同济大学计算机系

计算机组成原理实验报告

学　　号 **1951247**

姓　　名 钟伊凡

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

在本次实验中，完成 31 条指令 CPU 的设计、实现、调试、下板。主要步骤包括逐条指令建立数据通路、整理指令信号，并据此使用 Verilog HDL 语言进行实现，并参照 MARS 汇编器的输出结果进行调试，最终需要通过前仿真和后仿真。

二、数据通路和指令信号建模

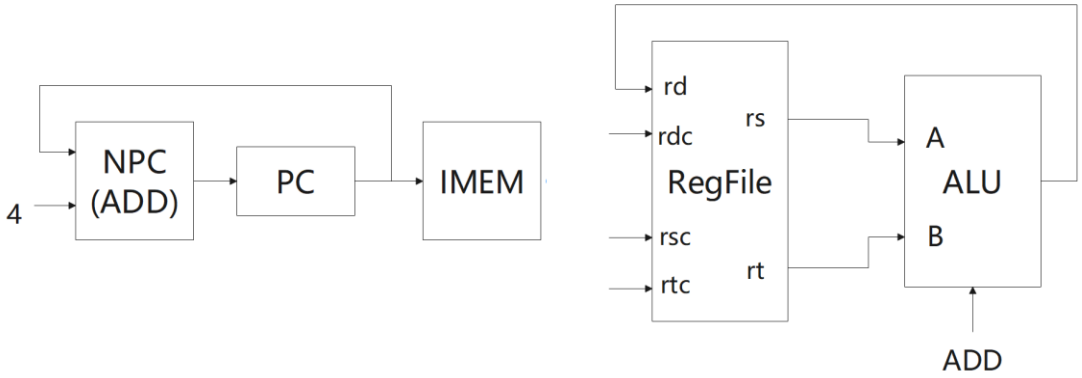
（一）数据通路设计

1. ADDU

格式：addu rd, rs, rt
描述：rd ← rs + rt
输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU	
				rd	A	B
ADDU	NPC	PC	PC	ALU	rs	rt

数据通路图：

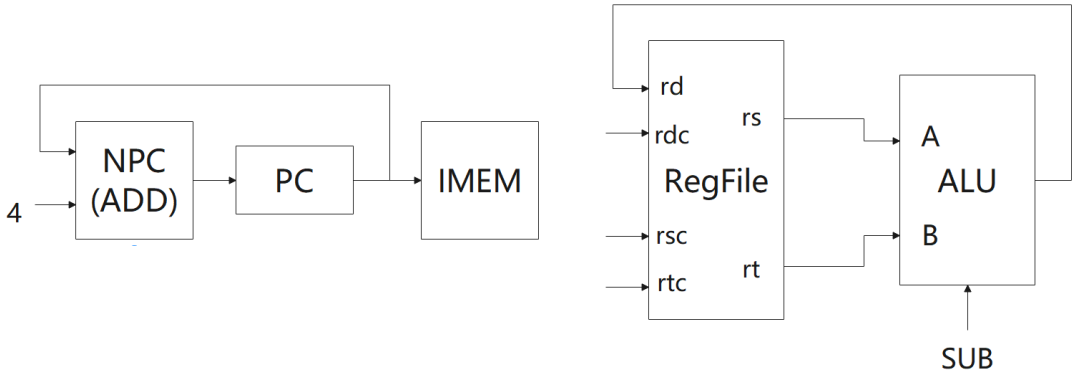


2. SUBU

格式：subu rd, rs, rt
描述：rd ← rs - rt
输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU	
				rd	A	B
SUBU	NPC	PC	PC	ALU	rs	rt

数据通路图：



3. ORI

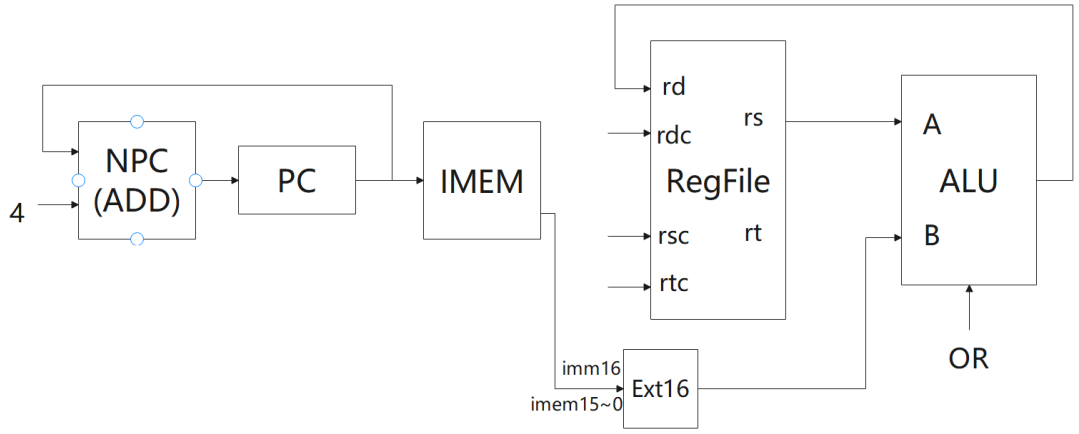
格式：ori rt, rs, imm16

描述：rt ← rs or imm16(zero_ext)

输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU		Ext5	Ext16
				rd	A	B		
ORI	NPC	PC	PC	ALU	rs	Ext16		imm16

数据通路图：



4. SLL

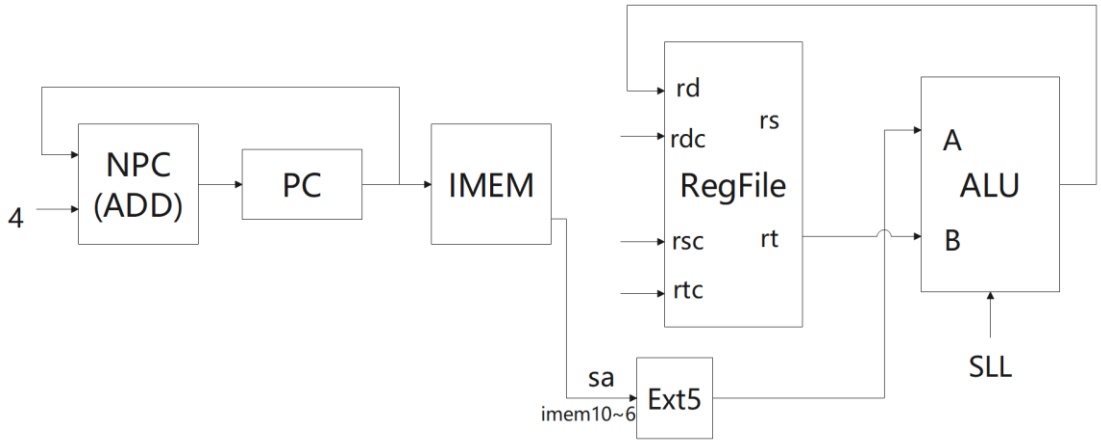
格式：sll rd, rt, sa

描述：rd ← rt << sa(sign_ext)

输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU		Ext5
				rd	A	B	
SLL	NPC	PC	PC	ALU	Ext5	rt	sa

数据通路图：



5. LW

格式：lw rt, offset(base)

31~26 LW(100011)

25~21 base

20~16 rt

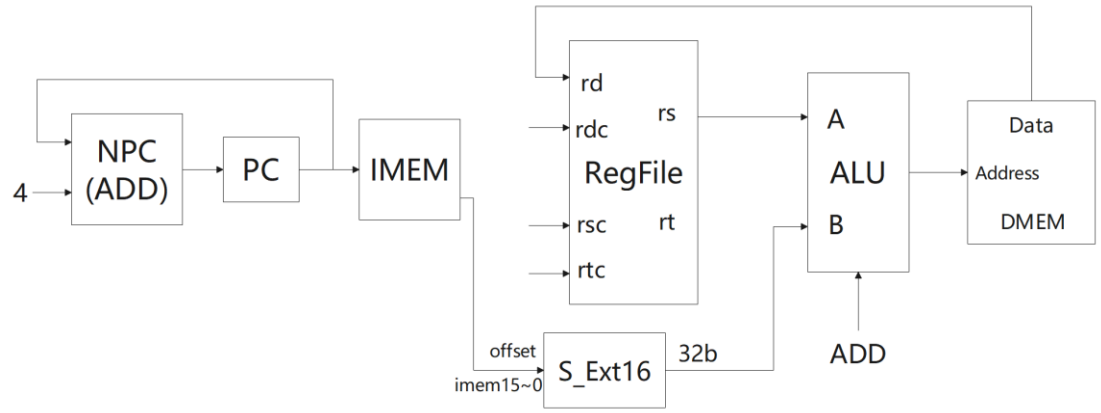
15~0 offset

描述：rt ← memory[offset + base]

输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU		Ext5	Ext16	DMEM		S_Ext16
				rd	A	B			Addr	Data	
LW	NPC	PC	PC	Data	rs	S_Ext16			ALU		offset

数据通路图：



6. SW

格式：sw rt, offset(base)

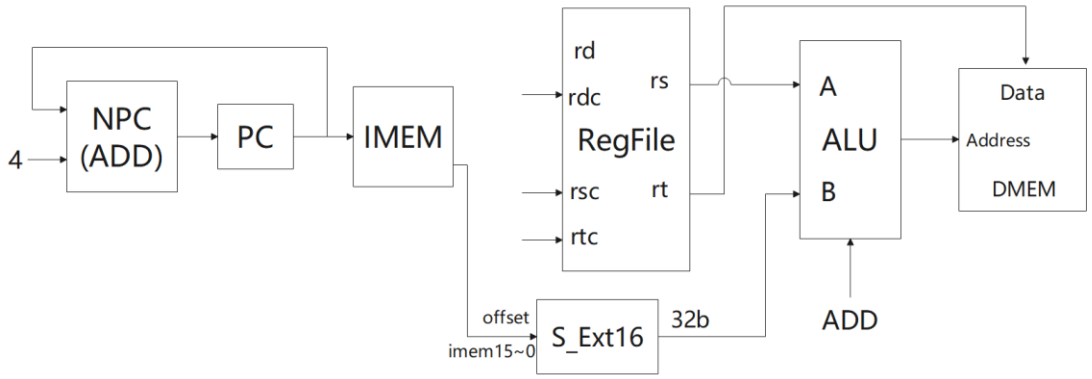
描述：memory[offset + base] ← rt

输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU		Ext5	Ext16	DMEM		S_Ext16
				rd	A	B			Addr	Data	

