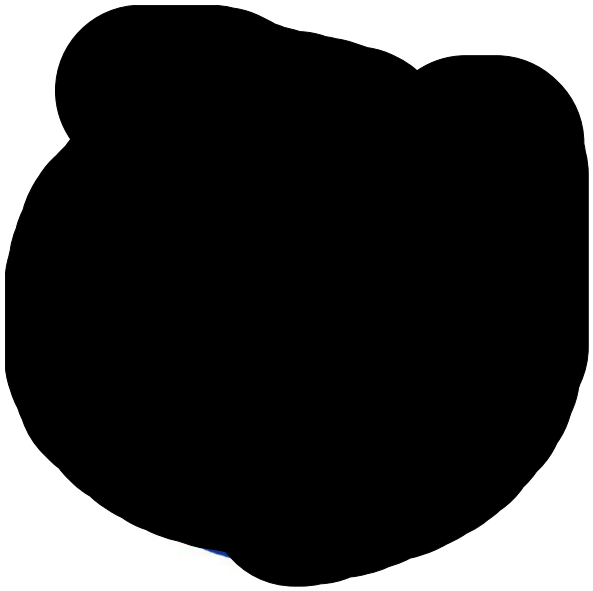
[redacted]

计算机组成原理课程实验报告



学　　号 [redacted]

姓　　名 [redacted]

专　　业 [redacted]

授课老师 [redacted]

一、实验内容

使用 Verilog HDL 实现 31 条 MIPS 指令的 CPU 的设计。

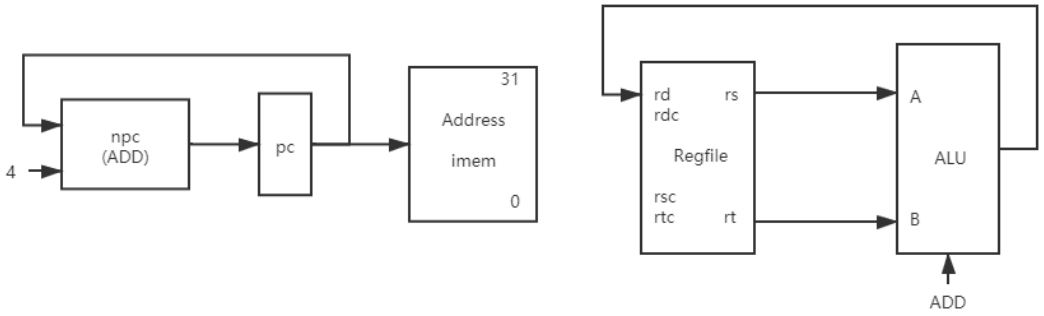
二、实验原理

1. ADD:

格式：ADD rd, rs, rt

操作：取指令，rd←rs+rt, PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
ADD	npc	pc	pc	ALU	rs	rt

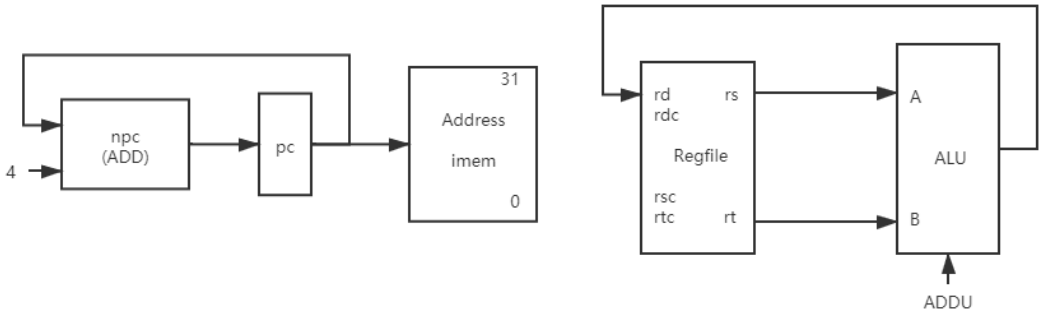


2. ADDU:

格式：ADDU rd, rs, rt

操作：取指令，rd←rs+rt, PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
ADDU	npc	pc	pc	ALU	rs	rt



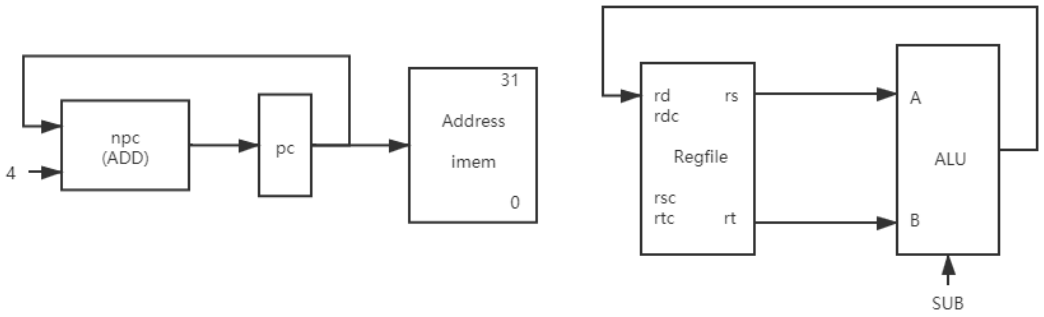
3. SUB:

格式：SUB rd, rs, rt

操作：取指令，rd←rs-rt, PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU

				Rd	A	B
SUB	npc	pc	pc	ALU	rs	rt

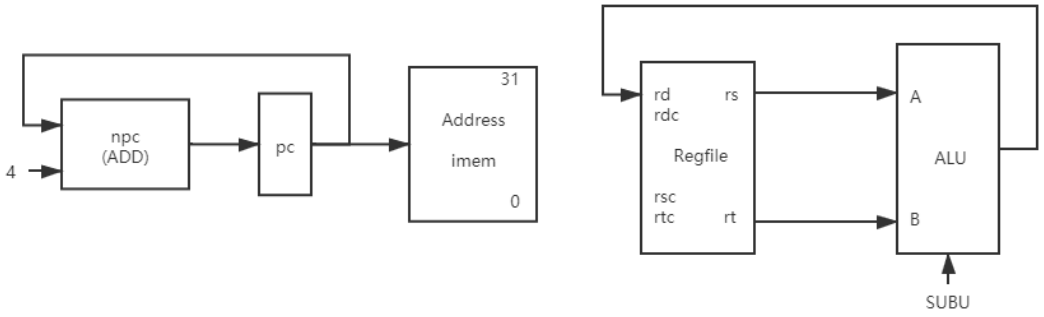


4. SUBU:

格式：SUBU rd, rs, rt

操作：取指令，rd←rs-rt, PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
SUBU	npc	pc	pc	ALU	rs	rt

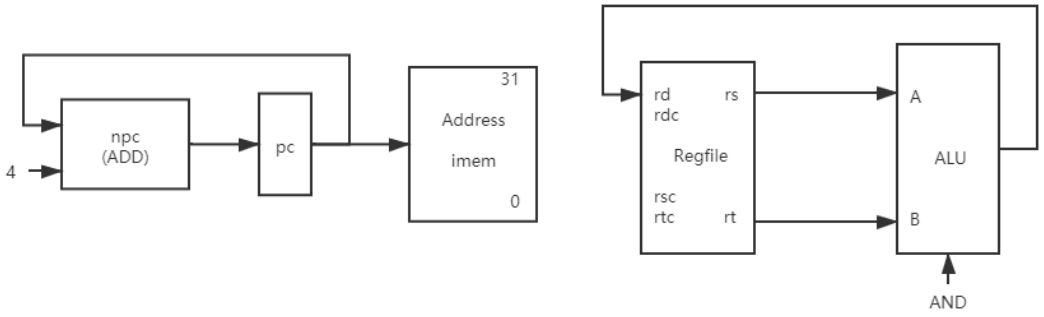


5. AND:

格式：AND rd, rs, rt

操作：取指令，rd←rs & rt, PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
AND	npc	pc	pc	ALU	rs	rt

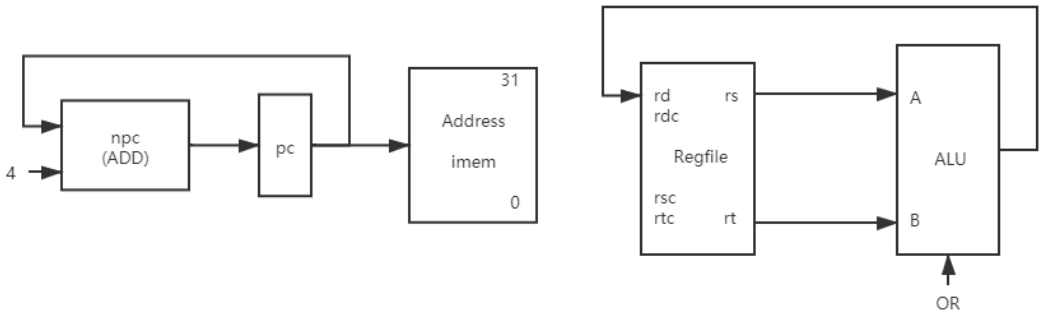


6. OR:

格式：OR rd, rs, rt

操作：取指令，rd←rs | rt, PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
OR	npc	pc	pc	ALU	rs	rt

