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 54 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，最终成功通过了前仿真、后仿真和下板，并生成了时序分析报告。

二、数据通路和指令信号建模

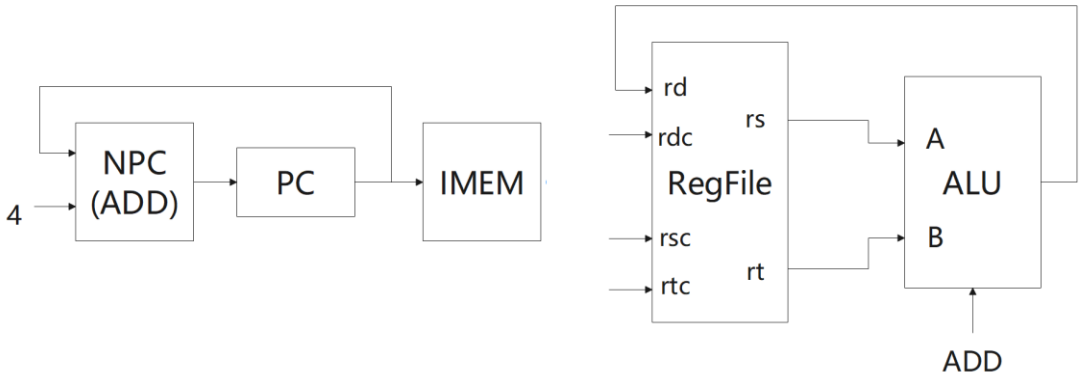
（一）数据通路设计

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：

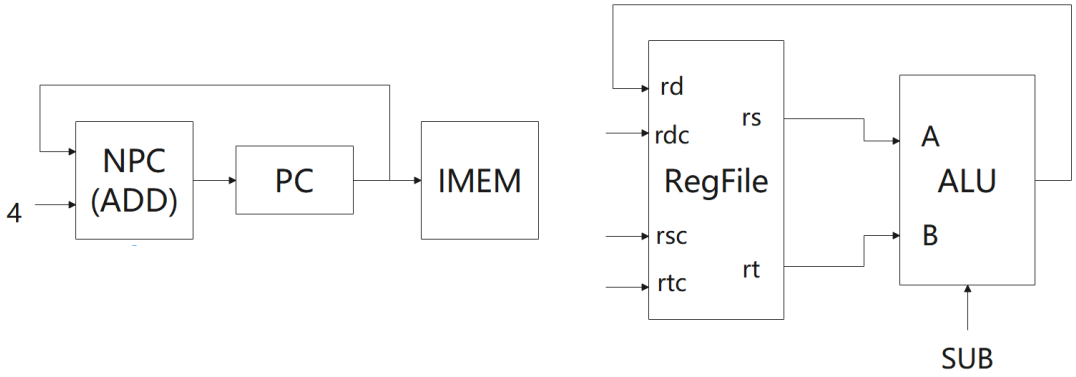


2. SUBU

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

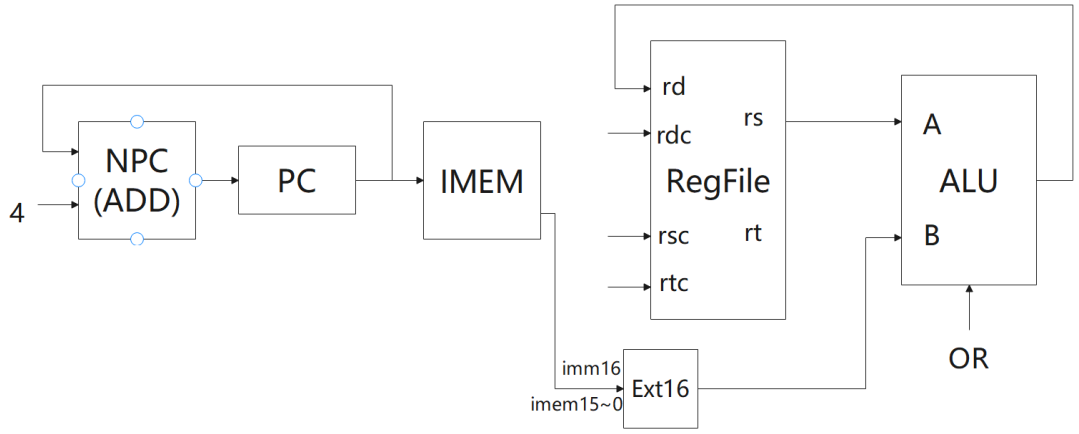
格式：ori rt, rs, imm16

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

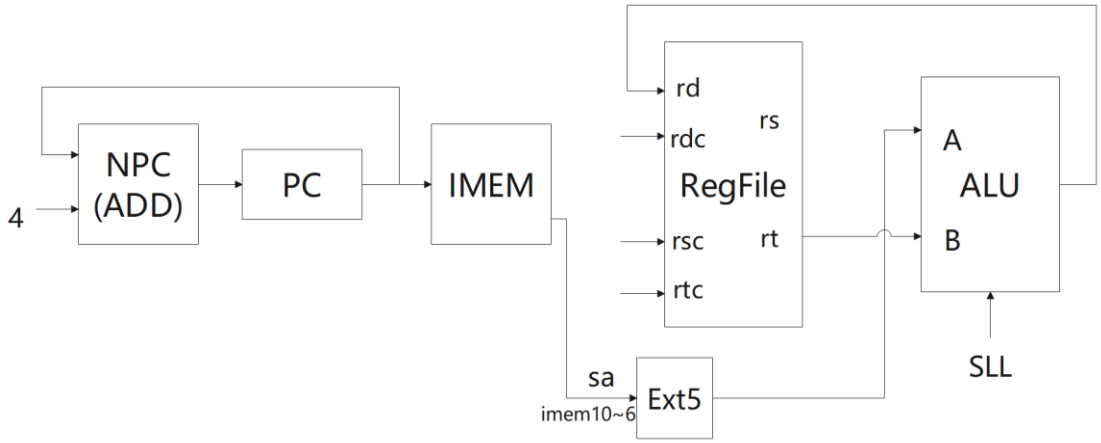
格式：sll rd, rt, sa

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

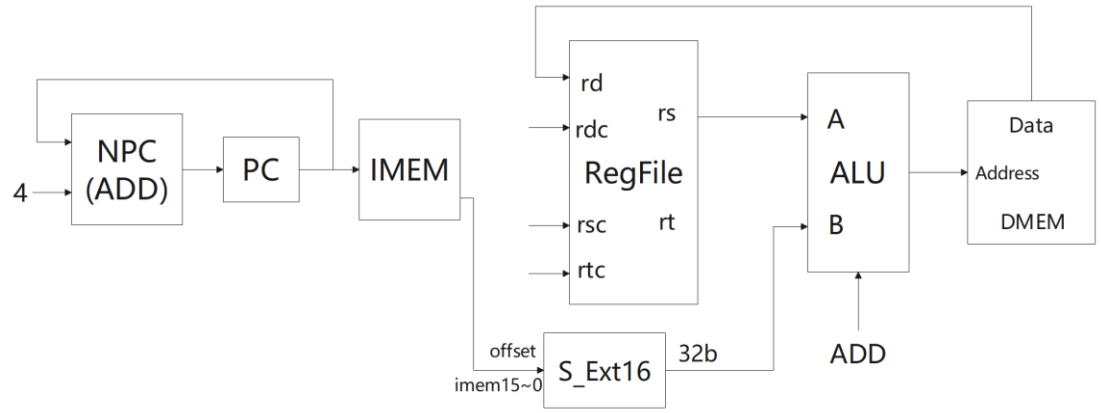
15~0 offset

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)