SW	NPC	PC	PC		rs	S_Ext16			ALU	rt	offset	

数据通路图：



7. BEQ

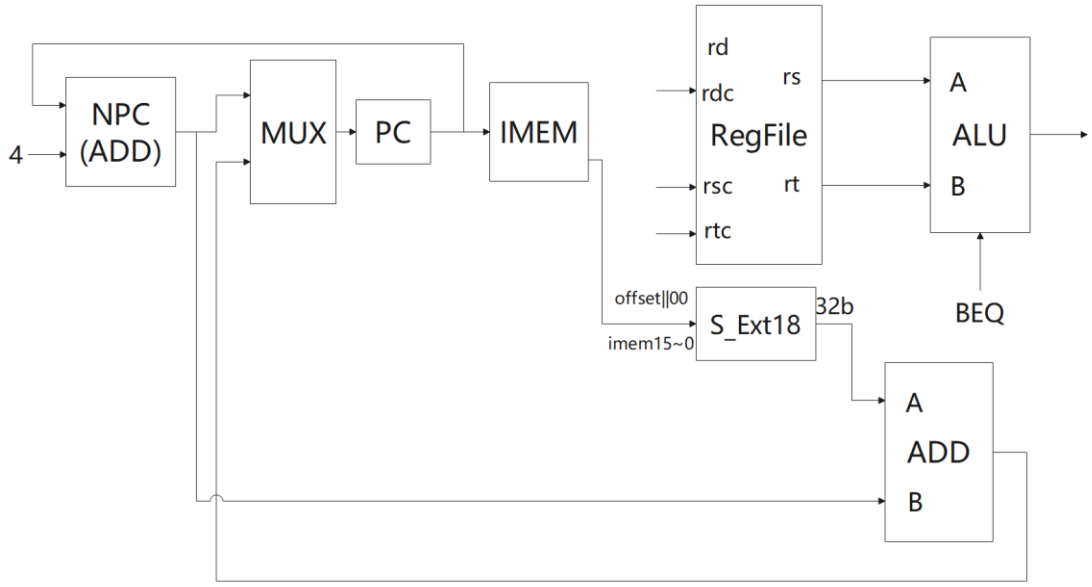
格式：beq rs, rt, offset

描述：if(rs != rt) pc ← pc + 4; else pc ← pc + sign_ext(offset || 00)

输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU		Ext5	Ext16	DMEM		S_Ext16	S_Ext18	ADD	
				rd	A	B			Addr	Data			A	B
ADDU	NPC	PC	PC	ALU	rs	rt								
SUBU	NPC	PC	PC	ALU	rs	rt								
ORI	NPC	PC	PC	ALU	rs	Ext16		imm16						
SLL	NPC	PC	PC	ALU	Ext5	rt	sa							
LW	NPC	PC	PC	Data	rs	S_Ext16			ALU		offset			
SW	NPC	PC	PC		rs	S_Ext16			ALU	rt	offset			
BEQ	ADD	PC	PC		rs	rt						offset	NPC	S_Ext18

数据通路图：



8. J

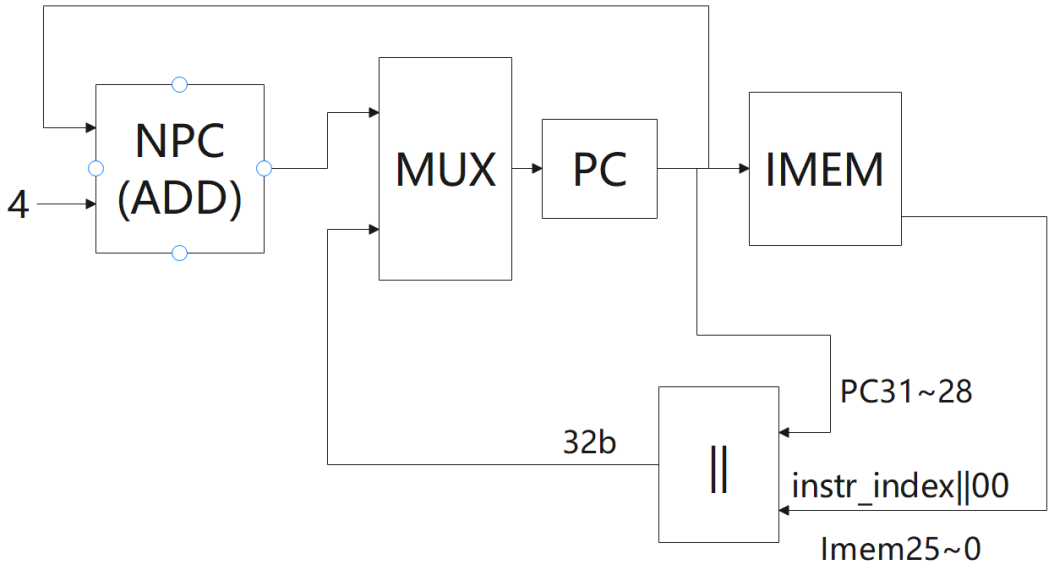
格式：J target

描述：PC ← {PC[31:28], instr_index, 00}

输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU		Ext5	Ext16	DMEM		S_Ext16	S_Ext18	ADD		||	
				rd	A	B			Addr	Data			A	B	A	B
ADDU	NPC	PC	PC	ALU	rs	rt										
SUBU	NPC	PC	PC	ALU	rs	rt										
ORI	NPC	PC	PC	ALU	rs	Ext16		imm16								
SLL	NPC	PC	PC	ALU	Ext5	rt	sa									
LW	NPC	PC	PC	Data	rs	S_Ext16			ALU		offset					
SW	NPC	PC	PC		rs	S_Ext16			ALU	rt	offset					
BEQ	ADD	PC	PC		rs	rt						offset	NPC	S_Ext18		
J	||	PC	PC												PC[31:28]	imem25~0

数据通路图：



9. ADD

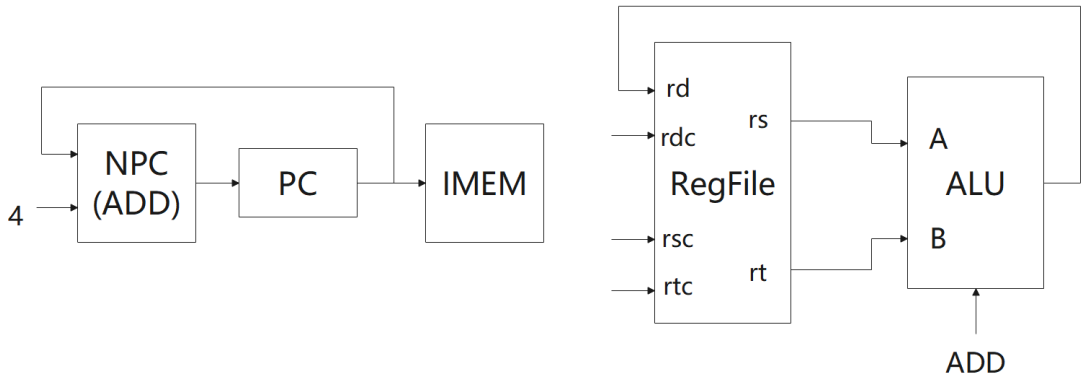
格式：add rd, rs, rt

描述：rd ← rs + rt

输入输出关系：

指令	PC	NPC	IMEM	RegFile	ALU		Ext5	Ext16	DMEM		S_Ext16	S_Ext18	ADD		||	
				rd	A	B			Addr	Data			A	B	A	B
ADDU	NPC	PC	PC	ALU	rs	rt										
SUBU	NPC	PC	PC	ALU	rs	rt										
ORI	NPC	PC	PC	ALU	rs	Ext16		imm16								
SLL	NPC	PC	PC	ALU	Ext5	rt	sa									
LW	NPC	PC	PC	Data	rs	S_Ext16			ALU		offset					
SW	NPC	PC	PC		rs	S_Ext16			ALU	rt	offset					
BEQ	ADD	PC	PC		rs	rt						offset	NPC	S_Ext18		
J	||	PC	PC												PC[31:28]	imem25~0
ADD	NPC	PC	PC	ALU	rs	rt										

