire [31:0] mux3_out;
      wire [31:0] mux4_out;
      wire [31:0] mux5_out;
      wire [31:0] mux6_out;
      wire [31:0] mux7_out;
      wire [4:0] mux8_out;

      wire [31:0] ext16_out;
      wire [31:0] s_ext16_out;
      wire [31:0] s_ext18_out;
      wire [31:0] ext5_out;

      wire [31:0] join_out;
     // wire [31:0] pc_out;
      wire [31:0] npc_out;
      //wire [31:0] imem_out;
     // wire [31:0] dmem_out;
     // wire [31:0] dmem_data_out;
      wire [31:0] add_out;
```

```
    wire [31:0] rs_out;
   // wire [31:0] rt_out;
    //wire [31:0] alu_out;

    /*------------------------*/


    //选择器部分
    selector21 mux1(mux6_out,mux5_out,M1,mux1_out);
    selector21 mux2(dmem_out,alu_out,M2,mux2_out);
    selector21 mux3(ext5_out,rs_out,M3,mux3_out);
    selector41 mux4(rt_out,ext16_out,s_ext16_out,32'b0,M4,mux4_out);
    selector21 mux5(npc_out,add_out,M5,mux5_out);
    selector21 mux6(rs_out,join_out,M6,mux6_out);
    selector21 mux7(mux2_out,npc_out,M7,mux7_out);
    selector41_5 mux8(imem_out[15:11],imem_out[20:16],5'b11111,5'b0,M8,mux8_out);

    //扩展器部分
    EXT_16 cpu_ext1 (imem_out[15:0],ext16_out);
    S_EXT_16 cpu_ext2 (imem_out[15:0],s_ext16_out);
    S_EXT_18 cpu_ext3 ({imem_out[15:0],2'b00},s_ext18_out);
    EXT_5 cpu_ext4 (imem_out[10:6],ext5_out);

    joint_pc_im jpi({imem_out[25:0],2'b00},pc_out[31:28],join_out);
    pc promc (clk_in,reset,mux1_out,pc_out);
    ADD npromc(pc_out,4,npc_out);
    ADD add2(s_ext18_out,npc_out,add_out);

    //
    Regfiles
cpu_ref(clk_in,reset,RF_W,imem_out[25:21],imem_out[20:16],mux8_out,mux7_out,rs_out,rt_out
);
    alu ALunit(mux3_out,mux4_out,ALUC,reset,alu_out,Z,C,N,O);



endmodule


module controller(

//input：输入的指令种类
input i_add,input i_addu,input i_addi,input i_addiu,input i_sub,input i_subu,
    input i_and,input i_andi,input i_or,input i_ori,input i_xor,input i_xori,input i_nor,
```

```
    input i_lui,
    input i_sll,input i_sllv,input i_sra,input i_srav,input i_srl,input i_srlv,
    input i_slt,input i_slti,input i_sltu,input i_sltiu,
    input i_beq,input i_bne,
    input i_j,input i_jal,input i_jr,
    input i_lw,input i_sw,

//input:ALU 状态信号，决定是否写入等操作
input Z,
input C,
input N,
input O,

//output：控制信号
//output    PC_CLK,
//output    IM_R,
//output    [4:0]rsc,
//output    [4:0]rtc,
//output    [4:0]rdc,
output    RF_W,
//output    RF_CLK,
output    [3:0]ALUC,

output    M1,
output    M2,
output    M3,
output    [1:0]M4,
output    M5,
output    M6,
output    M7,
output    [1:0]M8,

output    CS,
output    DM_R,
output    DM_W


    );

    //assign PC_CLK=1;
    //assign IM_R=1;

    //由此可见 regfle 的 rd 在下降沿写入，根据运算结果决定
    assign
```

```
RF_W=((~O)&(i_add|i_addi|i_sub))|(i_addu)|(i_addiu)|(i_subu)|(i_and)|(i_andi)|(i_or)|(i_ori)|
      (i_xor)|(i_xori)|(i_nor)|(i_lui)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl)|(i_srlv)|
      (i_slt)|(i_slti)|(i_sltu)|(i_sltiu)|(i_jal)|(i_lw);



