同济大学计算机系

数字逻辑课程实验报告

学　号	**1850772**
姓　名	张哲源
专　业	计算机科学与技术
授课老师	张冬冬老师

目录

一、实验内容

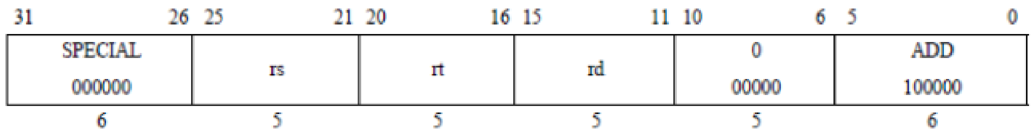
1. 使用 verilog 建模,实现 31 条 MIPS 指令的 CPU 的设计仿真和测试,验证结果

二、数据通路设计图

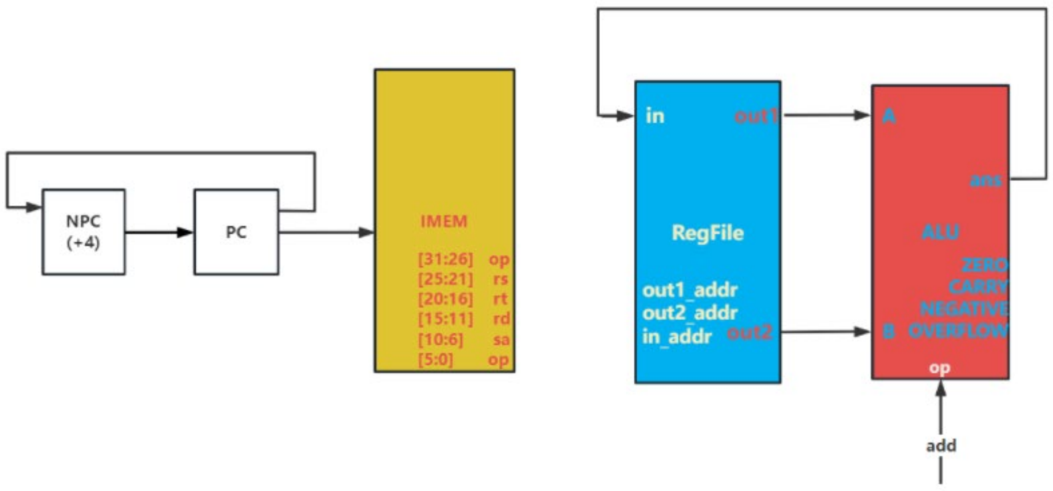
R 型指令

通路类型 1

1.ADD



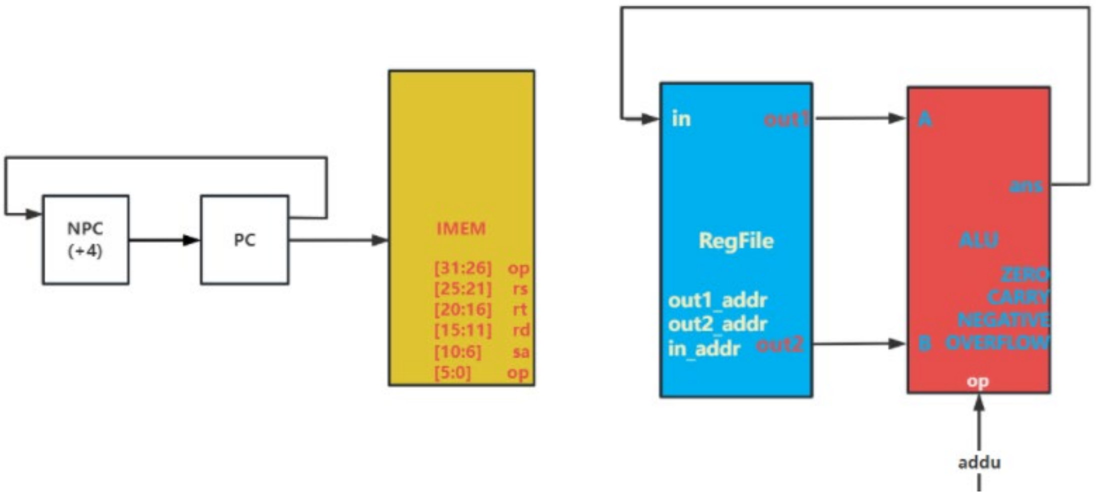
31　　26	25　　21	20　　16	15　　11	10　　6	5　　0
SPECIAL 000000	rs	rt	rd	0 00000	ADD 100000
6	5	5	5	5	6



```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A + B -> RES)
ans -> in
```

2.ADDU

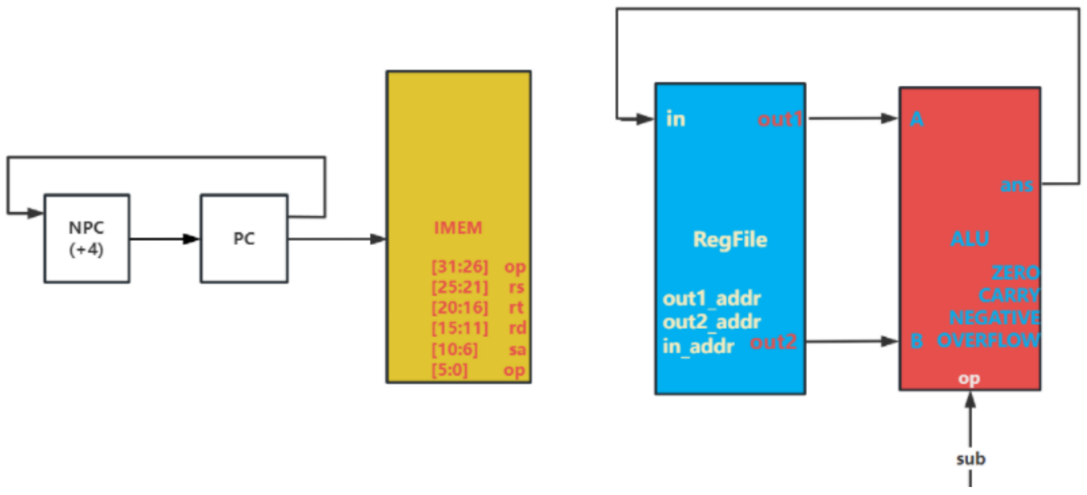
31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	ADDU 100001
6	5	5	5	5	6



```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A + B -> RES)
ans -> in
```

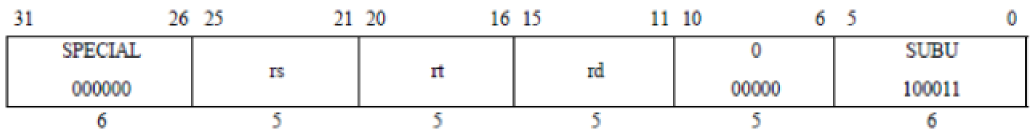
3.SUB

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SUB 100010
6	5	5	5	5	6

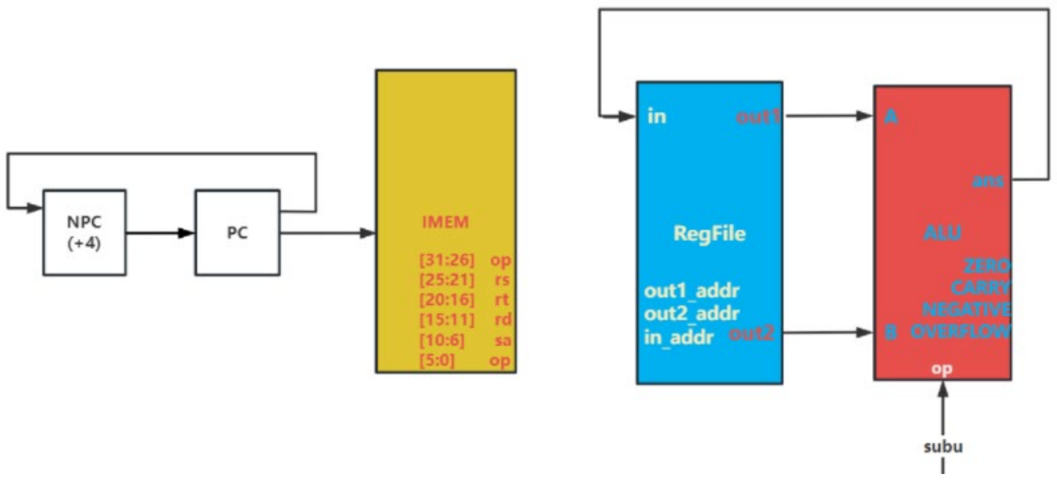


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A - B -> RES)
ans -> in
```

4.SUBU



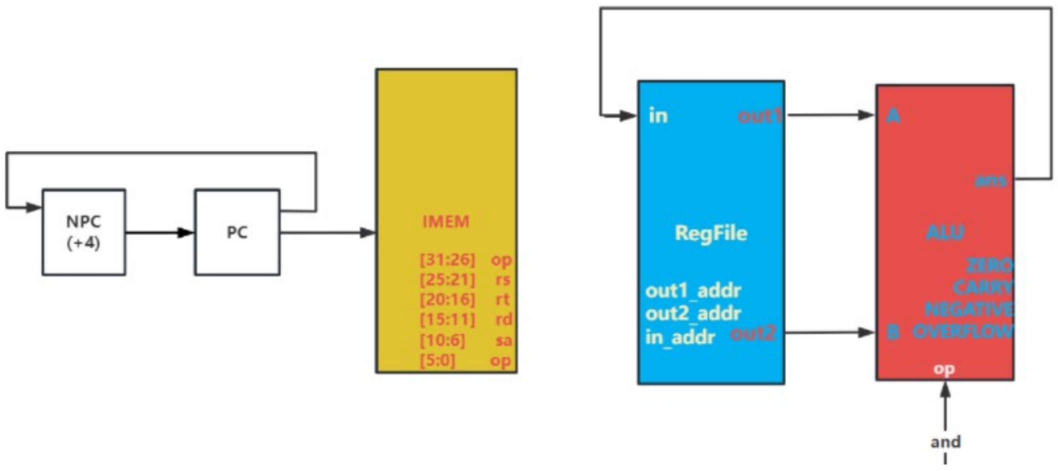
31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SUBU 100011
6	5	5	5	5	6



```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A - B -> RES)
ans -> in
```

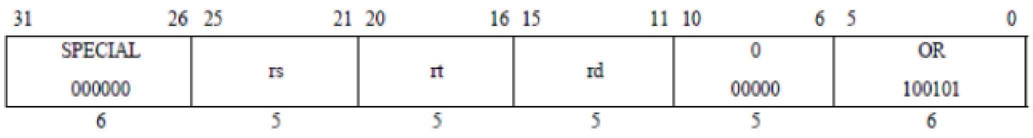
5.AND

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	AND 100100
6	5	5	5	5	6

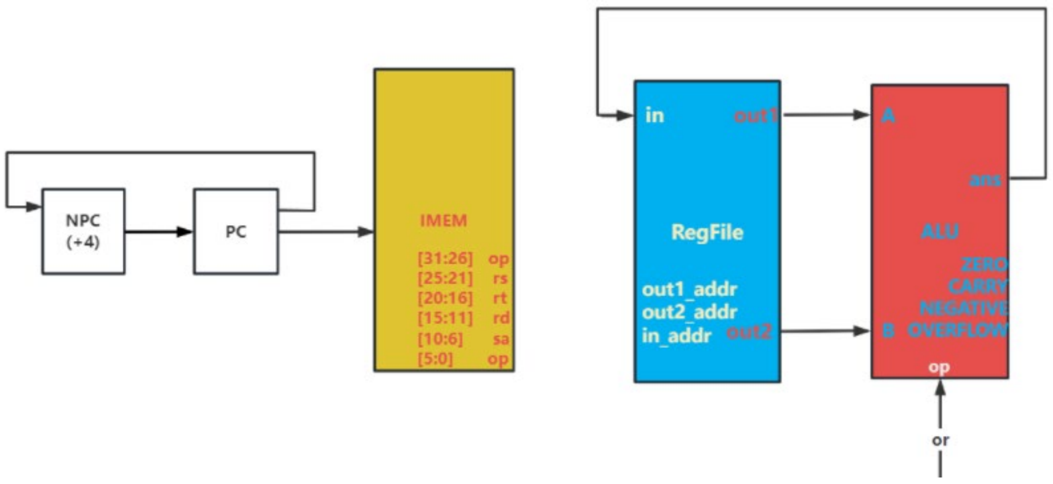


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A & B -> RES)
ans -> in
```

6.OR

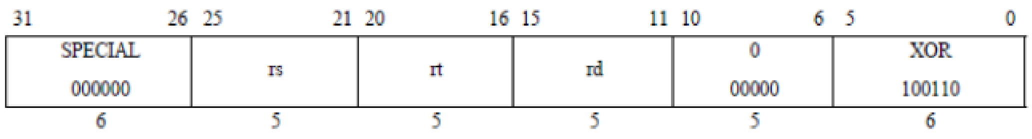


31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	OR 100101
6	5	5	5	5	6

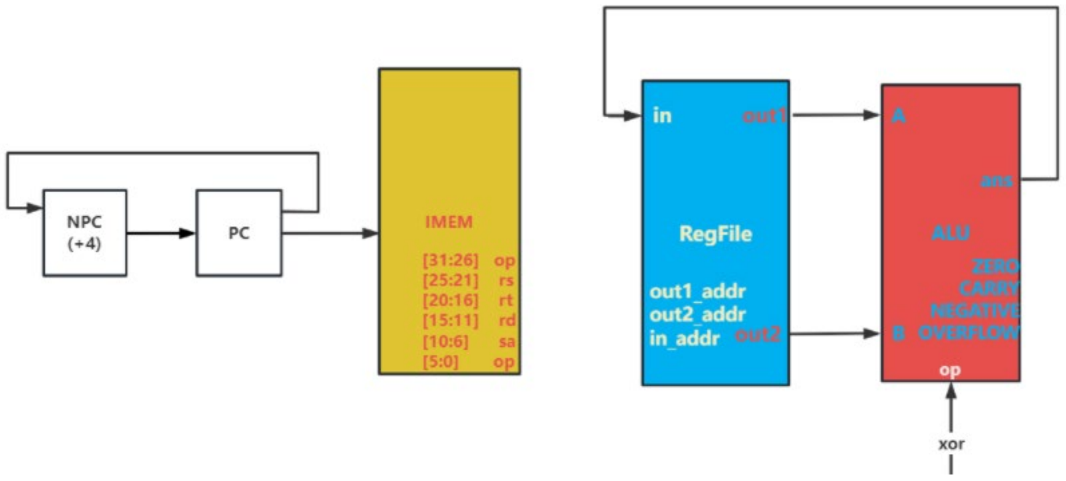


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A | B -> RES)
ans -> in
```

7.XOR

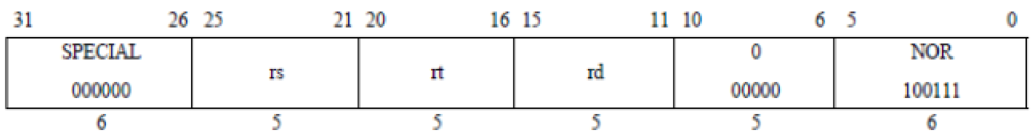


31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	XOR 100110
6	5	5	5	5	6

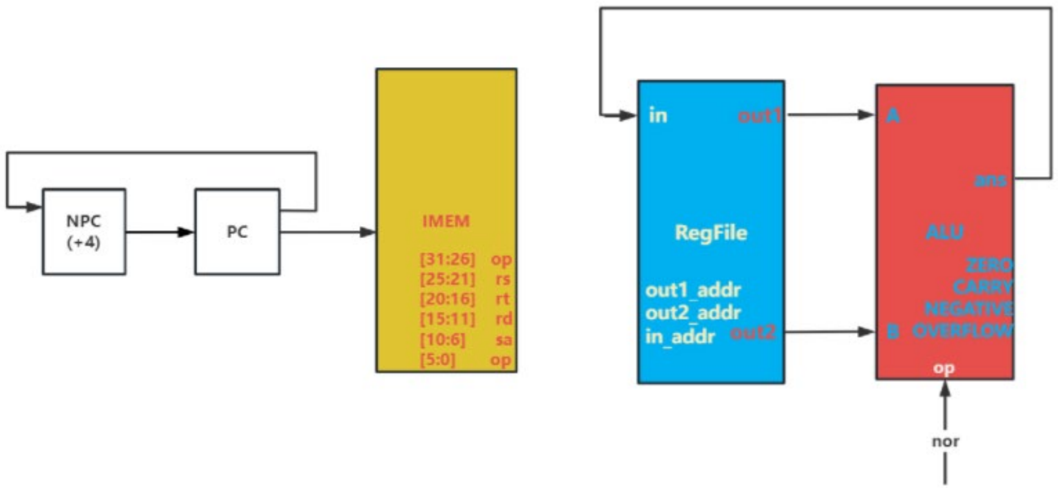


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A ⊕ B -> RES)
ans -> in
```