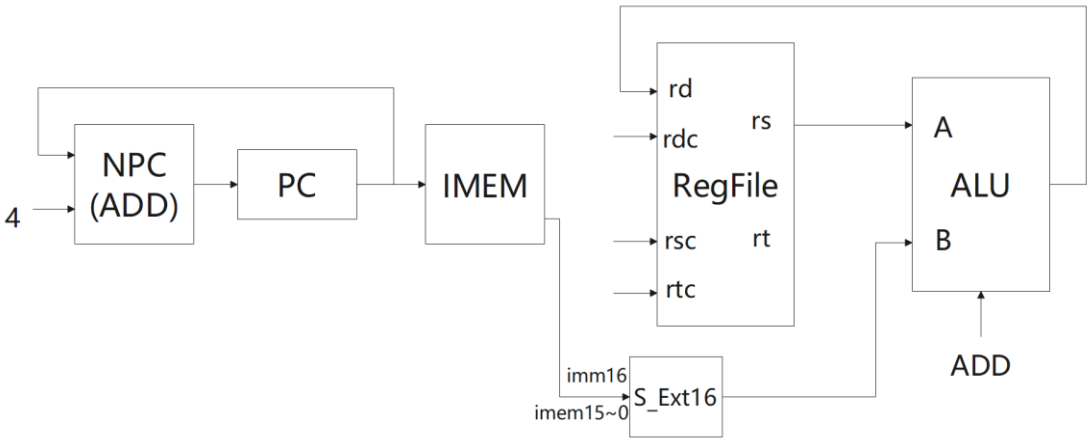
数据通路图：



【以下的输入输出关系在最后汇总一张总表】

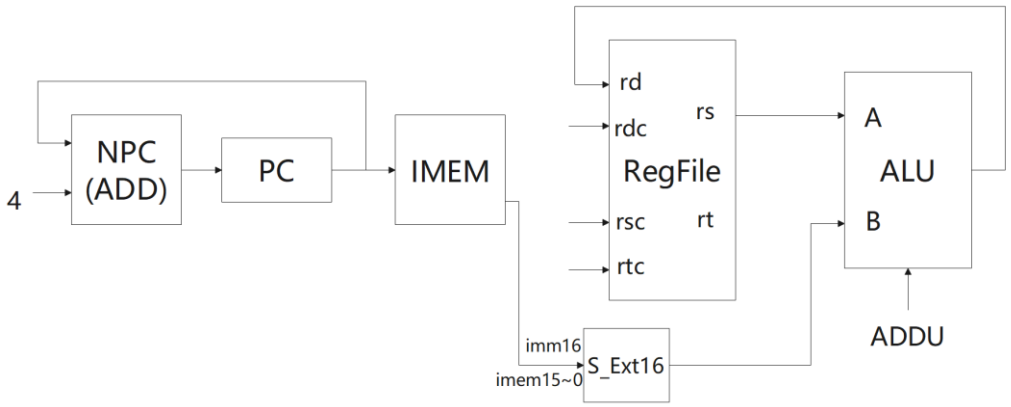
10.ADDI

格式：addi rt, rs, immediate
描述：rt ← rs + immediate
数据通路图：



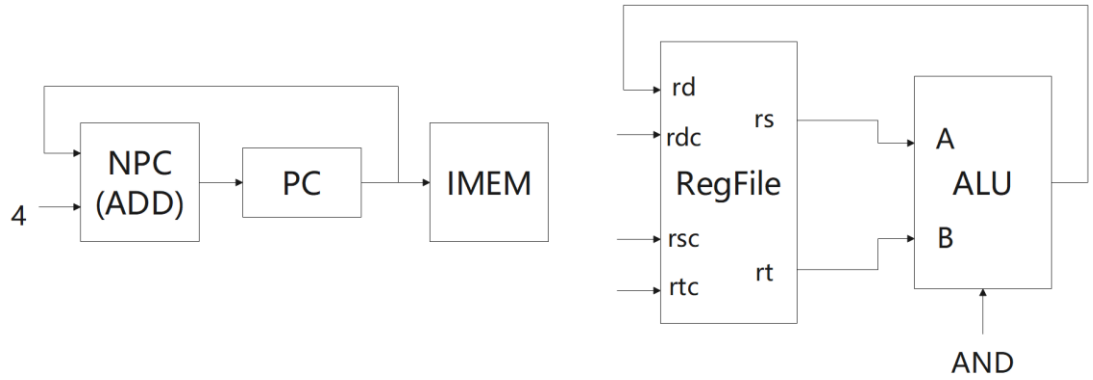
11.ADDIU

格式：addiu rt, rs, immediate
描述：rt ← rs + immediate
数据通路图：



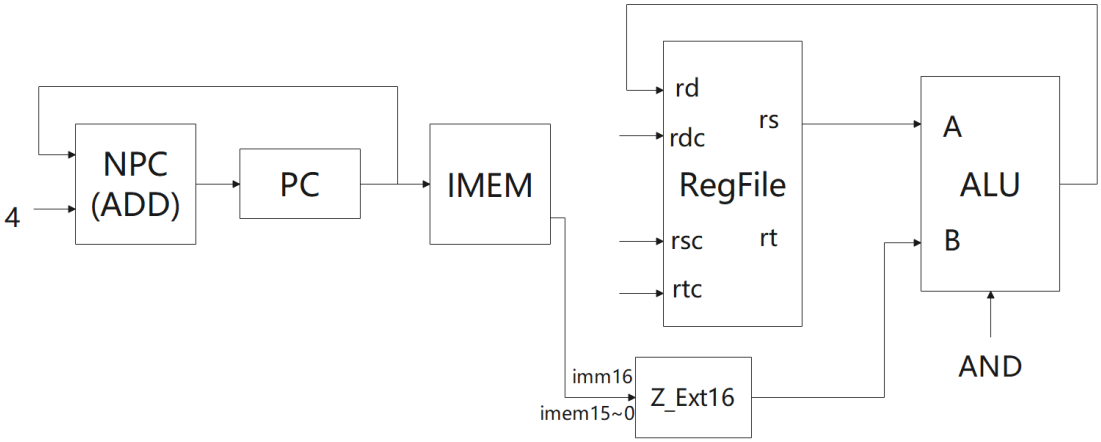
12.AND

格式：and rd, rs, rt
描述：rt ← rs AND rt
数据通路图：



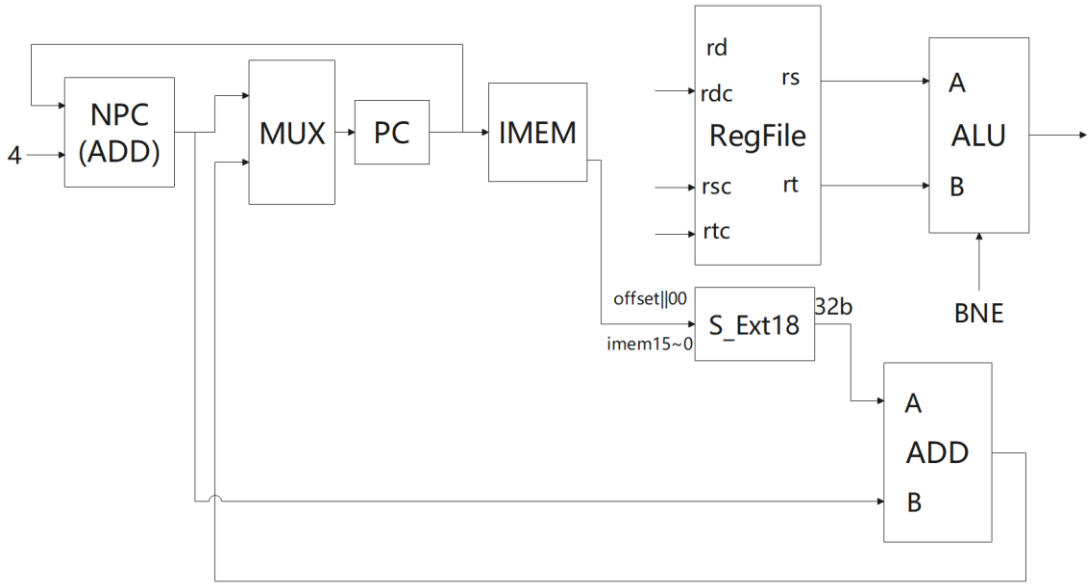
13.ANDI

格式：andi rt, rs, immediate
描述：rt ← rs AND immediate
数据通路图：



14.BNE

格式：bne rs, rt, offset
描述：**if(rs == rt) pc ← pc + 4; else pc ← pc + sign_ext(offset || 00)**
数据通路图：