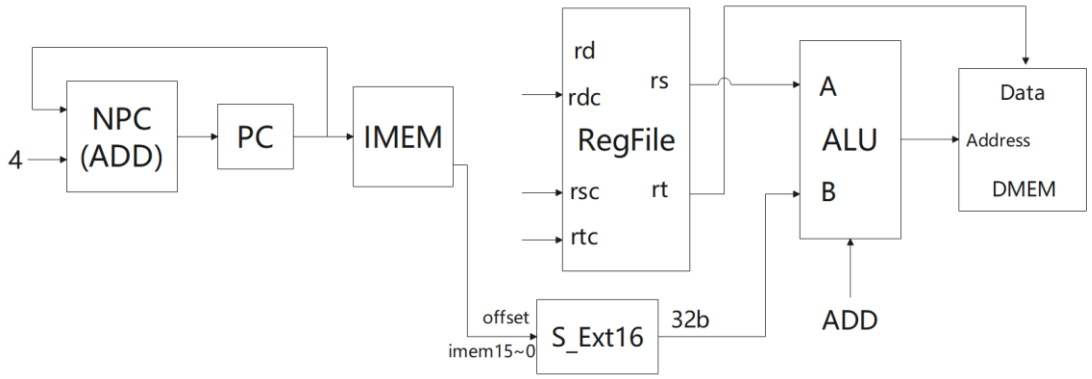
描述：memory[offset + base] ← rt

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

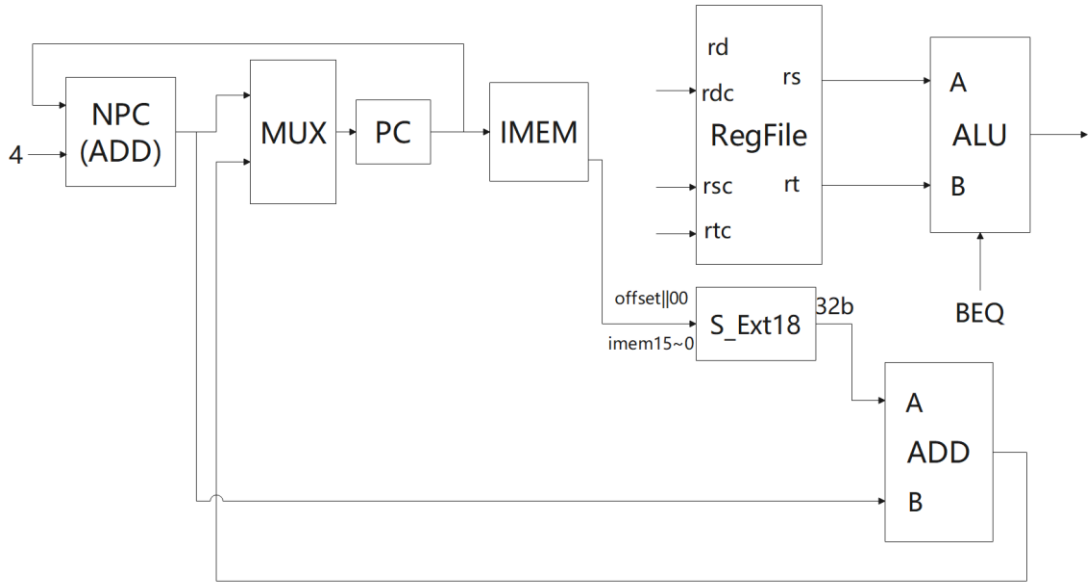
格式：beq rs, rt, offset

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

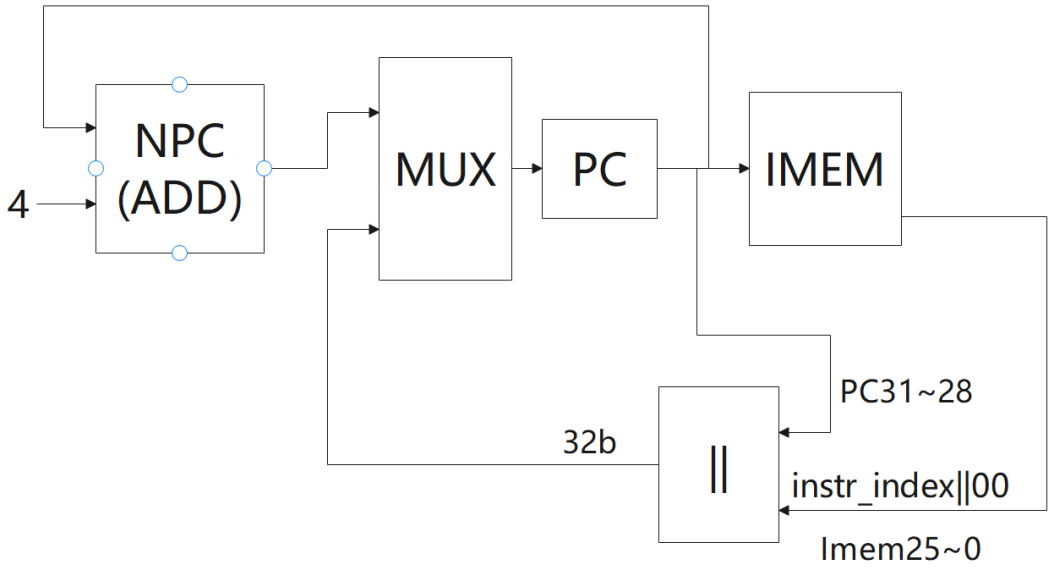
格式：J target

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

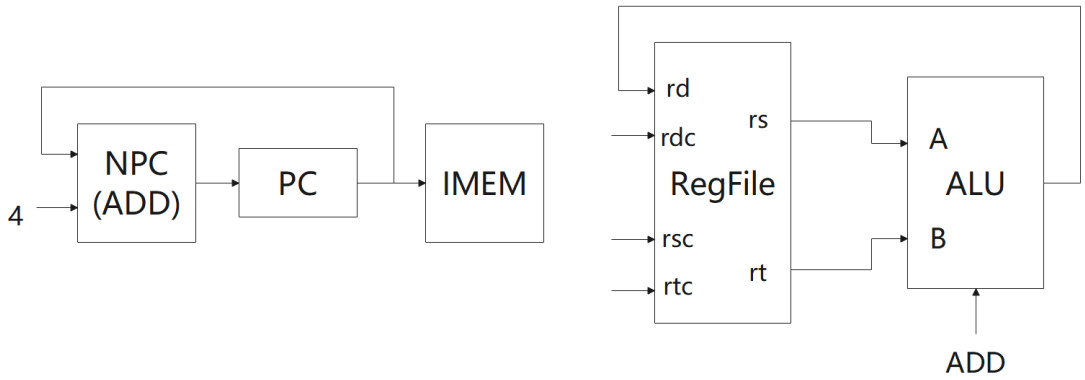
格式：add rd, rs, rt

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										

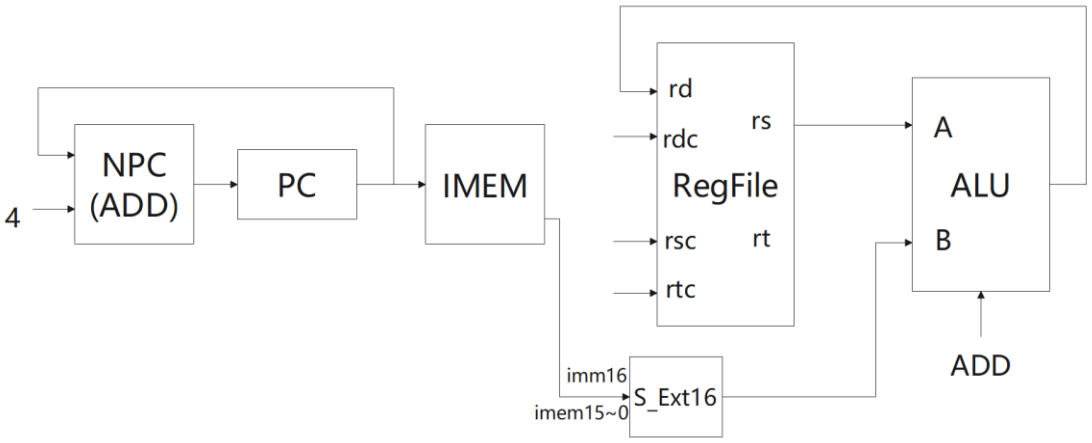
数据通路图：



【以下的输入输出关系在最后汇总一张总表】

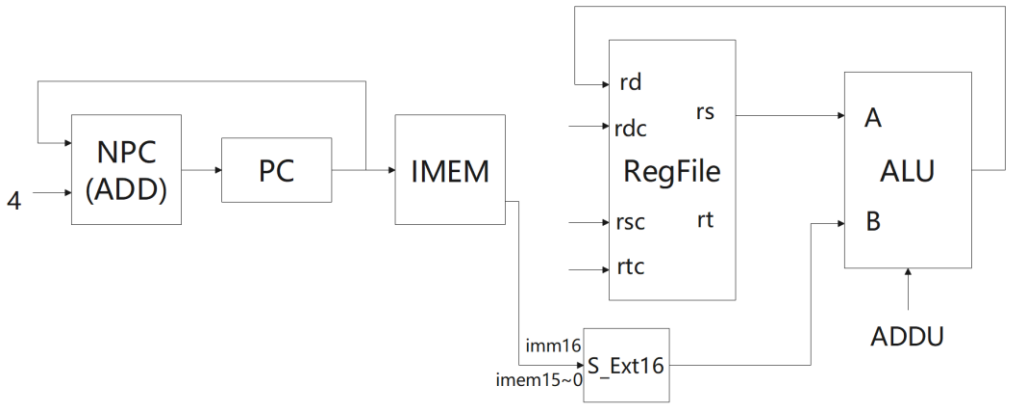
10.ADDI

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



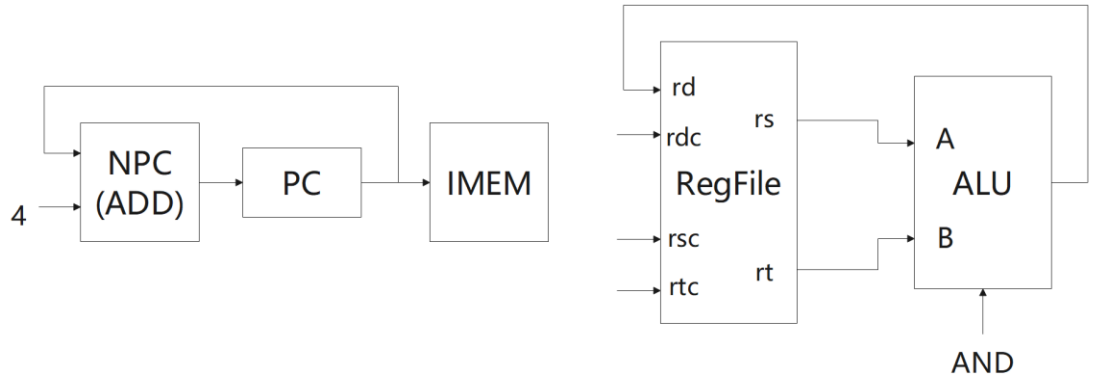
11.ADDIU

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



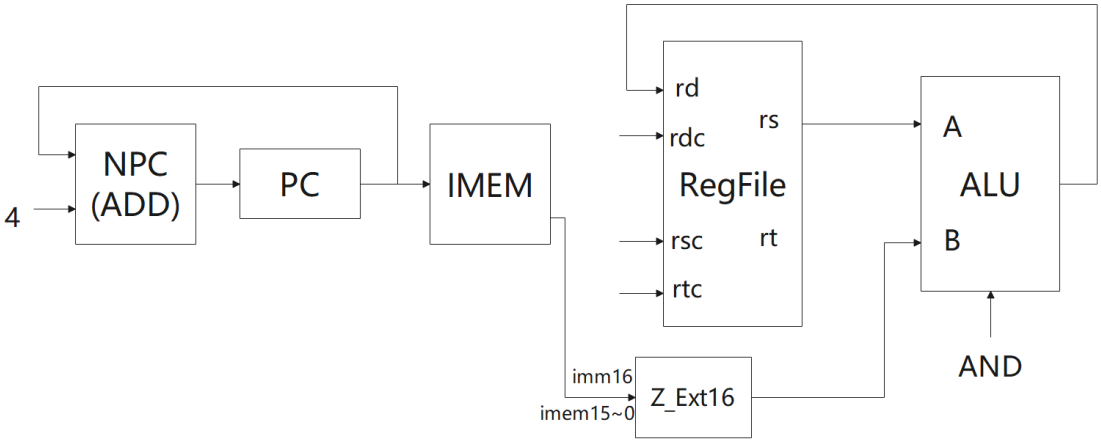
12.AND

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



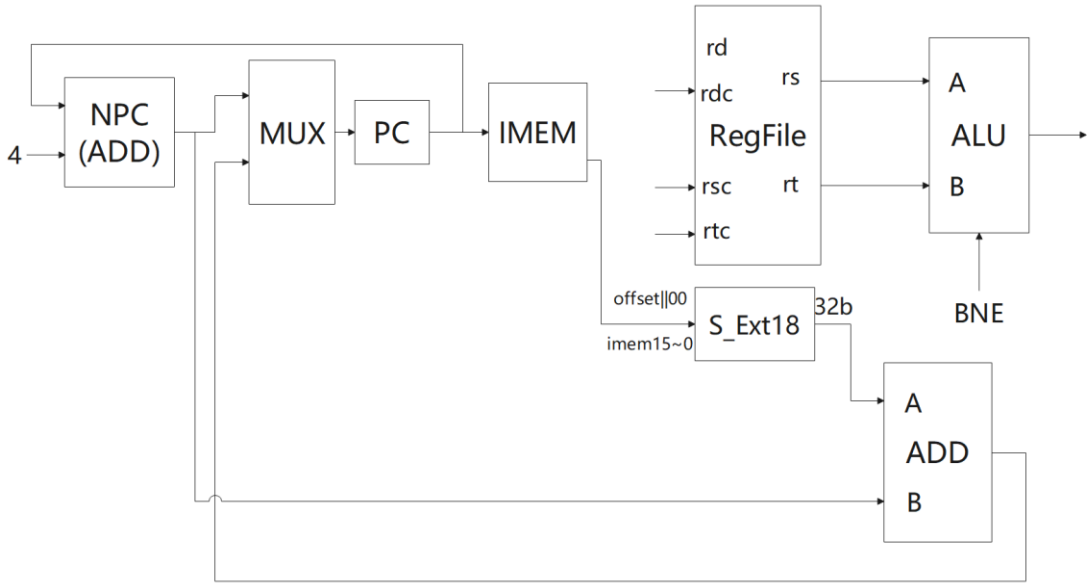
13.ANDI

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：

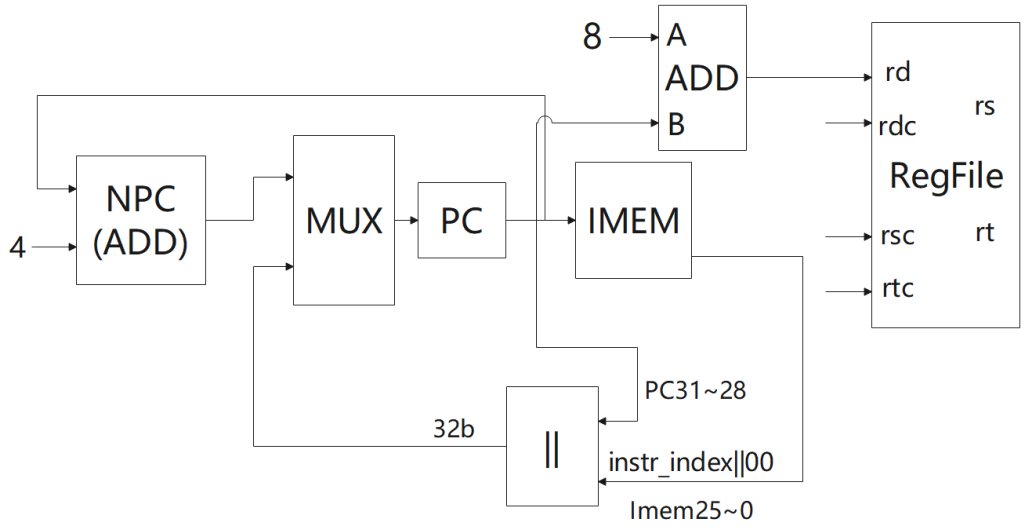


15.JAL

格式：jal target

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

数据通路图：

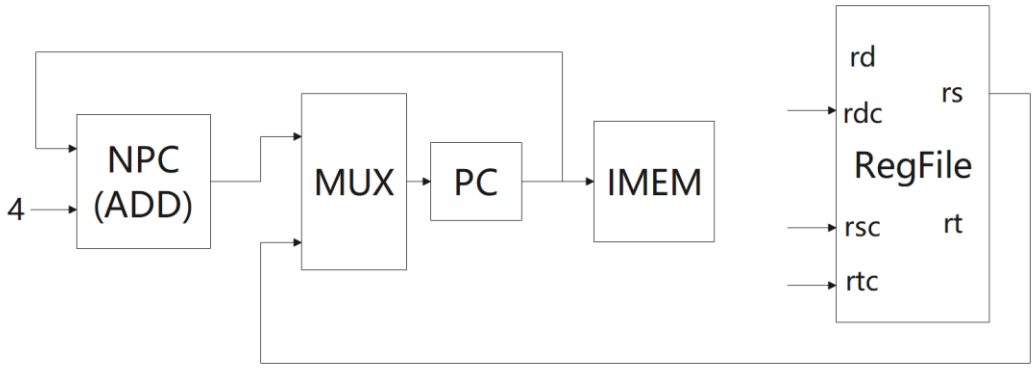


16.JR

格式：jr rs

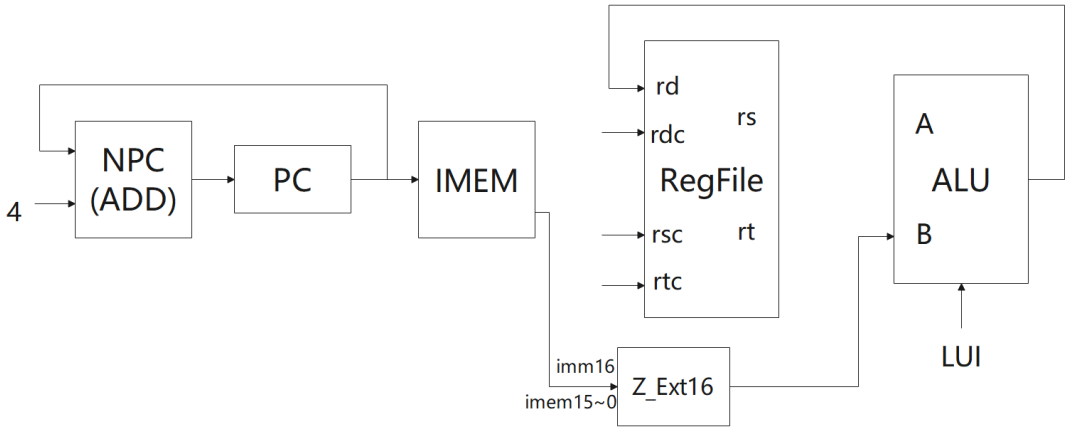
描述：PC ← RS

数据通路图：



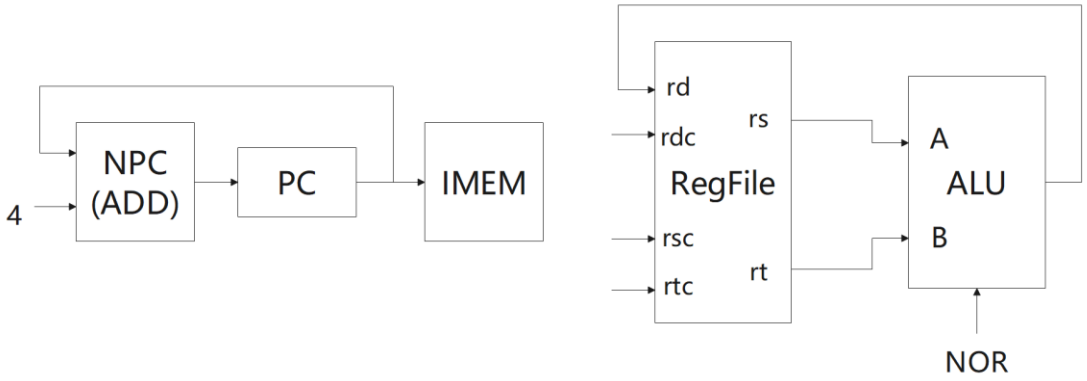
17.LUI

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



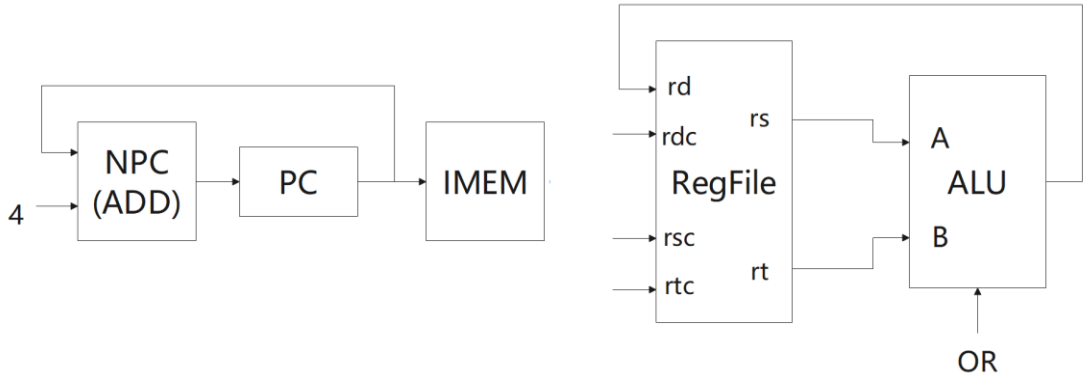
18.NOR

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



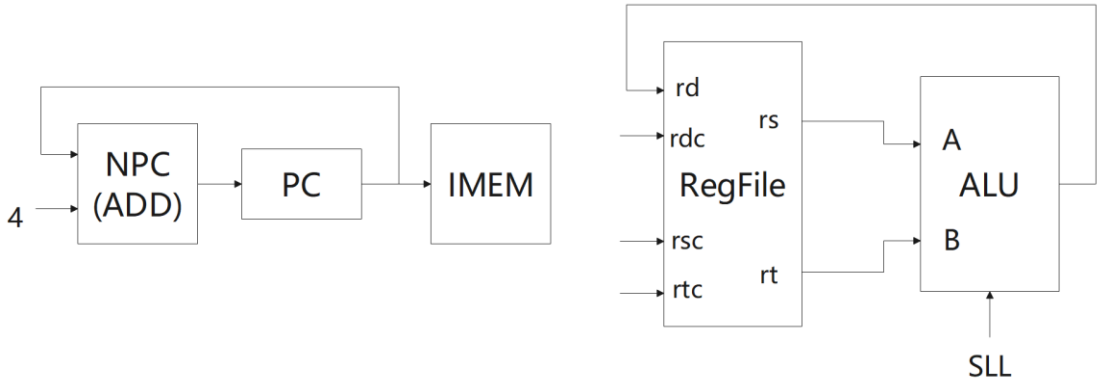
19.OR

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



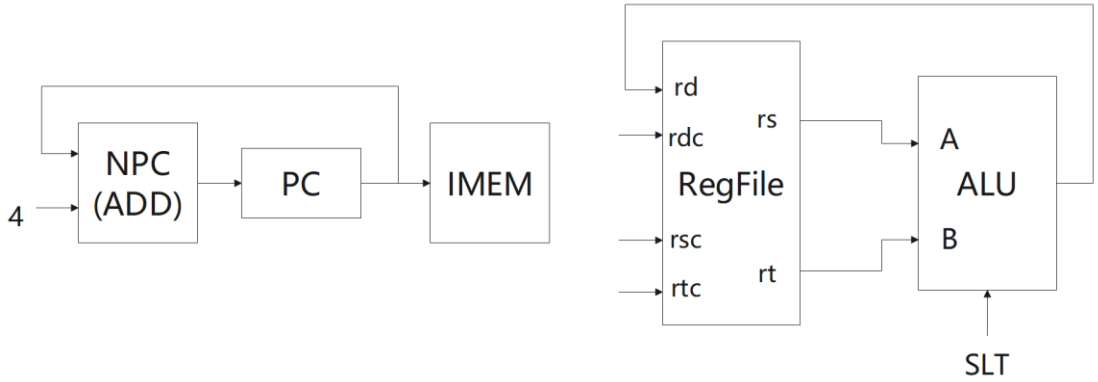
20.SLLV

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



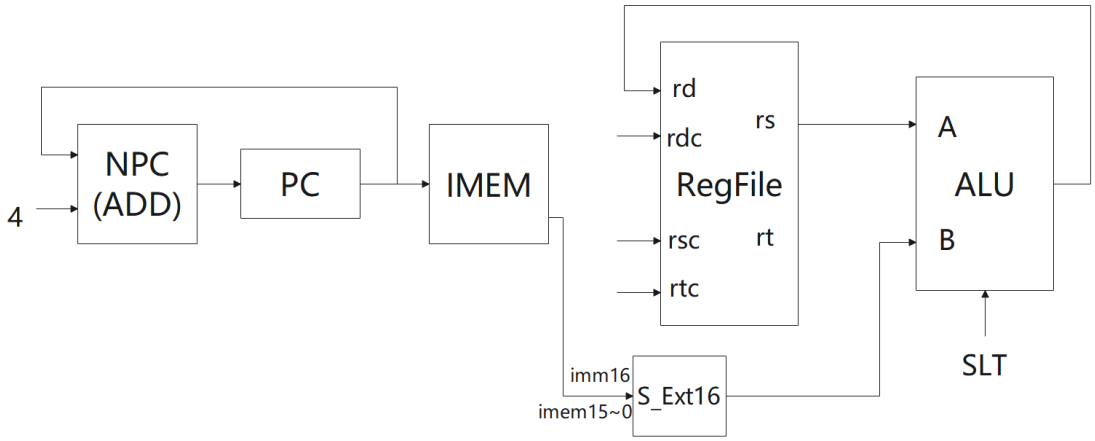
21.SLT

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



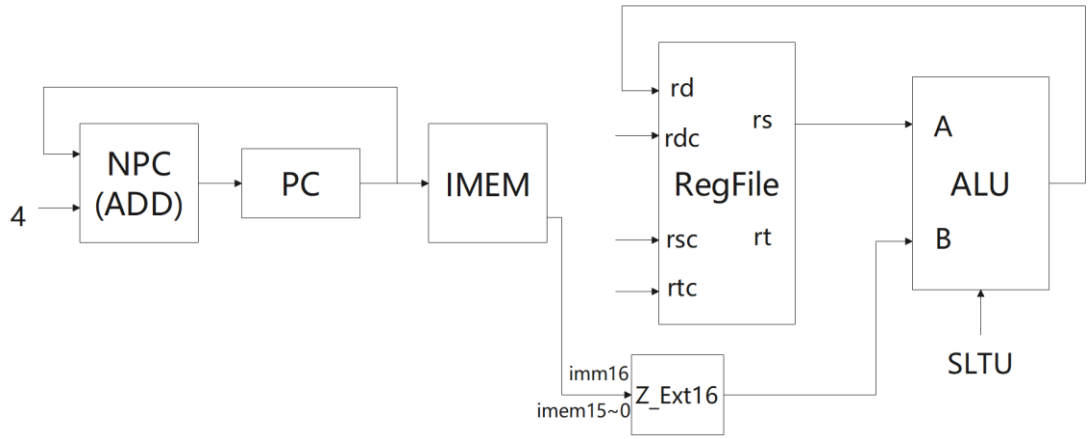
22.SLTI

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