8. NOR



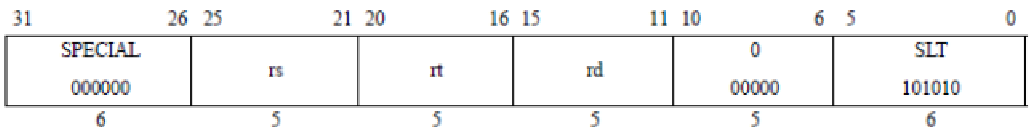
31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	NOR 100111
6	5	5	5	5	6



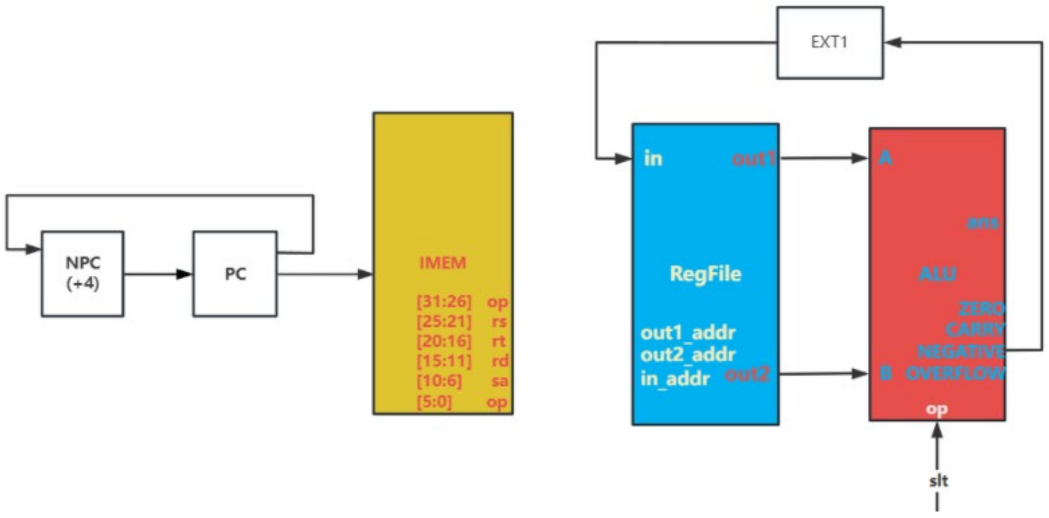
```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A ⨀ B -> RES)
ans -> in
```

通路类型 2

9.SLT

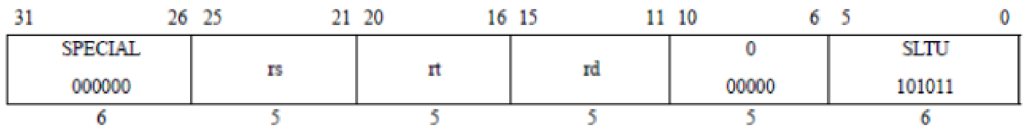


31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SLT 101010
6	5	5	5	5	6

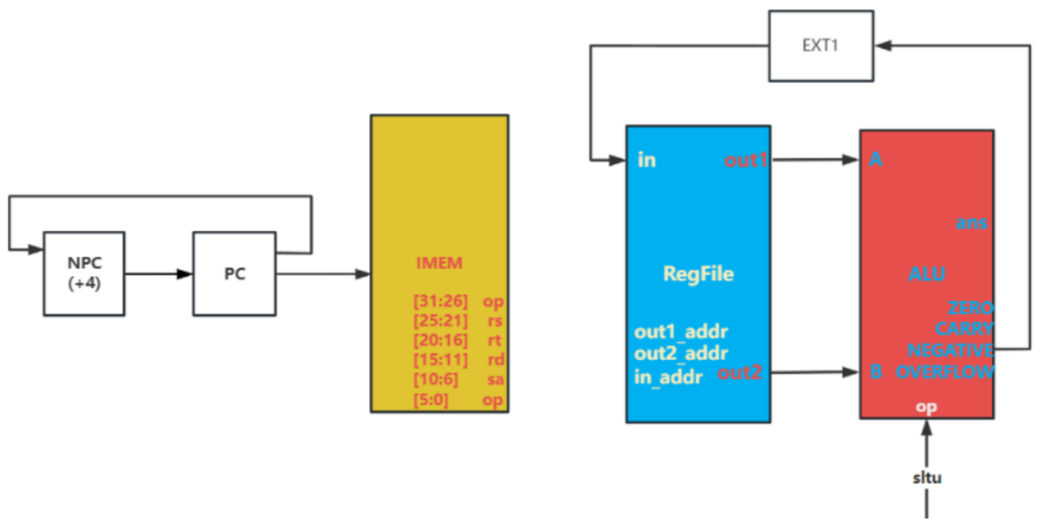


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A - B -> ans) //相减判断，负数则为 Rs 中数小
negative -> EXT1 //注意要做扩展
EXT1 -> in
```

10.SLTU



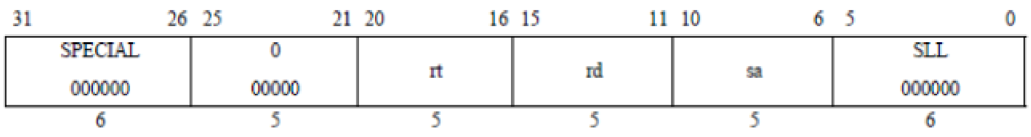
31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SLTU 101011
6	5	5	5	5	6



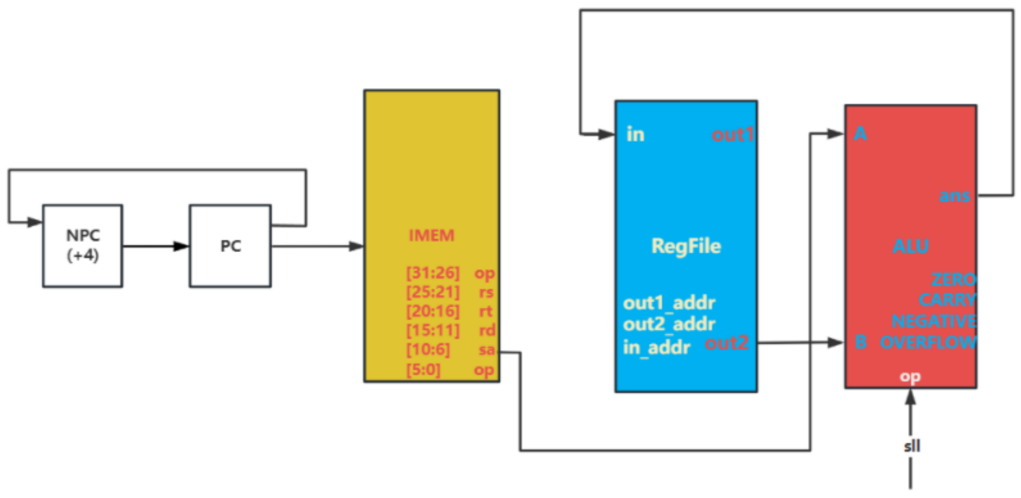
```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1 -> A , out2 -> B
(A - B -> ans) //相减判断，负数则为 Rs 中数小
negative -> EXT1 //注意要做扩展
EXT1 -> in
```

通路类型 3

11.SLL

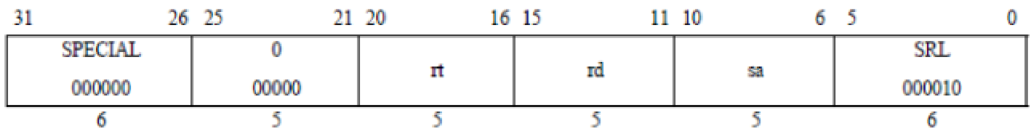


31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	0 00000	rt	rd	sa	SLL 000000
6	5	5	5	5	6

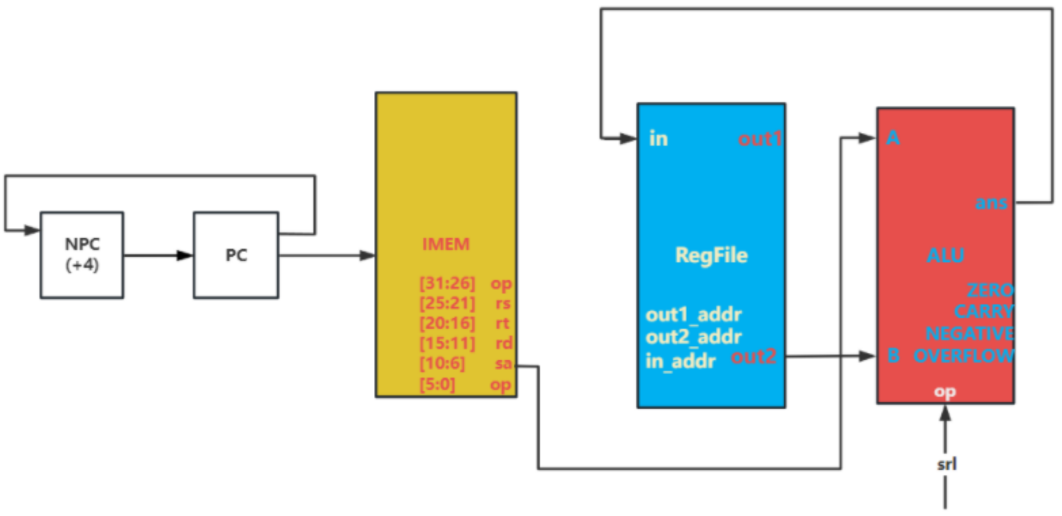


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5 -> A
out2 -> B
(B<<A -> ans)
ans -> in
```

12.SRL

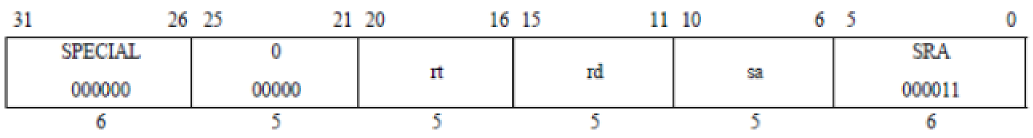


31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	0 00000	rt	rd	sa	SRL 000010
6	5	5	5	5	6

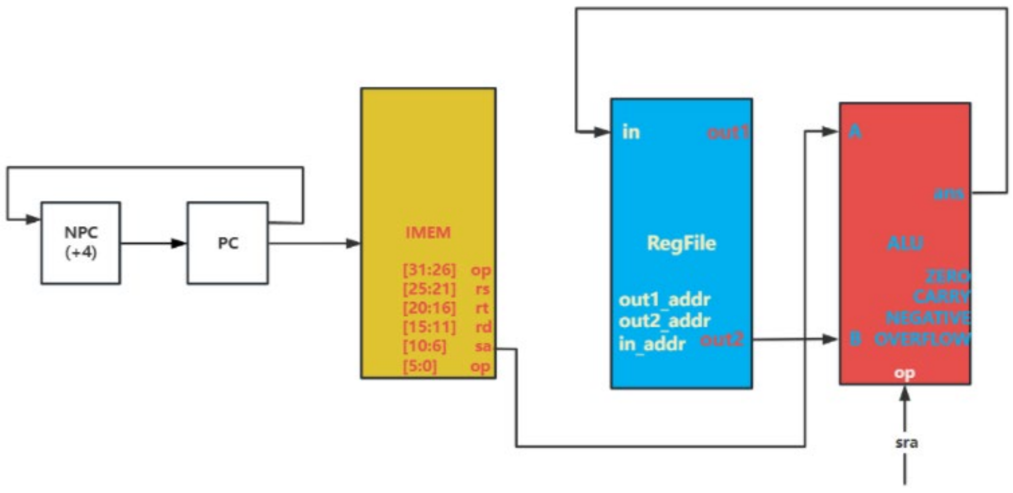


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5 -> A
out2 -> B
(B>>A -> ans)
ans -> in
```

13.SRA



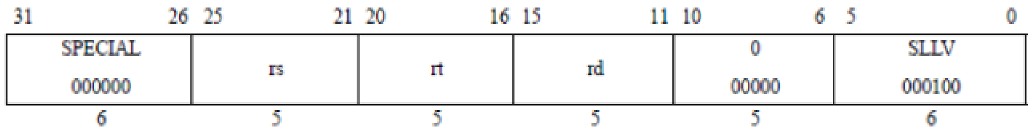
31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	0 00000	rt	rd	sa	SRA 000011
6	5	5	5	5	6



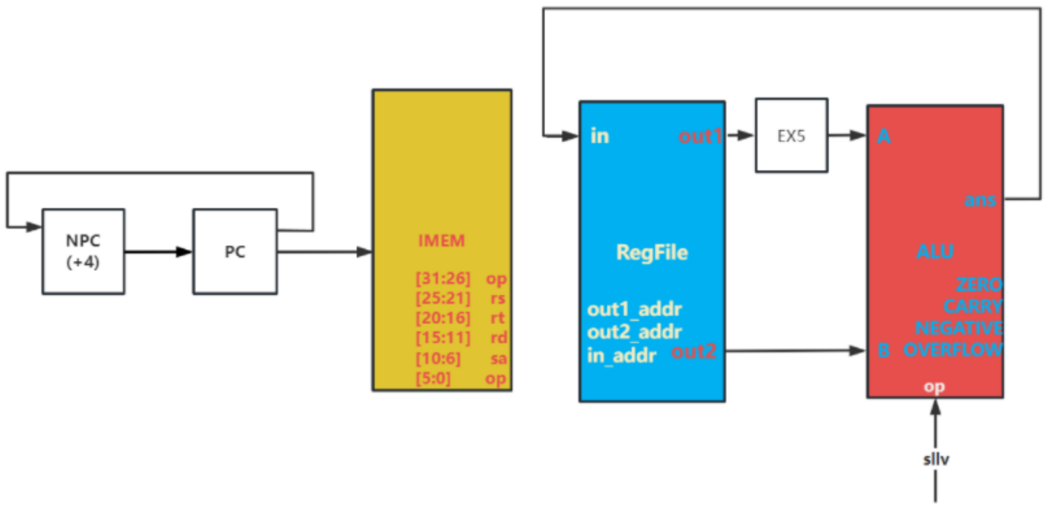
```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5 -> A
out2 -> B
(B>>A -> ans)
ans -> in
```

通路类型 4

14.SLLV

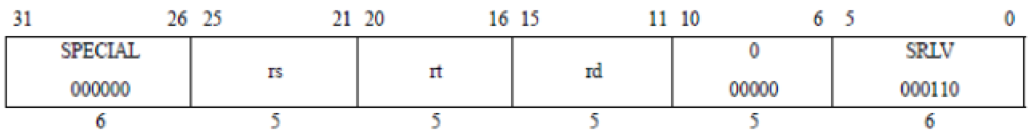


31　　26	25　　21	20　　16	15　　11	10　　6	5　　0
SPECIAL 000000	rs	rt	rd	0 00000	SLLV 000100
6	5	5	5	5	6

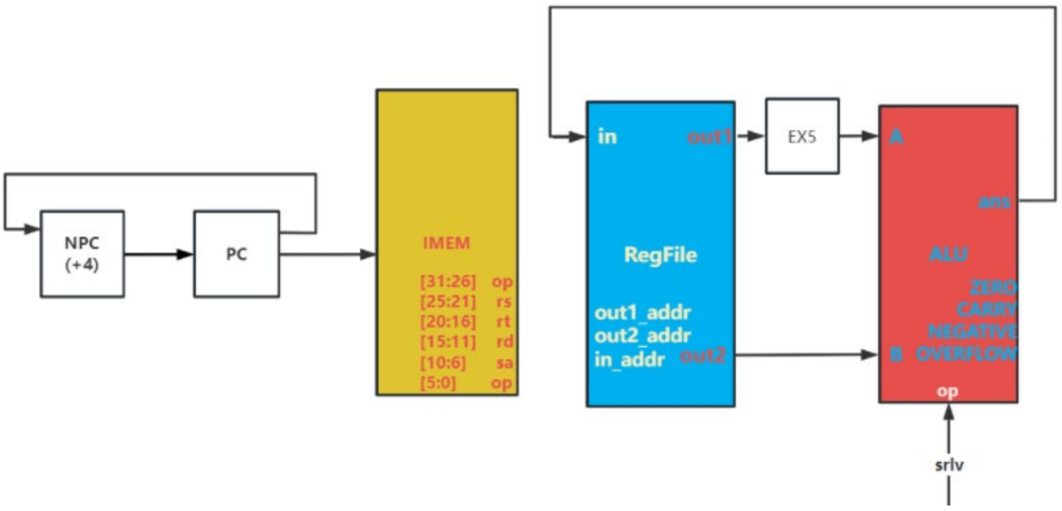


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1[4:0] -> EXT5
EXT5 -> A
out2 -> B
(A<<B -> ans)
ans -> in
```