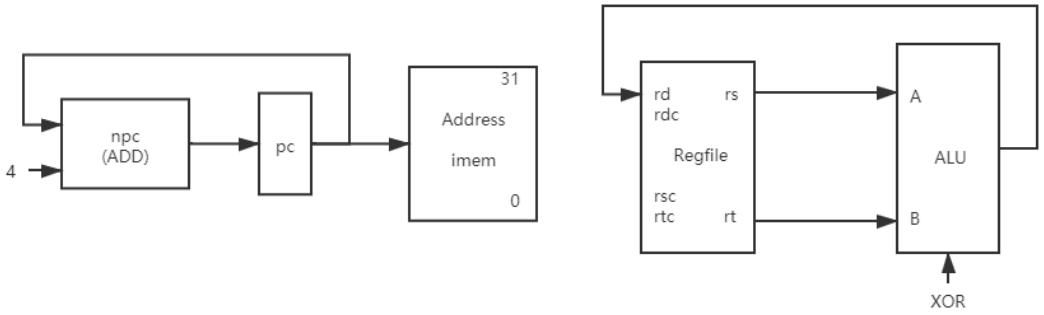


7. XOR:

格式：XOR rd, rs, rt

操作：取指令，rd←rs ^ rt, PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
XOR	npc	pc	pc	ALU	rs	rt

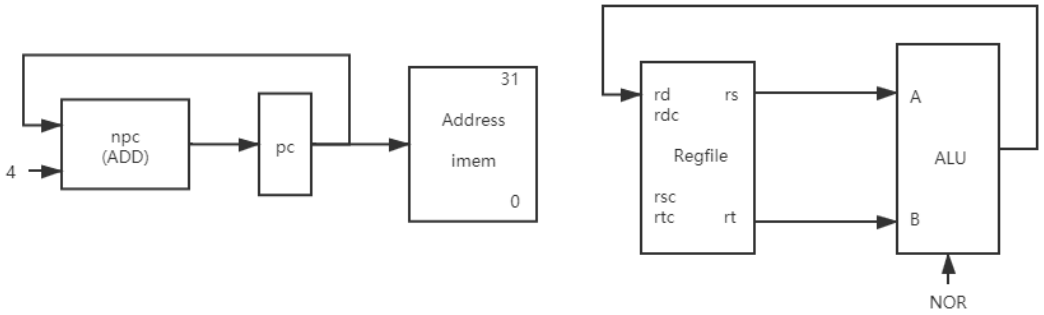


8. NOR:

格式：NOR rd, rs, rt

操作：取指令，rd←~(rs | rt), PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
NOR	npc	pc	pc	ALU	rs	rt

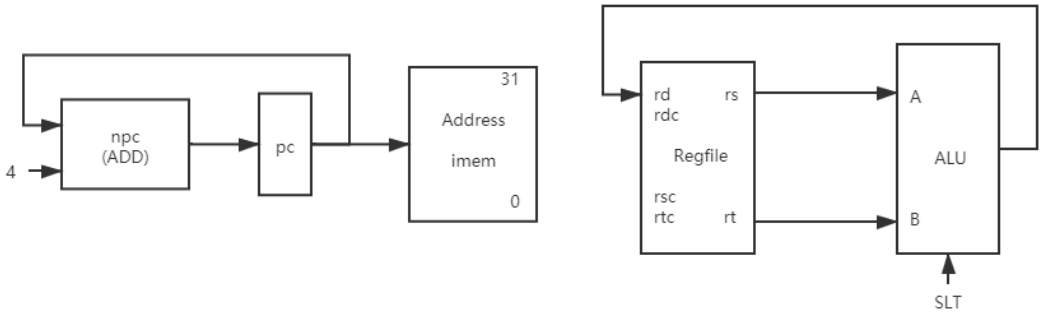


9. SLT:

格式：NOR rd, rs, rt

操作：取指令，rd←~(rs | rt), PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
SLT	npc	pc	pc	ALU	rs	rt

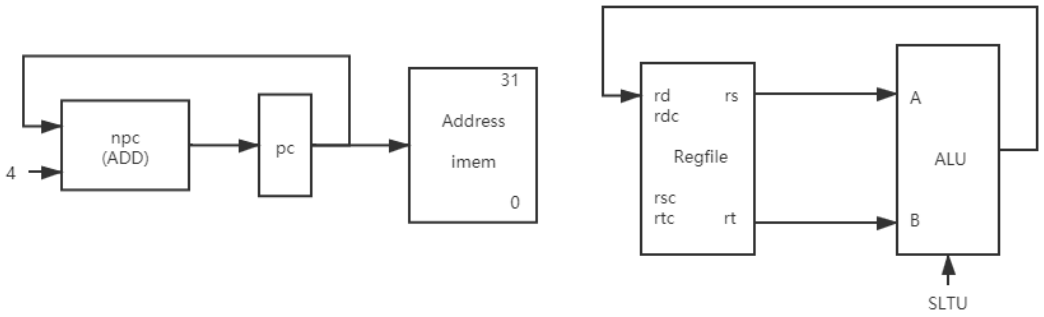


10. SLTU:

格式：SLTU rd, rs, rt

操作：取指令，rd← rs < rt , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
SLTU	npc	pc	pc	ALU	rs	rt

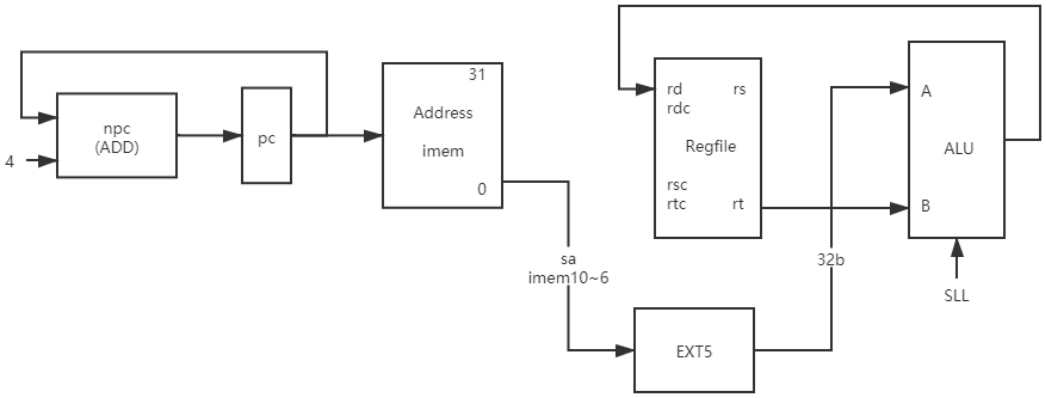


11. SLL:

格式：SLL rd, rt, sa

操作：取指令, rd←rt << sa , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext5
				Rd	A	B	
SLL	npc	pc	pc	ALU	Ext5	rt	sa

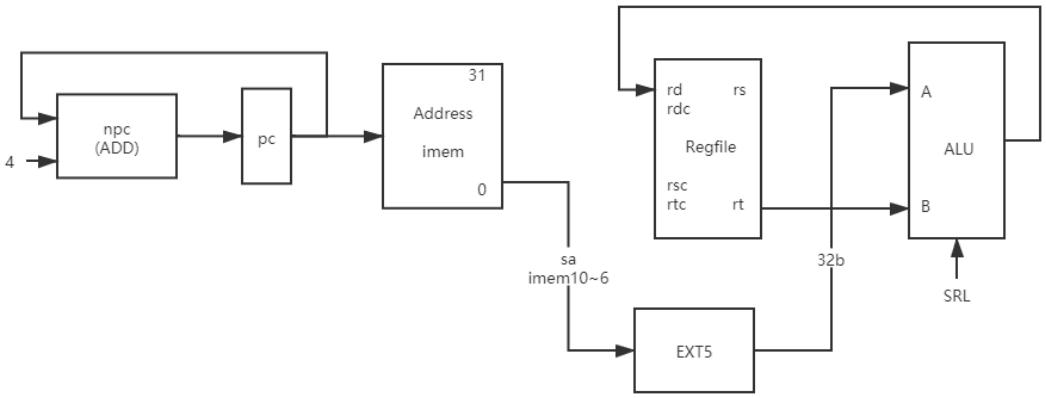


12. SRL:

格式：SRL rd, rt, sa

操作：取指令, rd←rt >> sa (logical) , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext5
				Rd	A	B	
SRL	npc	pc	pc	ALU	Ext5	rt	sa

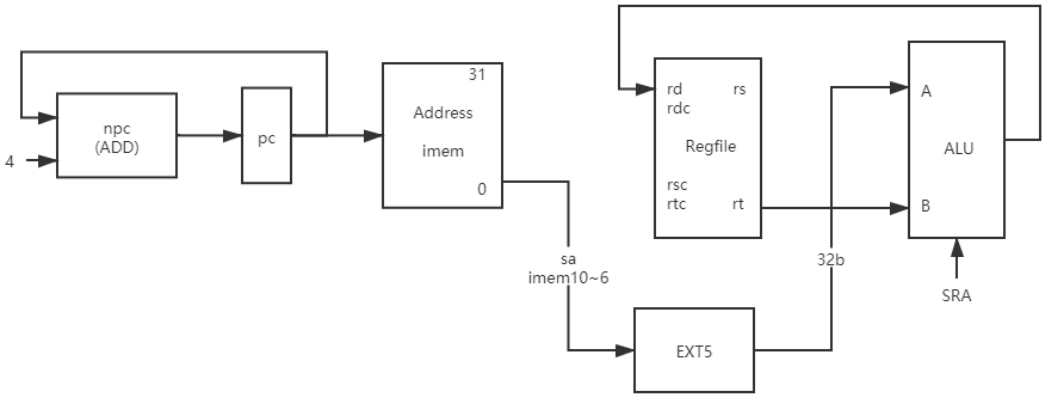


13.SRA:

格式：SRA rd, rt, sa

操作：取指令, rd←rt >> sa (arithmetic) , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext5
				Rd	A	B	
SRA	npc	pc	pc	ALU	Ext5	rt	sa