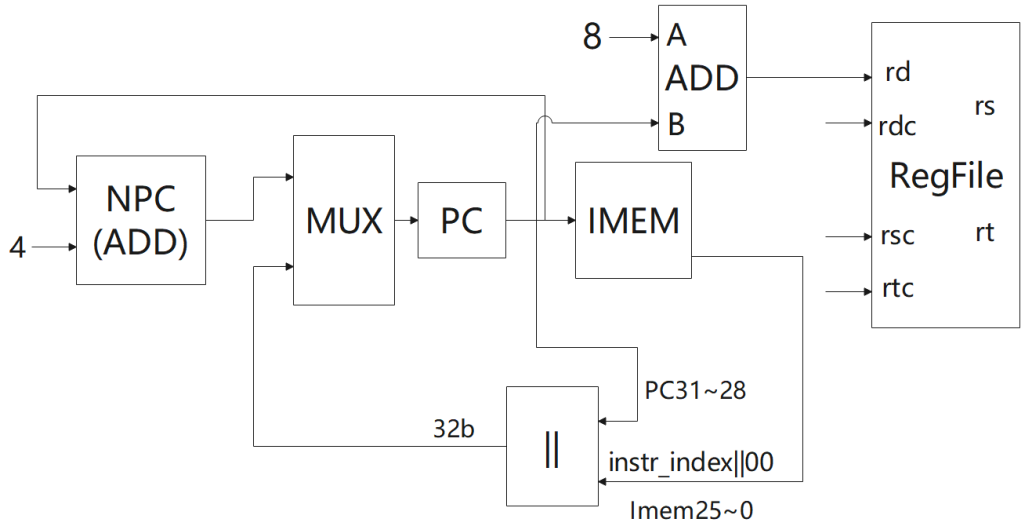


15.JAL

格式：jal target

描述：**PC ← {PC[31:28], instr_index, 00}; GPR[31] ← PC + 8**

数据通路图：

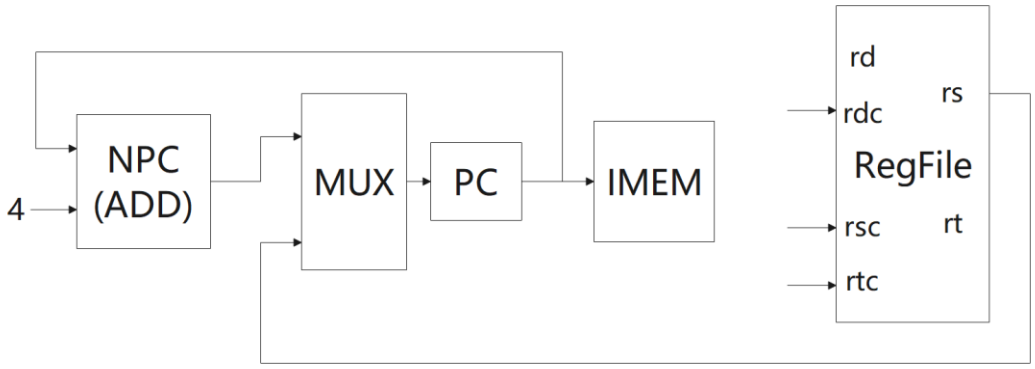


16.JR

格式：jr rs

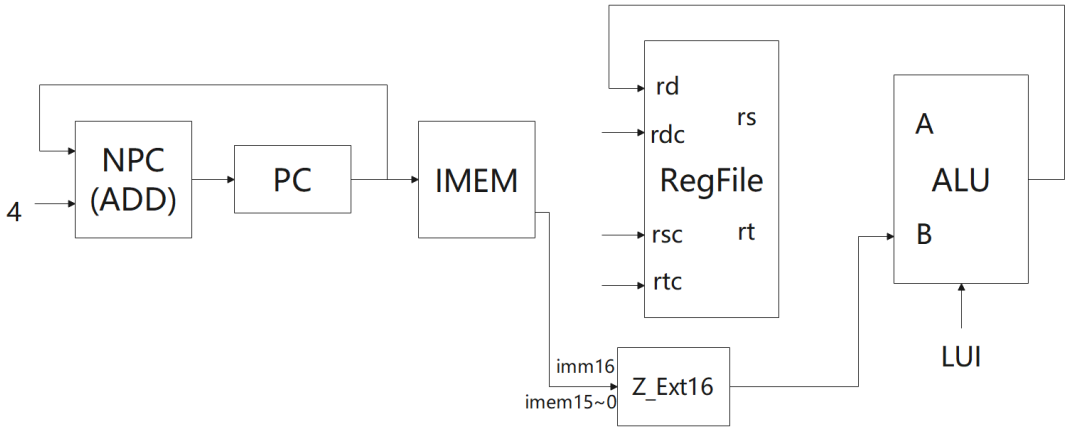
描述：PC ← RS

数据通路图：



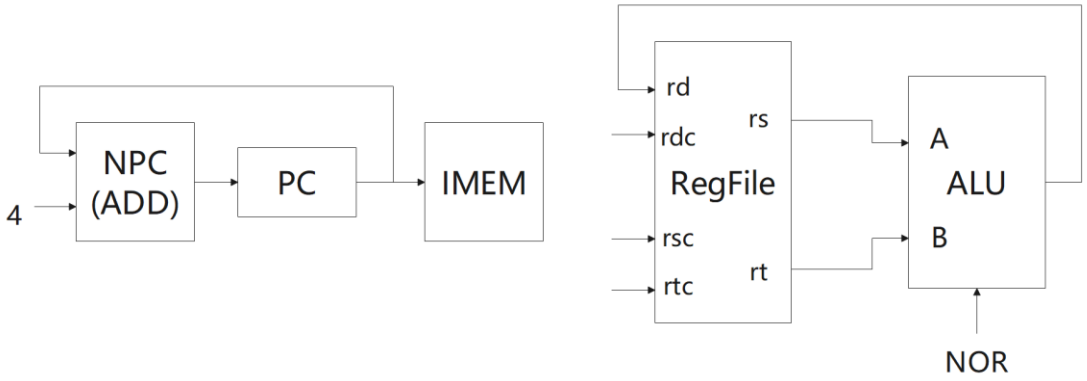
17.LUI

格式：LUI rt, immediate
描述：rt ← immediate || {16{1'b0}}
数据通路图：



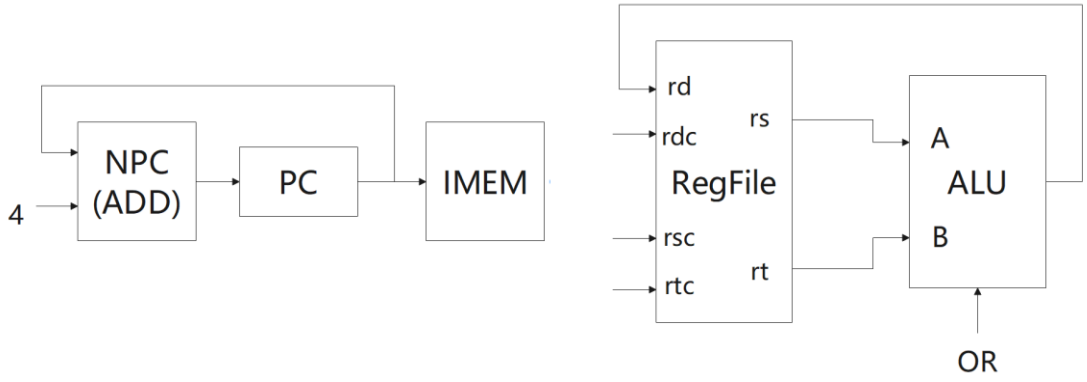
18.NOR

格式：nor rd, rs, rt
描述：rd ← rs NOR rt
数据通路图：



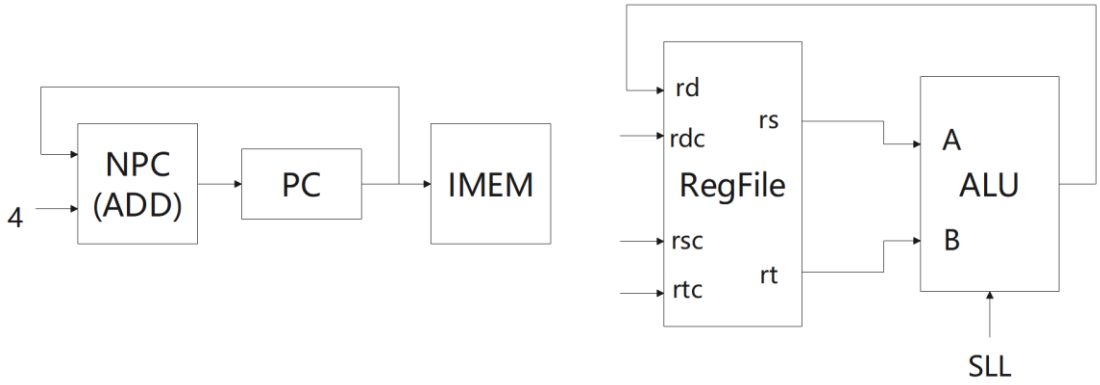
19.OR

格式：or rd, rs, rt
描述：rd ← rs OR rt
数据通路图：



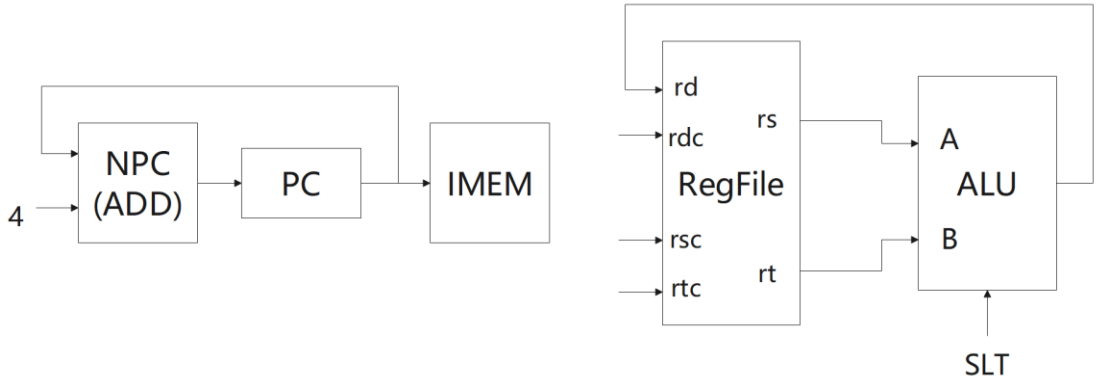
20.SLLV

格式：sllv rd, rs, rt
描述：rd ← rt << rs
数据通路图：



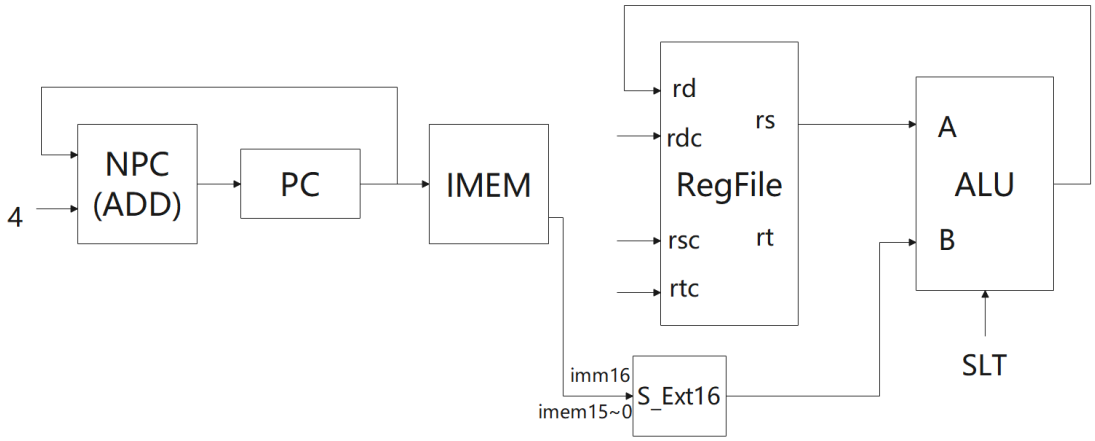
21.SLT

格式：slt rd, rs, rt
描述：rd ← (rs < rt)
数据通路图：



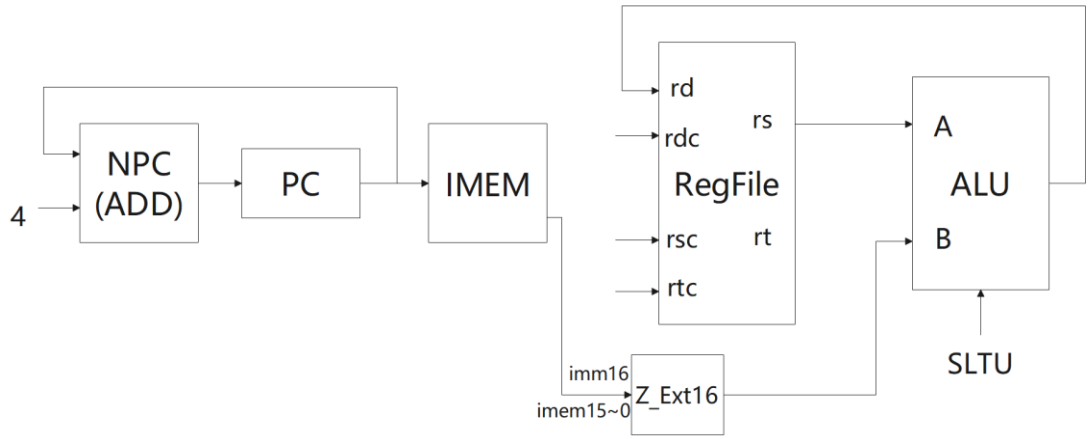
22.SLTI

格式：slti rt, rs, immediate
描述：rt ← (rs < immediate)
数据通路图：



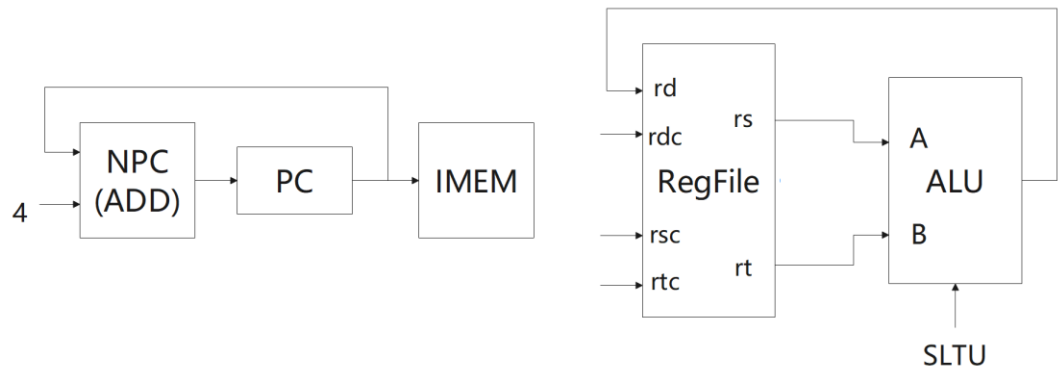
23.SLTIU

格式：sltiu rt, rs, immediate
描述：rt ← (rs < immediate)
数据通路图：



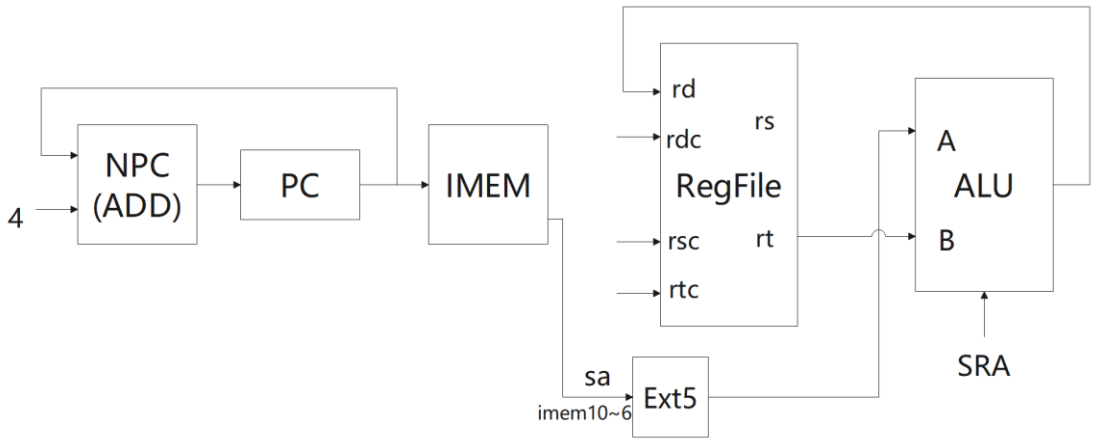
24.SLTU

格式：sltu rd, rs, rt
描述：rd ← (rs < rt)
数据通路图：



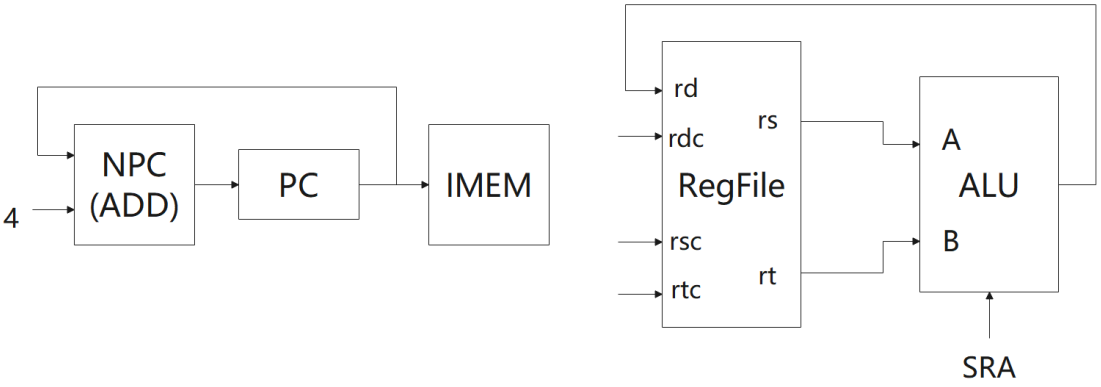
25.SRA

格式：sra rd, rt, sa
描述：rd ← rt >> sa (Arithmetic)
数据通路图：



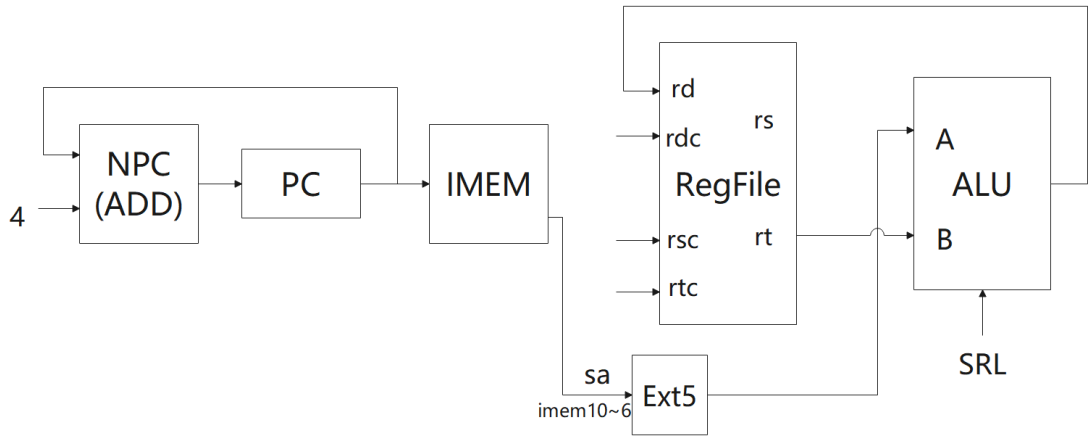
26.SRAV

格式：srav rd, rt, rs
描述：rd ← rt >> rs (arithmetic)
数据通路图：



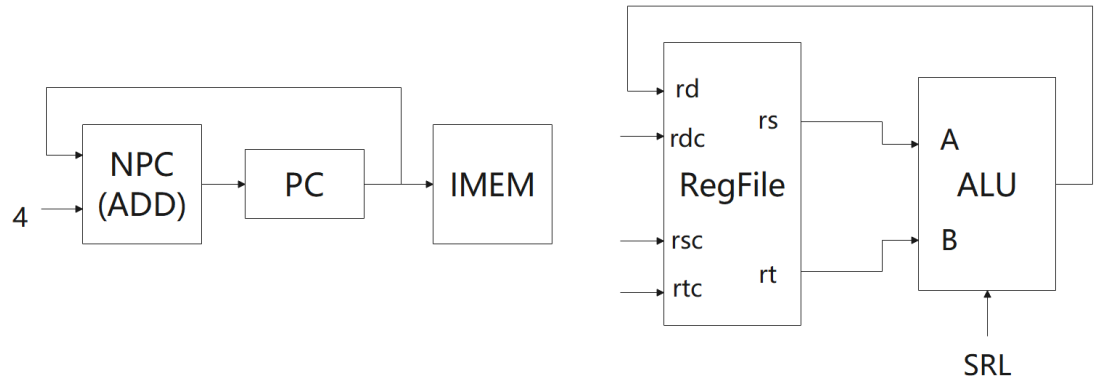
27.SRL

格式：srl rd, rt, sa
描述：rd ← rt >> sa (Arithmetic)
数据通路图：



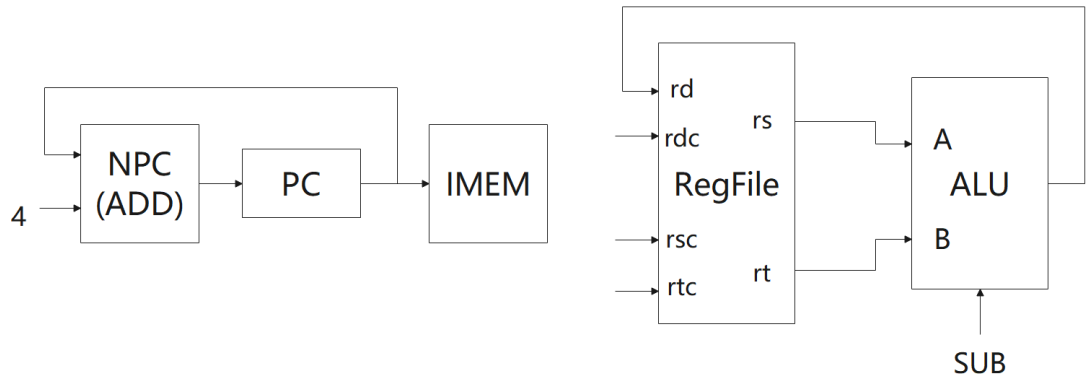
28.SRLV

格式：srlv rd, rt, rs
描述：rd ← rt >> rs (arithmetic)
数据通路图：



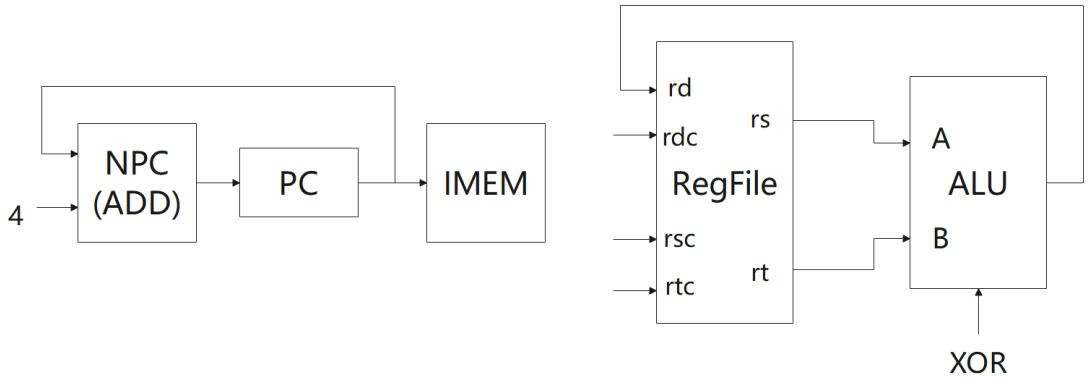
29.SUB

格式：sub rd, rs, rt
描述：rd ← rs - rt
数据通路图：



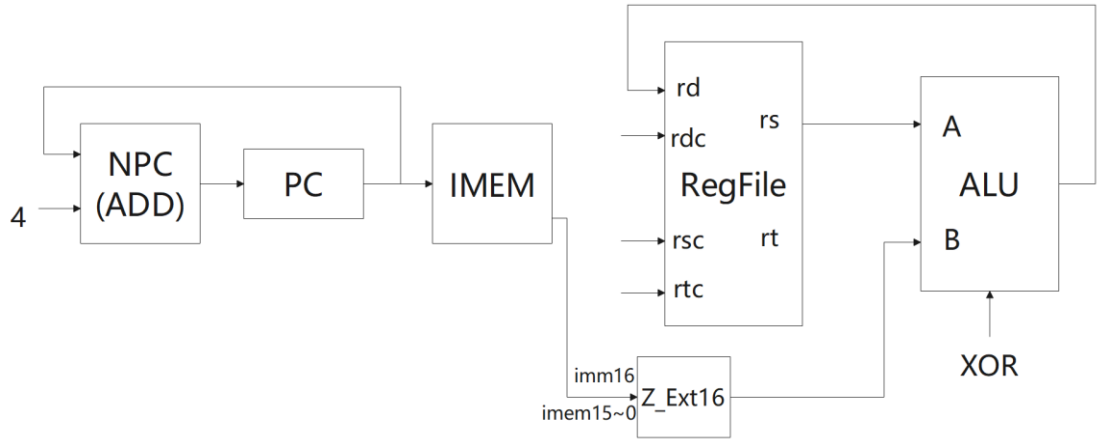
30.XOR

格式：xor rd, rs, rt
描述：rd ← rs XOR rt
数据通路图：



31.XORI