15.SRLV

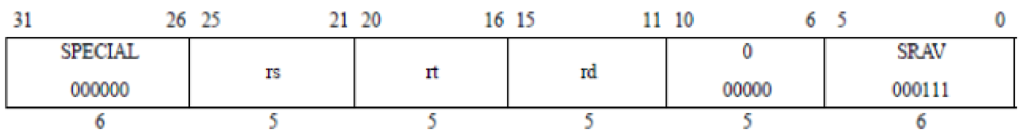


31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SRLV 000110
6	5	5	5	5	6

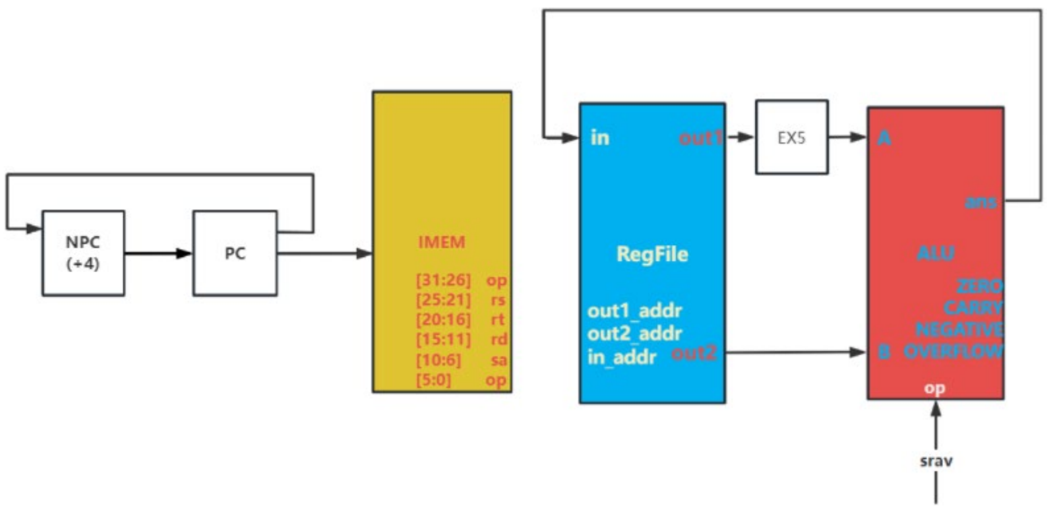


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1[4:0] -> EXT5
EXT5 -> A
out2 -> B
(A >> B -> ans)
ans -> in
```

16.SRAV



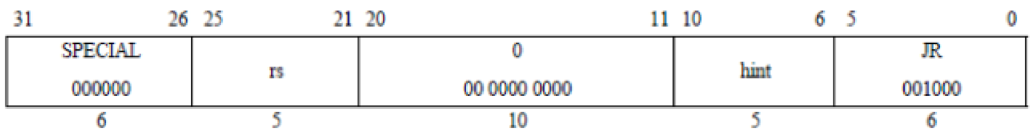
31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SRAV 000111
6	5	5	5	5	6



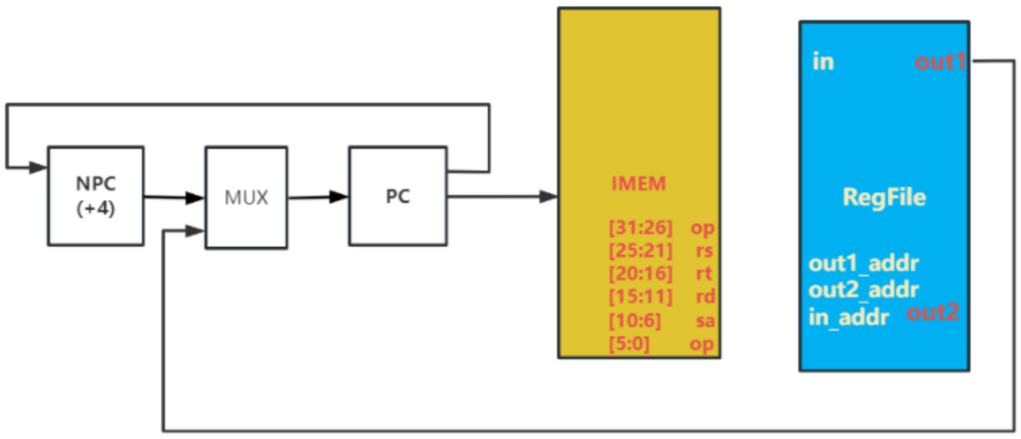
```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
out1[4:0] -> EXT5
EXT5 -> A
out2 -> B
(A >> B -> ans)
ans -> in
```

通路类型 5

17.JR



31 26	25 21	20 11	10 6	5 0
SPECIAL 000000	rs	0 00 0000 0000	hint	JR 001000
6	5	10	5	6

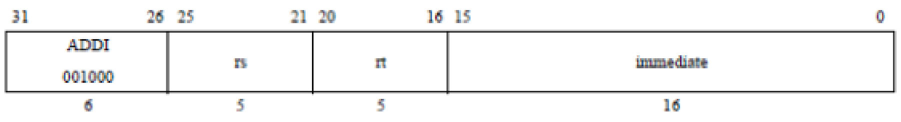


```
PC -> IMEM
PC + 4 -> NPC //无关指令
out1 -> MUX
MUX_OUT -> PC
NPC -> MUX //无关指令
```

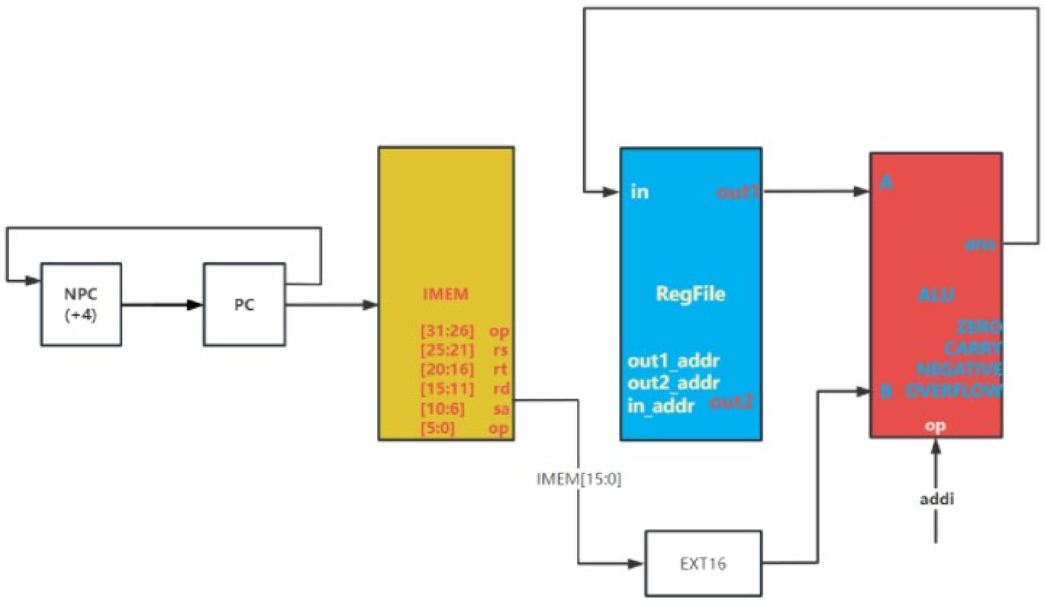
I 型指令

通路类型 6

18. ADDI

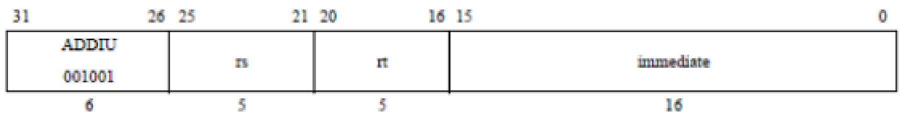


31 26	25 21	20 16	15 0
ADDI 001000	rs	rt	immediate
6	5	5	16

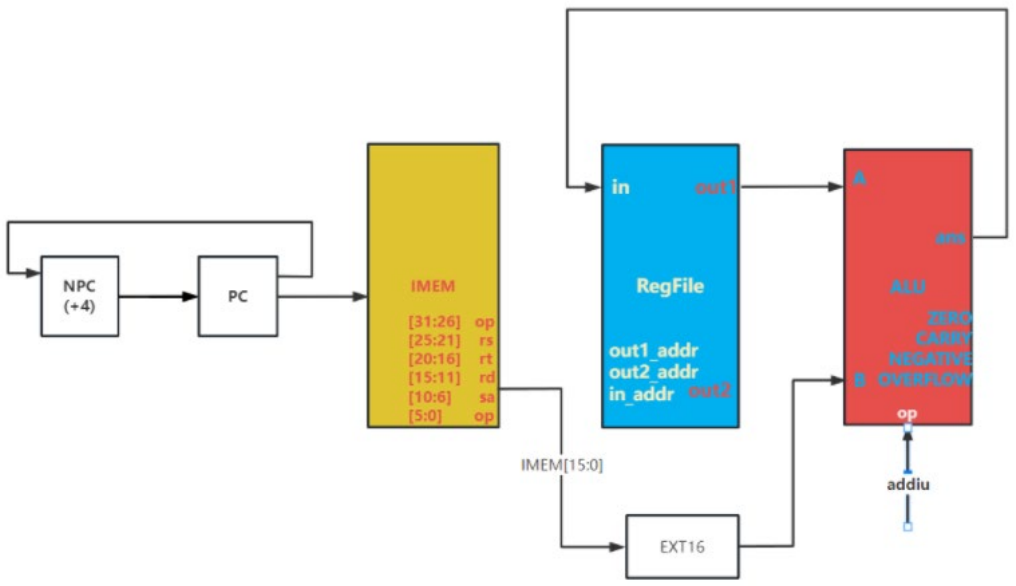


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A + B -> RES)
ans -> Rd
```

19.addiu

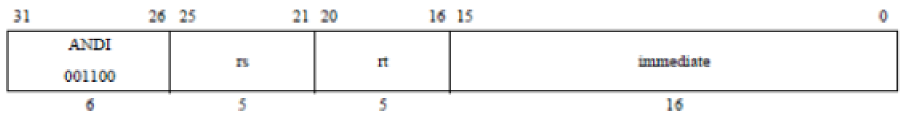


31 26	25 21	20 16	15 0
ADDIU 001001	rs	rt	immediate
6	5	5	16

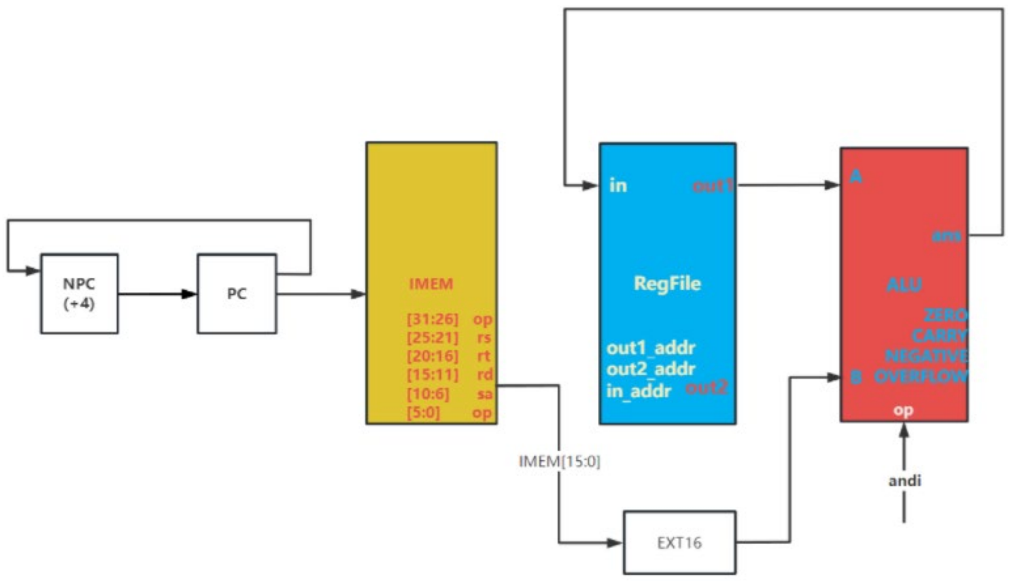


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A + B -> RES)
ans -> Rd
```

20.andi

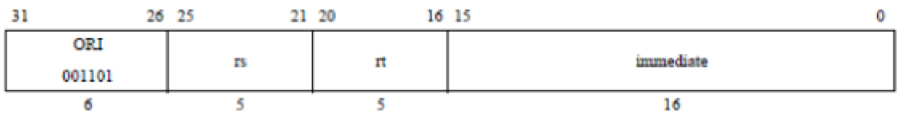


31 26	25 21	20 16	15 0
ANDI 001100	rs	rt	immediate
6	5	5	16

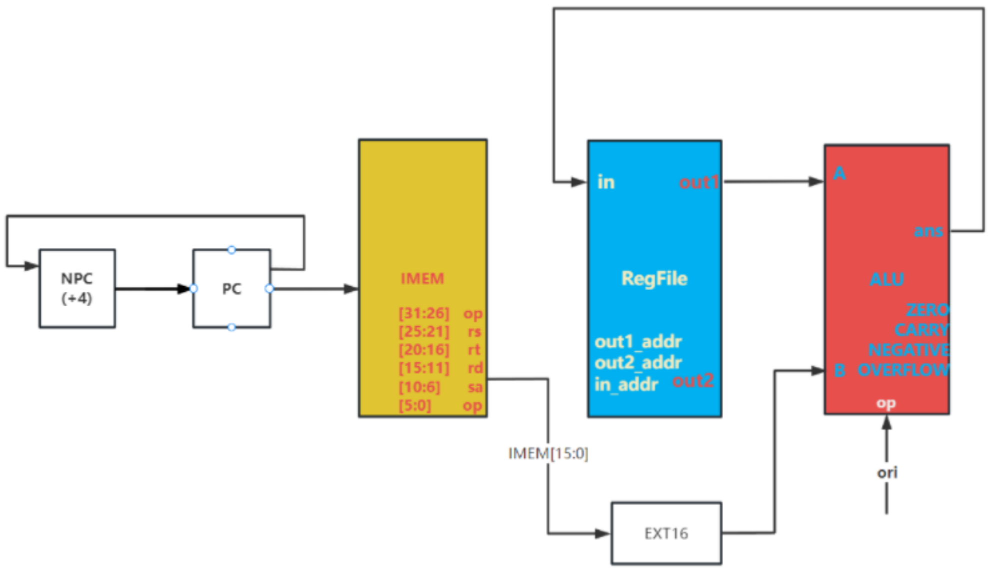


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A & B -> RES)
ans -> Rd
```