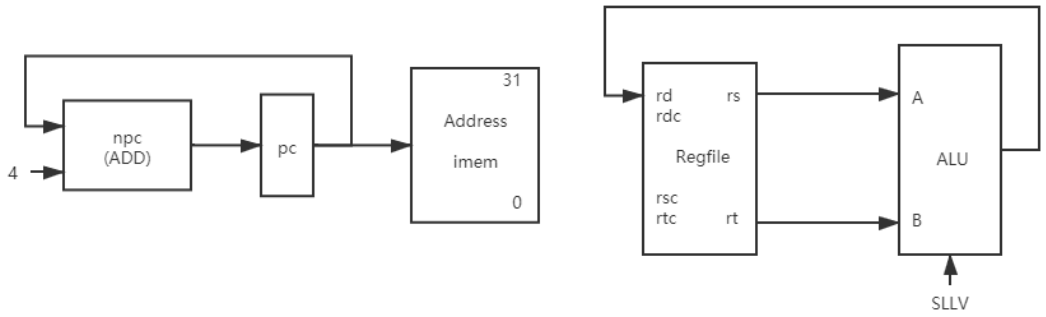


14 SLLV

格式：SLLV rd, rt, rs,

操作：取指令, rd←rt << rs , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
SLLV	npc	pc	pc	ALU	rs	rt

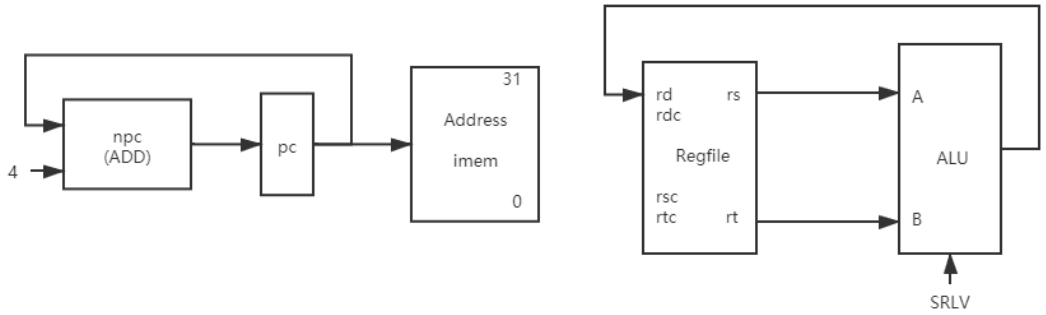


15 SRLV

格式：SRLV rd, rt,rs

操作：取指令, rd←rt >> rs (logical) , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
SRLV	npc	pc	pc	ALU	rs	rt

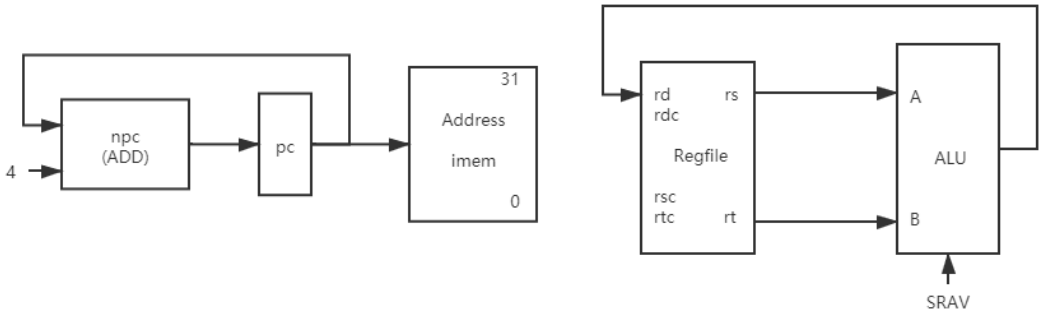


16 SRAV

格式：SRAV rd, rt,rs

操作：取指令, rd←rt >> rs (arithmetic) , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
SRAV	npc	pc	pc	ALU	rs	rt

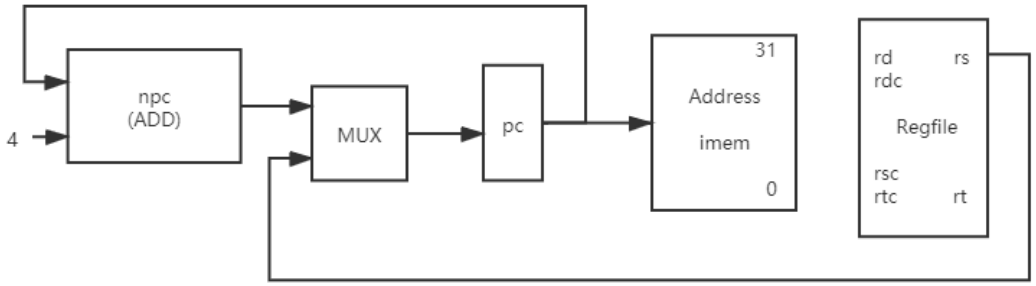


17.JR

格式：SRAV rd, rt,rs

操作：取指令, rd←rt >> rs (arithmetic) , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
JR	npc	pc	pc			

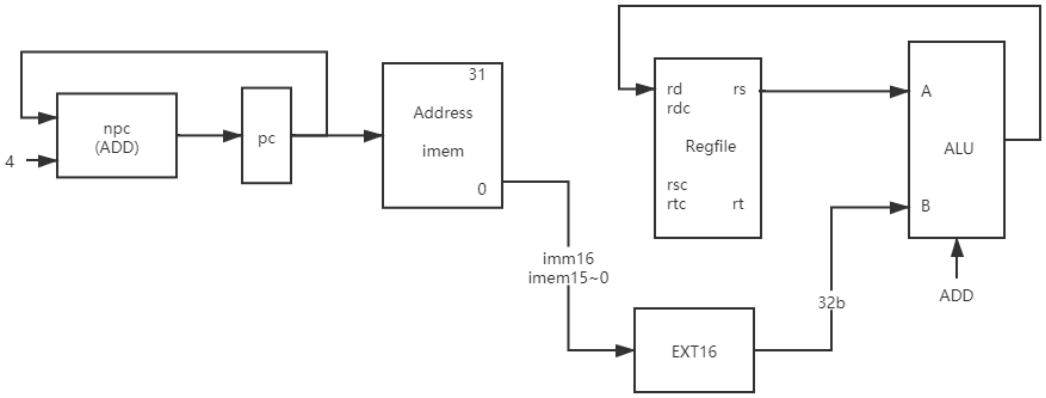


18.ADDI

格式：ADDI rt, rs, imm16

操作：取指令、rt←rs+imm16(sign_extend) 、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext16
				Rd	A	B	
ADDI	npc	pc	pc	ALU	rs	Ext16	Imm16

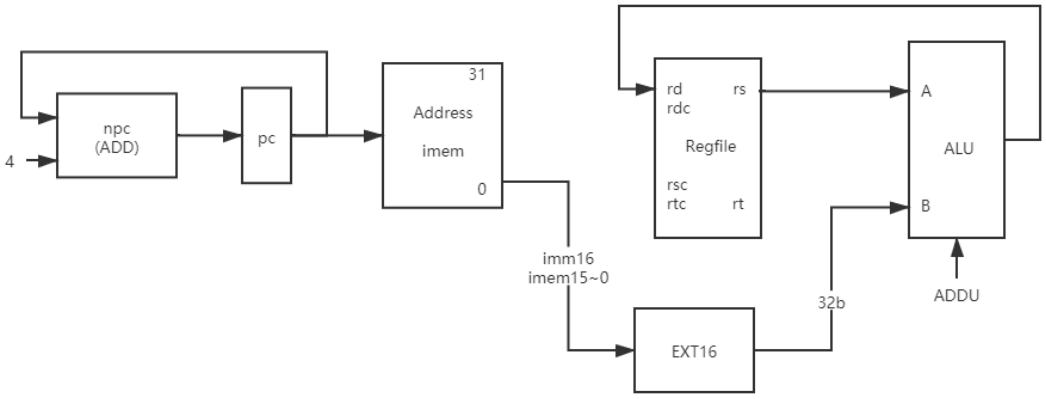


19.ADDIU

格式：ADDIU rt, rs, imm16

操作：取指令、rt←rs+imm16(sign_extend) 、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext16
				Rd	A	B	
ADDIU	npc	pc	pc	ALU	rs	Ext16	Imm16

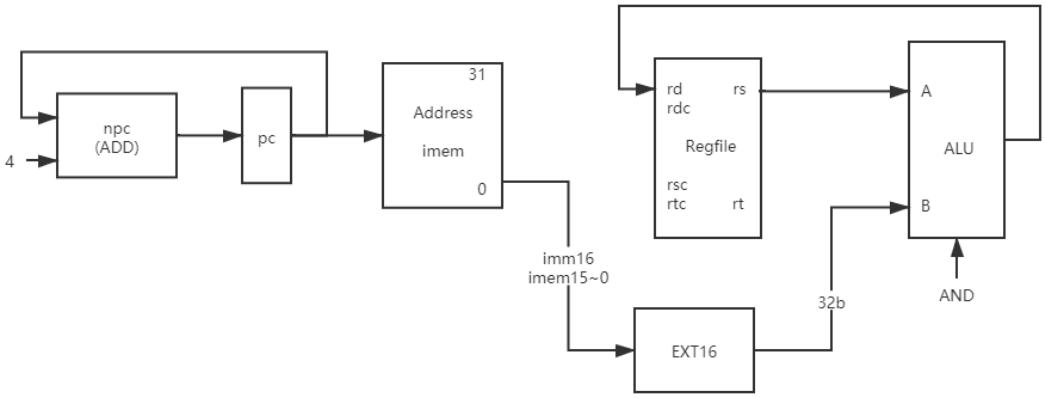


20.ANDI

格式：ANDI rt, rs, imm16

操作：取指令、rt←rs&imm16(zero_extend) 、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext16
				Rd	A	B	
ANDI	npc	pc	pc	ALU	rs	Ext16	Imm16