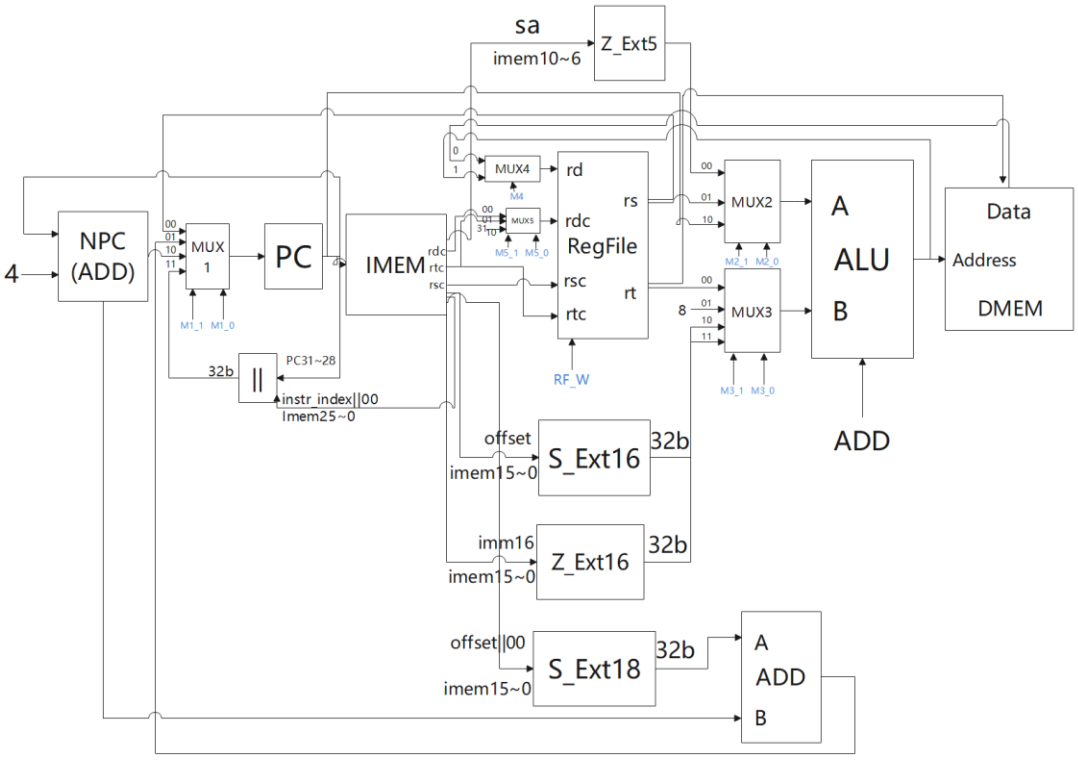
格式：xori rt, rs, immediate
描述：rt ← rs XOR immediate
数据通路图：



（二）信号输入输出关系总表

指令	PC	NPC	IMEM	RegFile	ALU		Z_Ext5	Z_Ext16	DMEM		S_Ext16	S_Ext18	ADD		||	
				rd	A	B			Addr	Data			A	B	A	B
ADDU	NPC	PC	PC	ALU	rs	rt										
SUBU	NPC	PC	PC	ALU	rs	rt										
ORI	NPC	PC	PC	ALU	rs	Z_Ext16		imm16								
SLL	NPC	PC	PC	ALU	Z_Ext5	rt	sa									
LW	NPC	PC	PC	Data	rs	S_Ext16			ALU		offset					
SW	NPC	PC	PC		rs	S_Ext16			ALU	rt	offset					
BEQ	ADD	PC	PC		rs	rt						offset	NPC	S_Ext18		
J	||	PC	PC												PC[31:28]	imem25~0||00
ADD	NPC	PC	PC	ALU	rs	rt										
ADDI	NPC	PC	PC	ALU	rs	S_Ext16					imm16					
ADDIU	NPC	PC	PC	ALU	rs	S_Ext16					imm16					
AND	NPC	PC	PC	ALU	rs	rt										
ANDI	NPC	PC	PC	ALU	rs	Z_Ext16		imm16								
BNE	ADD	PC	PC		rs	rt						offset	NPC	S_Ext18		
JAL	||	PC	PC	ALU	PC	8									PC[31:28]	imem25~0||00
JR	RegFile	PC	PC													
LUI	NPC	PC	PC	ALU		Z_Ext16		imm16								
NOR	NPC	PC	PC	ALU	rs	rt										
OR	NPC	PC	PC	ALU	rs	rt										
SLLV	NPC	PC	PC	ALU	rs	rt										
SLT	NPC	PC	PC	ALU	rs	rt										
SLTI	NPC	PC	PC	ALU	rs	S_Ext16					imm16					
SLTIU	NPC	PC	PC	ALU	rs	Z_Ext16		imm16								
SLTU	NPC	PC	PC	ALU	rs	rt										
SRA	NPC	PC	PC	ALU	Z_Ext5	rt	sa									
SRAV	NPC	PC	PC	ALU	rs	rt										
SRL	NPC	PC	PC	ALU	Z_Ext5	rt	sa									
SRLV	NPC	PC	PC	ALU	rs	rt										
SUB	NPC	PC	PC	ALU	rs	rt										
XOR	NPC	PC	PC	ALU	rs	rt										
XORI	NPC	PC	PC	ALU	rs	Z_Ext16		imm16								
附注：标红的可以直接使用ALU实现；																

（三）数据通路总图



（四）指令信号总表

指令	PC_CLK	IM_R	aluc3	aluc2	aluc1	aluc0	M1_1	M1_0	M2_1	M2_0	M3_1	M3_0	M4	M5_1	M5_0	RF_W	RF_CLK	CS	DM_R	DM_W
ADDU	1	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0