21.ori

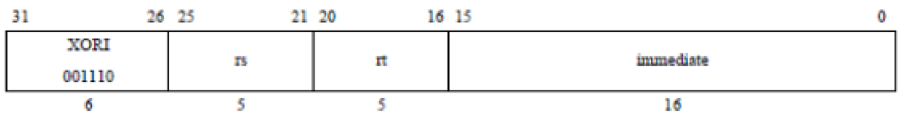


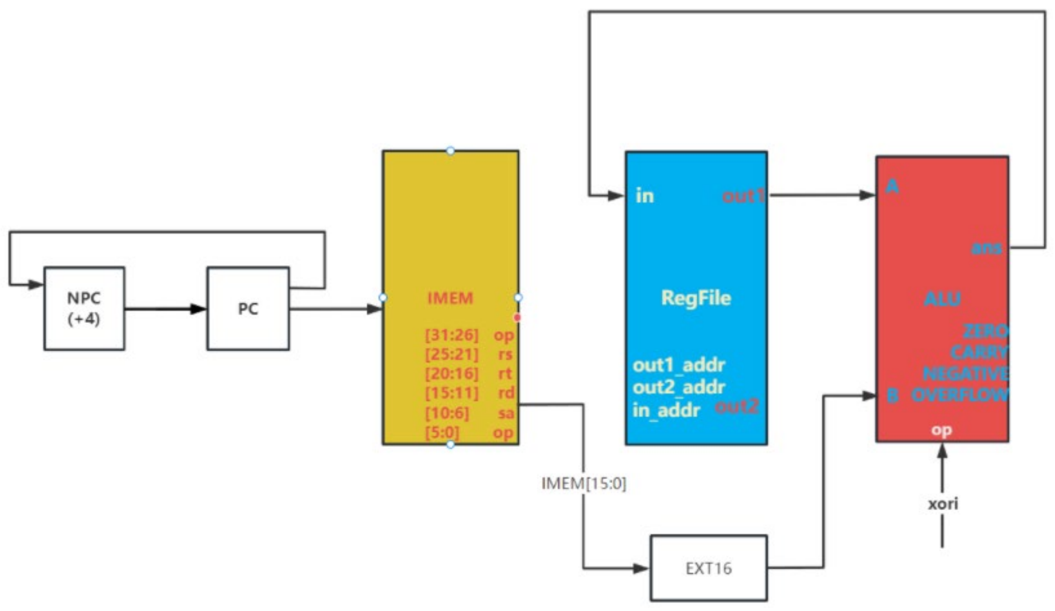
31 26	25 21	20 16	15 0
ORI 001101	rs	rt	immediate
6	5	5	16



```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A | B -> RES)
ans -> Rd
```

22.xori

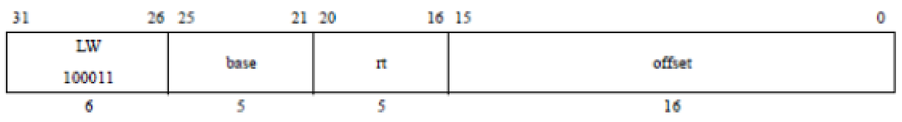


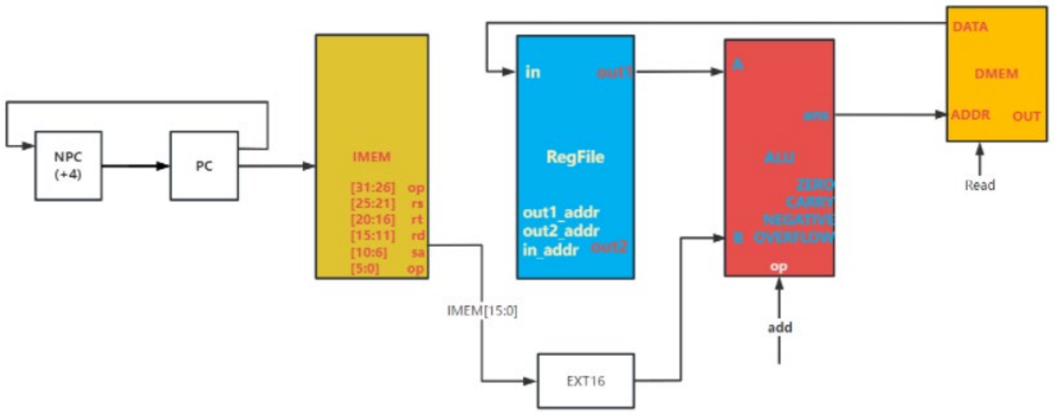


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16-> B
out1 -> A
(A ⊕ B -> RES)
ans -> Rd
```

通路类型 7

23.lw

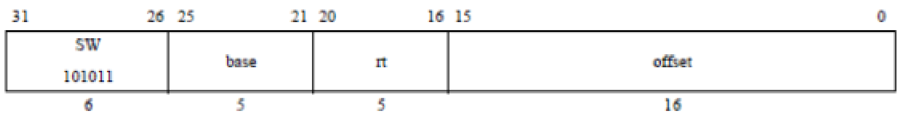


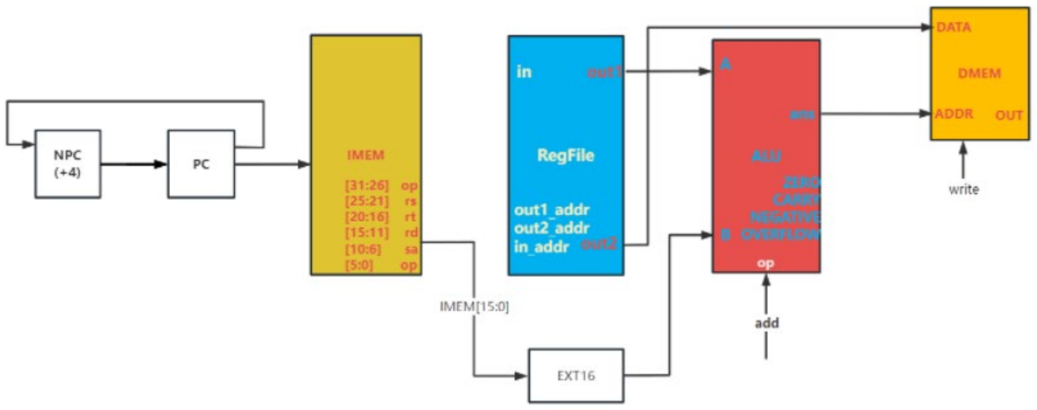


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16 -> B
out1 -> A
(A + B -> ans)
ans -> DMEM_ADDR
data -> in
```

通路类型 8

24.sw

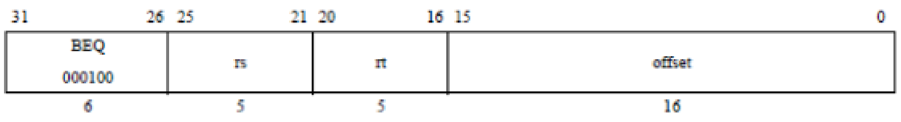


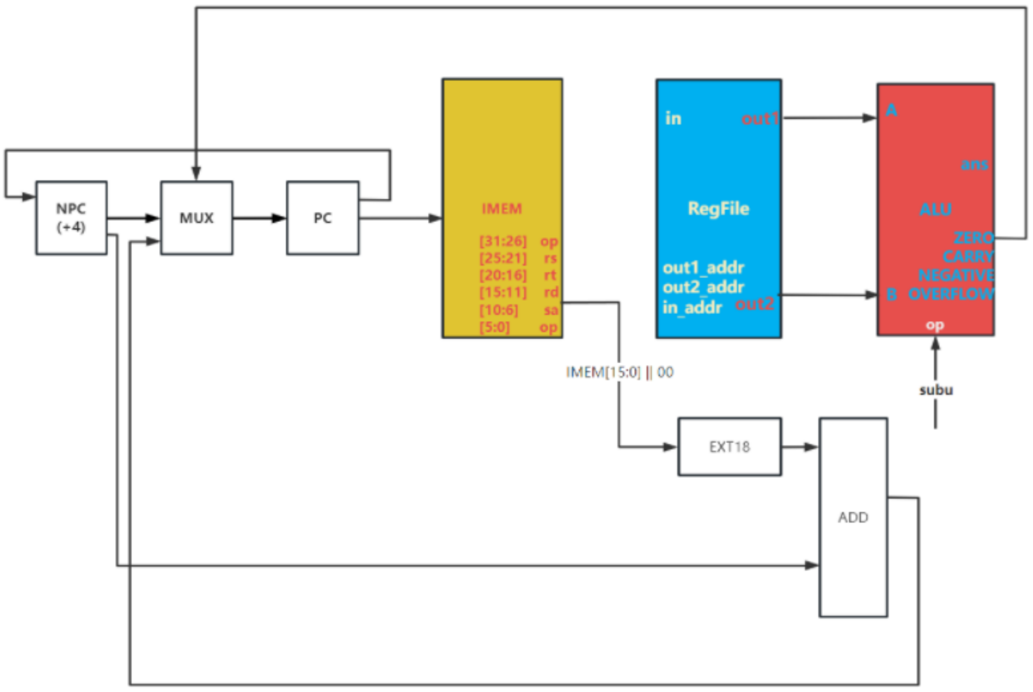


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_OUT -> B
out1 -> A
(A + B -> ans)
out2 -> DMEM
ans -> DMEM_ADDR
```

通路类型 9

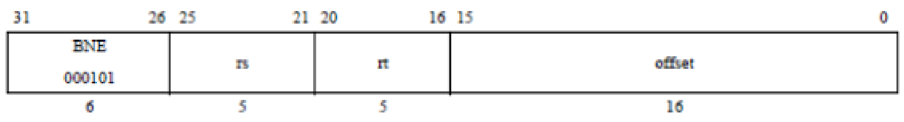
25.beq



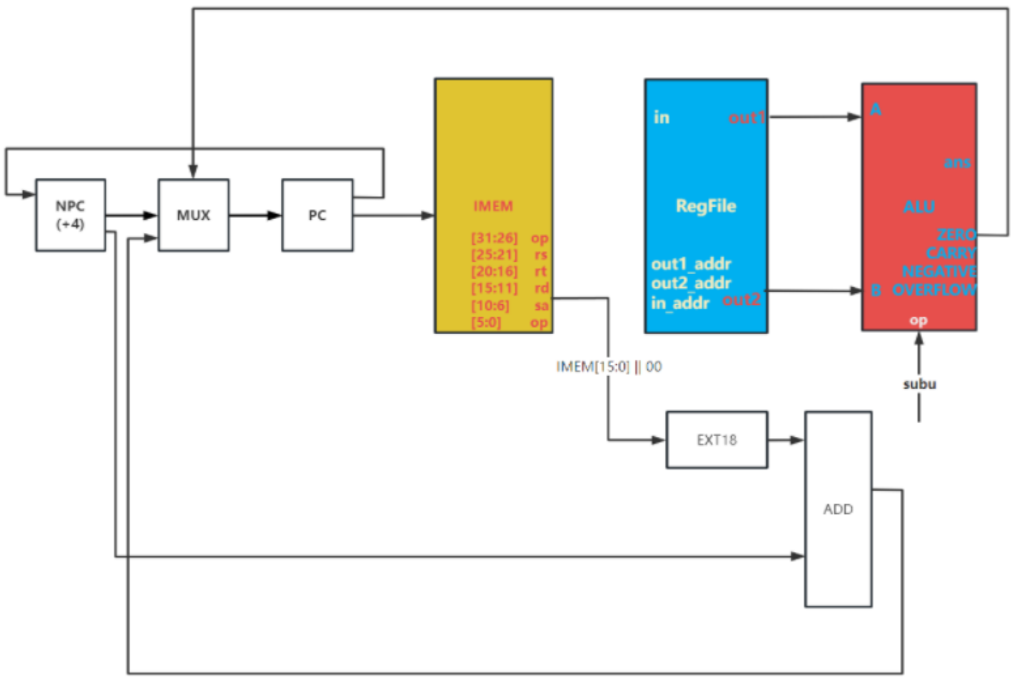


```
PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
IMEM[15:0] || 02 -> EXT18
EXT18_OUT -> ADD_A
NPC -> ADD_B
(ADD_A + ADD_B -> ADD_OUT)
ADD_OUT -> MUX
out1 -> A
out2 -> B
(A - B -> RES)
zero -> MUX
MUX -> PC
```

26.bne



31 26	25 21	20 16	15 0
BNE 000101	rs	rt	offset
6	5	5	16

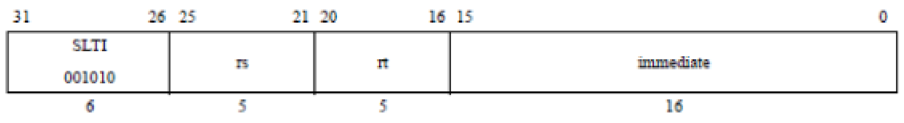


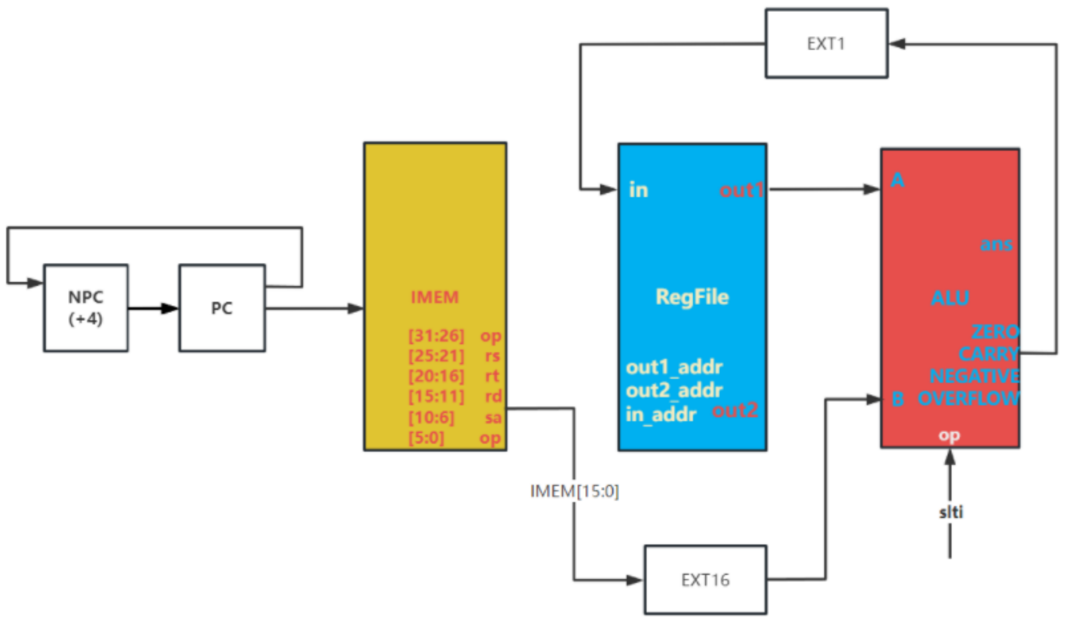
```
PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
IMEM[15:0] || 02 -> EXT18
EXT18_OUT -> ADD_A
NPC -> ADD_B
(ADD_A + ADD_B -> ADD_OUT)
ADD_OUT -> MUX
out1 -> A
out2 -> B
(A - B -> RES)
```

```
zero -> MUX
MUX -> PC
```

通路类型 10

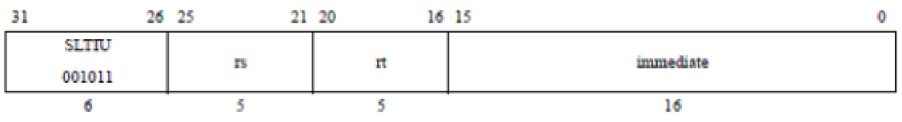
27.slti

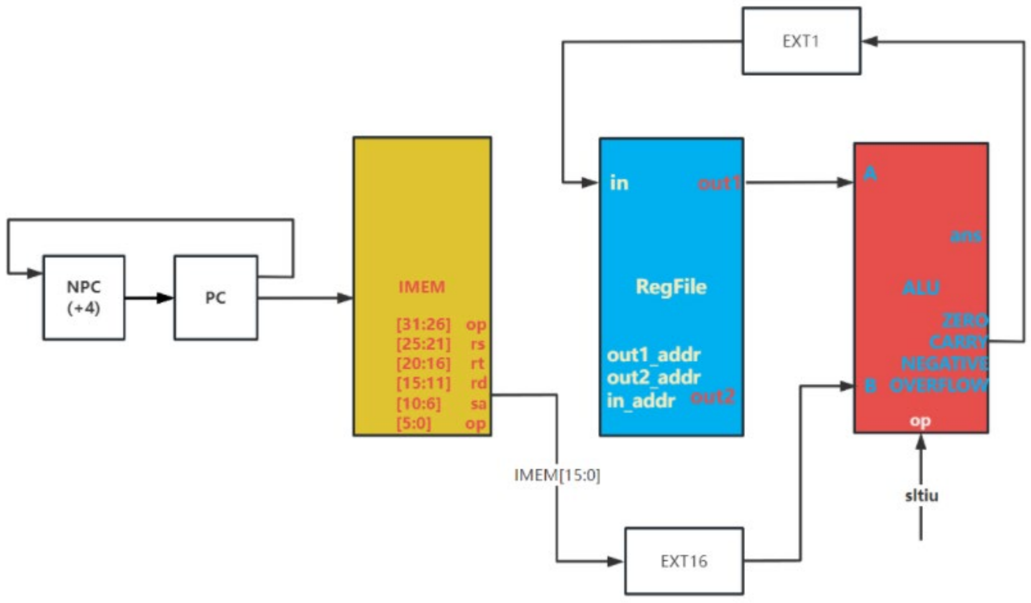