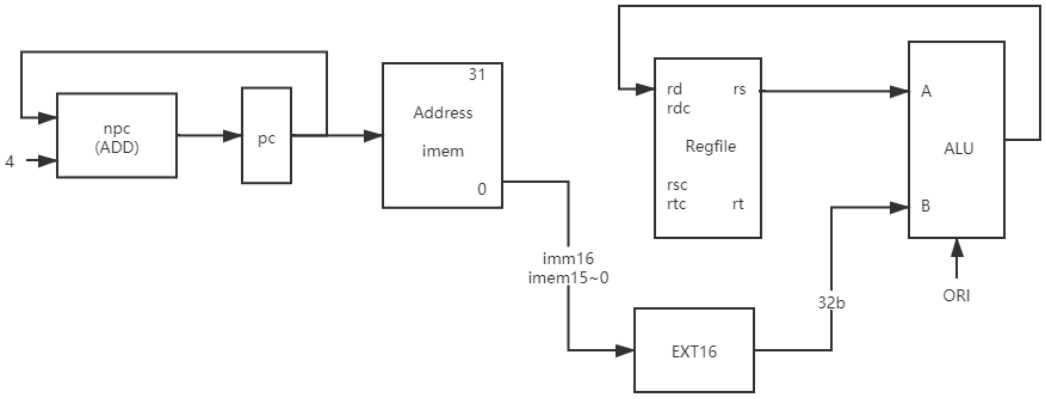


21.ORI

格式：ORI rt, rs, imm16

操作：取指令、rt←rs|imm16(zero_extend) 、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext16
				Rd	A	B	
ORI	npc	pc	pc	ALU	rs	Ext16	Imm16

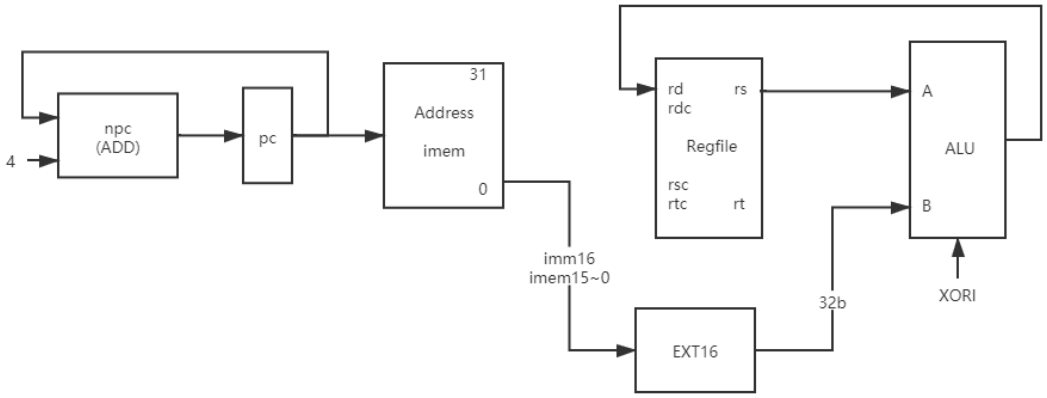


22.XORI

格式：XORI rt, rs, imm16

操作：取指令、rt←rs^imm16(zero_extend) 、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext16
				Rd	A	B	
XORI	npc	pc	pc	ALU	rs	Ext16	Imm16

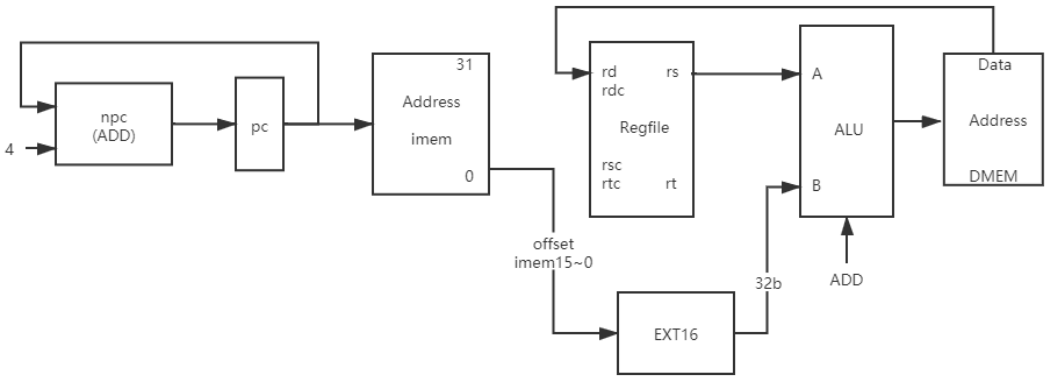


23.LW

格式：LW rt, offset(base)

操作：取指令、rt←memory[rs + offset]、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		DMEM		Ext16
				rd	A	B	Addr	Data	
LW	npc	pc	pc	Data	rs	Ext16	ALU		offset

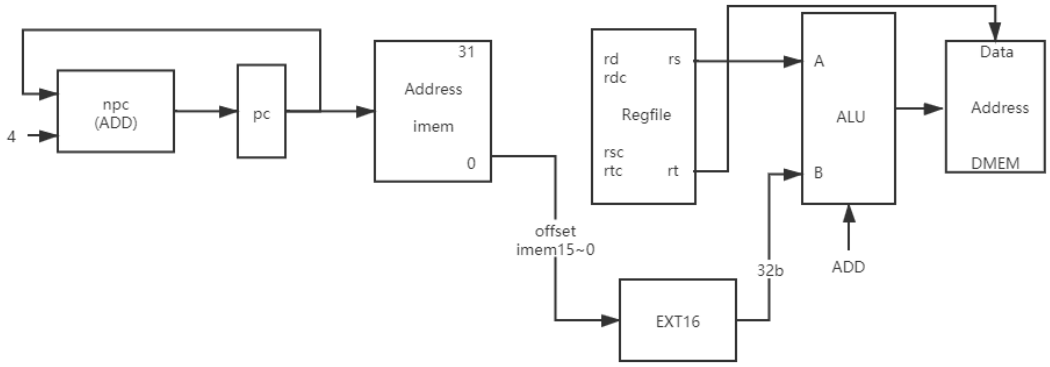


24.SW

格式：SW rt, offset(base)

操作：取指令、memory[base + offset]←rt、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		DMEM		Ext16
				rd	A	B	Addr	Data	
SW	npc	pc	pc		rs	Ext16	ALU	rt	offset



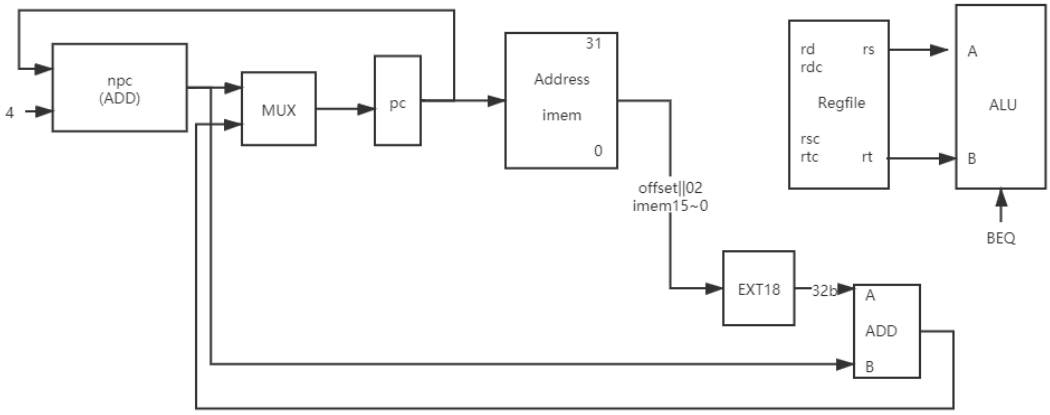
25.BEQ

格式：BEQ rs, rt, offset

操作: if (rs=rt)PC←NPC + Sign_ext(offset||02)

else PC← NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		ADD		Ext18
				rd	A	B	A	B	
BEQ	npc	pc	pc		rs	rt	Ext18	rt	offset



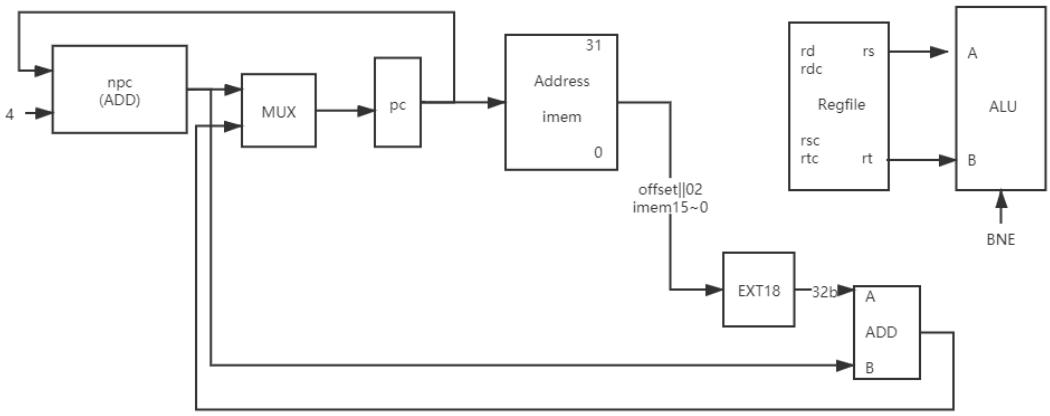
26.BNE

格式：BNE rs, rt, offset

操作：if (rs≠rt)PC←NPC + Sign_ext(offset||02)

else PC← NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		ADD		Ext18
				rd	A	B	A	B	
BNE	npc	pc	pc		rs	rt	Ext18	rt	offset