SUBU	1	1	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0
ORI	1	1	0	1	0	1	1	0	0	1	1	1	1	0	1	1	1	0	0	0
SLL	1	1	1	1	1	x	1	0	0	0	0	0	1	0	0	1	1	0	0	0
LW	1	1	0	0	1	0	1	0	0	1	1	0	0	0	1	1	1	1	1	0
SW	1	1	0	0	1	0	1	0	0	1	1	0				0	1	1	0	1
BEQ, z=1	1	1	0	0	0	1	0	1	0	1	0	0				0	1	0	0	0
BEQ, z=0	1	1	0	0	0	1	1	0	0	1	0	0				0	1	0	0	0
J	1	1					1	1								0	1	0	0	0
ADD	1	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0
ADDI	1	1	0	0	1	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0
ADDIU	1	1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0
AND	1	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0
ANDI	1	1	0	1	0	0	1	0	0	1	1	1	1	0	1	1	1	0	0	0
BNE, z=0	1	1	0	0	0	1	0	1	0	1	0	0				0	1	0	0	0
BNE, z=1	1	1	0	0	0	1	1	0	0	1	0	0				0	1	0	0	0
JAL	1	1					1	1	1	0	0	1	1	1	0	1	1	0	0	0
JR	1	1					0	0								0	1	0	0	0
LUI	1	1	1	0	0	x	1	0			1	1	1	0	1	1	1	0	0	0
NOR	1	1	0	1	1	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0
OR	1	1	0	1	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0
SLLV	1	1	1	1	1	x	1	0	0	1	0	0	1	0	0	1	1	0	0	0
SLT	1	1	1	0	1	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0
SLTI	1	1	1	0	1	1	1	0	0	1	1	0	1	0	1	1	1	0	0	0
SLTIU	1	1	1	0	1	0	1	0	0	1	1	1	1	0	1	1	1	0	0	0
SLTU	1	1	1	0	1	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0
SRA	1	1	1	1	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0
SRAV	1	1	1	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0
SRL	1	1	1	1	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0
SRLV	1	1	1	1	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0
SUB	1	1	0	0	1	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0
XOR	1	1	0	1	1	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0
XORI	1	1	0	1	1	0	1	0	0	1	1	1	1	0	1	1	1	0	0	0