```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16 -> B
out1 -> A
(A - B -> RES)
Carry -> EXT1
EXT1 -> in
```

28.sltiu

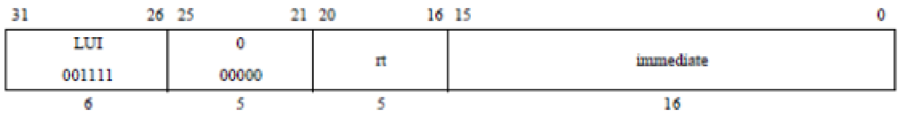


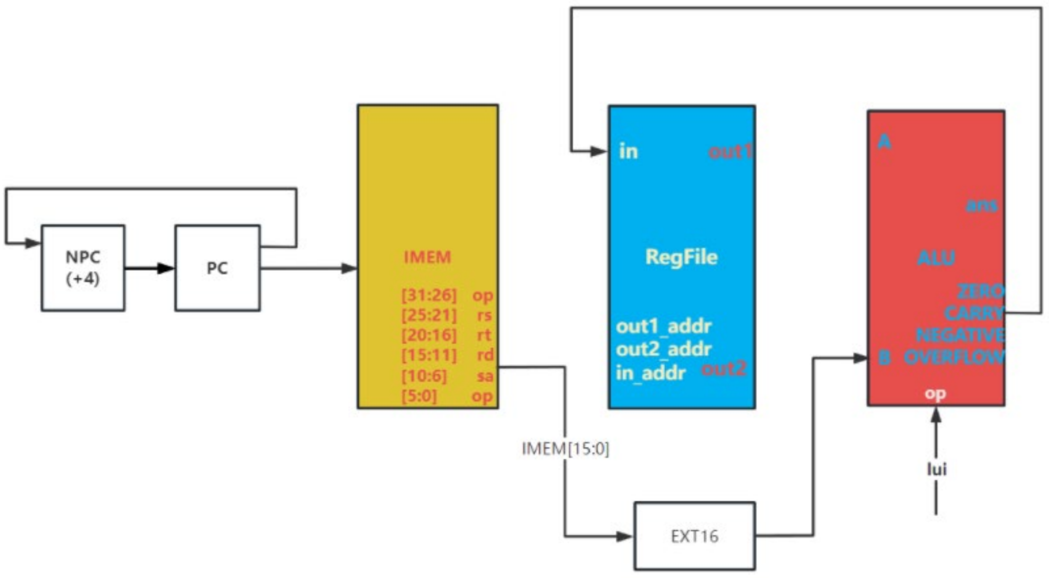


```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16 -> B
out1 -> A
(A - B -> ans)
Carry -> EXT1
EXT1 -> in
```

通路类型 11

29.lui



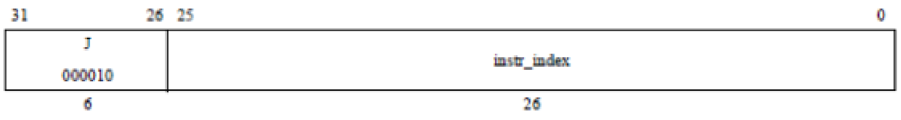


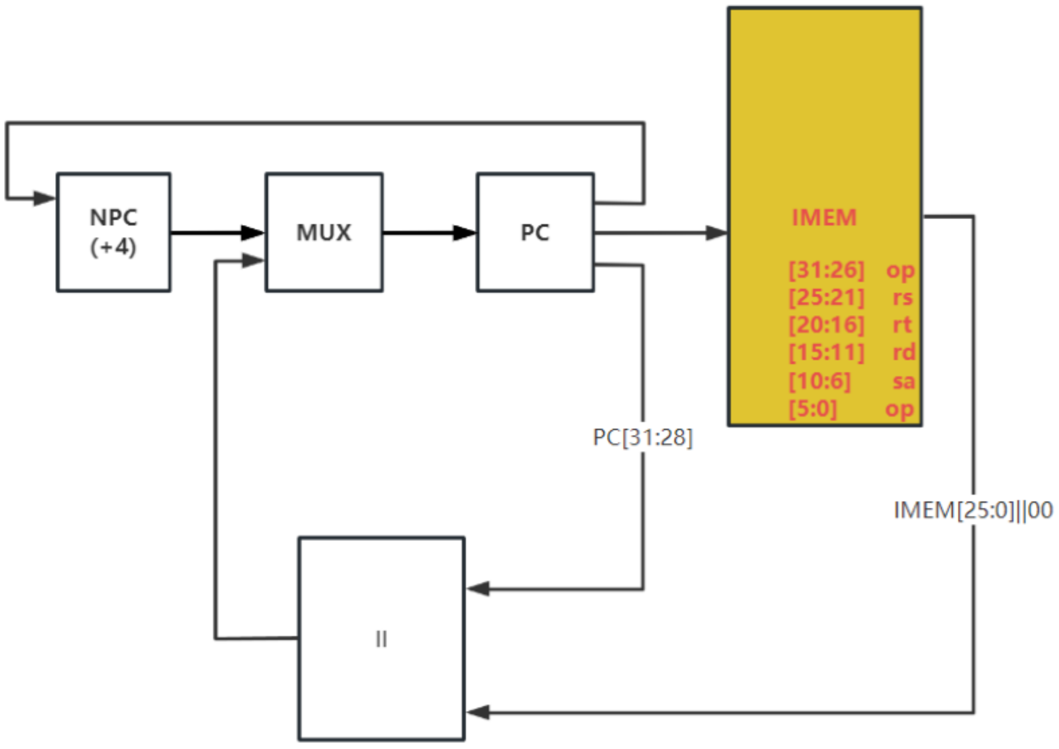
```
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16 -> B
ans -> in(!!!!!!上面画错了!!!!!!)
```

J 型指令

通路类型 12

30.j



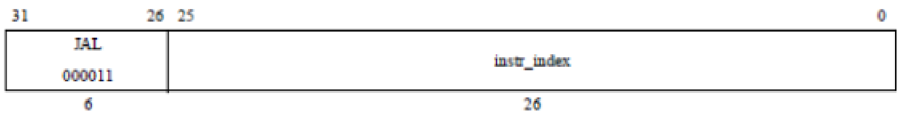


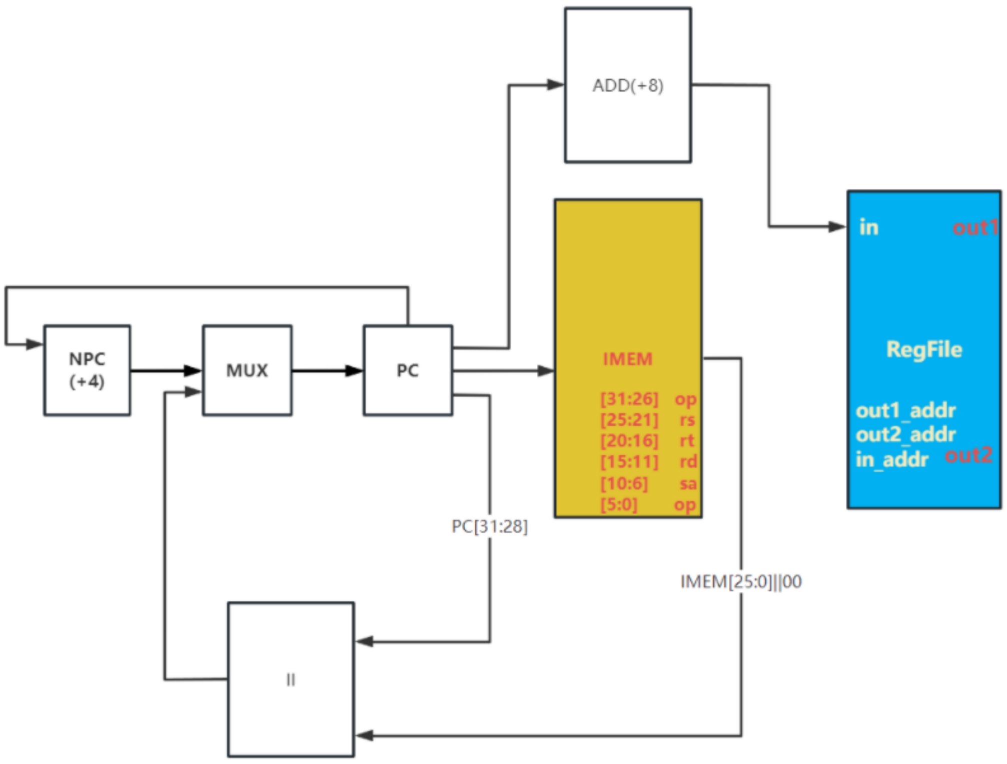
```
PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
PC[31:28] -> ||_A
IMEM[25,0] || 00 -> ||_B
```

```
||_OUT -> MUX
MUX_OUT -> PC
```

通路类型 13

31.jal

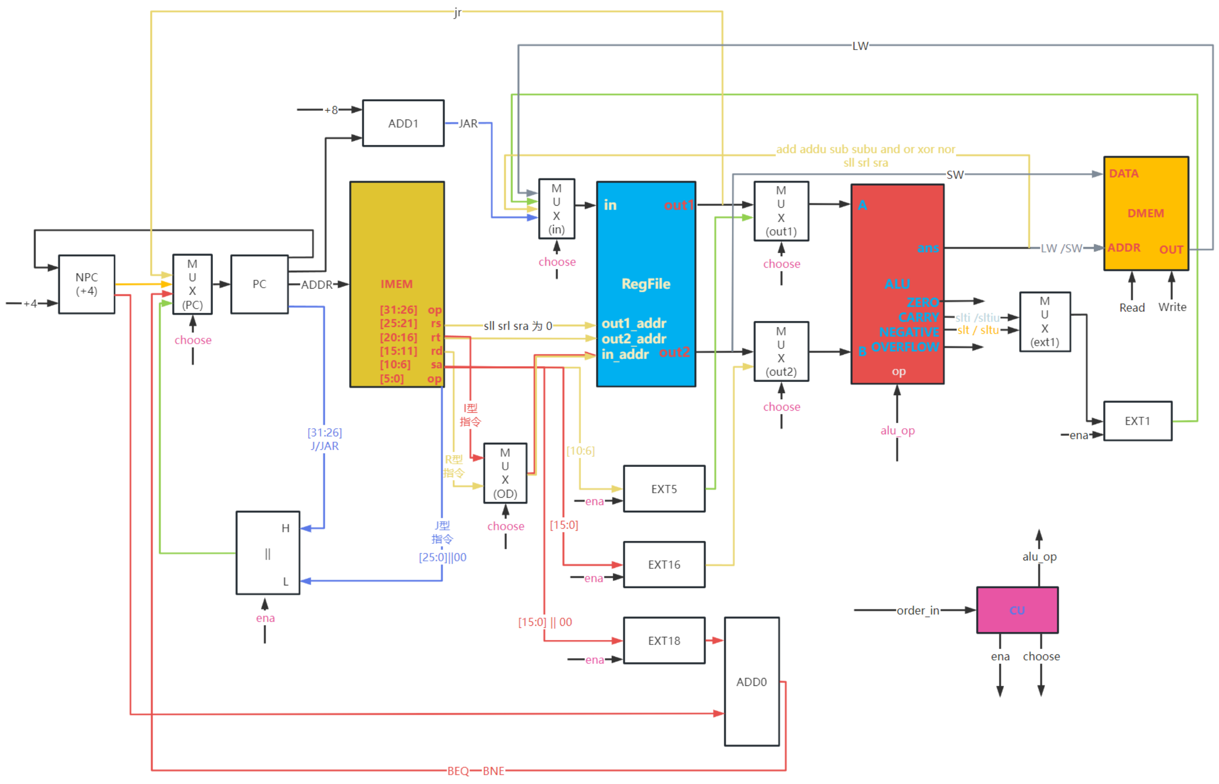


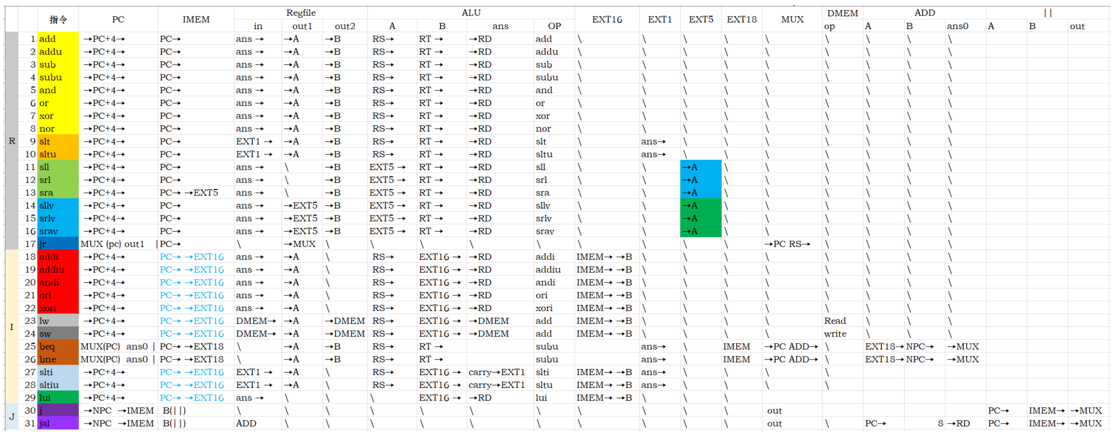


```
PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
8 -> ADD_A
PC -> ADD_B
(ADD_A + ADD_B -> ADD_OUT)
```