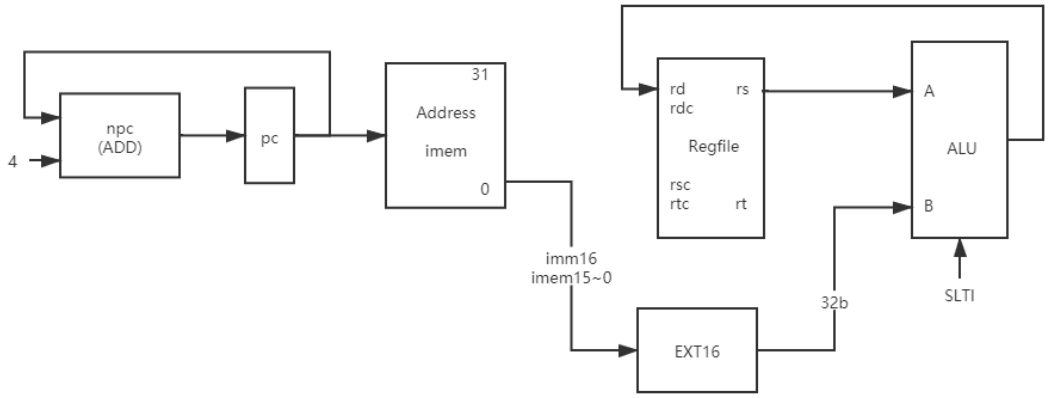


27.SLTI

格式：SLTI rt, rs, imm16

操作：取指令、rt←rs < ext.imm16(sign_extend) 、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext16
				Rd	A	B	
SLTI	npc	pc	pc	ALU	rs	Ext16	Imm16

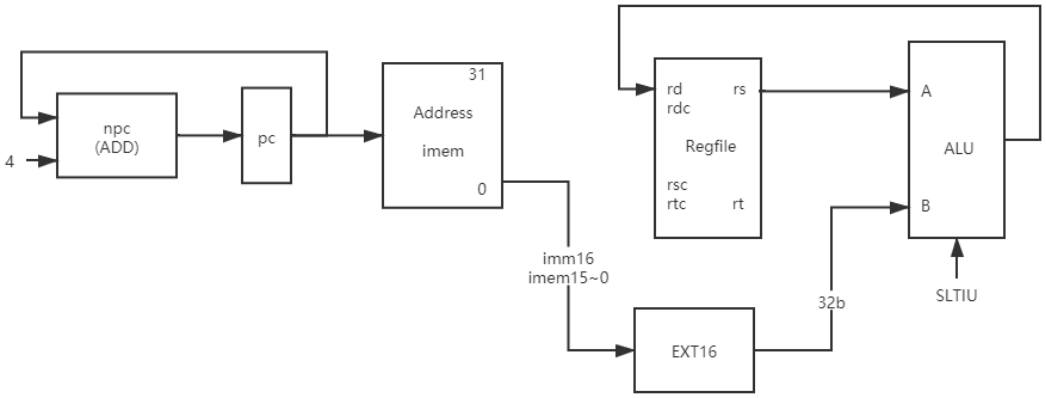


28.SLTIU

格式：SLTI rt, rs, imm16

操作：取指令、rt←rs < ext.imm16(sign_extend) 、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext16
				Rd	A	B	
SLTIU	npc	pc	pc	ALU	rs	Ext16	Imm16

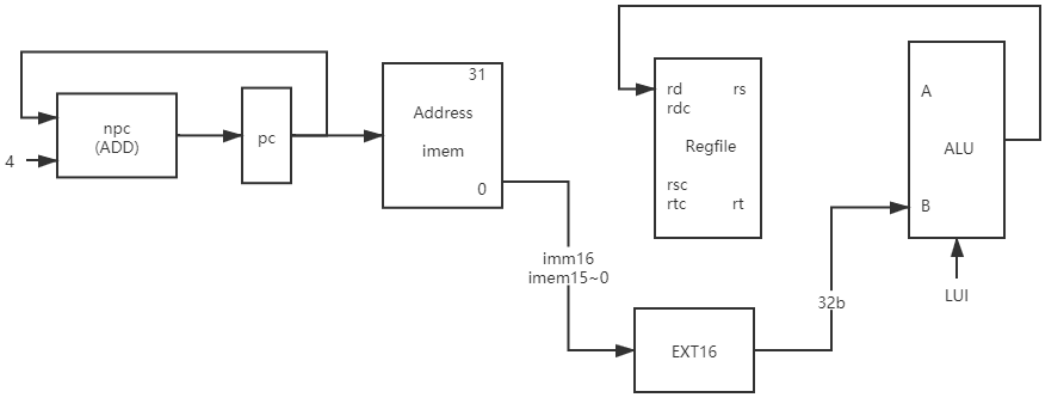


29.LUI

格式：LUI rt, imm16

操作：取指令、rt←imm16||0^{16} 、PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU		Ext16
				Rd	A	B	
LUI	npc	pc	pc	ALU		Ext16	Imm16

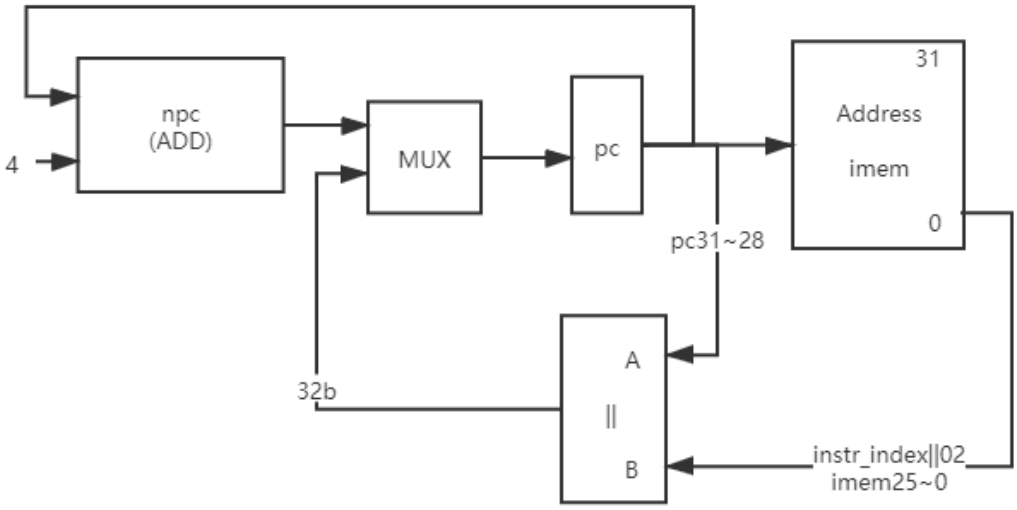


30.J

格式：J target

操作：取指令、PC ← PC_{31-28}||instr_index||0^2 , PC←NPC(PC+4)