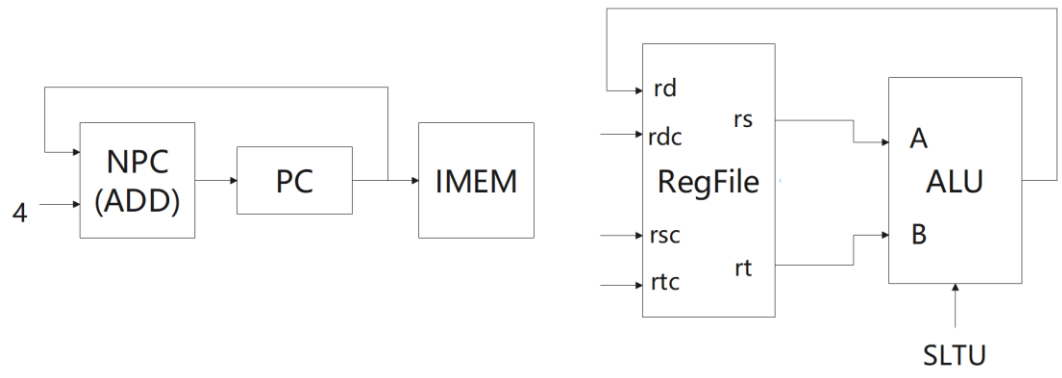
23.SLTIU

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



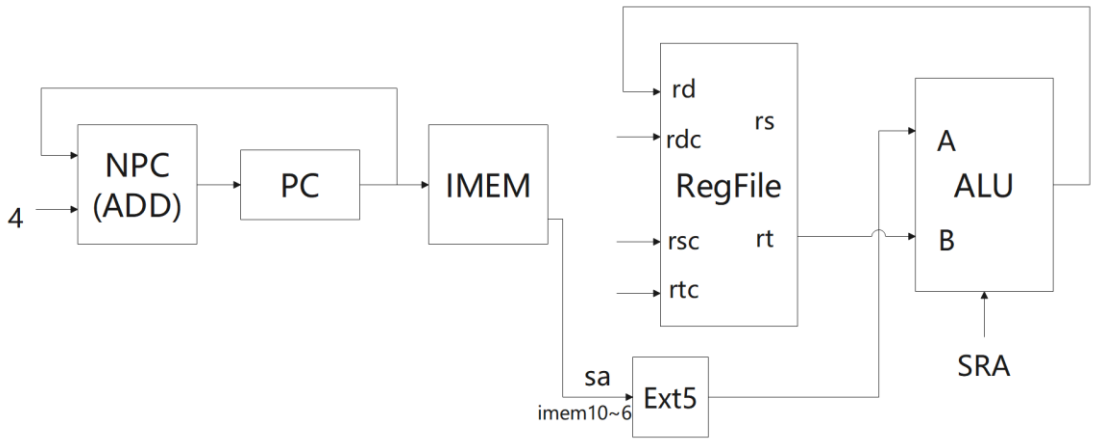
24.SLTU

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



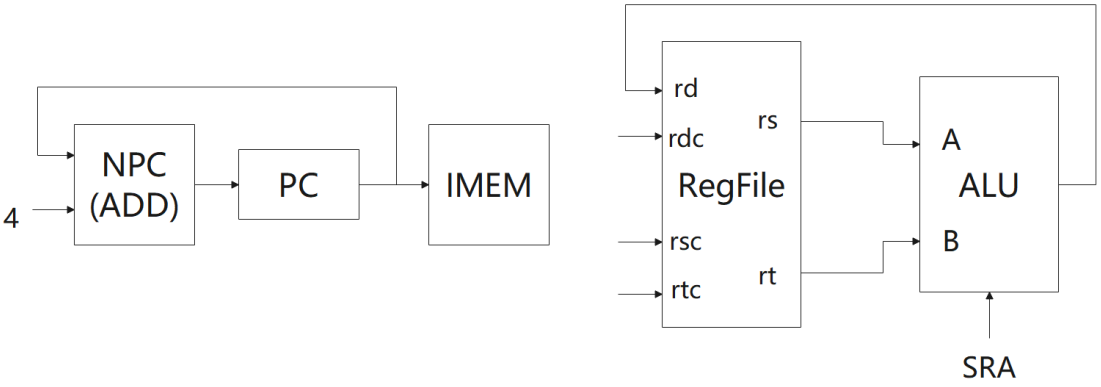
25.SRA

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



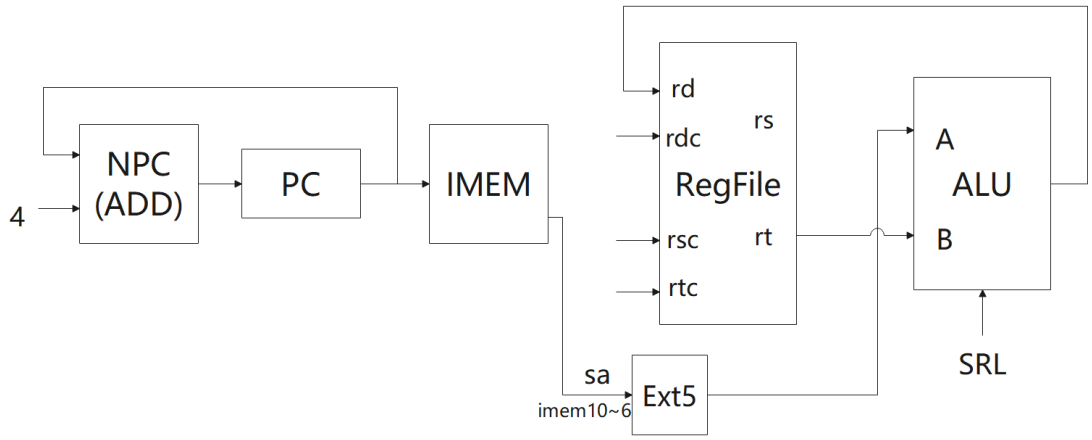
26.SRAV

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



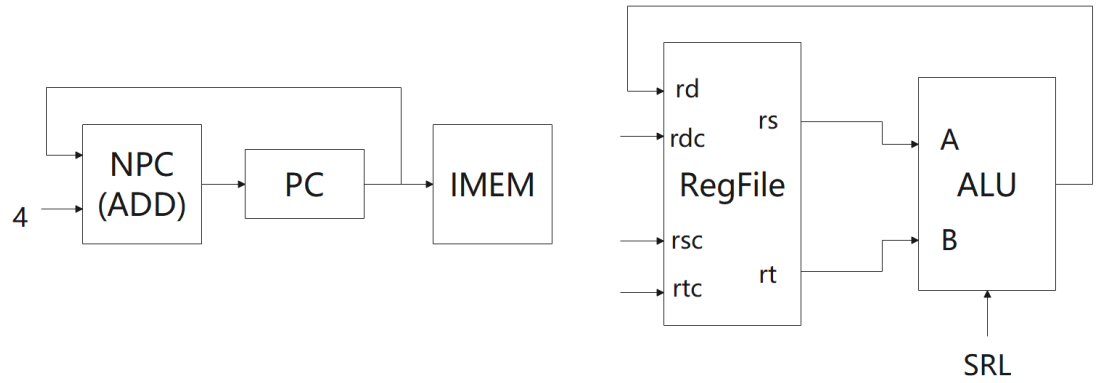
27.SRL

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



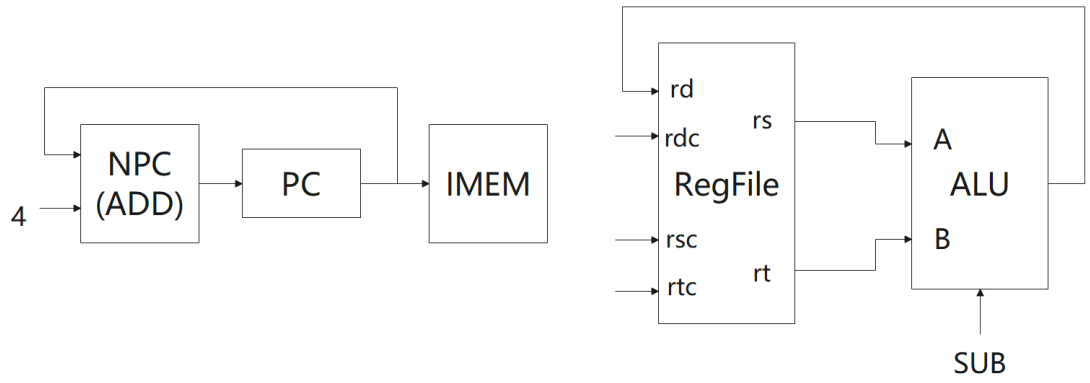
28.SRLV

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



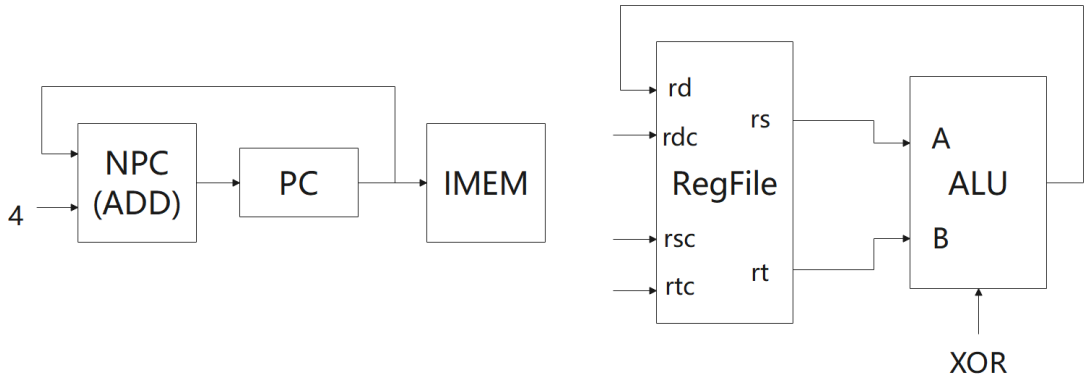
29.SUB

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



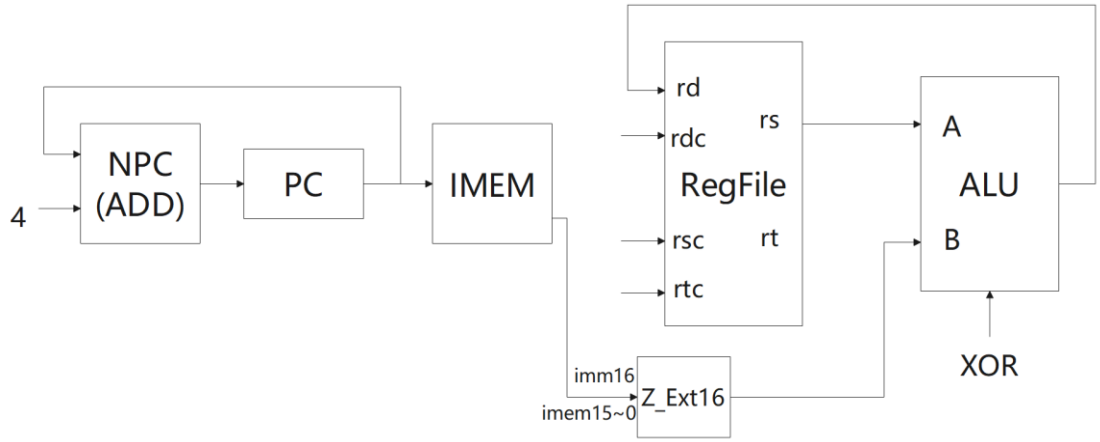
30.XOR

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



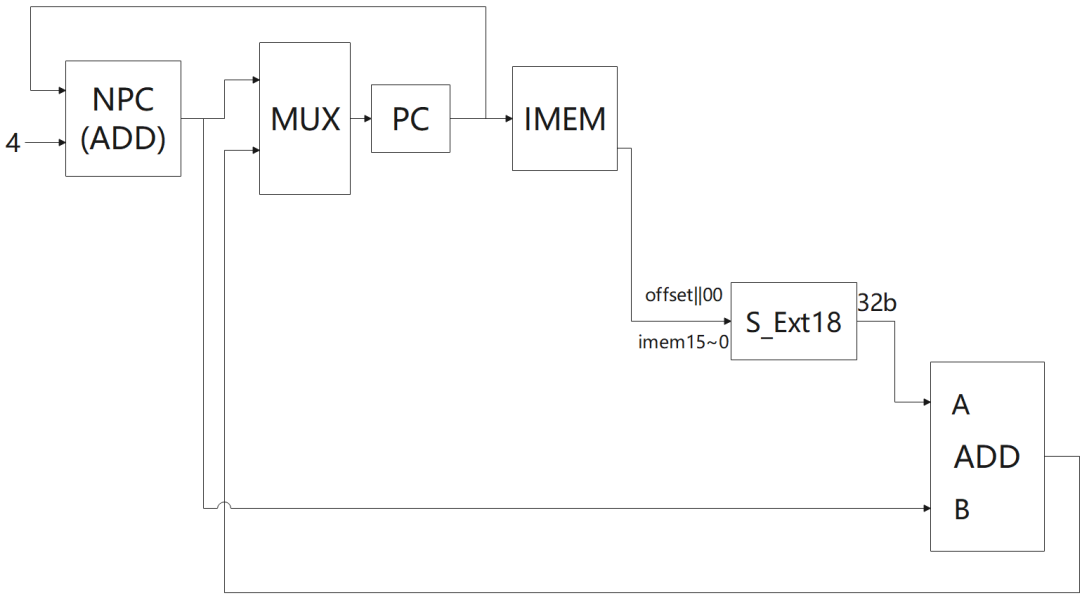
31.XORI

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



32.BGEZ

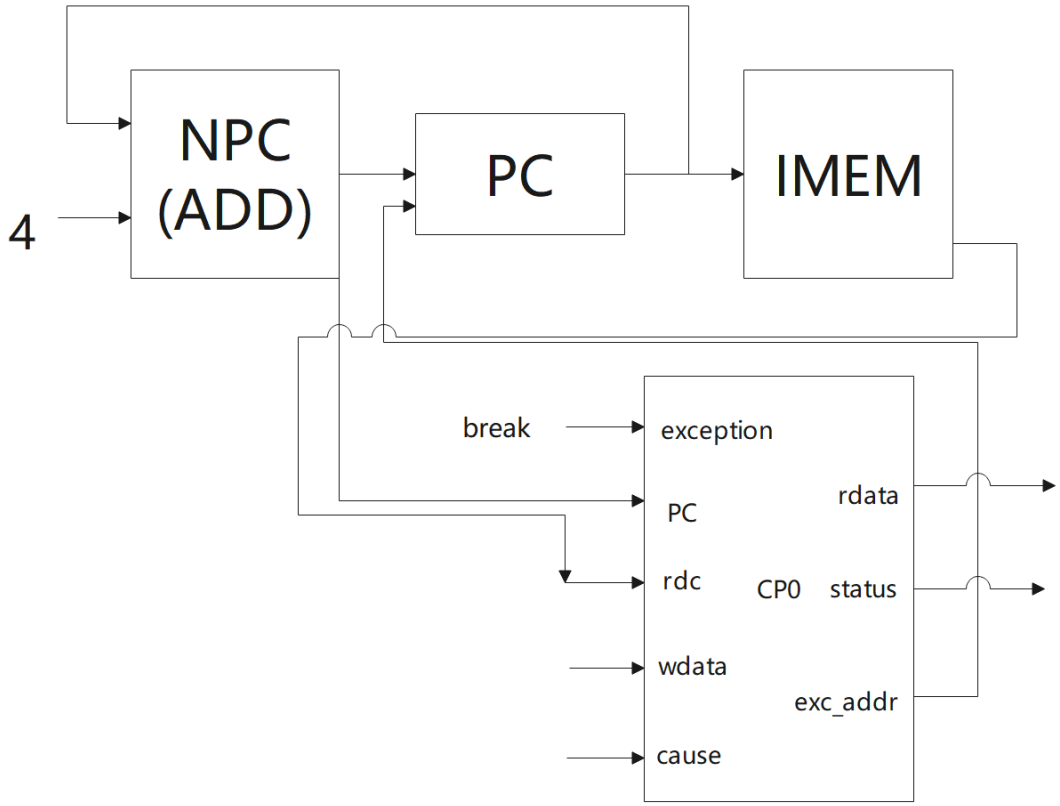
格式：BGEZ rs, offset
数据通路图：



33.BREAK

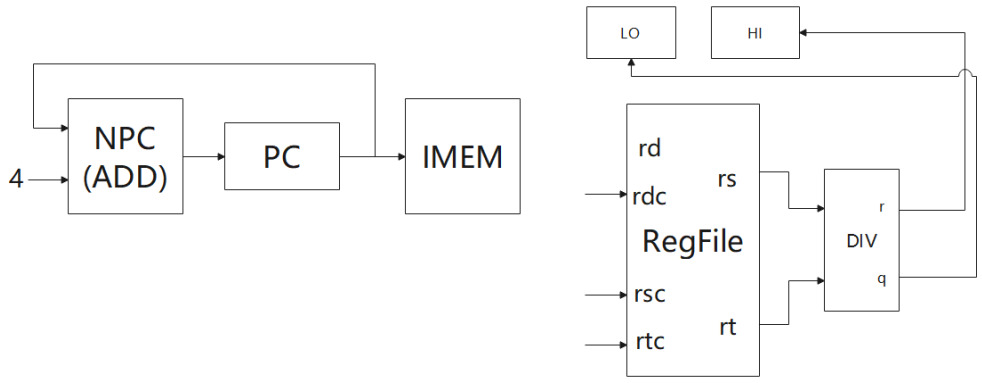
格式：BREAK MIPS32

数据通路图：



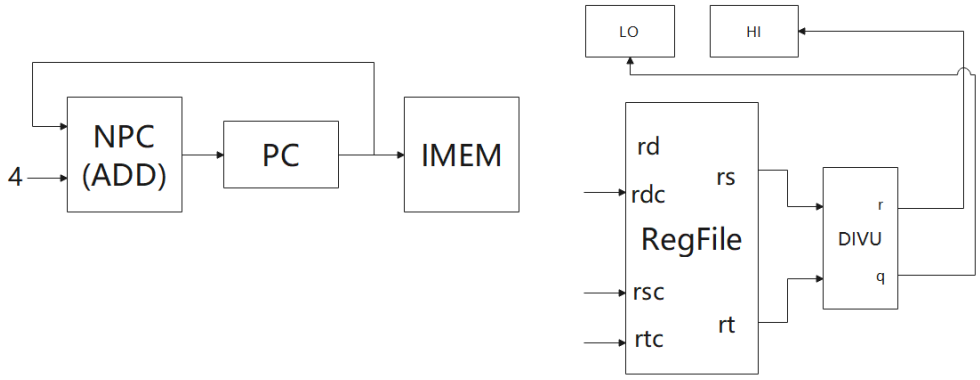
34.DIV

格式：DIV rs, rt
数据通路图：



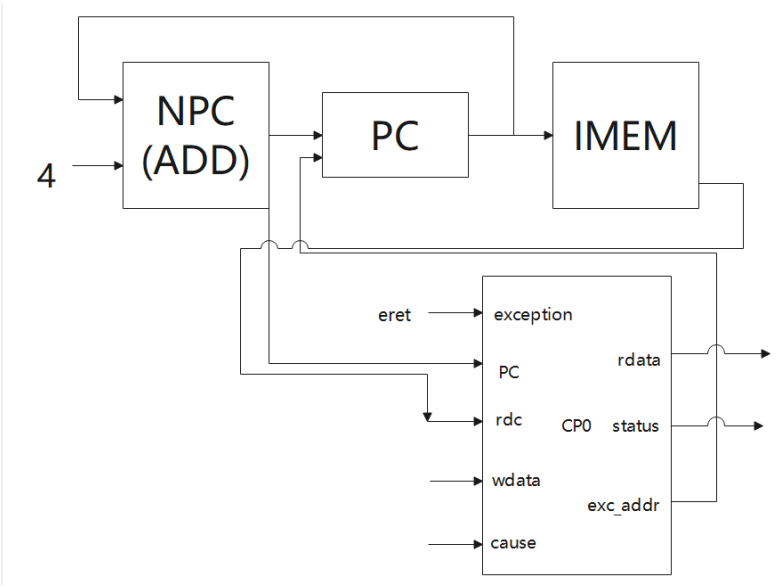
35.DIVU

格式：DIVU rs, rt
数据通路图：



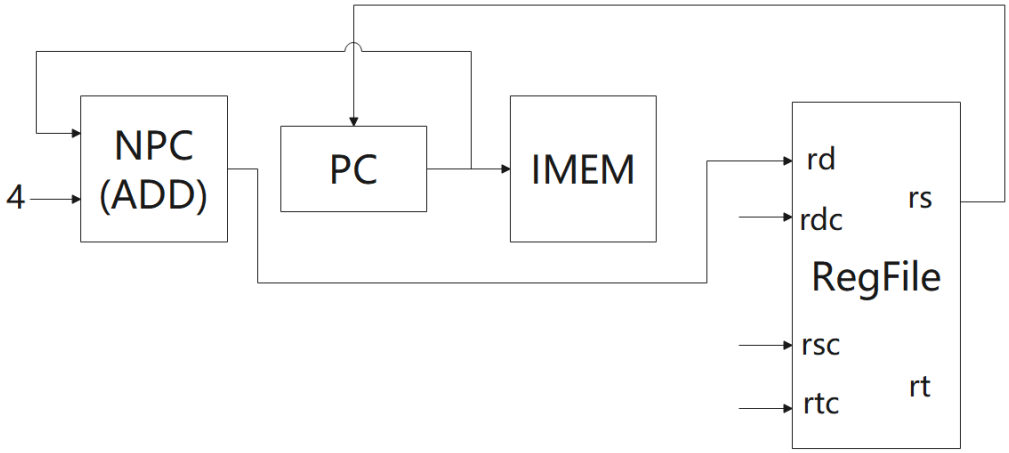
36.ERET

格式：ERET
数据通路图：



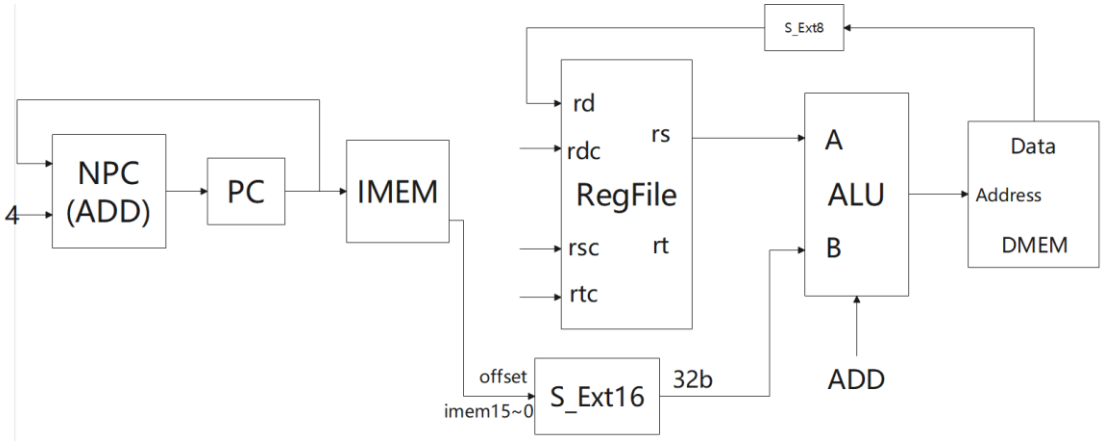
37.JALR

格式：JALR rd, rs
数据通路图：



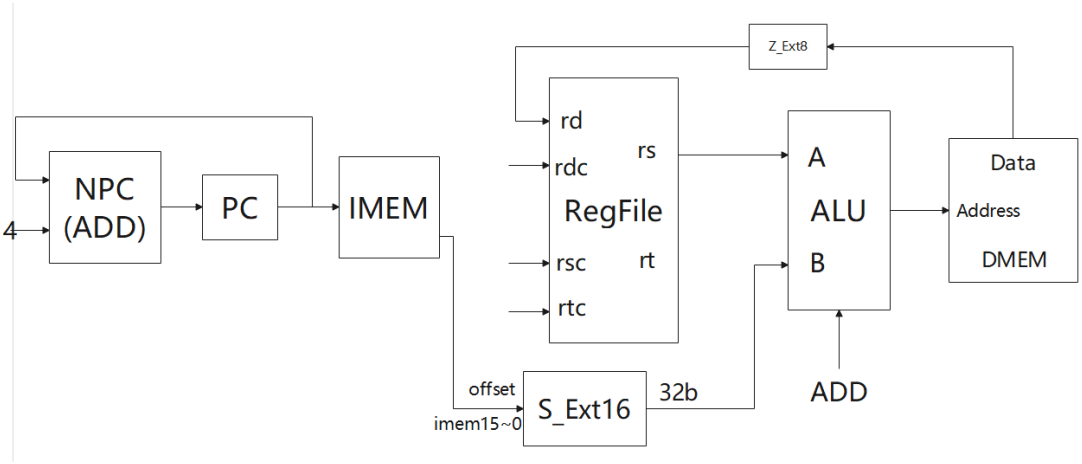
38.LB

格式：LB rt, offset(base)
数据通路图：



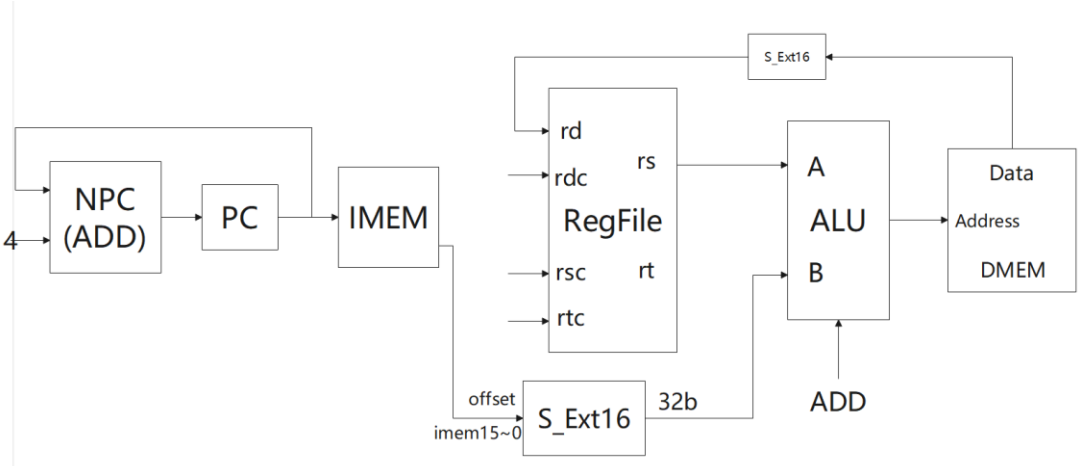
39.LBU

格式：LBU rt, offset(base)
数据通路图：



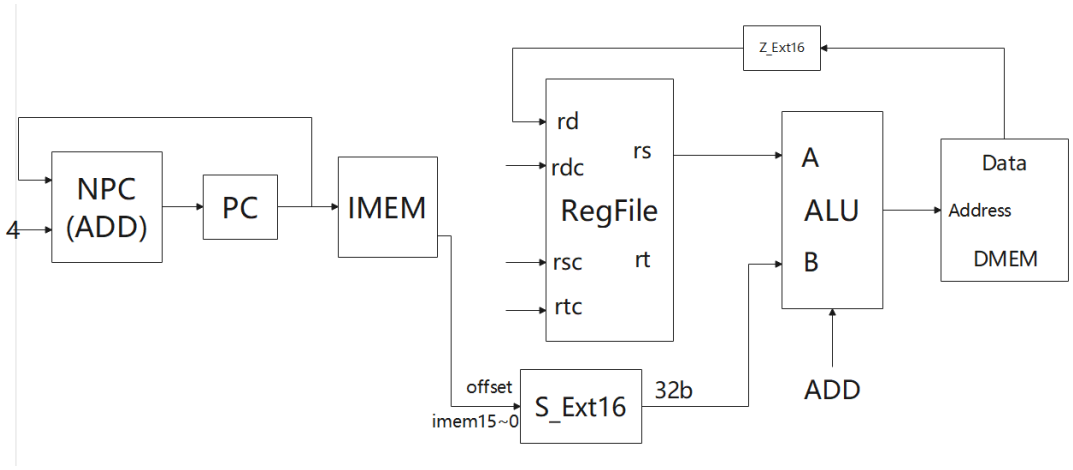
40.LH

格式：LH rt, offset(base)
数据通路图：



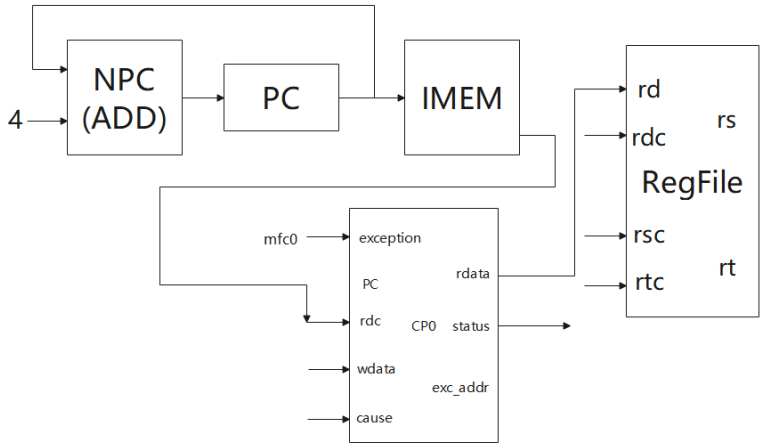
41.LHU

格式：LHU rt, offset(base)
数据通路图：



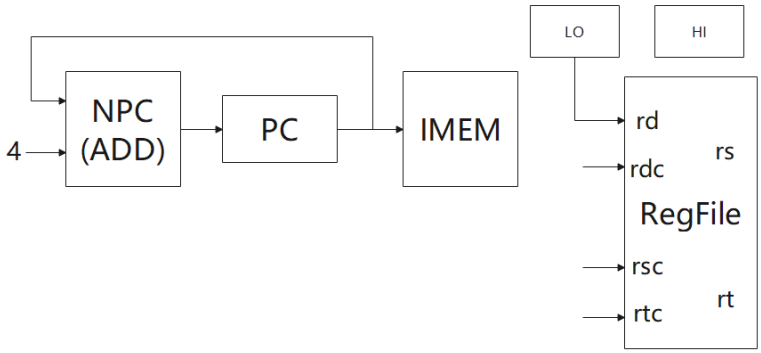
42.MFC0

格式：MFC0 rt, rd
数据通路图：



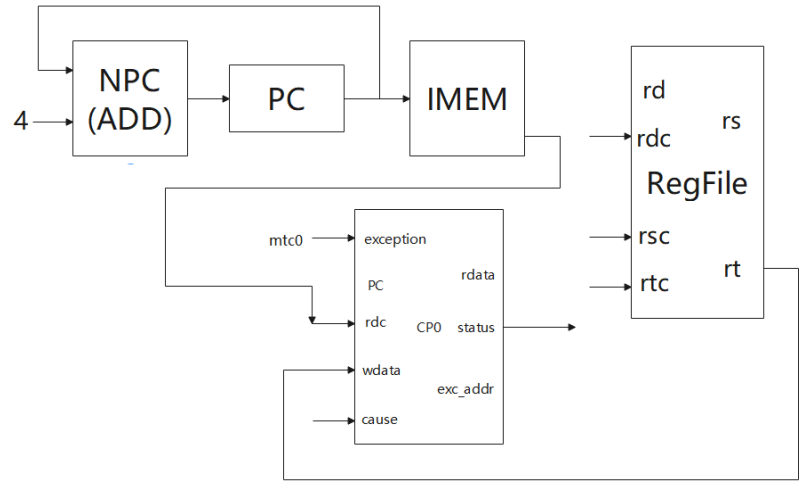
43.MFLO

格式：MFLO rd
数据通路图：