```
ADD_OUT -> Rd
PC[31:28] -> ||_A
IMEM[25,0] || 02 -> ||_B
||_OUT -> MUX
MUX_OUT -> PC
```

数据通路总图





		指令	PC	IMEM	Regfile			ALU				EXT16	EXT1	EXT5	EXT18	MUX	DMEM	ADD			||		
					in	out1	out2	A	B	ans	OP						op	A	B	ans0	A	B	out
R	1	add	→PC+4→	PC→	ans →	→A	→B	RS→	RT →	→RD	add	\	\	\	\	\	\	\	\	\			
	2	addu	→PC+4→	PC→	ans →	→A	→B	RS→	RT →	→RD	addu	\	\	\	\	\	\	\	\	\			
	3	sub	→PC+4→	PC→	ans →	→A	→B	RS→	RT →	→RD	sub	\	\	\	\	\	\	\	\	\			
	4	subu	→PC+4→	PC→	ans →	→A	→B	RS→	RT →	→RD	subu	\	\	\	\	\	\	\	\	\			
	5	and	→PC+4→	PC→	ans →	→A	→B	RS→	RT →	→RD	and	\	\	\	\	\	\	\	\	\			
	6	or	→PC+4→	PC→	ans →	→A	→B	RS→	RT →	→RD	or	\	\	\	\	\	\	\	\	\			
	7	xor	→PC+4→	PC→	ans →	→A	→B	RS→	RT →	→RD	xor	\	\	\	\	\	\	\	\	\			
	8	nor	→PC+4→	PC→	ans →	→A	→B	RS→	RT →	→RD	nor	\	\	\	\	\	\	\	\	\			
	9	slt	→PC+4→	PC→	EXT1 →	→A	→B	RS→	RT →	→RD	slt	\	ans→	\	\	\	\	\	\	\			
	10	sltu	→PC+4→	PC→	EXT1 →	→A	→B	RS→	RT →	→RD	sltu	\	ans→	\	\	\	\	\	\	\			
	11	sll	→PC+4→	PC→	ans →	\	→B	EXT5 →	RT →	→RD	sll	\	\	→A	\	\	\	\	\	\			
	12	srl	→PC+4→	PC→	ans →	\	→B	EXT5 →	RT →	→RD	srl	\	\	→A	\	\	\	\	\	\			
	13	sra	→PC+4→	PC→ →EXT5	ans →	\	→B	EXT5 →	RT →	→RD	sra	\	\	→A	\	\	\	\	\	\			
	14	sllv	→PC+4→	PC→	ans →	→EXT5	→B	EXT5 →	RT →	→RD	sllv	\	\	→A	\	\	\	\	\	\			
	15	srlv	→PC+4→	PC→	ans →	→EXT5	→B	EXT5 →	RT →	→RD	srlv	\	\	→A	\	\	\	\	\	\			
	16	srav	→PC+4→	PC→	ans →	→EXT5	→B	EXT5 →	RT →	→RD	srav	\	\	→A	\	\	\	\	\	\			
	17	jr	MUX (pc) out1 |	PC→	\	→MUX	\	\	\	\	\	\	\	\	\	→PC RS→	\	\	\	\			
I	18	addi	→PC+4→	PC→ →EXT16	ans →	→A	\	RS→	EXT16 →	→RD	addi	IMEM→ →B	\	\	\	\	\	\	\	\			
	19	addiu	→PC+4→	PC→ →EXT16	ans →	→A	\	RS→	EXT16 →	→RD	addiu	IMEM→ →B	\	\	\	\	\	\	\	\			
	20	andi	→PC+4→	PC→ →EXT16	ans →	→A	\	RS→	EXT16 →	→RD	andi	IMEM→ →B	\	\	\	\	\	\	\	\			
	21	ori	→PC+4→	PC→ →EXT16	ans →	→A	\	RS→	EXT16 →	→RD	ori	IMEM→ →B	\	\	\	\	\	\	\	\			
	22	xori	→PC+4→	PC→ →EXT16	ans →	→A	\	RS→	EXT16 →	→RD	xori	IMEM→ →B	\	\	\	\	\	\	\	\			
	23	lw	→PC+4→	PC→ →EXT16	DMEM→	→A	→DMEM	RS→	EXT16 →	→DMEM	add	IMEM→ →B	\	\	\	\	Read	\	\	\			
	24	sw	→PC+4→	PC→ →EXT16	DMEM→	→A	→DMEM	RS→	EXT16 →	→DMEM	add	IMEM→ →B	\	\	\	\	write	\	\	\			
	25	beq	MUX(PC) ans0 |	PC→ →EXT18	\	→A	→B	RS→	RT →		subu		ans→	\	IMEM	→PC ADD→	\	EXT18→	NPC→	→MUX			
	26	bne	MUX(PC) ans0 |	PC→ →EXT18	\	→A	→B	RS→	RT →		subu		ans→	\	IMEM	→PC ADD→	\	EXT18→	NPC→	→MUX			
	27	slti	→PC+4→	PC→ →EXT16	EXT1 →	→A	\	RS→	EXT16 →	carry→EXT1	slti	IMEM→ →B	ans→	\	\	\	\						
	28	sltiu	→PC+4→	PC→ →EXT16	EXT1 →	→A	\	RS→	EXT16 →	carry→EXT1	sltu	IMEM→ →B	ans→	\	\	\	\						
	29	lui	→PC+4→	PC→ →EXT16	ans →	\	\	\	EXT16 →	→RD	lui	IMEM→ →B	\	\	\								
J	30	j	→NPC →IMEM	B(||)	\	\	\	\	\	\	\	\	\	\	\	out					PC→	IMEM→	→MUX
	31	jal	→NPC →IMEM	B(||)	ADD	\	\	\	\	\	\	\	\	\	\	out	\	PC→	8	→RD	PC→	IMEM→	→MUX

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 sccomp_dataflow.v 模块

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns/1ps
module sccomp_dataflow(
    input clk_in,   //别改,提交就是这个规则
    input reset,
    output [31:0] inst,     //测试用,检测是什么指令
    output [31:0] pc        // 测试用,检测 pc 有没有正常前进


    /******数码管显示接口*******/
    // input cs,
    // output [7:0] o_seg,
    // output [7:0] o_sel

);
    wire dmem_ena,dmem_read,dmem_write;
    wire [31:0]dmem_addr,dmem_addr_temp,dmem_read_data,dmem_write_data;
    assign dmem_addr = (dmem_addr_temp - 32'h1001_0000) /4;
    wire [31:0]imem_addr;
/*****************下板时使用慢时钟********************/
    wire slow_clk;
    DIVIDER #(.div(1))
divide_clk(.clk(clk_in),.rst(reset),.new_clk(slow_clk));

    /* DMEM */
    DMEM dmem(
        .clk(slow_clk),
        .ena(1'b1),
        .write(dmem_write),
        .read(dmem_read),

        .addr(dmem_addr[10:0]),
        .in(dmem_write_data),
        .out(dmem_read_data)
    );
    /* CPU */
    CPU31 sccpu(
        .clk(slow_clk), .ena(1'b1),.rst(reset),

        .imem_out(inst),
        .imem_addr(imem_addr),

        //.alu_ans() 不接
        .dmem_addr(dmem_addr_temp),
        .dmem_out(dmem_read_data),
        .dmem_in(dmem_write_data),

        .dmem_write(dmem_write),
        .dmem_read(dmem_read)
    );
    /* IMEM */
    assign pc = imem_addr;

    IMEM imem(
        .in(imem_addr[12:2]),
        .out(inst)
    );


    /*********数码管接口**********/
    // seg7x16 show_pc (
    //      .clk(slow_clk),
    //      .reset(reset),
    //      .cs(cs),
    //      .i_data(pc),
    //      .o_seg(o_seg),
    //      .o_sel(o_sel)
    // );

endmodule
```

寄存器堆 REGFILES.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns/1ps
module REGFILES(
    input ena,
    input rst,
    input clk,
    input [4:0]in_addr,
    input [4:0]out1_addr,
    input [4:0]out2_addr,

    input write,

    input [31:0] in,
    output [31:0] out1,
    output [31:0] out2
```

```verilog
);
reg [31:0] array_reg[31:0];
always @(posedge clk or posedge rst)
begin
    if(ena)
    begin
        if(rst)
        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
            if(write && in_addr!=0)
            array_reg[in_addr] <= in;
    end
end
assign out1 = ena? array_reg[out1_addr] : 32'bz;
assign out2 = ena? array_reg[out2_addr] : 32'bz;
```

```
endmodule
```

PC.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns/1ps
module PC(
    input clk,
    input rst,
    input ena,
    input [31:0] in,
    output [31:0] out
);

reg [31:0] register;
always @(negedge clk) begin
    if (ena)
    begin
        if(rst)
            register <= 32'h0040_0000;
        else
            register <= in;
    end
end
assign out  = rst ? 32'h0040_0000:(ena ? register :32'bz);
endmodule
```

MUX.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns / 1ps
module MUX #(parameter WIDTH = 32)
(
    input [WIDTH-1:0] in0,
    input [WIDTH-1:0] in1,
    input [WIDTH-1:0] in2,
    input [WIDTH-1:0] in3,
    input [1:0] choose,
    output reg[WIDTH-1:0] out
);

    always@(*)
    begin
        case(choose)
        2'b00:out <=in0;
        2'b01:out <=in1;
        2'b10:out <=in2;
        2'b11:out <=in3;
        endcase
    end
endmodule
```

IMEM.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns/1ps
module IMEM(
    input [10:0] in,
    output [31:0] out

);
   /*ip 核*/
dist_mem_gen_0 imem(
    .a(in),
    .spo(out)
);
endmodule
```

II.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns / 1ps
module II(
    input [3:0]H,
    input [25:0]L,
    input ena,
    output [31:0] out_d
    );
assign out_d = ena? {{H},{L},2'b00}:32'bz;
endmodule
```

EXT.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns/1ps
module EXT #(parameter WIDTH = 16)(
input [WIDTH-1:0] in,
input sign,
input ena,
output [31:0] out
);
assign out=ena?(sign? {{(32-WIDTH){in[WIDTH-1]}},in} : {{(32-
WIDTH){1'b0}},in}) : 32'bz;
endmodule
```

DMEM.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns / 1ps
module DMEM(
    input clk,
    input ena,
    input write,
    input read,

    input [10:0] addr,
    input [31:0] in,
    output [31:0] out
);

reg [31:0] memory [0:31];
always@(posedge clk)
begin
    if(ena)begin
       if(write)
        memory[addr] <= in;
    end
end
assign out  = (read && ena)? memory[addr] : 32'bz;
endmodule
```

DIVIDER.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns/1ps
module DIVIDER
#(parameter div = 32'd20)//分频为原来频率的 1/n
(
    input clk,//要分频的时钟
    input rst,//高电平复位信号
    output reg new_clk//分频后的时钟
);
    localparam T = div / 2;
    reg [31:0] cnt = 0;
    initial
    begin
        new_clk <= 0;
    end
    always @(posedge clk)
    begin
        if(rst == 1)
        begin
            cnt <= 0;
            new_clk <= 0;
        end
        else
        begin
            if(cnt == T)
            begin
                cnt <= 1;
                new_clk <= ~new_clk;
            end
            else
                cnt <= cnt + 1;
        end
    end
endmodule
```

DECODER.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns / 1ps
module DECODER(
    input  [31:0] order,    //需要译码的指令，也就是当前要执行的指令

    output [31:0] which_order,//哪一个指令
    output [4:0]  Rs,          //Rs 对应的寄存器的地址
    output [4:0]  Rt,          //Rt 对应的寄存器的地址
    output [4:0]  Rd,          //Rd 对应的寄存器的地址
    output [4:0]  Sa,        //位移偏移量（SLL，SRL，SRA 用）
    output [15:0] i_immediate,    //立即数（I 型指令用）
    output [25:0] j_immediate       //跳转地址（J 型指令用）
    );

/* R 型指令 */
parameter ADD    = 6'b100000;
parameter ADDU   = 6'b100001;
parameter SUB    = 6'b100010;
parameter SUBU   = 6'b100011;
parameter AND    = 6'b100100;
parameter OR     = 6'b100101;
parameter XOR    = 6'b100110;
parameter NOR    = 6'b100111;
parameter SLT    = 6'b101010;
parameter SLTU   = 6'b101011;
parameter SLL    = 6'b000000;
parameter SRL    = 6'b000010;
parameter SRA    = 6'b000011;
parameter SLLV   = 6'b000100;
parameter SRLV   = 6'b000110;
parameter SRAV   = 6'b000111;
parameter JR     = 6'b001000;
/* I 型指令 */
parameter ADDI   = 6'b001000;
parameter ADDIU  = 6'b001001;
parameter ANDI   = 6'b001100;
parameter ORI    = 6'b001101;
parameter XORI   = 6'b001110;
parameter LW     = 6'b100011;
parameter SW     = 6'b101011;
parameter BEQ    = 6'b000100;
parameter BNE    = 6'b000101;
parameter SLTI   = 6'b001010;
parameter SLTIU  = 6'b001011;
parameter LUI    = 6'b001111;
/* J 型指令*/
parameter J      = 6'b000010;
parameter JAL    = 6'b000011;
/************************************R*********************************
*************/
wire add_  = ((order[31:26] == 6'h0) && (order[5:0] == ADD  )) ? 1'b1 :
1'b0;
wire addu_ = ((order[31:26] == 6'h0) && (order[5:0] == ADDU )) ? 1'b1 :
```

```
1'b0;
wire sub_  = ((order[31:26] == 6'h0) && (order[5:0] == SUB  )) ? 1'b1 :
1'b0;
wire subu_ = ((order[31:26] == 6'h0) && (order[5:0] == SUBU )) ? 1'b1 :
1'b0;
wire and_  = ((order[31:26] == 6'h0) && (order[5:0] == AND  )) ? 1'b1 :
1'b0;
wire or_   = ((order[31:26] == 6'h0) && (order[5:0] == OR   )) ? 1'b1 :
1'b0;
wire xor_  = ((order[31:26] == 6'h0) && (order[5:0] == XOR  )) ? 1'b1 :
1'b0;
wire nor_  = ((order[31:26] == 6'h0) && (order[5:0] == NOR  )) ? 1'b1 :
1'b0;

wire slt_  = ((order[31:26] == 6'h0) && (order[5:0] == SLT  )) ? 1'b1 :
1'b0;
wire sltu_ = ((order[31:26] == 6'h0) && (order[5:0] == SLTU )) ? 1'b1 :
1'b0;

wire sll_  = ((order[31:26] == 6'h0) && (order[5:0] == SLL  )) ? 1'b1 :
1'b0;
wire srl_  = ((order[31:26] == 6'h0) && (order[5:0] == SRL  )) ? 1'b1 :
1'b0;
wire sra_  = ((order[31:26] == 6'h0) && (order[5:0] == SRA  )) ? 1'b1 :
1'b0;

wire sllv_ = ((order[31:26] == 6'h0) && (order[5:0] == SLLV )) ? 1'b1 :
1'b0;
wire srlv_ = ((order[31:26] == 6'h0) && (order[5:0] == SRLV )) ? 1'b1 :
1'b0;
wire srav_ = ((order[31:26] == 6'h0) && (order[5:0] == SRAV )) ? 1'b1 :
1'b0;

wire jr_   = ((order[31:26] == 6'h0) && (order[5:0] == JR   )) ? 1'b1 :
1'b0;
/*********************************************************************
****************/
wire addi_  = (order[31:26] == ADDI  ) ? 1'b1 : 1'b0;
wire addiu_ = (order[31:26] == ADDIU ) ? 1'b1 : 1'b0;
wire andi_  = (order[31:26] == ANDI  ) ? 1'b1 : 1'b0;
wire ori_   = (order[31:26] == ORI   ) ? 1'b1 : 1'b0;
wire xori_  = (order[31:26] == XORI  ) ? 1'b1 : 1'b0;

wire lw_    = (order[31:26] == LW    ) ? 1'b1 : 1'b0;

wire sw_    = (order[31:26] == SW    ) ? 1'b1 : 1'b0;

wire beq_   = (order[31:26] == BEQ   ) ? 1'b1 : 1'b0;
```

```verilog
wire bne_   = (order[31:26] == BNE   ) ? 1'b1 : 1'b0;

wire slti_  = (order[31:26] == SLTI  ) ? 1'b1 : 1'b0;
wire sltiu_ = (order[31:26] == SLTIU ) ? 1'b1 : 1'b0;

wire lui_   = (order[31:26] == LUI   ) ? 1'b1 : 1'b0;
/*************************************J*******************************
*****************/
wire j_     = (order[31:26] == J     ) ? 1'b1 : 1'b0;

wire jal_   = (order[31:26] == JAL   ) ? 1'b1 : 1'b0;
/*********************************************************************
*****************/
assign   which_order [1]  = add_ ;         //ADD
assign   which_order [2]  = addu_ ;         //ADDU
assign   which_order [3]  = sub_ ;         //SUB
assign   which_order [4]  = subu_;         //SUBU
assign   which_order [5]  = and_ ;         //AND
assign   which_order [6]  = or_  ;         //OR
assign   which_order [7]  = xor_ ;         //XOR
assign   which_order [8]  = nor_ ;         //NOR
assign   which_order [9]  = slt_ ;         //SLT
assign   which_order [10] = sltu_ ;        //SLTU
assign   which_order [11] = sll_  ;        //SLL
assign   which_order [12] = srl_  ;        //SRL
assign   which_order [13] = sra_  ;        //SRA
assign   which_order [14] = sllv_ ;        //SLLV
assign   which_order [15] = srlv_ ;        //SRLV
assign   which_order [16] = srav_ ;        //SRAV
assign   which_order [17] = jr_   ;        //JR
assign   which_order [18] = addi_ ;        //ADDI
assign   which_order [19] = addiu_;        //ADDIU
assign   which_order [20] = andi_ ;        //ANDI
assign   which_order [21] = ori_  ;        //ORI
assign   which_order [22] = xori_ ;        //XORI
assign   which_order [23] = lw_   ;        //LW
assign   which_order [24] = sw_   ;        //SW
assign   which_order [25] = beq_  ;        //BEQ
assign   which_order [26] = bne_  ;        //BNE
assign   which_order [27] = slti_ ;        //SLTI
assign   which_order [28] = sltiu_;        //SLTIU
assign   which_order [29] = lui_  ;        //LUI
assign   which_order [30] = j_    ;        //J
assign   which_order [31] = jal_  ;        //JAL
/* 取出指令中各部分的值 */
assign Rs  = (sll_|srl_|sra_|lui_|j_|jal_) ? 5'hz :order[25:21];

assign Rt  = (jr_|j_|jal_) ? 5'hz : order[20:16];
assign Rd  = 
(jr_|addi_|addiu_|andi_|ori_|xori_|lw_|sw_|beq_|bne_|slti_|sltiu_|lui_|
j_|jal_) ? 5'bz : order[15:11];

assign Sa = (sll_ | srl_ | sra_) ? order[10:6] : 5'hz;

assign i_immediate = (addi_ | addiu_ | andi_  | ori_ |
                      xori_ | lw_    | sw_    | beq_ |
                      bne_  | slti_  | sltiu_ | lui_) ? order[15:0] : 
16'hz;

assign j_immediate = (j_ | jal_) ? order[25:0] : 26'hz;