指令	pc	npc	imem	RegFile	ALU	
				Rd	A	B
J	||	pc	pc			

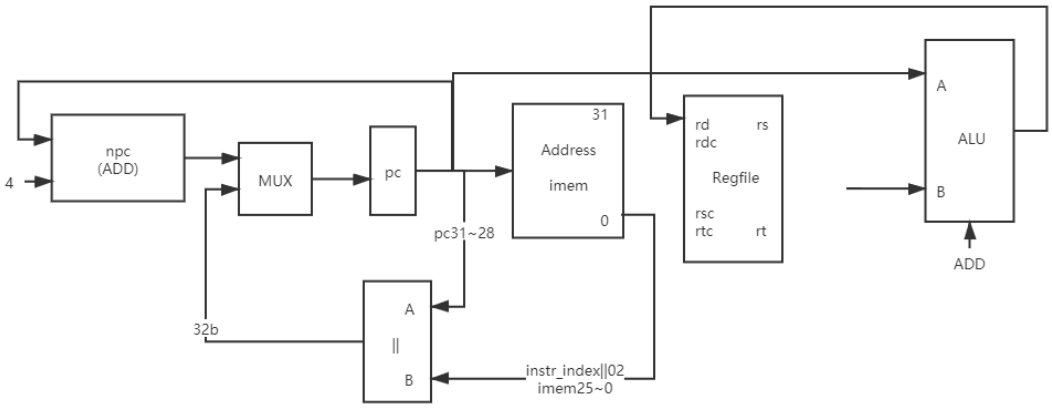


31.JAL

格式：JAL target

操作：取指令、 R[31] ← PC + 8,PC ← PC_{31-28}||instr_index||0^2 , PC←PC+4

	PC	NPC	IMEM	Rgefile		ALU		
				Rd	Rdc	A	B	
JAL	||	PC	PC	ALU		PC	8	

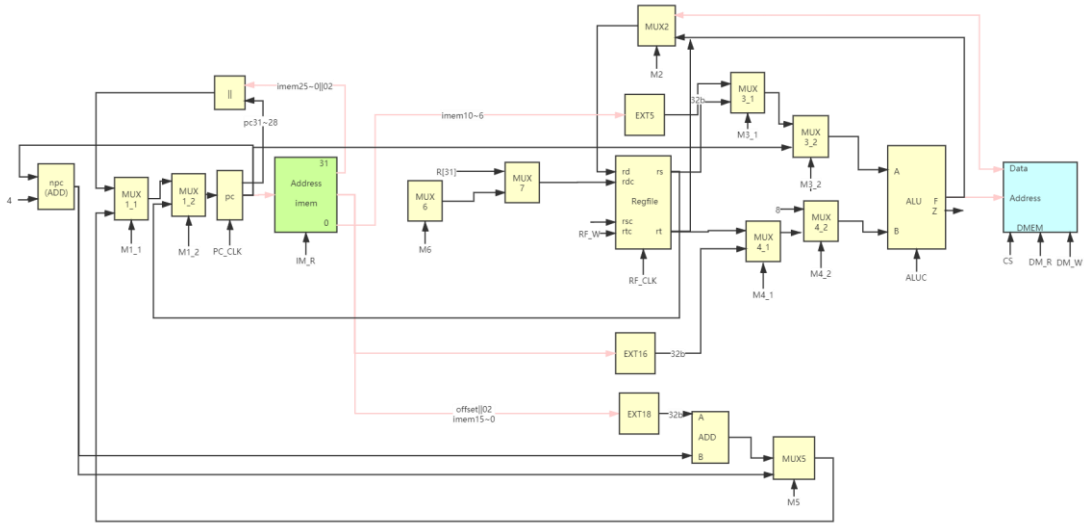


		pc	npc	imem	RegFile		ALU		Ext5	Ext16	Ext18	DMEM	
					rd	rdc	A	B				Addr	Data
1	add	npc	pc	pc	ALU	15-11	rs	rt					
2	addu	npc	pc	pc	ALU	15-11	rs	rt					
3	sub	npc	pc	pc	ALU	15-11	rs	rt					
4	subu	npc	pc	pc	ALU	15-11	rs	rt					
5	and	npc	pc	pc	ALU	15-11	rs	rt					
6	or	npc	pc	pc	ALU	15-11	rs	rt					
7	xor	npc	pc	pc	ALU	15-11	rs	rt					
8	nor	npc	pc	pc	ALU	15-11	rs	rt					

9	slt	npc	pc	pc	ALU	15-11	rs	rt					
10	sltu	npc	pc	pc	ALU	15-11	rs	rt					
11	sllv	npc	pc	pc	ALU	15-11	rs	rt					
12	srlv	npc	pc	pc	ALU	15-11	rs	rt					
13	srav	npc	pc	pc	ALU	15-11	rs	rt					
14	sll	npc	pc	pc	ALU	15-11	Ext5	rt	sa				
15	srl	npc	pc	pc	ALU	15-11	Ext5	rt	sa				
16	sra	npc	pc	pc	ALU	15-11	Ext5	rt	sa				
17	addi	npc	pc	pc	ALU	20-16	rs	Ext16		imm16			
18	addiu	npc	pc	pc	ALU	20-16	rs	Ext16		imm16			
19	andi	npc	pc	pc	ALU	20-16	rs	Ext16		imm16			
20	ori	npc	pc	pc	ALU	20-16	rs	Ext16		imm16			
21	xori	npc	pc	pc	ALU	20-16	rs	Ext16		imm16			
22	slti	npc	pc	pc	ALU	20-16	rs	Ext16		imm16			
23	sltiu	npc	pc	pc	ALU	20-16	rs	Ext16		imm16			
24	lui	npc	pc	pc	ALU	20-16		Ext16		imm16			
25	lw	npc	pc	pc	DMEM	20-16	rs	Ext16		offset		ALU	
26	sw	npc	pc	pc		20-16	rs	Ext16		offset		ALU	rt

		pc	npc	imem	RegFile		ALU		Ext18	ADD		||	
					rd	rdc	A	B		A	B	A	B
27	beq	ADD	pc	pc			rs	rt	offset	Ext18	npc		
28	bne	ADD	pc	pc			rs	rt	offset	Ext18	npc		
29	j	||	pc	pc								pc31-28	imem25-0
30	jal	||	pc	pc	ALU		pc					pc31-28	imem25-0
31	jr	rs	pc	pc									

总图：



三、模块建模

add.v:

```
`timescale 1ns / 1ps
module add(
    input [31:0] add_1,
    input [31:0] add_2,
    output [31:0] out_data
    );
    assign out_data=add_1+add_2;
endmodule
```

alu.v:

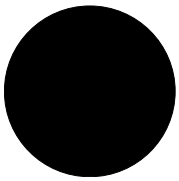
```
`timescale 1ns / 1ps
module alu(
    input [31:0] a,
    input [31:0] b,
    input [3:0] aluc,
    output reg [31:0] r,
    output reg zero,
    output reg carry,
    output reg negative,
    output reg overflow
    );
    reg [32:0] tempu;
    reg [32:0] tempru;
    reg signed [32:0] tempr;
    reg signed [32:0] temp;

    always@(*)
    begin
    if(aluc==4'b0000)
    begin
    tempu<=a+b;
    r<=tempu[31:0];
    if(r==0)
        zero<=1;
    else
        zero<=0;
    carry<=tempu[32];
    negative<=r[31];
    overflow<=0;
    end
```

```
else if(aluc==4'b0010)
begin
temp<=$signed(a)+$signed(b);
r<=temp[31:0];
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
if($signed(r)<0)
	negative<=1;
else
	negative<=0;
if($signed(a)<0&&$signed(b)<0)
begin
	if((-$signed(a))+(-$signed(b))>32'h7fffffff)
	overflow=1;
	else
	overflow=0;
end
else if($signed(a)>0&&$signed(b)>0)
begin
	if($signed(a)+$signed(b)>32'h7fffffff)
		overflow=1;
		else
		overflow=0;
end
else
overflow=0;
end

else if(aluc==4'b0001)
begin
tempu<=a-b;
r<=tempu[31:0];
if(r==0)
	zero<=1;
else
	zero<=0;
if(a<b)
	carry<=1;
else
	carry<=0;
negative=r[31];
```

```
overflow<=0;
end

else if(aluc==4'b0011)
begin
tempu<=a-b;
temp<=$signed(a)-$signed(b);
r<=temp[31:0];
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
if($signed(r)<0)
	negative<=1;
else
	negative<=0;
if($signed(a)<0&&$signed(b)>0)
	begin
		if((-$signed(a))+$signed(b)>32'h7fffffff)
		overflow=1;
		else
		overflow=0;
	end
	else if($signed(a)>0&&$signed(b)<0)
	begin
		if($signed(a)+(-$signed(b))>32'h7fffffff)
			overflow=1;
			else
			overflow=0;
	end
	else
	overflow=0;
end

else if(aluc==4'b0100)
begin
r<=a&b;
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
negative<=r[31];
```

```
overflow<=0;
end

else if(aluc==4'b0101)
begin
r<=a|b;
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b0110)
begin
r<=a^b;
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b0111)
begin
r<=~(a|b);
if(r==0)
	zero<=1;
else
	zero<=0;
carry<=0;
negative<=r[31];
overflow<=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;
carry<=0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b1011)
begin
r<=($signed(a)<$signed(b))? 1:0;
if($signed(a)==$signed(b))
        zero<=1;
else
        zero<=0;
carry<=0;
if($signed(a)<$signed(b))
        negative<=1;
else
        negative<=0;
overflow<=0;
end

else if(aluc==4'b1010)
begin
r<=(a<b)? 1:0;
if(a==b)
        zero<=1;
else
        zero<=0;
carry<=(a<b)? 1:0;
negative<=r[31];
overflow<=0;
end

else if(aluc==4'b1100)
begin
if(a>0&&a<=32)
carry<=b[a-1];
else if(a==0)
carry<=0;
else
carry<=b[31];
r<=$signed(b)>>>a;
if(r==0)
```



```
            zero<=1;
        else
            zero<=0;
        negative<=r[31];
        overflow<=0;
        end

        else if(aluc==4'b1110||aluc==4'b1111)
        begin
        tempu<=b<<a;
        r<=tempu[31:0];
        carry=tempu[32];
        if(r==0)
            zero<=1;
        else
            zero<=0;
        negative<=r[31];
        overflow<=0;
        end

        else if(aluc==4'b1101)
        begin
        if(a>0&&a<=32)
        carry<=b[a-1];
        else if(a==0)
        carry<=0;
        else
        carry<=0;
        r<=b>>a;
        if(r==0)
            zero<=1;
        else
            zero<=0;
        negative<=r[31];
        overflow<=0;
        end

        end
endmodule
```