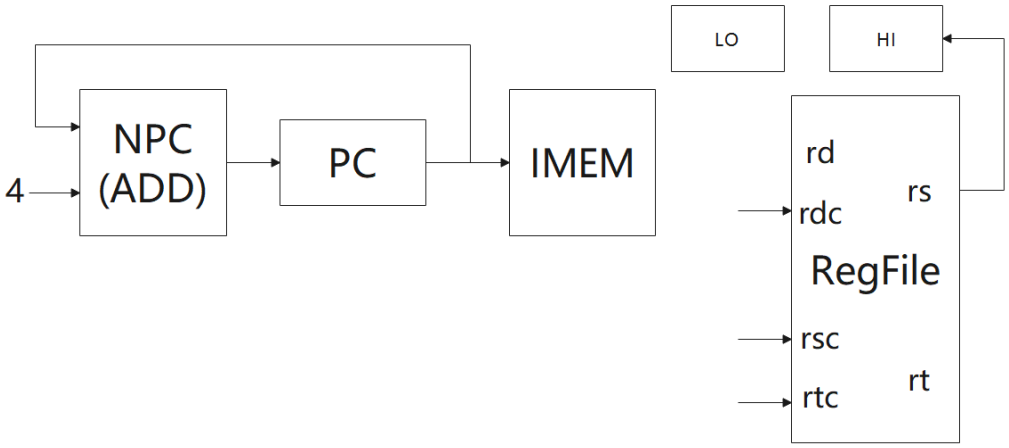
44.MTC0

格式：MTC0 rt, rd
数据通路图：



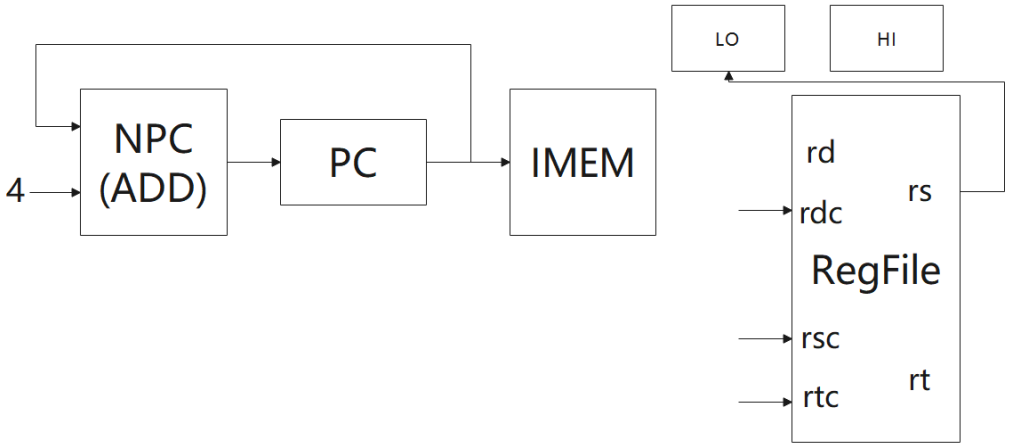
45.MTHI

格式：MTLO rs
数据通路图：



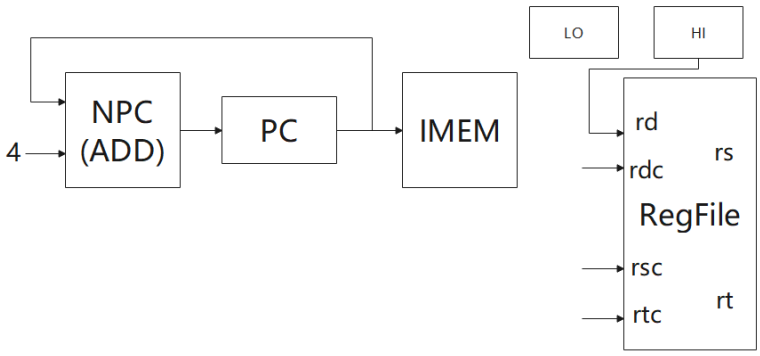
46.MTLO

格式：MTLO rs
数据通路图：



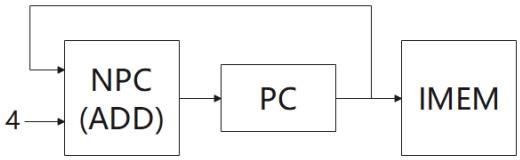
47.MFHI

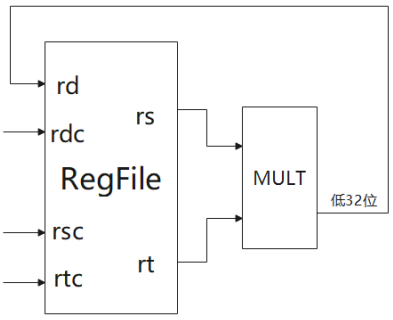
格式：MFHI rd
数据通路图：



48.MUL

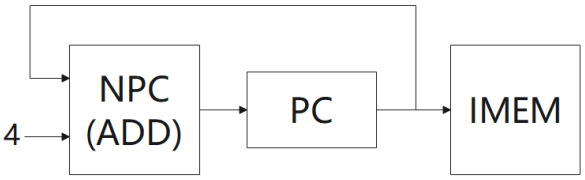
格式：MUL rd, rs, rt
数据通路图：

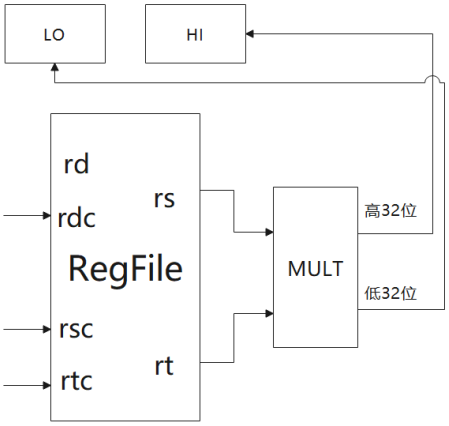




49.MULT

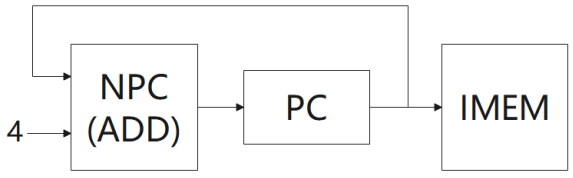
格式：MULT rs, rt
数据通路图：

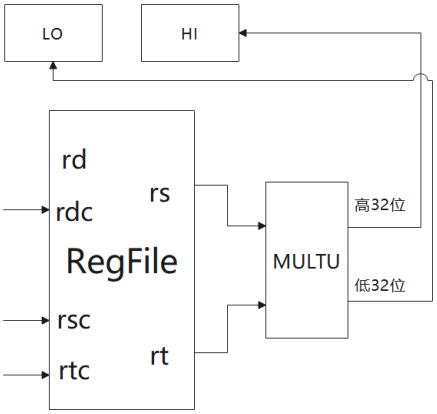




50.MULTU

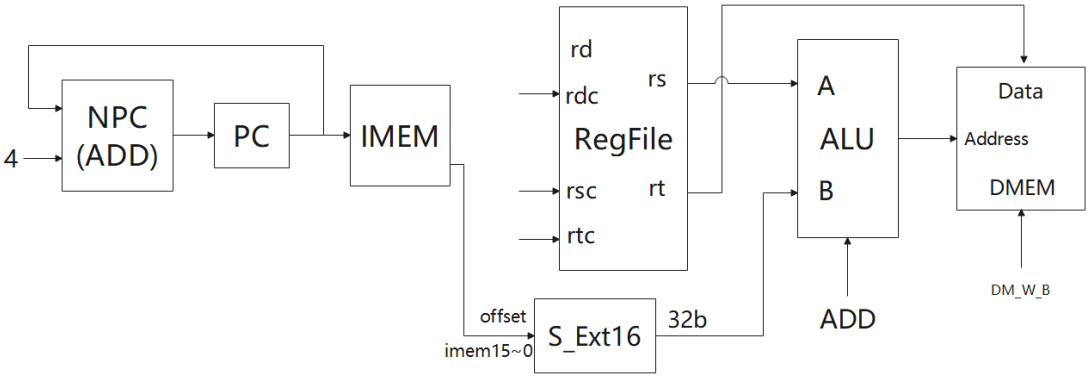
格式：MULTU rs, rt
数据通路图：





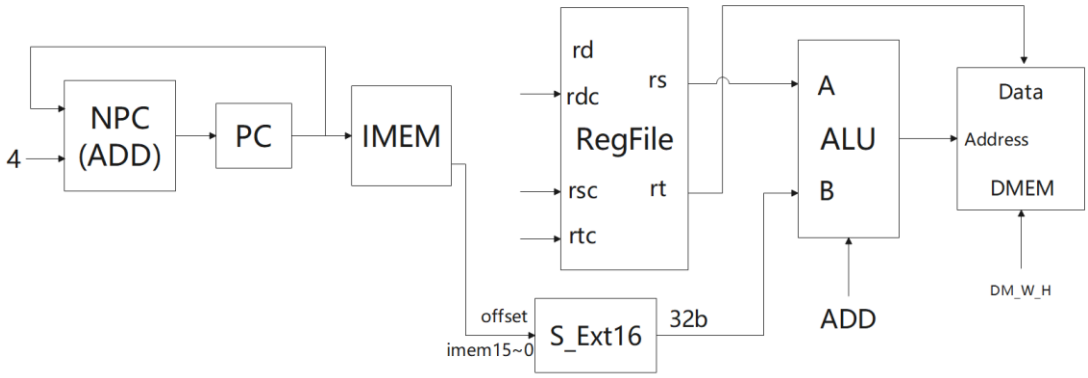
51.SB

格式：SB rt, offset(base)
数据通路图：



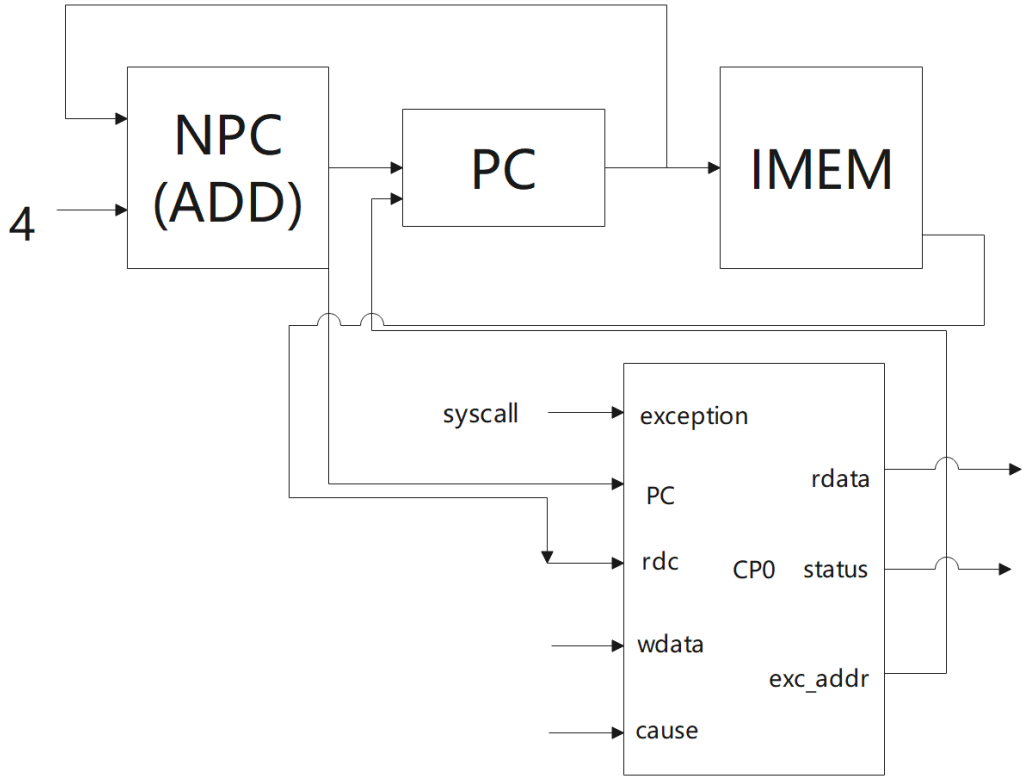
52.SH

格式：SH rt, offset(base)
数据通路图：



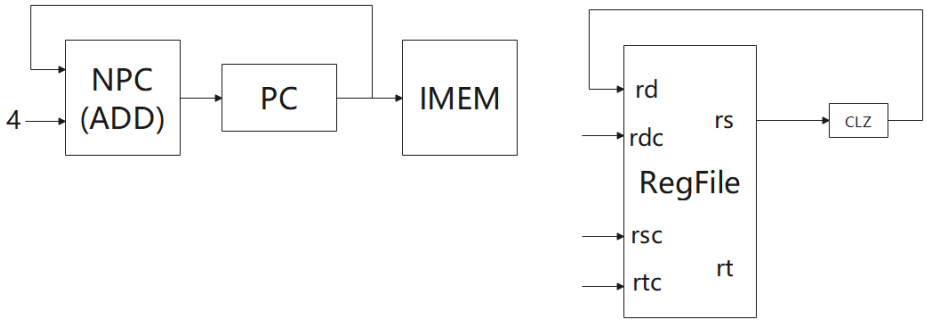
53.SYSCALL

格式：SYSCALL
数据通路图：



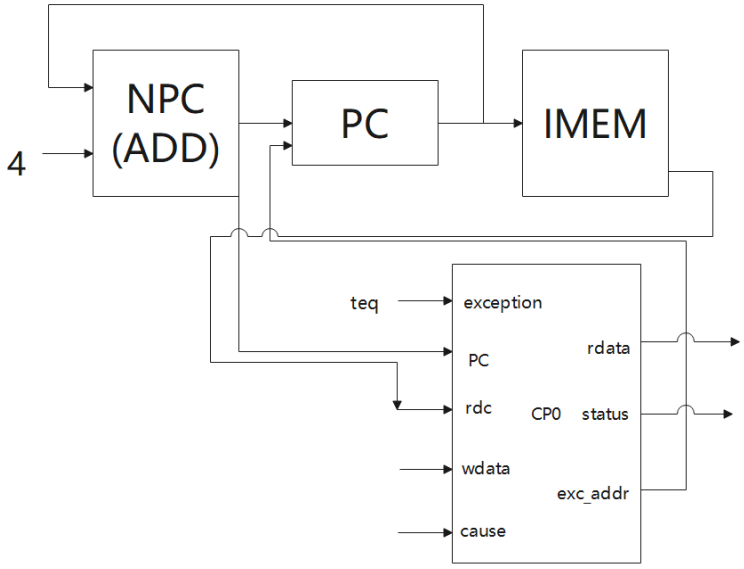
54.CLZ

格式：CLZ rd, rs
数据通路图：



55.TEQ

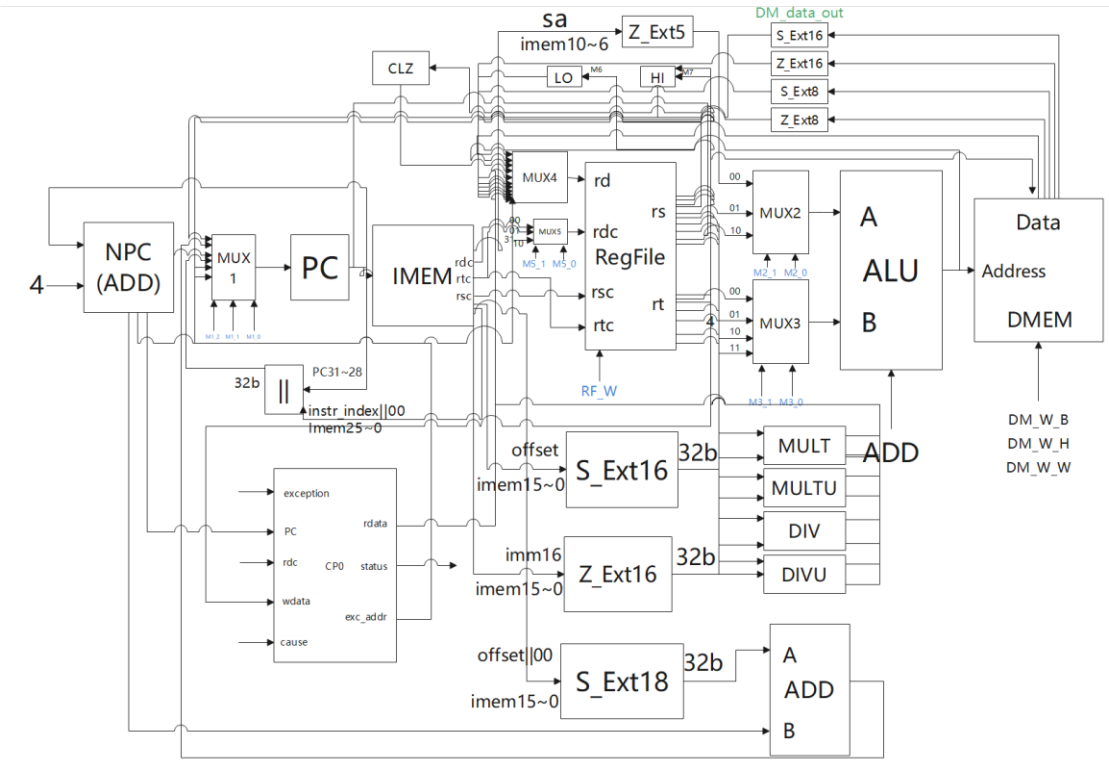
格式：TEQ rs, rt
数据通路图：



（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		||		CP0	LO	HI	DM_data_out_S_Ext8	DM_data_out_Z_Ext8	DM_data_out_S_Ext16	DM_data_out_Z_Ext16	MULT	MULTU
				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																	
DIV	NPC	PC	PC															DIV[31:0]	DIV[63:32]					Regfile	
DIVU	NPC	PC	PC															DIV[31:0]	DIV[63:32]					Regfile	
MULT	NPC	PC	PC															MULT[31:0]	MULT[63:32]					Regfile	
MUL	NPC	PC	PC	MULT																				Regfile	
MULTU	NPC	PC	PC															MULT[31:0]	MULT[63:32]						Regfile
BGEZ	ADD	PC	PC									offset	NPC	S_Ext18											
JALR	regfile	PC	PC	NPC																					
LBU	NPC	PC	PC	DM_data_out_Z_Ext8	rs	S_Ext16			ALU		offset											Data			
LHU	NPC	PC	PC	DM_data_out_Z_Ext16	rs	S_Ext16			ALU		offset												Data		
LB	NPC	PC	PC	DM_data_out_S_Ext8	rs	S_Ext16			ALU		offset									Data					