      //assign RF_CLK=

     // assign ALUC=2;
      assign
ALUC[3]=(i_lui)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl)|(i_srlv)|(i_slt)|(i_slti)|(i_sltu)|(i_sltiu);
      assign
ALUC[2]=(i_and)|(i_andi)|(i_or)|(i_ori)|(i_xor)|(i_xori)|(i_nor)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl
)|(i_srlv);
      assign
ALUC[1]=(i_add)|(i_addi)|(i_sub)|(i_xor)|(i_xori)|(i_nor)|(i_sll)|(i_sllv)|(i_slt)|(i_slti)|(i_sltu)|(i_slt
iu)|(i_lw)|(i_sw);
      assign
ALUC[0]=(i_sub)|(i_subu)|(i_or)|(i_ori)|(i_nor)|(i_srl)|(i_srlv)|(i_slt)|(i_slti)|(i_beq)|(i_bne);



      assign M1=(i_add)|(i_addu)|(i_addi)|(i_addiu)|(i_sub)|(i_subu)|(i_and)|(i_andi)|(i_or)|(i_ori)|
      (i_xor)|(i_xori)|(i_nor)|(i_lui)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl)|(i_srlv)|
      (i_slt)|(i_slti)|(i_sltu)|(i_sltiu)|(i_beq)|(i_bne)|(i_lw)|(i_sw);

      assign M2=(i_add)|(i_addu)|(i_addi)|(i_addiu)|(i_sub)|(i_subu)|(i_and)|(i_andi)|(i_or)|(i_ori)|
      (i_xor)|(i_xori)|(i_nor)|(i_lui)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl)|(i_srlv)|
      (i_slt)|(i_slti)|(i_sltu)|(i_sltiu);

      assign M3=(i_add)|(i_addu)|(i_addi)|(i_addiu)|(i_sub)|(i_subu)|(i_and)|(i_andi)|(i_or)|(i_ori)|
      (i_xor)|(i_xori)|(i_nor)|(i_lui)|(i_sllv)|(i_srav)|(i_srlv)|
      (i_slt)|(i_slti)|(i_sltu)|(i_sltiu)|(i_beq)|(i_bne)|(i_j)|(i_jal)|(i_jr)|(i_lw)|(i_sw);

      assign M4[1]=(i_addi)|(i_addiu)|(i_slti)|(i_sltiu)|(i_lw)|(i_sw);
      assign M4[0]=(i_andi)|(i_ori)|(i_xori)|(i_lui);

      assign M5=(Z&i_beq)|(~Z&i_bne);//由此可见 pc 在下降沿写入，根据运算结果决定

      assign M6=(i_j)|(i_jal);

      assign M7=i_jal;

      assign M8[1]=i_jal;
      assign M8[0]=(i_addi)|(i_addiu)|(i_andi)|(i_ori)|
```

```
        (i_xori)|(i_lui)|
        (i_slti)|(i_sltiu)|(i_beq)|(i_bne)|(i_j)|(i_jr)|(i_lw)|(i_sw);

        assign CS=(i_lw)|(i_sw);
        assign DM_R=(i_lw);
        assign DM_W=(i_sw);


endmodule


module alu(
input [31:0] a,
input [31:0] b,
input [3:0] aluc,
input reset,
output reg [31:0] r,
output reg zero,
output reg carry,
output reg negative,
output reg overflow
        );

        reg [31:0]kkk;

        always @(*)
        begin

        if(reset==1'b1)
        begin
        r=32'b0;
        zero=0;
        carry=0;
        negative=0;
        overflow=0;
        end
        else
        begin

        if (aluc == 4'b0000)
        begin
        r=a+b;

        if(r==0)
```

```
zero=1;
else
zero=0;
if(r<a||r<b)
carry=1;
else
carry=0;
if(r[31]==1)
negative=1;
else
negative=0;

end
else if (aluc == 4'b0010)
begin
r=a+b;

if(r==0)
zero=1;
else
zero=0;
if(r[31]==1)
negative=1;
else
negative=0;
if((a[31]==0&&b[31]==0&&r[31]==1)||(a[31]==1&&b[31]==1&&r[31]==0))
overflow=1;
else
overflow=0;


end
else if (aluc == 4'b0001)
begin
r=a-b;

if(r==0)
zero=1;
else
zero=0;
if(r>a)
carry=1;
else
carry=0;
```

```
  if(r[31]==1)
  negative=1;
  else
  negative=0;


  end
  else if (aluc == 4'b0011)
  begin
  r=a-b;