controller.v:

```
`timescale 1ns / 1ps

module controller(
```

```
        input clk,
        input z,
        input [31:0] i,
        output PC_CLK,
        output IM_R,
        output M1_1,
        output M1_2,
        output M2,
        output M3_1,
        output M3_2,
        output M4_1,
        output M4_2,
        output M5,
        output M6_1,
        output M6_2,
        output [3:0] ALUC,
        output RF_W,
        output RF_CLK,
        output DM_w,
        output DM_r,
        output DM_cs,
        output C_EXT16
);
        assign _add  =i[0];
        assign _addu =i[1];
        assign _sub  =i[2];
        assign _subu =i[3];
        assign _and  =i[4];
        assign _or   =i[5];
        assign _xor  =i[6];
        assign _nor  =i[7];
        assign _slt  =i[8];
        assign _sltu =i[9];
        assign _sll  =i[10];
        assign _srl  =i[11];
        assign _sra  =i[12];
        assign _sllv =i[13];
        assign _srlv =i[14];
        assign _srav =i[15];
        assign _jr   =i[16];
        assign _addi =i[17];
        assign _addiu=i[18];
        assign _andi =i[19];
        assign _ori  =i[20];
```

```
    assign _xori =i[21];
    assign _lw      =i[22];
    assign _sw      =i[23];
    assign _beq    =i[24];
    assign _bne    =i[25];
    assign _slti =i[26];
    assign _sltiu=i[27];
    assign _lui    =i[28];
    assign _j       =i[29];
    assign _jal    =i[30];


    assign PC_CLK    = ~clk;
    assign IM_R       = 1;
    assign M1_1       = ~(_j |_jr |_jal);
    assign M1_2       =    _jr;
    assign M2          = ~_lw;
    assign M3_1       = _sll|_srl|_sra;//1
    assign M3_2       = _jal;
    assign M4_1       = _lui|_addi|_addiu|_andi|_ori|_xori|_lw|_sw|_slti|_sltiu;//0
    assign M4_2       = _jal;
    assign M5          =(_beq&z)|(_bne&~z);
    assign M6_1       = _lui|_addi|_addiu|_andi|_ori|_xori|_lw|_slti|_sltiu;//0
    assign M6_2       =_jal;
    assign ALUC[3] =_slt|_sltu|_sll|_srl|_sra|_sllv|_srlv|_srav|_lui|_slti|_sltiu;//1
    assign ALUC[2] =_and|_or|_xor|_nor|_sll|_srl|_sra|_sllv|_srlv|_srav|_andi|_ori|_xori;//1
    assign ALUC[1] =_add|_sub|_xor|_nor|_slt|_sltu|_sll|_sllv|_addi|_xori|_slti|_sltiu;//1
    assign ALUC[0] =_sub|_subu|_or|_nor|_slt|_srl|_srlv|_ori|_beq|_bne|_slti;//0
    assign RF_W=~(_jr|_sw|_beq|_bne|_j);//1
    assign RF_CLK    =~clk;
    assign DM_cs=_lw|_sw;
    assign DM_r=_lw;
    assign DM_w=_sw;
    assign C_EXT16 =~(_andi|_ori|_xori|_lui);//1

endmodule
```

cpu.v:

```
`timescale 1ns / 1ps

module cpu(
    input               clk,
    input               reset,
    input    [31:0] inst,
```

```
    input    [31:0] rdata,
    output [31:0] pc,
    output [31:0] DM_addr,
    output [31:0] DM_wdata,
    output        DM_CS,
    output        DM_R,
    output        DM_W
);

    wire PC_CLK;
    wire PC_ENA;
    wire M1_1, M2, M3_1,M3_2, M4_1,M4_2,M5,M6_1, M6_2;
    wire [3:0] ALUC;
    wire RF_W;
    wire RF_CLK;
    wire C_EXT16;

    wire zero;
    wire carry;
    wire negative;
    wire overflow;
    wire add_overflow;

    wire [31:0] ins_code;

    wire [31:0] ALU;
    wire [31:0] PC;
    wire [31:0] RF;
    wire [31:0] Rs;
    wire [31:0] Rt;
    wire [31:0] IM;
    wire [31:0] DM;
    wire [31:0] Mux1_1;
    wire [31:0] Mux1_2;
    wire [31:0] Mux2;
    wire [31:0] Mux3_1;
    wire [31:0] Mux3_2;
    wire [31:0] Mux4_1;
    wire [31:0] Mux4_2;
    wire [31:0] Mux5;
    wire [4:0]    Mux6_1;
    wire [4:0]    Mux6_2;
    wire [31:0] Mux7;
    wire [31:0] Mux8;
```

```
    wire [31:0] Mux9;
    wire [31:0] Mux10;
    wire        Mux11;

    wire [31:0] EXT1;
    wire [31:0] EXT5;
    wire [31:0] EXT16;
    wire [31:0] EXT18;
    wire [31:0] ADD;
    wire [31:0] ADD8;
    wire [31:0] NPC;
    wire [31:0] ii;

    assign PC_ENA = 1;

    assign pc = PC;
    assign DM_addr = ALU;
    assign DM_wdata = Rt;

    instr_decode cpu_inst (
        .instr_raw(inst),
        .instr_code(ins_code)
    );

    controller cpu_control (
        .clk(clk), .z(zero),
        .i(ins_code), .PC_CLK(PC_CLK),
        .IM_R(IM_R), .M1_1(M1_1), .M1_2(M1_2), .M2(M2), .M3_1(M3_1), .M3_2(M3_2), .
M4_1(M4_1), .M4_2(M4_2),.M5(M5), .M6_1(M6_1),.M6_2(M6_2),
        .ALUC(ALUC), .RF_W(RF_W), .RF_CLK(RF_CLK), .DM_w(DM_W), .DM_r(DM_
R), .DM_cs(DM_CS), .C_EXT16(C_EXT16)
    );

    pcreg cpu_pc (
        .clk(PC_CLK), .rst(reset), .ena(PC_ENA),
        .data_in(Mux1_2), .data_out(PC)
    );

    npc cpu_npc    (
        .in(PC), .out(NPC)
    );

    alu cpu_alu (
        .a(Mux3_2), .b(Mux4_2),
```

```
        .aluc(ALUC), .r(ALU),
        .zero(zero), .carry(carry), .negative(negative), .overflow(overflow)
);

II cpu_ii(
        .a(PC[31:28]),
        .b(inst[25:0]),
        .out_data(ii)
);

mux cpu_mux1_1(
        .a(ii),
        .b(Mux5),
        .choice(M1_1),
        .out_data(Mux1_1)
);

mux cpu_mux1_2(
        .a(Mux1_1),
        .b(Rs),
        .choice(M1_2),
        .out_data(Mux1_2)
);

mux cpu_mux2(
        .a(rdata),
        .b(ALU),
        .choice(M2),
        .out_data(Mux2)
);

mux cpu_mux3_1(
        .a(Rs),
        .b(EXT5),
        .choice(M3_1),
        .out_data(Mux3_1)
);

mux cpu_mux3_2(
        .a(Mux3_1),
        .b(PC),
        .choice(M3_2),
        .out_data(Mux3_2)
);
```

```
mux cpu_mux4_1(
        .a(Rt),
        .b(EXT16),
        .choice(M4_1),
        .out_data(Mux4_1)
);

mux cpu_mux4_2(
        .a(Mux4_1),
        .b(32'd4),
        .choice(M4_2),
        .out_data(Mux4_2)
);

mux cpu_mux5(
        .a(NPC),
        .b(ADD),
        .choice(M5),
        .out_data(Mux5)
);

mux_5bits cpu_mux6_1(
        .a(inst[15:11]),
        .b(inst[20:16]),
        .choice(M6_1),
        .out_data(Mux6_1)
);

mux_5bits cpu_mux6_2(
        .a(Mux6_1),
        .b(5'd31),
        .choice(M6_2),
        .out_data(Mux6_2)
);

ext5 cpu_ext5(
        .a(inst[10:6]),
        .b(EXT5)
);

ext16 cpu_ext16 (
        .a(inst[15:0]),
        .sext(C_EXT16),
```

```
        .b(EXT16)
    );

    ext18 cpu_ext18(
        .a(inst[15:0]),
        .b(EXT18)
    );

    add cpu_add(
        .add_1(EXT18),
        .add_2(NPC),
        .out_data(ADD)
    );

    regfile cpu_ref(
        .clk(RF_CLK),.rst(reset), .we(RF_W),
        .Rsc(inst[25:21]), .Rtc(inst[20:16]), .Rdc(Mux6_2),
        .Rs(Rs), .Rt(Rt), .Rd(Mux2)
    );

endmodule
```

DMEM.v:

```
`timescale 1ns / 1ps
module DMEM(
    input clk,
    input ena,
    input write,
    input read,
    input [10:0] addr,
    input [31:0] wdata,
    output [31:0] rdata
);

    reg [31:0] data[0:31];

    always @ (negedge clk)
    begin
        if (write && ena)
        begin
            data[addr] <= wdata;
        end
    end
```

```
    assign rdata = (read && ena) ? data[addr] : 32'bz;

endmodule
```

ext5.v:

```
`timescale 1ns / 1ps
module ext5 #(parameter WIDTH=5)(
    input [WIDTH-1:0] a,
    output [31:0] b
    );
    assign b={{(32-WIDTH){0}},a};
endmodule
```

ext16.v:

```
`timescale 1ns / 1ps
module ext16 #(parameter WIDTH=16)(
    input [WIDTH-1:0] a,
    input sext,
    output [31:0] b
    );
    assign b=sext?{{(32-WIDTH){a[WIDTH-1]}},a}:{27'b0,a};
endmodule
```

ext18.v:

```
`timescale 1ns / 1ps
module ext18 (
    input [15:0] a,
    output [31:0] b
    );
    assign b={{14{a[15]}},a,2'b0};
endmodule
```