LH	NPC	PC	PC	DM_data_out_S_Ext16	rs	S_Ext16			ALU		offset											Data			
SB	NPC	PC	PC		rs	S_Ext16			ALU	rt	offset														
SH	NPC	PC	PC		rs	S_Ext16			ALU	rt	offset														
BREAK	CP0	PC	PC																						
SYSCALL	CP0	PC	PC																						
ERET	CP0	PC	PC																						
MFHI	NPC	PC	PC	HI																					
MFLO	NPC	PC	PC	LO																					
MTHI	NPC	PC	PC																Regfile						
MTLO	NPC	PC	PC															Regfile							
MFC0	NPC	PC	PC	CP0																					
MTC0	NPC	PC	PC														Regfile								
CLZ	NPC	PC	PC																						
TEQ	CP0	PC	PC																						

（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_2	M1_1	M1_0	M2	M3_1	M3_0	M4_3	M4_2	M4_1	M4_0	M5_1	M5_0	M6_2	M6_1	M6_0	M7_2	M7_1	M7_0	RF_W	DM_CS	DM_R	DM_W_W	DM_W_H	DM_W_B	HI_W	LO_W
ADDU	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
SUBU	1	1	0	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
ORI	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1							1	0	0	0				
SLL	1	1	1	1	1	x	0	1	0	0	0	0	0	0	0	1	0	0							1	0	0	0				
LW	1	1	0	0	1	0	0	1	0	1	1	0	0	1	1	0	0	1							1	1	1	0				
SW	1	1	0	0	1	0	0	1	0	1	1	0													0	1	0	1				
BEQ, z=1	1	1	0	0	0	1	0	1	0	0	1	0													0	0	0	0				
BEQ, z=0	1	1	0	0	0	1	0	1	0	1	0	0													0	0	0	0				
J	1	1					0	1	1																0	0	0	0				
ADD	1	1	0	0	1	0	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
ADDI	1	1	0	0	1	0	0	1	0	1	1	0	0	0	0	1	0	1							1	0	0	0				
ADDIU	1	1	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	1							1	0	0	0				
AND	1	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
ANDI	1	1	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	1							1	0	0	0				
BNE, z=0	1	1	0	0	0	1	0	0	1	1	0	0													0	0	0	0				
BNE, z=1	1	1	0	0	0	1	0	1	0	1	0	0													0	0	0	0				
JAL	1	1					0	1	1	0			0	1	1	1	1	0							1	0	0	0				