  if(r==0)
  zero=1;
  else
  zero=0;
  if(r[31]==1)
  negative=1;
  else
  negative=0;
if((a[31]==0&&b[31]==1&&r[31]==1)||(a[31]==1&&b[31]==0&&r[31]==0))
  overflow=1;
  else
  overflow=0;

  end
  else if (aluc == 4'b0100)
  begin
  r=a&b;

  if(r==0)
  zero=1;
  else
  zero=0;
  if(r[31]==1)
  negative=1;
  else
  negative=0;

  end
  else if (aluc == 4'b0101)
  begin
  r=a|b;

  if(r==0)
```

```
zero=1;
else
zero=0;
if(r[31]==1)
negative=1;
else
negative=0;

end
else if (aluc == 4'b0110)
begin
r=a^b;

if(r==0)
zero=1;
else
zero=0;
if(r[31]==1)
negative=1;
else
negative=0;

end
else if (aluc == 4'b0111)
begin
r=~(a|b);

if(r==0)
zero=1;
else
zero=0;
if(r[31]==1)
negative=1;
else
negative=0;

end
else if (aluc == 4'b1000||aluc == 4'b1001)
begin
r={b[15:0],16'b0};

if(r==0)
zero=1;
else
```

```
  zero=0;
  if(r[31]==1)
  negative=1;
  else
  negative=0;

  end
  else if (aluc == 4'b1011)
  begin
// r=(a<b)?1:0;

if(a[31]==0&&b[31]==0)
r=(a<b)?1:0;
else if(a[31]==0&&b[31]==1)
r=0;
else if(a[31]==1&&b[31]==0)
r=1;
else
r=(a<b)?1:0;



  if(a-b==0)
  zero=1;
  else
  zero=0;

  kkk=a-b;

  if(kkk[31]==1)
  negative=1;
  else
  negative=0;

  end
  else if (aluc == 4'b1010)
  begin
  r=(a<b)?1:0;

  if(a-b==0)
  zero=1;
  else
  zero=0;
  if(a<b)
```

```
  carry=1;
  else
  carry=0;
  if(r[31]==1)
  negative=1;
  else
  negative=0;

  end
  else if (aluc == 4'b1100)
  begin
// r=b>>>a;
  r=($signed(b)) >>> a;

  if(r==0)
  zero=1;
  else
  zero=0;


  if(a==0)
  carry=0;
  else if(a<=32)
  carry=b[a-1];
  else
  carry=b[31];
  //carry=

  if(r[31]==1)
  negative=1;
  else
  negative=0;


  end
  else if (aluc == 4'b1110||aluc == 4'b1111)
  begin
  r=b<<a;

  if(r==0)
  zero=1;
  else
  zero=0;
```

```
        if(a==0)
        carry=0;
        else if(a<=32)
        carry=b[32-a];
        else
        carry=0;

        if(r[31]==1)
        negative=1;
        else
        negative=0;

        end
        else if (aluc == 4'b1101)
        begin
        r=b>>a;

        if(r==0)
        zero=1;
        else
        zero=0;


        if(a==0)
        carry=0;
        else if(a<=32)
        carry=b[a-1];
        else
        carry=0;
        //carry=

        if(r[31]==1)
        negative=1;
        else
        negative=0;

        end
        else
        begin

        end
        end
        end
endmodule
```

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

四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

```
`timescale 1ns / 1ns
module test(
    );
    reg clk, rst;
    wire [31:0] inst, pc;

    integer file_output;
    integer counter=0 ;
    initial begin
        file_output = $fopen("result.txt", "a");
        clk = 1'b0;
        rst = 1'b1;
        #50 rst=1'b0;
    end

    always begin
        #20 clk = !clk;
    end

    always @(posedge clk) begin
        if(counter>=400)
        begin
        $fclose(file_output);
        end
        else
        begin
        counter=counter+1;
```