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）

[sccomp_dataflow.v]

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

wire cs, DM_R, DM_W;
wire [31:0] DM_data_in;
wire [31:0] DM_data_out;
wire [31:0] pc_imem;
wire [10:0] DM_addr_cpu;
wire [10:0] DM_addr_DM;

assign pc_imem = (pc - 32'h0040_0000) / 4;
assign DM_addr_DM = (DM_addr_cpu - 32'h1001_0000) / 4;

imem imem_inst(.a(pc_imem), .spo(inst));
```

```
dmem dmem_inst(.clk(clk_in), .cs(cs), .DM_R(DM_R), .DM_W(DM_W), .DM_addr(DM_addr_DM), .DM_data_in(DM_data_in), .DM_data_out(DM_data_out));

cpu sccpu(.clk(clk_in), .rst(reset), .inst(inst), .DM_data_out(DM_data_out), .pc(pc), .DM_cs(cs), .DM_R(DM_R), .DM_W(DM_W), .DM_addr(DM_addr_cpu), .DM_data_in(DM_data_in));

endmodule
```

[dmem.v]

```
module dmem (
    input clk,
    input cs,
    input DM_R,
    input DM_W,
    input [10:0] DM_addr,
    input [31:0] DM_data_in,
    output [31:0] DM_data_out
);

reg [31:0] mem [0:127];
always @ (posedge clk)
begin
    if(cs && DM_W)
    begin
        mem[DM_addr] = DM_data_in;
    end
end

assign DM_data_out = (cs && DM_R) ? mem[DM_addr] : {32{1'bz}};

endmodule
```

[imem.v]

```
module imem (
    input [10:0] addr,
    output [31:0] inst
);
reg [31:0] mem [2047:0];
initial begin
    $readmemh("D:/Undergraduate/0 大二下课程/计组/实验/CPU/资料/31 条指令CPU 实验指导书和网站提交说明/网上提交测试所用coe/mips_31_mars_simulate.txt", mem);
```

```
end
assign inst = mem[addr];
endmodule
```

[cpu.v]

```
module cpu (
    input clk,
    input rst,
    input [31:0] inst,
    input [31:0] DM_data_out,
    output [31:0] pc,
    output DM_cs,
    output DM_R,
    output DM_W,
    output [10:0] DM_addr,
    output [31:0] DM_data_in
);

wire [3:0] aluc;
wire [1:0] M1;
wire [1:0] M2;
wire [1:0] M3;
wire M4;
wire [1:0] M5;
wire RF_W;
wire [31:0] npc;
wire [31:0] newpc;
wire [5:0] op;
wire [4:0] rsc;
wire [4:0] rtc;
wire [4:0] im_rdc;
wire [4:0] sa;
wire [5:0] func;
wire [4:0] ref_rdc;
wire [31:0] ref_rd;
wire [31:0] alu_a;
wire [31:0] alu_b;
wire [31:0] alu_out;
wire [31:0] rs;
wire [31:0] rt;
wire [31:0] add_out;
wire [31:0] concat;
wire [31:0] zext5;
wire [31:0] sext16;
```

```
wire [31:0] zext16;
wire [31:0] sext18;
wire [31:0] M1_temp1;
wire [31:0] M1_temp2;
wire [31:0] M2_temp;
wire [31:0] M3_temp1;
wire [31:0] M3_temp2;
wire zero, carry, negative, overflow;

assign npc = pc + 4;
assign op = inst[31:26];
assign rsc = inst[25:21];
assign rtc = inst[20:16];
assign im_rdc = inst[15:11];
assign sa = inst[10:6];
assign func = inst[5:0];
assign concat = {pc[31:28], inst[25:0], 2'b00};
assign zext5 = {27'b0, sa};
assign sext16 = {{16{inst[15]}}, inst[15:0]};
assign zext16 = {16'b0, inst[15:0]};
assign sext18 = {{14{inst[15]}}, inst[15:0], 2'b0};
assign add_out = sext18 + npc;

wire _add_, _addu_, _sub_, _subu_, _and_, _or_, _xor_, _nor_, _slt_, _s
ltu_, _sll_, _srl_, _sra_, _sllv_, _srlv_, _srav_, _jr_;
wire _addi_, _addiu_, _andi_, _ori_, _xori_, _lw_, _sw_, _beq_, _bne_,
_slti_, _sltiu_, _lui_, _j_, _jal_;

assign _add_ = (op == 6'b000000 && func == 6'b100000) ? 1'b1 : 1'b0;
assign _addu_ = (op == 6'b000000 && func == 6'b100001) ? 1'b1 : 1'b0;
assign _sub_ = (op == 6'b000000 && func == 6'b100010) ? 1'b1 : 1'b0;
assign _subu_ = (op == 6'b000000 && func == 6'b100011) ? 1'b1 : 1'b0;
assign _and_ = (op == 6'b000000 && func == 6'b100100) ? 1'b1 : 1'b0;
assign _or_ = (op == 6'b000000 && func == 6'b100101) ? 1'b1 : 1'b0;
assign _xor_ = (op == 6'b000000 && func == 6'b100110) ? 1'b1 : 1'b0;
assign _nor_ = (op == 6'b000000 && func == 6'b100111) ? 1'b1 : 1'b0;
assign _slt_ = (op == 6'b000000 && func == 6'b101010) ? 1'b1 : 1'b0;
assign _sltu_ = (op == 6'b000000 && func == 6'b101011) ? 1'b1 : 1'b0;
assign _sll_ = (op == 6'b000000 && func == 6'b000000) ? 1'b1 : 1'b0;
assign _srl_ = (op == 6'b000000 && func == 6'b000010) ? 1'b1 : 1'b0;
assign _sra_ = (op == 6'b000000 && func == 6'b000011) ? 1'b1 : 1'b0;
assign _sllv_ = (op == 6'b000000 && func == 6'b000100) ? 1'b1 : 1'b0;
assign _srlv_ = (op == 6'b000000 && func == 6'b000110) ? 1'b1 : 1'b0;
assign _srav_ = (op == 6'b000000 && func == 6'b000111) ? 1'b1 : 1'b0;
```

```
assign _jr_ = (op == 6'b000000 && func == 6'b001000) ? 1'b1 : 1'b0;
assign _addi_ = (op == 6'b001000) ? 1'b1 : 1'b0;
assign _addiu_ = (op == 6'b001001) ? 1'b1 : 1'b0;
assign _andi_ = (op == 6'b001100) ? 1'b1 : 1'b0;
assign _ori_ = (op == 6'b001101) ? 1'b1 : 1'b0;
assign _xori_ = (op == 6'b001110) ? 1'b1 : 1'b0;
assign _lw_ = (op == 6'b100011) ? 1'b1 : 1'b0;
assign _sw_ = (op == 6'b101011) ? 1'b1 : 1'b0;
assign _beq_ = (op == 6'b000100) ? 1'b1 : 1'b0;
assign _bne_ = (op == 6'b000101) ? 1'b1 : 1'b0;
assign _slti_ = (op == 6'b001010) ? 1'b1 : 1'b0;
assign _sltiu_ = (op == 6'b001011) ? 1'b1 : 1'b0;
assign _lui_ = (op == 6'b001111) ? 1'b1 : 1'b0;
assign _j_ = (op == 6'b000010) ? 1'b1 : 1'b0;
assign _jal_ = (op == 6'b000011) ? 1'b1 : 1'b0;

assign aluc[3] = _sll_ || _lui_ || _sllv_ || _slt_ || _slti_ || _sltiu_
 || _sltu_ || _sra_ || _srav_ || _srl_ || _srlv_;

assign aluc[2] = _ori_ || _sll_ || _and_ || _andi_ || _nor_ || _or_ ||
_sllv_ || _sra_ || _srav_ || _srl_ || _srlv_ || _xor_ || _xori_;

assign aluc[1] = _sll_ || _add_ || _addi_ || _nor_ || _sllv_ || _slt_ |
| _slti_ || _sltiu_ || _sltu_ || _sub_ || _xor_ || _xori_ || _sw_ || _l
w_;

assign aluc[0] = _subu_ || _ori_ || _nor_ || _or_ || _slt_ || _slti_ ||
 _srl_ || _srlv_ || _sub_ || _beq_ || _bne_;

assign M1[1] = _add_ || _addu_ || _sub_ || _subu_ || _and_ || _or_ || _
xor_ || _nor_ || _slt_ || _sltu_ || _sll_ || _srl_ || _sra_ || _sllv_ |
| _srlv_ || _srav_ || _addi_ || _addiu_ || _andi_ || _ori_ || _xori_ ||
 _lw_ || _sw_ || (_beq_ == 1 && zero == 0) || (_bne_ == 1 && zero == 1)
 || _slti_ || _sltiu_ || _lui_ || _j_ || _jal_;

assign M1[0] = (_beq_ == 1 && zero == 1) || (_j_ == 1) || (_bne_ == 1 &
& zero == 0) || (_jal_ == 1);

assign M2[1] = _jal_;

assign M2[0] = _add_ || _addu_ || _sub_ || _subu_ || _and_ || _or_ || _
xor_ || _nor_ || _slt_ || _sltu_ || _addi_ || _addiu_ || _andi_ || _ori
_ || _xori_ || _lw_ || _sw_ || _beq_ || _bne_ || _slti_ || _sltiu_ || _
sllv_ || _srav_ || _srlv_;
```