JR	1	1					0	0	0																0	0	0	0				
LUI	1	1	1	0	0	x	0	1	0		0	1	0	0	0	1	0	1							1	0	0	0				
NOR	1	1	0	1	1	1	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
OR	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
SLLV	1	1	1	1	1	x	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
SLT	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
SLTI	1	1	1	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1							1	0	0	0				
SLTIU	1	1	1	0	1	0	0	1	0	1	0	1	0	0	0	1	0	1							1	0	0	0				
SLTU	1	1	1	0	1	0	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
SRA	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0							1	0	0	0				
SRAV	1	1	1	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
SRL	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0							1	0	0	0				
SRLV	1	1	1	1	0	1	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
SUB	1	1	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
XOR	1	1	0	1	1	0	0	1	0	1	0	0	0	0	0	1	0	0							1	0	0	0				
XORI	1	1	0	1	1	0	0	1	0	1	0	1	0	0	0	1	0	1							1	0	0	0				
DIV && start							1	0	1										0	1	1	0	1	1							1	1
DIV && busy	1	1					1	0	1										0	1	1	0	1	1							1	1
&& !busy &&	1	1					0	1	0										0	1	1	0	1	1							1	1
DIVU && start							1	0	1										1	0	0	1	0	0							1	1
DIVU && bus	1	1					1	0	1										1	0	0	1	0	0							1	1
&& !busy &&	1	1					0	1	0										1	0	0	1	0	0							1	1
MUL	1	1					0	1	0				1	0	1	1	0	0							1							
MULT	1	1					0	1	0										0	0	1	0	0	1							1	1
MULTU	1	1					0	1	0										0	1	0	0	1	0							1	1
BEZ, rs[31] ==	1	1					0	0	1																							
BEZ, rs[31] ==	1	1					0	1	0																							
JALR	1	1					0	0	0				0	1	1	1	1	0							1							
LBU	1	1	0	0	1	0	0	1	0	1	1	0	0	0	1	1	0	1							1	1	1					
LHU	1	1	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	1							1	1	1					
LB	1	1	0	0	1	0	0	1	0	1	1	0	0	0	1	0	0	1							1	1	1					