```
        $fdisplay(file_output, "pc: %h", pc);
        $fdisplay(file_output, "instr: %h", inst);
        $fdisplay(file_output, "regfile0: %h", test.uut.sccpu.cpu_ref.array_reg[0]);
        $fdisplay(file_output, "regfile1: %h", test.uut.sccpu.cpu_ref.array_reg[1]);
        $fdisplay(file_output, "regfile2: %h", test.uut.sccpu.cpu_ref.array_reg[2]);
        $fdisplay(file_output, "regfile3: %h", test.uut.sccpu.cpu_ref.array_reg[3]);
        $fdisplay(file_output, "regfile4: %h", test.uut.sccpu.cpu_ref.array_reg[4]);
        $fdisplay(file_output, "regfile5: %h", test.uut.sccpu.cpu_ref.array_reg[5]);
        $fdisplay(file_output, "regfile6: %h", test.uut.sccpu.cpu_ref.array_reg[6]);
        $fdisplay(file_output, "regfile7: %h", test.uut.sccpu.cpu_ref.array_reg[7]);
        $fdisplay(file_output, "regfile8: %h", test.uut.sccpu.cpu_ref.array_reg[8]);
        $fdisplay(file_output, "regfile9: %h", test.uut.sccpu.cpu_ref.array_reg[9]);
        $fdisplay(file_output, "regfile10: %h", test.uut.sccpu.cpu_ref.array_reg[10]);
        $fdisplay(file_output, "regfile11: %h", test.uut.sccpu.cpu_ref.array_reg[11]);
        $fdisplay(file_output, "regfile12: %h", test.uut.sccpu.cpu_ref.array_reg[12]);
        $fdisplay(file_output, "regfile13: %h", test.uut.sccpu.cpu_ref.array_reg[13]);
        $fdisplay(file_output, "regfile14: %h", test.uut.sccpu.cpu_ref.array_reg[14]);
        $fdisplay(file_output, "regfile15: %h", test.uut.sccpu.cpu_ref.array_reg[15]);
        $fdisplay(file_output, "regfile16: %h", test.uut.sccpu.cpu_ref.array_reg[16]);
        $fdisplay(file_output, "regfile17: %h", test.uut.sccpu.cpu_ref.array_reg[17]);
        $fdisplay(file_output, "regfile18: %h", test.uut.sccpu.cpu_ref.array_reg[18]);
        $fdisplay(file_output, "regfile19: %h", test.uut.sccpu.cpu_ref.array_reg[19]);
        $fdisplay(file_output, "regfile20: %h", test.uut.sccpu.cpu_ref.array_reg[20]);
        $fdisplay(file_output, "regfile21: %h", test.uut.sccpu.cpu_ref.array_reg[21]);
        $fdisplay(file_output, "regfile22: %h", test.uut.sccpu.cpu_ref.array_reg[22]);
        $fdisplay(file_output, "regfile23: %h", test.uut.sccpu.cpu_ref.array_reg[23]);
        $fdisplay(file_output, "regfile24: %h", test.uut.sccpu.cpu_ref.array_reg[24]);
        $fdisplay(file_output, "regfile25: %h", test.uut.sccpu.cpu_ref.array_reg[25]);
        $fdisplay(file_output, "regfile26: %h", test.uut.sccpu.cpu_ref.array_reg[26]);
        $fdisplay(file_output, "regfile27: %h", test.uut.sccpu.cpu_ref.array_reg[27]);
        $fdisplay(file_output, "regfile28: %h", test.uut.sccpu.cpu_ref.array_reg[28]);
        $fdisplay(file_output, "regfile29: %h", test.uut.sccpu.cpu_ref.array_reg[29]);
        $fdisplay(file_output, "regfile30: %h", test.uut.sccpu.cpu_ref.array_reg[30]);
        $fdisplay(file_output, "regfile31: %h", test.uut.sccpu.cpu_ref.array_reg[31]);
        end