```
assign M3[1] = _ori_ || _lw_ || _sw_ || _addi_ || _addiu_ || _andi_ ||
_lui_ || _slti_ || _sltiu_ || _xori_;

assign M3[0] = _ori_ || _andi_ || _jal_ || _lui_ || _xori_ || _sltiu_;

assign M4 = _add_ || _addu_ || _sub_ || _subu_ || _and_ || _or_ || _xor
_ || _nor_ || _slt_ || _sltu_ || _sll_ || _srl_ || _sra_ || _sllv_ || _
srlv_ || _srav_ || _addi_ || _addiu_ || _andi_ || _ori_ || _xori_ || _s
lti_ || _sltiu_ || _lui_ || _jal_;

assign M5[1] = _jal_;

assign M5[0] = _ori_ || _lw_ || _addi_ || _addiu_ || _andi_ || _lui_ ||
 _slti_ || _sltiu_ || _xori_;

assign RF_W = _add_ || _addu_ || _sub_ || _subu_ || _and_ || _or_ || _x
or_ || _nor_ || _slt_ || _sltu_ || _sll_ || _srl_ || _sra_ || _sllv_ ||
 _srlv_ || _srav_ || _addi_ || _addiu_ || _andi_ || _ori_ || _xori_ ||
_slti_ || _sltiu_ || _lui_ || _jal_ || _lw_;

assign DM_cs = _lw_ || _sw_;

assign DM_R = _lw_;

assign DM_W = _sw_;

assign M1_temp1 = M1[0] ? add_out : rs;
assign M1_temp2 = M1[0] ? concat : npc;
assign newpc = M1[1] ? M1_temp2 : M1_temp1;

assign M2_temp = M2[1] ? pc : zext5;
assign alu_a = M2[0] ? rs : M2_temp;

assign M3_temp1 = M3[0] ? zext16 : sext16;
assign M3_temp2 = M3[0] ? 4 : rt;
assign alu_b = M3[1] ? M3_temp1 : M3_temp2;

assign ref_rd = M4 ? alu_out : DM_data_out;

assign ref_rdc = (M5 == 2'b00) ? im_rdc : (M5 == 2'b01) ? rtc : 31;

assign DM_addr = alu_out;
```

```
assign DM_data_in = rt;

alu ALU(.a(alu_a), .b(alu_b), .aluc(aluc), .r(alu_out), .zero(zero), .c
arry(carry), .negative(negative), .overflow(overflow));

regfile cpu_ref(.clk(clk), .rst(rst), .RF_W(RF_W), .rd(ref_rd), .rdc(re
f_rdc), .rtc(rtc), .rsc(rsc), .rs(rs), .rt(rt));

pcreg PCREG(.clk(clk), .rst(rst), .newpc(newpc), .pc(pc));

endmodule
```

[pcreg.v]

```
module pcreg (
    input clk,
    input rst,
    input [31:0] newpc,
    output reg [31:0] pc
);

always @ (negedge clk or posedge rst)
begin
    if(rst)
        pc = 32'h0040_0000;
    else
        pc = newpc;
end

endmodule
```

[alu.v]

```
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] temp;
always@(*)
begin
```

```
    case(aluc)
        4'b0000:
            begin
                temp = {1'b0, a} + {1'b0, b};
                r = temp[31:0];
                zero = (r == 0) ? 1 : 0;
                carry = temp[32];
                negative = r[31];
            end
        4'b0001:
            begin
                carry = (a < b) ? 1 : 0;
                r = a - b;
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
        4'b0010:
            begin
                r = a + b;
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
                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
        4'b0011:
            begin
                r = a - b;
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
                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
        4'b0100:
            begin
                r = a & b;
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
```

```
        4'b0101:
            begin
                r = a | b;
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
        4'b0110:
            begin
                r = a ^ b;
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
        4'b0111:
            begin
                r = ~(a | b);
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
        4'b1000:
            begin
                r = {b[15:0], 16'b0};
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
        4'b1001:
            begin
                r = {b[15:0], 16'b0};
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
        4'b1010:
            begin
                r = (a < b) ? 1 : 0;
                zero = (a == b) ? 1 : 0;
                negative = r[31];
                carry = (a < b) ? 1 : 0;
            end
        4'b1011:
            begin
                if((a[31] == 1 && b[31] == 0) || (a[31] == 1 && b[31] =
= 1 && a < b) || (a[31] == 0 && b[31] == 0 && a < b))
                    r = 1;
                else
                    r = 0;
```

```
                zero = (a == b) ? 1 : 0;
                negative = (r == 1) ? 1 : 0;
            end
        4'b1100:
            begin
                r = ($signed(b)) >>> a;
                if(a > 32)
                    carry = b[31];
                else if(a == 0)
                    carry = 0;
                else
                    carry = b[a - 1];
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
        4'b1101:
            begin
                r = b >> a;
                if(a > 32)
                    carry = 0;
                else if(a == 0)
                    carry = 0;
                else
                    carry = b[a - 1];
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
            end
        4'b1110:
            begin
                r = b << a;
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
                if(a == 0)
                    carry = 0;
                else if(a > 32)
                    carry = 0;
                else
                    carry = b[32 - a];
            end
        4'b1111:
            begin
                r = b << a;
                zero = (r == 0) ? 1 : 0;
                negative = r[31];
```

```
                if(a == 0)
                    carry = 0;
                else if(a > 32)
                    carry = 0;
                else
                    carry = b[32 - a];
            end
    endcase
end
endmodule
```

[regfile.v]

```
module regfile (
    input clk,
    input rst,
    input RF_W,
    input [31:0] rd,
    input [4:0] rdc,
    input [4:0] rtc,
    input [4:0] rsc,
    output [31:0] rs,
    output [31:0] rt
);
reg [31:0] array_reg[31:0];

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(RF_W && rdc != 0)
        begin
            array_reg[rdc] <= rd;
            array_reg[0] <= 32'b0;
        end
    end
end

assign rs = array_reg[rsc];
assign rt = array_reg[rtc];

endmodule
```

四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

[cpu_tb.v]

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

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

```
integer file_output;

initial begin
    file_output = $fopen("D:/Undergraduate/0 大二下课程/计组/实验
/CPU/tools/mars/result_mine.txt");
    clk = 0;
    rst = 1;
    #225 rst = 0;
end

always
begin
    #50 clk = ~clk;
end

always@(posedge clk)
begin
    if(rst == 0)
    begin
        $fdisplay(file_output, "pc: %h", pc);
        $fdisplay(file_output, "instr: %h", inst);
        // $fdisplay(file_output, "dmem0: %h", uut.dmem_inst.mem[0]);
        // $fdisplay(file_output, "dmem1: %h", uut.dmem_inst.mem[1]);
        // $fdisplay(file_output, "dmem2: %h", uut.dmem_inst.mem[2]);
        // $fdisplay(file_output, "dmem3: %h", uut.dmem_inst.mem[3]);
        $fdisplay(file_output, "regfile0: %h", uut.sccpu.cpu_ref.array_
reg[0]);
        $fdisplay(file_output, "regfile1: %h", uut.sccpu.cpu_ref.array_
reg[1]);
        $fdisplay(file_output, "regfile2: %h", uut.sccpu.cpu_ref.array_
reg[2]);
        $fdisplay(file_output, "regfile3: %h", uut.sccpu.cpu_ref.array_
reg[3]);
        $fdisplay(file_output, "regfile4: %h", uut.sccpu.cpu_ref.array_
reg[4]);
        $fdisplay(file_output, "regfile5: %h", uut.sccpu.cpu_ref.array_
reg[5]);
        $fdisplay(file_output, "regfile6: %h", uut.sccpu.cpu_ref.array_
reg[6]);
        $fdisplay(file_output, "regfile7: %h", uut.sccpu.cpu_ref.array_
reg[7]);
        $fdisplay(file_output, "regfile8: %h", uut.sccpu.cpu_ref.array_
reg[8]);
```

```
        $fdisplay(file_output, "regfile9: %h", uut.sccpu.cpu_ref.array_
reg[9]);
        $fdisplay(file_output, "regfile10: %h", uut.sccpu.cpu_ref.array
_reg[10]);
        $fdisplay(file_output, "regfile11: %h", uut.sccpu.cpu_ref.array
_reg[11]);
        $fdisplay(file_output, "regfile12: %h", uut.sccpu.cpu_ref.array
_reg[12]);
        $fdisplay(file_output, "regfile13: %h", uut.sccpu.cpu_ref.array
_reg[13]);
        $fdisplay(file_output, "regfile14: %h", uut.sccpu.cpu_ref.array
_reg[14]);
        $fdisplay(file_output, "regfile15: %h", uut.sccpu.cpu_ref.array
_reg[15]);
        $fdisplay(file_output, "regfile16: %h", uut.sccpu.cpu_ref.array
_reg[16]);
        $fdisplay(file_output, "regfile17: %h", uut.sccpu.cpu_ref.array
_reg[17]);
        $fdisplay(file_output, "regfile18: %h", uut.sccpu.cpu_ref.array
_reg[18]);
        $fdisplay(file_output, "regfile19: %h", uut.sccpu.cpu_ref.array
_reg[19]);
        $fdisplay(file_output, "regfile20: %h", uut.sccpu.cpu_ref.array
_reg[20]);
        $fdisplay(file_output, "regfile21: %h", uut.sccpu.cpu_ref.array
_reg[21]);
        $fdisplay(file_output, "regfile22: %h", uut.sccpu.cpu_ref.array
_reg[22]);
        $fdisplay(file_output, "regfile23: %h", uut.sccpu.cpu_ref.array
_reg[23]);
        $fdisplay(file_output, "regfile24: %h", uut.sccpu.cpu_ref.array
_reg[24]);
        $fdisplay(file_output, "regfile25: %h", uut.sccpu.cpu_ref.array
_reg[25]);
        $fdisplay(file_output, "regfile26: %h", uut.sccpu.cpu_ref.array
_reg[26]);
        $fdisplay(file_output, "regfile27: %h", uut.sccpu.cpu_ref.array
_reg[27]);
        $fdisplay(file_output, "regfile28: %h", uut.sccpu.cpu_ref.array
_reg[28]);
        $fdisplay(file_output, "regfile29: %h", uut.sccpu.cpu_ref.array
_reg[29]);
        $fdisplay(file_output, "regfile30: %h", uut.sccpu.cpu_ref.array
_reg[30]);
```

```
        $fdisplay(file_output, "regfile31: %h", uut.sccpu.cpu_ref.array
_reg[31]);
    end
end

endmodule
```