LH	1	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	1							1	1	1					
SB	1	1	0	0	1	0	0	1	0	1	1	0														1				1		
SH	1	1	0	0	1	0	0	1	0	1	1	0														1			1			
BREAK	1	1					1	0	0																							
SYSCALL	1	1					1	0	0																							
ERET	1	1					1	0	0																							
MFHI	1	1					0	1	0				1	0	0	1	0	0							1							
MFLO	1	1					0	1	0				1	0	1	0	0	0							1							
MTHI	1	1					0	1	0													0	0	0							1	
MTLO	1	1					0	1	0										0	0	0											1
MFC0	1	1					0	1	0				1	0	0	0	0	1							1							
MTC0	1	1					0	1	0																							
CLZ	1	1					0	1	0				0	0	0	0	0	0							1							
TEQ, rs == rt	1	1					1	0	0																							
TEQ, rs != rt	1	1					0	1	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_W, DM_W_H, DM_W_B;
wire [31:0] DM_data_in;
wire [31:0] DM_data_out;
wire [10:0] 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;

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

dmem dmem_inst(.clk(clk_in), .cs(cs), .DM_R(DM_R), .DM_W_W(DM_W_W), .DM
_W_H(DM_W_H), .DM_W_B(DM_W_B), .DM_addr(DM_addr_DM), .DM_data_in(DM_dat
a_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_W(DM_W_W), .DM_W_H(DM_W_H
), .DM_W_B(DM_W_B), .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_W,
    input DM_W_H,
    input DM_W_B,
    input [10:0] DM_addr,
```

```
    input [31:0] DM_data_in,
    output [31:0] DM_data_out
);

reg [7:0] mem [0:1023];
always @ (posedge clk)
begin
    if(cs && DM_W_W)
    begin
        mem[DM_addr] <= DM_data_in[7:0];
        mem[DM_addr + 1] <= DM_data_in[15:8];
        mem[DM_addr + 2] <= DM_data_in[23:16];
        mem[DM_addr + 3] <= DM_data_in[31:24];
    end
    else if(cs && DM_W_H)
    begin
        mem[DM_addr] <= DM_data_in[7:0];
        mem[DM_addr + 1] <= DM_data_in[15:8];
    end
    else if(cs && DM_W_B)
    begin
        mem[DM_addr] <= DM_data_in[7:0];
    end
end