end

sccomp_dataflow uut(
        .clk_in(clk),
        .reset(rst),
        .inst(inst),
        .pc(pc)
```

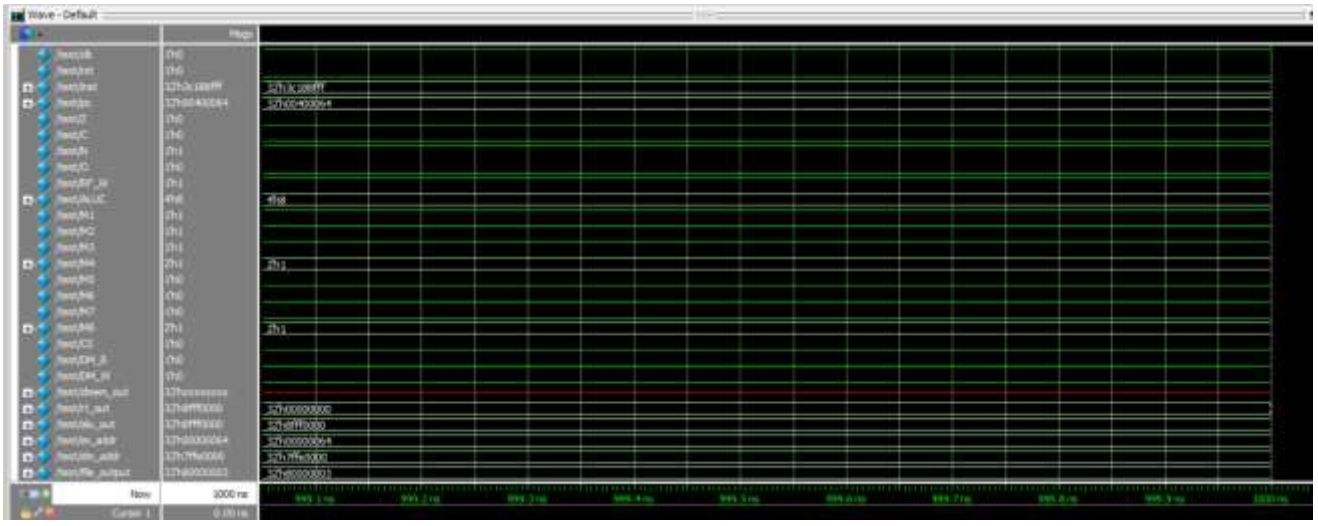
```
    );
endmodule
```

五、实验结果

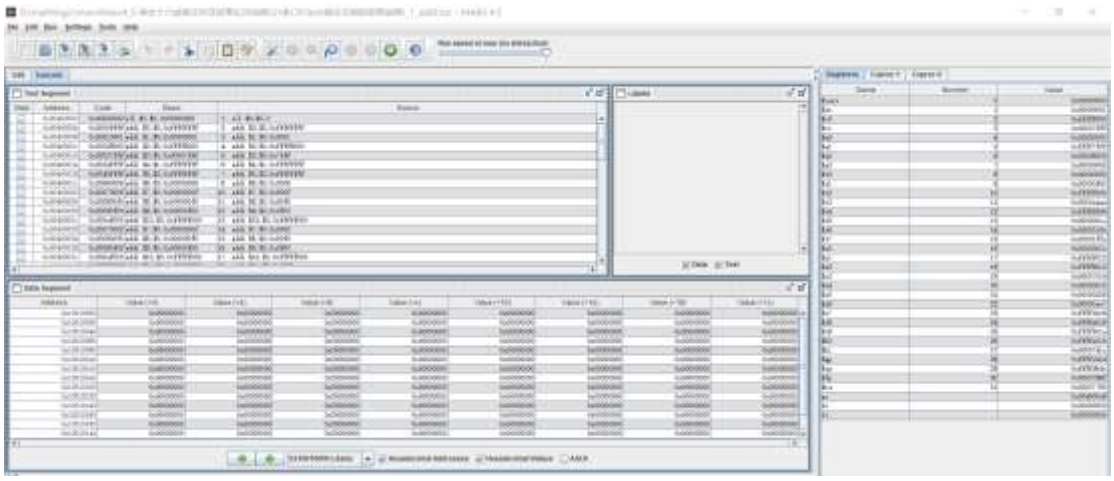
（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

顶层模块对外接口波形图：

包含控制信号的波形图：



从 Mars 中获取 coe 文件和 result 文件：



Mars 导出的 coe 文件：

test.coe - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
memory_initialization_radix = 16;
memory_initialization_vector =
00000000
2000ffff
20010001
20028000
20037fff
2024ffff
2045ffff
20660006
2007000f
200800f0
20090f00
200af000
2007000f
200800f0
20090f00
200af000
200b5555
200caaaa
```

过程中打印的 result.txt 文件：

result.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
pc: 00400000
instr: 3c1d1001
regfile0: 00000000
regfile1: 00000000
regfile2: 00000000
regfile3: 00000000
regfile4: 00000000
regfile5: 00000000
regfile6: 00000000
regfile7: 00000000
regfile8: 00000000
regfile9: 00000000
regfile10: 00000000
regfile11: 00000000
regfile12: 00000000
regfile13: 00000000
regfile14: 00000000
regfile15: 00000000
regfile16: 00000000
regfile17: 00000000
regfile18: 00000000
regfile19: 00000000
regfile20: 00000000
regfile21: 00000000
```

生成的 result 文件与 Mars 的执行结果在文本比较器中比较：