II.v:

```
`timescale 1ns / 1ps

module II(
    input [3:0] a,
    input [25:0] b,
    output [31:0] out_data
);

    assign out_data = {a, b<<2};

endmodule
```

```
`timescale 1ns / 1ps

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

	dist_mem_gen_0 imem(
		.a(addr),
		.spo(instr)
	);
endmodule
```

instr_decode.v:

```
`timescale 1ns / 1ps
module instr_decode(
	input [31:0] instr_raw,
	output reg [31:0] instr_code
);
	always @ (*) begin
		casex ({instr_raw[31:26], instr_raw[5:0]})
			12'b000000_100000 : instr_code <= 32'b00000000000000000000000000000001; //
add
			12'b000000_100001 : instr_code <= 32'b00000000000000000000000000000010; //
addu
			12'b000000_100010 : instr_code <= 32'b00000000000000000000000000000100; //
sub
			12'b000000_100011 : instr_code <= 32'b00000000000000000000000000001000; //
subu
			12'b000000_100100 : instr_code <= 32'b00000000000000000000000000010000; //
and
			12'b000000_100101 : instr_code <= 32'b00000000000000000000000000100000; //
or
			12'b000000_100110 : instr_code <= 32'b00000000000000000000000001000000; //
xor
			12'b000000_100111 : instr_code <= 32'b00000000000000000000000010000000; //
nor
			12'b000000_101010 : instr_code <= 32'b00000000000000000000000100000000; //
slt
			12'b000000_101011 : instr_code <= 32'b00000000000000000000001000000000; //
sltu
			12'b000000_000000 : instr_code <= 32'b00000000000000000000010000000000; //
sll
```

```
                12'b000000_000010 : instr_code <= 32'b00000000000000000000100000000000; //
srl
                12'b000000_000011 : instr_code <= 32'b00000000000000000001000000000000; //
sra
                12'b000000_000100 : instr_code <= 32'b00000000000000000010000000000000; //
sllv
                12'b000000_000110 : instr_code <= 32'b00000000000000000100000000000000; //
srlv
                12'b000000_000111 : instr_code <= 32'b00000000000000001000000000000000; //
srav
                12'b000000_001000 : instr_code <= 32'b00000000000000010000000000000000; //
jr
                12'b001000_?????? : instr_code <= 32'b00000000000000100000000000000000; //
addi
                12'b001001_?????? : instr_code <= 32'b00000000000001000000000000000000; //
addiu
                12'b001100_?????? : instr_code <= 32'b00000000000010000000000000000000; //
andi
                12'b001101_?????? : instr_code <= 32'b00000000000100000000000000000000; //
ori
                12'b001110_?????? : instr_code <= 32'b00000000001000000000000000000000; //
xori
                12'b100011_?????? : instr_code <= 32'b00000000010000000000000000000000; //
lw
                12'b101011_?????? : instr_code <= 32'b00000000100000000000000000000000; //
sw
                12'b000100_?????? : instr_code <= 32'b00000001000000000000000000000000; //
beq
                12'b000101_?????? : instr_code <= 32'b00000010000000000000000000000000; //
bne
                12'b001010_?????? : instr_code <= 32'b00000100000000000000000000000000; //
slti
                12'b001011_?????? : instr_code <= 32'b00001000000000000000000000000000; //
sltiu
                12'b001111_?????? : instr_code <= 32'b00010000000000000000000000000000; //
lui
                12'b000010_?????? : instr_code <= 32'b00100000000000000000000000000000; //
j
                12'b000011_?????? : instr_code <= 32'b01000000000000000000000000000000; //
jal
                default:                instr_code <= 32'bzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz; //
ERROR
            endcase
        end
```

```
endmodule
```

mux.v:

```
`timescale 1ns / 1ps
module mux(
	input [31:0] a,
	input [31:0] b,
	input choice,
	output [31:0] out_data
	);
	assign out_data=(choice)?b:a;
endmodule
```

mux_5bits.v:

```
`timescale 1ns / 1ps
module mux_5bits(
	input [4:0] a,
	input [4:0] b,
	input choice,
	output [4:0] out_data
	);
	assign out_data=(choice)?b:a;
endmodule
```

npc.v:

```
`timescale 1ns / 1ps
module npc(
	input [31:0] in,
	output [31:0] out
	);
	assign out=in+4;
endmodule
```

pcreg.v:

```
`timescale 1ns / 1ps
module pcreg(
	input clk,
	input rst,
	input ena,
	input [31:0] data_in,
	output [31:0] data_out
	);

	reg [31:0] data=32'b0;
```

```
    always @(posedge clk or posedge rst) begin
        if(rst) data<=32'h00400000;
        else begin
            if(ena) data<=data_in;
        end
    end

    assign data_out     =     data;

endmodule
```

regfile.v:

```
`timescale 1ns / 1ps

module regfile(
    input clk,
    input rst,
    input we,
    input [4:0] Rsc,
    input [4:0] Rtc,
    output [31:0] Rs,
    output [31:0] Rt,
    input [4:0] Rdc,
    input [31:0] Rd
);

    reg [31:0] array_reg[0:31];

    always @ (posedge clk or posedge rst) 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 begin
                if (we&& Rdc != 5'b0) begin
                    array_reg[Rdc] <= Rd;
                end
            end
        end

        assign Rs = array_reg[Rsc];
        assign Rt = array_reg[Rtc];

endmodule
```

sccomp_dataflow.v:

```
`timescale 1ns / 1ps

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

        wire dmw, dmr, dena;
        wire [31:0] wdata, rdata;
        wire [31:0] instr, dma;
```

```
    wire [10:0] dm_addr;
    wire [10:0] im_addr;

    IMEM imem(
        .addr(im_addr),
        .instr(instr)
    );

    DMEM dmem(
        .clk(clk_in), .ena(dena),
        .write(dmw), .read(dmr),
        .addr(dm_addr[10:0]), .wdata(wdata), .rdata(rdata)
    );

    cpu sccpu(
        .clk(clk_in), .reset(reset),
        .inst(instr), .rdata(rdata),
        .DM_CS(dena), .DM_R(dmr), .DM_W(dmw), .DM_addr(dma), .DM_wdata(wdata),
        .pc(pc)
    );
    assign inst = instr;
    assign im_addr = (pc    - 32'h00400000) / 4;
    assign dm_addr = (dma - 32'h10010000) / 4;

endmodule
```

四、测试模块建模

```
 module test();
    reg clk;
    reg rst;
    wire [31:0] inst;
    wire [31:0] pc;

    integer file_output;
    integer counter = 0;


    sccomp_dataflow uut(.clk_in(clk),.reset(rst),.inst(inst),.pc(pc)//.addr(addr)
    );


        initial
        begin
```

```
        file_output = $fopen("D:/result.txt");
        clk = 0;
        rst = 1;
        #225;
        rst = 0;

end

always
  begin
#50;
  clk = ~clk;
if (clk == 1'b1)
begin
if (counter == 5000)
begin
  $fclose(file_output);
end
#2
      counter = counter + 1;
      if (clk==1'b1&&rst==0)
     begin
      $fdisplay(file_output,"regfiles0 = %h",uut.sccpu.cpu_ref.array_reg[0]);
      $fdisplay(file_output,"regfiles1 = %h",uut.sccpu.cpu_ref.array_reg[1]);
      $fdisplay(file_output,"regfiles2 = %h",uut.sccpu.cpu_ref.array_reg[2]);
      $fdisplay(file_output,"regfiles3 = %h",uut.sccpu.cpu_ref.array_reg[3]);
      $fdisplay(file_output,"regfiles4 = %h",uut.sccpu.cpu_ref.array_reg[4]);
      $fdisplay(file_output,"regfiles5 = %h",uut.sccpu.cpu_ref.array_reg[5]);
      $fdisplay(file_output,"regfiles6 = %h",uut.sccpu.cpu_ref.array_reg[6]);
      $fdisplay(file_output,"regfiles7 = %h",uut.sccpu.cpu_ref.array_reg[7]);
      $fdisplay(file_output,"regfiles8 = %h",uut.sccpu.cpu_ref.array_reg[8]);
      $fdisplay(file_output,"regfiles9 = %h",uut.sccpu.cpu_ref.array_reg[9]);
      $fdisplay(file_output,"regfiles10 = %h",uut.sccpu.cpu_ref.array_reg[10]);
      $fdisplay(file_output,"regfiles11 = %h",uut.sccpu.cpu_ref.array_reg[11]);
      $fdisplay(file_output,"regfiles12 = %h",uut.sccpu.cpu_ref.array_reg[12]);
      $fdisplay(file_output,"regfiles13 = %h",uut.sccpu.cpu_ref.array_reg[13]);
      $fdisplay(file_output,"regfiles14 = %h",uut.sccpu.cpu_ref.array_reg[14]);
      $fdisplay(file_output,"regfiles15 = %h",uut.sccpu.cpu_ref.array_reg[15]);
      $fdisplay(file_output,"regfiles16 = %h",uut.sccpu.cpu_ref.array_reg[16]);
      $fdisplay(file_output,"regfiles17 = %h",uut.sccpu.cpu_ref.array_reg[17]);
      $fdisplay(file_output,"regfiles18 = %h",uut.sccpu.cpu_ref.array_reg[18]);
      $fdisplay(file_output,"regfiles19 = %h",uut.sccpu.cpu_ref.array_reg[19]);
      $fdisplay(file_output,"regfiles20 = %h",uut.sccpu.cpu_ref.array_reg[20]);
      $fdisplay(file_output,"regfiles21 = %h",uut.sccpu.cpu_ref.array_reg[21]);
```

```
            $fdisplay(file_output,"regfiles22 = %h",uut.sccpu.cpu_ref.array_reg[22]);
            $fdisplay(file_output,"regfiles23 = %h",uut.sccpu.cpu_ref.array_reg[23]);
            $fdisplay(file_output,"regfiles24 = %h",uut.sccpu.cpu_ref.array_reg[24]);
            $fdisplay(file_output,"regfiles25 = %h",uut.sccpu.cpu_ref.array_reg[25]);
            $fdisplay(file_output,"regfiles26 = %h",uut.sccpu.cpu_ref.array_reg[26]);
            $fdisplay(file_output,"regfiles27 = %h",uut.sccpu.cpu_ref.array_reg[27]);
            $fdisplay(file_output,"regfiles28 = %h",uut.sccpu.cpu_ref.array_reg[28]);
            $fdisplay(file_output,"regfiles29 = %h",uut.sccpu.cpu_ref.array_reg[29]);
            $fdisplay(file_output,"regfiles30 = %h",uut.sccpu.cpu_ref.array_reg[30]);
            $fdisplay(file_output,"regfiles31 = %h",uut.sccpu.cpu_ref.array_reg[31]);
            $fdisplay(file_output,"instr = %h",uut.inst);
            $fdisplay(file_output,"pc = %h",pc);
          end
        end
        end
endmodule
```

五、实验结果