endmodule
```

CU.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns/1ps
module CU(
    input [31:0]order_flag,
    input zero,
    input overflow,

    output regfile_write,

    output dmem_write,
    output dmem_read,

    output [18:0] ext_ena,
    output ext16_sign,
    output II_ena,
    output [4:0] alu_op,

    output [1:0] choose_mux_pc,
    output [1:0] choose_mux_od,
    output [1:0] choose_mux_in,

    output choose_mux_out1,
    output choose_mux_out2,
    output choose_mux_ext1

);
wire   add_  =  order_flag [1] ;        //ADD
```

```verilog
wire   addu_ =  order_flag [2] ;        //ADDU
wire   sub_  =  order_flag [3] ;        //SUB
wire   subu_ =  order_flag [4] ;        //SUBU
wire   and_  =  order_flag [5] ;        //AND
wire   or_   =  order_flag [6] ;        //OR
wire   xor_  =  order_flag [7] ;        //XOR
wire   nor_  =  order_flag [8] ;        //NOR
wire   slt_  =  order_flag [9] ;        //SLT
wire   sltu_ =  order_flag [10];        //SLTU
wire   sll_  =  order_flag [11];        //SLL
wire   srl_  =  order_flag [12];        //SRL
wire   sra_  =  order_flag [13];        //SRA
wire   sllv_ =  order_flag [14];        //SLLV
wire   srlv_ =  order_flag [15];        //SRLV
wire   srav_ =  order_flag [16];        //SRAV
wire   jr_   =  order_flag [17];        //JR
wire   addi_ =  order_flag [18];        //ADDI
wire   addiu_=  order_flag [19];        //ADDIU
wire   andi_ =  order_flag [20];        //ANDI
wire   ori_  =  order_flag [21];        //ORI
wire   xori_ =  order_flag [22];        //XORI
wire   lw_   =  order_flag [23];        //LW
wire   sw_   =  order_flag [24];        //SW
wire   beq_  =  order_flag [25];        //BEQ
wire   bne_  =  order_flag [26];        //BNE
wire   slti_ =  order_flag [27];        //SLTI
wire   sltiu_=  order_flag [28];        //SLTIU
wire   lui_  =  order_flag [29];        //LUI
wire   j_    =  order_flag [30];        //J
wire   jal_  =  order_flag [31];        //JAL

assign choose_mux_out1 = (sll_|srl_|sra_) ? 1 : 0;
assign choose_mux_out2 = (addi_|addiu_ |
andi_|ori_|xori_|lw_|sw_|slti_|sltiu_|lui_)?1:0;

assign choose_mux_od[0]=
(addi_|addiu_|andi_|ori_|xori_|lw_|slti_|sltiu_|lui_)?0:1;
assign choose_mux_od[1]= jal_ ? 1 :0;

assign choose_mux_ext1 =(slt_ |sltu_)?1:0;

assign choose_mux_pc[1]= ((beq_ & zero )| (bne_ & ~zero)| j_ | jal_) ?
1 : 0;
assign choose_mux_pc[0] = (jr_ |(beq_&zero)|(bne_ & ~zero))?0:1;

assign choose_mux_in[1] = (slt_|sltu_|lw_|slti_|sltiu_)?0:1;
assign choose_mux_in[0] = (slt_|sltu_|slti_|sltiu_|jal_)?1:0;
assign regfile_write =(jr_|sw_|beq_|bne_|j_)?0:1;
assign dmem_write = sw_?1:0;
assign dmem_read = lw_?1:0;
assign ext_ena[1] = (slt_|sltu_|slti_|sltiu_)?1:0;
assign ext_ena[5] = (sll_|srl_|sra_)?1:0;
assign ext_ena[16] =
(addi_|addiu_|andi_|ori_|xori_|lw_|sw_|slti_|sltiu_|lui_)?1:0;
assign ext_ena[18] = (beq_|bne_)?1:0;
assign ext16_sign = (addi_|addiu_|slti_)? 1 : 0;
assign II_ena = (j_|jal_)?1:0;

assign alu_op[4] = (lui_ | j_|jal_|jr_)?1:0;
assign alu_op[3] = (slt_|sltu_|sll_|srl_|sra_|sllv_|srlv_|srav_)?1:0;
assign alu_op[2] =
(and_|andi_|or_|ori_|xor_|xori_|nor_|sra_|sllv_|srlv_|srav_)?1:0;
assign alu_op[1] =
(sub_|slti_|subu_|beq_|bne_|sltiu_|xor_|xori_|nor_|sll_|srl_|srlv_|srav
_)?1:0;
assign alu_op[0] =
(addu_|subu_|beq_|bne_|sltiu_|or_|ori_|nor_|sltu_|srl_|sllv_|srav_|j_|j
al_|jr_)?1:0;

endmodule
```

CPU31.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns / 1ps
module CPU31(
    input clk,
    input ena,
    input rst,
    input [31:0] imem_out,
    output [31:0] imem_addr,

    output [31:0] alu_ans,

    output [31:0] dmem_addr,
    input [31:0] dmem_out,
    output [31:0] dmem_in,

    output dmem_read,
    output dmem_write
    );
/*寄存器堆 regfile*/
wire regfile_write;
wire[31:0] regfile_in,regfile_out1,regfile_out2;
```

```
assign dmem_in = regfile_out2;
wire[4:0]out1_addr,out2_addr,in_addr;
/*各种扩展器*/
wire [31:0] ext1_out,ext5_out,ext16_out,ext18_out;
wire ext16_sign,ext1_in;
wire [18:0]ext_ena;
/*单独的加法器,拼接器*/
wire [31:0] add0_out , add1_out,II_out;
wire II_ena;
/*各种选择器的控制端*/
wire [1:0]
mux_pc_choose,mux_od_choose,mux_out1_choose,mux_out2_choose,mux_in_choo
se,mux_ext1_choose;
/*PC 与 NPC*/
wire [31:0] pc_in,pc_out,npc_in,npc_out;
assign npc_in = pc_out;
assign imem_addr  = pc_out ; //pc 的接口接到 imem 上
/*ALU 运算器*/
wire [31:0] alu_a,alu_b;
wire [4:0] alu_op;
wire zero,carry,negative,overflow;//wire[31:0] alu_ans 在输出端
assign dmem_addr = alu_ans; //也接到 dmem_addr 上
/*IMEM 这个不属于 CPU,但是指令是必要的*/
wire [4:0]rs ;
wire [4:0]rt ;
wire [4:0]rd ;
wire [4:0]sa ;
/*i 指令和 j 指令*/
wire [15:0]i_immediate ;
wire [25:0]j_immediate ;
/*指令信号,单个 1 位信号对应单个条指令 flag order_flag[1] 对应 ADD(第一条指
令)*/
wire [31:0] order_flag;
//DECODER
DECODER decoder(
    .order(imem_out),
    .which_order(order_flag),
    .Rs(rs) , .Rt(rt), .Rd(rd),.Sa(sa),
    .i_immediate(i_immediate),
    .j_immediate(j_immediate)
);
//PC
PC pc(
    .clk(clk),.ena(ena),.rst(rst),
    .in(pc_in),
    .out(pc_out)
);
//NPC 这里不能用 PC 不然会第二次才能写入
```

```verilog
ADD npc(
    .a(npc_in),
    .b(4),
    .c(npc_out)
);
//寄存器堆
assign out1_addr = rs;
assign out2_addr = rt;
REGFILES cpu_ref(
    .clk(clk), .rst(rst) , .ena(ena),
    .in_addr(in_addr),.out1_addr(out1_addr),.out2_addr(out2_addr),
    .write(regfile_write),
    .in(regfile_in),.out1(regfile_out1),.out2(regfile_out2)
);
//ALU
ALU alu(
    .a(alu_a), .b(alu_b),
    .op(alu_op),
    .ans(alu_ans),
    .zero(zero),.carry(carry),
    .negative(negative),.overflow(overflow)
);
//add0
ADD add0(
    .a(ext18_out), .b(npc_out),
    .c(add0_out)
);
//add1
ADD add1(
    .a(4), .b(pc_out),
    .c(add1_out)
);
II ii(
    .H(pc_out[31:28]),
    .L(j_immediate[25:0]),
    .ena(II_ena),
    .out_d(II_out)
);
//mux_pc
MUX mux_pc(
    .in0(regfile_out1),
    .in1(npc_out),
    .in2(add0_out),
    .in3(II_out),
    .choose(mux_pc_choose),
    .out(pc_in)
);
//mux_od
```

```
MUX #(.WIDTH(5))mux_od(
    .in0(rt),.in1(rd),
    .in3(5'd31),
    .choose(mux_od_choose),
    .out(in_addr)
);
//mux_in
assign mux_out1_choose[1] = 0;//数据选择器写的 4 路的,
assign mux_out2_choose[1] = 0;//这里只用两路,就把高位写死为 0 了
assign mux_ext1_choose[1] = 0;
MUX mux_in(
    .in0(dmem_out),.in1(ext1_out),
    .in2(alu_ans),.in3(add1_out),
    .choose(mux_in_choose),
    .out(regfile_in)
);
//mux_out1
MUX mux_out1(
    .in0(regfile_out1),
    .in1(ext5_out),
    .choose(mux_out1_choose),
    .out(alu_a)
);
//mux_out2
MUX mux_out2(
    .in0(regfile_out2),
    .in1(ext16_out),
    .choose(mux_out2_choose),
    .out(alu_b)
);
//mux_ext1
MUX #(.WIDTH(1))mux_ext1(
    .in0(carry),
    .in1(negative),
    .choose(mux_ext1_choose),
    .out(ext1_in)
);
//ext1
EXT #(.WIDTH (1))
ext1(.in(ext1_in),.ena(ext_ena[1]),.sign(1'b0),.out(ext1_out));
//ext5
EXT #(.WIDTH (5))
ext5(.in(sa),.ena(ext_ena[5]),.sign(1'b0),.out(ext5_out));
//ext16
EXT #(.WIDTH
(16))ext16(.in(i_immediate),.ena(ext_ena[16]),.sign(ext16_sign),.out(ex
t16_out));
//ext18
```

```
EXT #(.WIDTH
(18))ext18(.in({{i_immediate},{2'b00}}),.ena(ext_ena[18]),.sign(1'b1),.
out(ext18_out));
// CU
CU controller(
    .order_flag(order_flag),
    .zero(zero),
    .overflow(overflow),

    .regfile_write(regfile_write),

    .dmem_write(dmem_write),
    .dmem_read(dmem_read),

    .ext_ena(ext_ena),
    .ext16_sign(ext16_sign),
    .II_ena(II_ena),
    .alu_op(alu_op),
    .choose_mux_pc(mux_pc_choose),
    .choose_mux_od(mux_od_choose),
    .choose_mux_in(mux_in_choose),

    .choose_mux_out1(mux_out1_choose[0]),
    .choose_mux_out2(mux_out2_choose[0]),
    .choose_mux_ext1(mux_ext1_choose[0])
);

endmodule
```

ALU.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns / 1ps
module ALU (
    input [31:0]a,       //32bit input opeater 1
    input [31:0]b,       //32bit input opeater 2
    input [4:0]op,       //6bit input control alu
    output [31:0]ans,    //32bit output,a,b,controled by aluc
    output zero,         // 0 flag
    output carry,        // carry flag
    output negative,     // symbol flag
    output overflow      //over flag
);
parameter ADD = 5'b00000 , ADDU = 5'b00001 , SUB = 5'b00010 , SUBU =
5'b00011;
```

```
parameter AND = 5'b00100 , OR = 5'b00101 , XOR = 5'b00110 ,NOR =
5'b00111; //1

parameter SLT = 5'b01000 , SLTU = 5'b01001 ; //2
parameter SLL = 5'b01010 , SRL = 5'b01011 , SRA =  5'b01100; //3
parameter SLLV = 5'b01101 , SRLV = 5'b1110 , SRAV = 5'b01111 ; //4

parameter LUI = 5'b10000;
parameter default_op =5'b10001 ;
/*  用于做有符号的计算  */
wire signed [31:0] signed_a , signed_b ;
reg [32:0] R;
assign signed_a = a;
assign signed_b = b;

always@(*)
begin
    case (op)
        ADD:    R <= signed_a + signed_b;
        ADDU:   R <= a + b;
        SUB:    R <= signed_a - signed_b;
        SUBU:   R <= a - b;
        AND:    R <= a & b;
        OR:     R <= a | b;
        XOR:    R <= a ^ b;
        NOR:    R <= ~(a | b);

        SLT:    R <= (signed_a - signed_b);
        SLTU:   R <= (a - b);

        SLL:    R <= b << a;
        SRL:    R <= b >> a;
        SRA:    R <= signed_b >>> signed_a;

        SLLV:   R <= b << a[4:0];
        SRLV:   R <= b >> a[4:0];
        SRAV:   R <= signed_b >>> signed_a[4:0];

        LUI:    R <= {b[15:0], 16'b0};
        default_op: R<=32'bz;
        default: R<=32'bz;
    endcase
end
assign ans = R[31:0];
assign zero = (R == 32'b0) ? 1'b1 : 1'b0;
assign carry = R[32];
assign overflow = R[32];
assign negative = (op == SLT ? (signed_a < signed_b) : ((op == SLTU) ?
(a < b) : 1'b0));
endmodule
```

ADD.v

用 Verilog 代码进行描述数据流型描述如下：

```
`timescale 1ns / 1ps
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

Tb_cpu 顶层测试模块

检测设计是否符合逻辑，进行 TestBeach 的测试， TestBeach 代码如下：

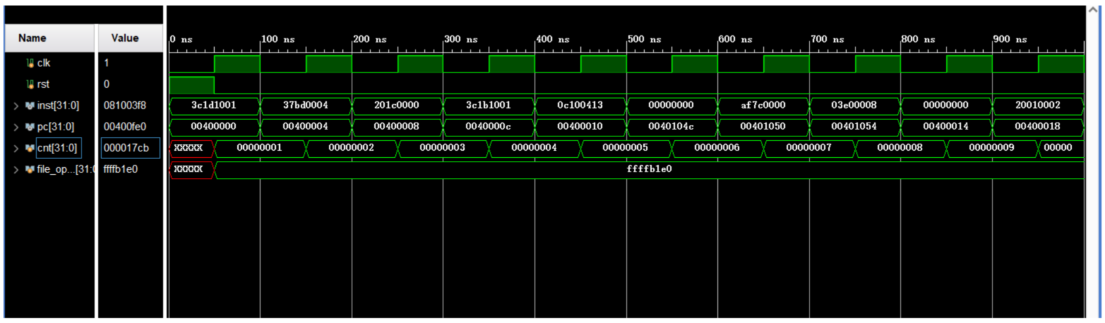
```
`timescale 1ns / 1ps

module cpu_tb;
reg clk;              //时钟信号
reg rst;              //复位信号
wire [31:0] inst;     //要执行的指令
wire [31:0] pc;       //下一条指令的地址
reg  [31:0] cnt;      //计数器，已经执行了几条指令?
// integer file_open;

parameter T = 200;
initial
begin
    clk = 1'b0;
    rst = 1'b1;
    #(T) rst = 1'b0;
    cnt = 0;
end

always  begin
```

```
#(T);
clk = ~clk;
end

/* always @ (posedge clk)
begin
    cnt <= cnt + 1'b1;
    file_open =
$fopen("D:\\verilog_projects\\test040_mips31cpu\\output.txt","a+");
    $fdisplay(file_open, "OP: %d", cnt);
    $fdisplay(file_open, "Instr_addr = %h", sc_inst.inst);
    $fdisplay(file_open, "$zero reg00= %h",
sc_inst.sccpu.cpu_ref.array_reg[0]);
    $fdisplay(file_open, "$at   reg01= %h",
sc_inst.sccpu.cpu_ref.array_reg[1]);
    $fdisplay(file_open, "$v0   reg02= %h",
sc_inst.sccpu.cpu_ref.array_reg[2]);
    $fdisplay(file_open, "$v1   reg03= %h",
sc_inst.sccpu.cpu_ref.array_reg[3]);
    $fdisplay(file_open, "$a0   reg04= %h",
sc_inst.sccpu.cpu_ref.array_reg[4]);
    $fdisplay(file_open, "$a1   reg05= %h",
sc_inst.sccpu.cpu_ref.array_reg[5]);
    $fdisplay(file_open, "$a2   reg06= %h",
sc_inst.sccpu.cpu_ref.array_reg[6]);
    $fdisplay(file_open, "$a3   reg07= %h",
sc_inst.sccpu.cpu_ref.array_reg[7]);
    $fdisplay(file_open, "$t0   reg08= %h",
sc_inst.sccpu.cpu_ref.array_reg[8]);
    $fdisplay(file_open, "$t1   reg09= %h",
sc_inst.sccpu.cpu_ref.array_reg[9]);
    $fdisplay(file_open, "$t2   reg10= %h",
sc_inst.sccpu.cpu_ref.array_reg[10]);
    $fdisplay(file_open, "$t3   reg11= %h",
sc_inst.sccpu.cpu_ref.array_reg[11]);
    $fdisplay(file_open, "$t4   reg12= %h",
sc_inst.sccpu.cpu_ref.array_reg[12]);
    $fdisplay(file_open, "$t5   reg13= %h",
sc_inst.sccpu.cpu_ref.array_reg[13]);
    $fdisplay(file_open, "$t6   reg14= %h",
sc_inst.sccpu.cpu_ref.array_reg[14]);
    $fdisplay(file_open, "$t7   reg15= %h",
sc_inst.sccpu.cpu_ref.array_reg[15]);
    $fdisplay(file_open, "$s0   reg16= %h",
sc_inst.sccpu.cpu_ref.array_reg[16]);
    $fdisplay(file_open, "$s1   reg17= %h",
sc_inst.sccpu.cpu_ref.array_reg[17]);
    $fdisplay(file_open, "$s2   reg18= %h",
sc_inst.sccpu.cpu_ref.array_reg[18]);
    $fdisplay(file_open, "$s3   reg19= %h",
sc_inst.sccpu.cpu_ref.array_reg[19]);
    $fdisplay(file_open, "$s4   reg20= %h",
sc_inst.sccpu.cpu_ref.array_reg[20]);
    $fdisplay(file_open, "$s5   reg21= %h",
sc_inst.sccpu.cpu_ref.array_reg[21]);
    $fdisplay(file_open, "$s6   reg22= %h",
sc_inst.sccpu.cpu_ref.array_reg[22]);
    $fdisplay(file_open, "$s7   reg23= %h",
sc_inst.sccpu.cpu_ref.array_reg[23]);
    $fdisplay(file_open, "$t8   reg24= %h",
sc_inst.sccpu.cpu_ref.array_reg[24]);
    $fdisplay(file_open, "$t9   reg25= %h",
sc_inst.sccpu.cpu_ref.array_reg[25]);
    $fdisplay(file_open, "$k0   reg26= %h",
sc_inst.sccpu.cpu_ref.array_reg[26]);
    $fdisplay(file_open, "$k1   reg27= %h",
sc_inst.sccpu.cpu_ref.array_reg[27]);
    $fdisplay(file_open, "$gp   reg28= %h",
sc_inst.sccpu.cpu_ref.array_reg[28]);
    $fdisplay(file_open, "$sp   reg29= %h",
sc_inst.sccpu.cpu_ref.array_reg[29]);
    $fdisplay(file_open, "$fp   reg30= %h",
sc_inst.sccpu.cpu_ref.array_reg[30]);
    $fdisplay(file_open, "$ra   reg31= %h",
sc_inst.sccpu.cpu_ref.array_reg[31]);
    /**************************test_dmem*********************/
   /*$fdisplay(file_open, "dmem_addr = %h", sc_inst.dmem.dm_addr);
   $fdisplay(file_open, "dm_data_in = %h", sc_inst.dmem.dm_data_in);
   $fdisplay(file_open, "dm_data_out = %h",
sc_inst.dmem.dm_data_out);
   $fdisplay(file_open, "$dmem0 = %h", sc_inst.dmem.dmem[0]);
   $fdisplay(file_open, "$dmem1 = %h", sc_inst.dmem.dmem[1]);
   $fdisplay(file_open, "$dmem2 = %h", sc_inst.dmem.dmem[2]);
   $fdisplay(file_open, "$dmem3 = %h", sc_inst.dmem.dmem[3]);
   $fdisplay(file_open, "$dmem4 = %h", sc_inst.dmem.dmem[4]);
   $fdisplay(file_open, "$dmem5 = %h", sc_inst.dmem.dmem[5]);
   $fdisplay(file_open, "$dmem6 = %h", sc_inst.dmem.dmem[6]);
   $fdisplay(file_open, "$dmem7 = %h", sc_inst.dmem.dmem[7]);
   $fdisplay(file_open, "$dmem8 = %h", sc_inst.dmem.dmem[8]);
   $fdisplay(file_open, "$dmem9 = %h", sc_inst.dmem.dmem[9]);
   $fdisplay(file_open, "$dmem10 = %h", sc_inst.dmem.dmem[10]);
   $fdisplay(file_open, "$dmem11 = %h", sc_inst.dmem.dmem[11]);
   $fdisplay(file_open, "$dmem12 = %h", sc_inst.dmem.dmem[12]);
   $fdisplay(file_open, "$dmem13 = %h", sc_inst.dmem.dmem[13]);
   $fdisplay(file_open, "$dmem14 = %h",
sc_inst.dmem.dmem[14]);
```

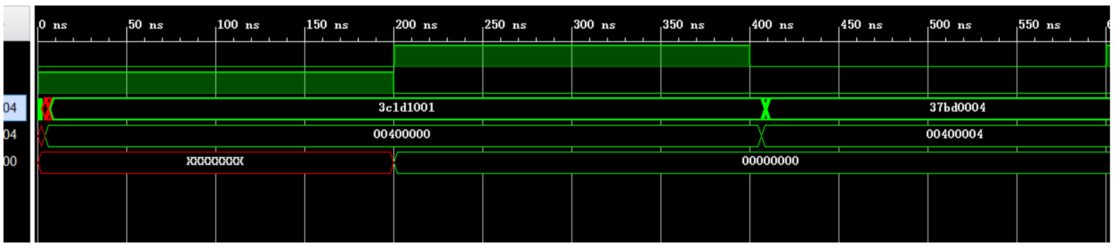
```
   $fdisplay(file_open, "$dmem15 = %h", sc_inst.dmem.dmem[15]);
   $fdisplay(file_open, "$dmem16 = %h", sc_inst.dmem.dmem[16]);
   $fdisplay(file_open, "$dmem17 = %h", sc_inst.dmem.dmem[17]);
   $fdisplay(file_open, "$dmem18 = %h", sc_inst.dmem.dmem[18]);
   $fdisplay(file_open, "$dmem19 = %h", sc_inst.dmem.dmem[19]);
   $fdisplay(file_open, "$dmem20 = %h", sc_inst.dmem.dmem[20]);
   $fdisplay(file_open, "$dmem21 = %h", sc_inst.dmem.dmem[21]);
   $fdisplay(file_open, "$dmem22 = %h", sc_inst.dmem.dmem[22]);
   $fdisplay(file_open, "$dmem23 = %h",
sc_inst.dmem.dmem[23]);

   $fdisplay(file_open, "$dmem24 = %h", sc_inst.dmem.dmem[24]);
   $fdisplay(file_open, "$dmem25 = %h", sc_inst.dmem.dmem[25]);
   $fdisplay(file_open, "$dmem26 = %h", sc_inst.dmem.dmem[26]);
   $fdisplay(file_open, "$dmem27 = %h", sc_inst.dmem.dmem[27]);
   $fdisplay(file_open, "$dmem28 = %h", sc_inst.dmem.dmem[28]);
   $fdisplay(file_open, "$dmem29 = %h", sc_inst.dmem.dmem[29]);
   $fdisplay(file_open, "$dmem30 = %h", sc_inst.dmem.dmem[30]);
   $fdisplay(file_open, "$dmem31 = %h", sc_inst.dmem.dmem[31]);

   $fdisplay(file_open, "$pc    = %h\n", sc_inst.pc);
   $fclose(file_open);
end
*/

sccomp_dataflow sc_inst(
    .clk_in(clk),
    .reset(rst),
    .inst(inst),
    .pc(pc)
);

endmodule
```

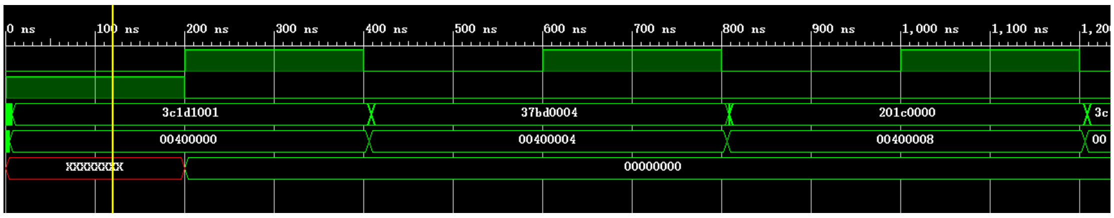
五、实验结果

前仿真结果

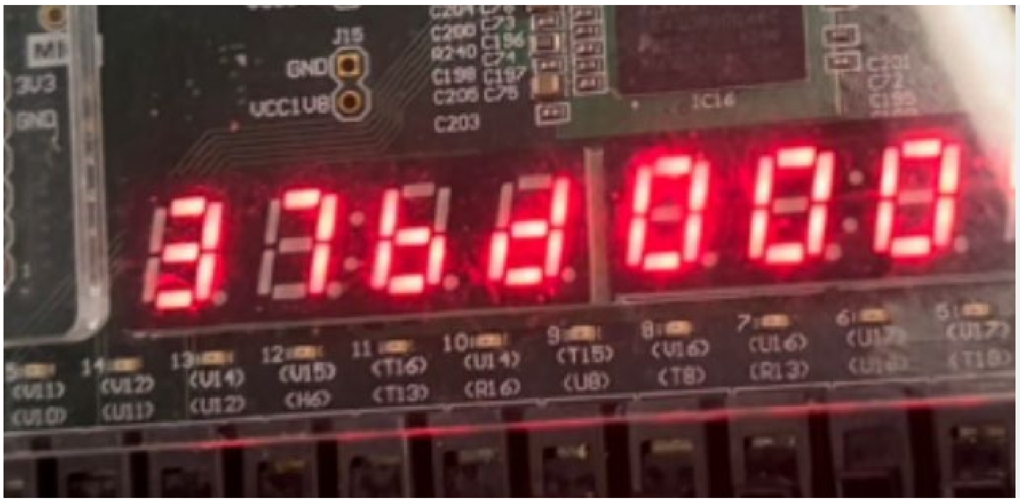

Name
Value
clk
1
rst
0
inst[31:0]
081003f8
pc[31:0]
00400fe0
cnt[31:0]
000017cb
file_op...[31:0]
ffffb1e0
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
3c1d1001 37bd0004 201c0000 3c1b1001 0c100413 00000000 af7c0000 03e00008 00000000 20010002
00400000 00400004 00400008 0040000c 00400010 0040104c 00401050 00401054 00400014 00400018
XXXXX 00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008 00000009 00000
XXXXX ffffb1e0

后仿真结果

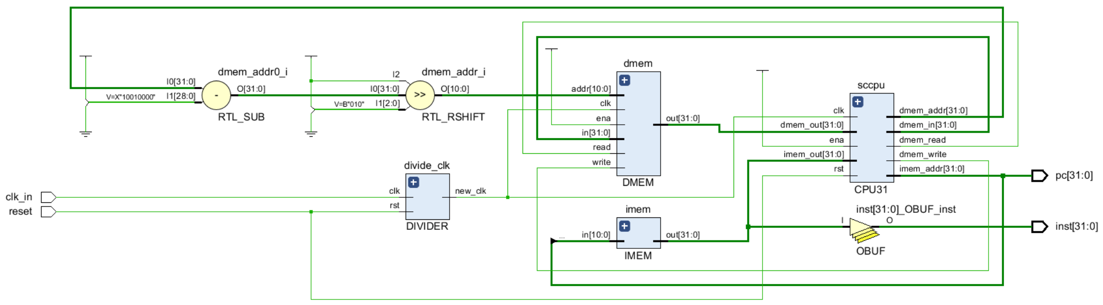




分频 1/1000000 下板结果(显示 PC)



RTL 图预览

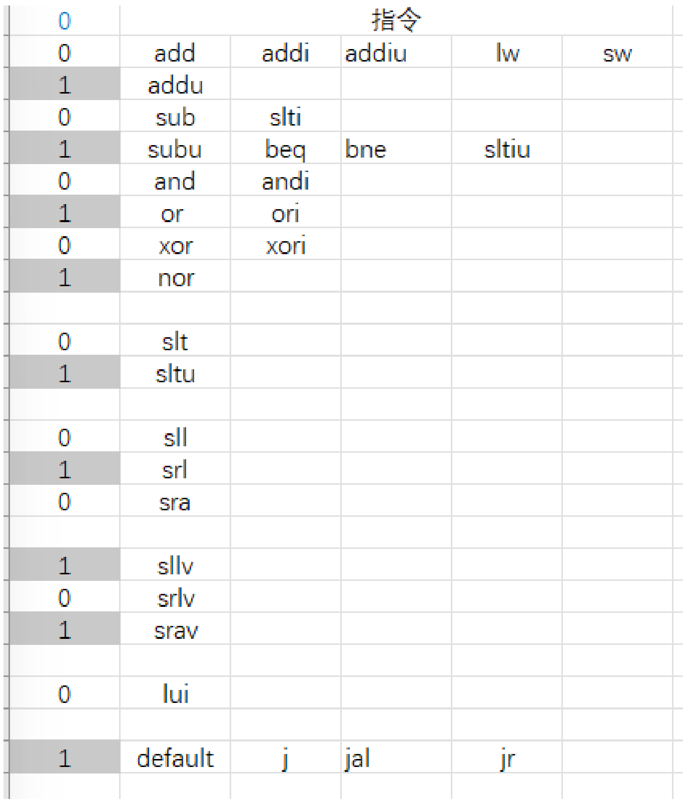


dmem_addr0_i
I0[31:0]
V=X"10010000"
I1[28:0]
O[31:0]
RTL_SUB
I2
dmem_addr_i
I0[31:0]
V=B"010"
I1[2:0]
O[10:0]
RTL_RSHIFT
dmem
addr[10:0]
clk
ena
in[31:0]
read
write
out[31:0]
DMEM
sccpu
clk
dmem_out[31:0]
ena
imem_out[31:0]
rst
dmem_addr[31:0]
dmem_in[31:0]
dmem_read
dmem_write
imem_addr[31:0]
CPU31
pc[31:0]
divide_clk
clk
rst
new_clk
DIVIDER
clk_in
reset
imem
in[10:0]
out[31:0]
IMEM
inst[31:0]_OBUF_inst
I
O
OBUF
inst[31:0]

顶层控制器逻辑

MUX(PC)		MUX(IN)		ALU			reg_write	(x表示不用,0表示无符号,1表示有符号)				DMEM	
[1]	[0]	[1]	[0]	op	zero	overflow		ext1	ext5	ext16	ext18	write	read
0	1	1	0	add	x		~overflow	x	x	x	x	0	0
0	1	1	0	addu	x		1	x	x	x	x	0	0
0	1	1	0	sub	x		1	x	x	x	x	0	0
0	1	1	0	subu	x		1	x	x	x	x	0	0
0	1	1	0	and	x		1	x	x	x	x	0	0
0	1	1	0	or	x		1	x	x	x	x	0	0
0	1	1	0	xor	x		1	x	x	x	x	0	0
0	1	1	0	nor	x		1	x	x	x	x	0	0
0	1	0	1	slt	x		1	0	x	x	x	0	0
0	1	0	1	sltu	x		1	0	x	x	x	0	0
0	1	1	0	sll	x		1	x	0	x	x	0	0
0	1	1	0	srl	x		1	x	0	x	x	0	0
0	1	1	0	sra	x		1	x	0	x	x	0	0
0	1	1	0	sllv	x		1	x	x	x	x	0	0
0	1	1	0	srlv	x		1	x	x	x	x	0	0
0	1	1	0	srav	x		1	x	x	x	x	0	0
0	0	x	x	default	x		0	x	x	x	x	0	0
0	1	1	0	add	x		1	x	x	1	x	0	0
0	1	1	0	add	x		1	x	x	1	x	0	0
0	1	1	0	and	x		1	x	x	0	x	0	0
0	1	1	0	or	x		1	x	x	0	x	0	0
0	1	1	0	xor	x		1	x	x	0	x	0	0
0	1	0	0	add	x		1	x	x	0	x	0	1
0	1	x	x	add	x		0	x	x	0	x	1	0
1	0	x	x	subu	1		0	x	x	x	0	0	0
0	1				0								
0	1	x	x	subu	1		0	x	x	x	0	0	0
1	0				0								
0	1	0	1	sub	x		1	0	x	1	x	0	0
0	1	0	1	subu	x		1	0	x	0	x	0	0
0	1	1	0	lui	x		1	x	x	0	x	0	0
1	1	x	x	default	x		0	x	x	x	x	0	0
1	1	1	1	default	x		1	x	x	x	x	0	0

ALU 编码



0	指令				
0	add	addi	addiu	lw	sw
1	addu				
0	sub	slti			
1	subu	beq	bne	sltiu	
0	and	andi			
1	or	ori			
0	xor	xori			
1	nor				
0	slt				
1	sltu				
0	sll				
1	srl				
0	sra				
1	sllv				
0	srlv				
1	srav				
0	lui				
1	default	j	jal	jr	