assign DM_data_out[7:0] = (cs && DM_R) ? mem[DM_addr] : {32{1'bz}};
assign DM_data_out[15:8] = (cs && DM_R) ? mem[DM_addr + 1] : {32{1'bz}}
;
assign DM_data_out[23:16] = (cs && DM_R) ? mem[DM_addr + 2] : {32{1'bz}
};
assign DM_data_out[31:24] = (cs && DM_R) ? mem[DM_addr + 3] : {32{1'bz}
};

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_W,
    output DM_W_H,
    output DM_W_B,
    output [10:0] DM_addr,
    output [31:0] DM_data_in
);

wire [3:0] aluc;
wire [2:0] M1;
wire M2;
wire [1:0] M3;
wire [3:0] M4;
wire [1:0] M5;
wire [2:0] M6;
wire [2:0] M7;
wire RF_W;
wire HI_W, LO_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;
reg [31:0] last_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 zero, carry, negative, overflow;
wire exception;
wire [4:0] cause;
wire [31:0] cp0_rdata;
```

```
wire [31:0] cp0_status;
wire [31:0] exc_addr;
wire [31:0] clz_output;
wire [31:0] DM_data_out_Z_Ext8;
wire [31:0] DM_data_out_S_Ext8;
wire [31:0] DM_data_out_Z_Ext16;
wire [31:0] DM_data_out_S_Ext16;
wire [31:0] hi_output;
wire [31:0] lo_output;
wire [31:0] hi_input;
wire [31:0] lo_input;
wire [63:0] MULT_result;
wire [63:0] MULTU_result;
reg div_start;
reg divu_start;
wire div_busy;
wire divu_busy;
wire [31:0] div_q;
wire [31:0] div_r;
wire [31:0] divu_q;
wire [31:0] divu_r;
wire [31:0] M1_temp1;
wire [31:0] M1_temp2;
wire [31:0] M1_temp3;
wire [31:0] M1_temp4;
wire [31:0] M3_temp;
wire [31:0] M4_temp1;
wire [31:0] M4_temp2;
wire [31:0] M4_temp3;
wire [31:0] M4_temp4;
wire [31:0] M4_temp5;
wire [31:0] M4_temp6;
wire [31:0] M4_temp7;
wire [31:0] M4_temp8;
wire [31:0] M4_temp9;
wire [31:0] M4_temp10;
wire [31:0] M5_temp;
wire [31:0] M6_temp1;
wire [31:0] M6_temp2;
wire [31:0] M6_temp3;
wire [31:0] M7_temp1;
wire [31:0] M7_temp2;
wire [31:0] M7_temp3;
```

```
assign npc = pc + 4;
assign op = inst[31:26];
assign rsc = inst[25:21]; // 指令中的5位rs
assign rtc = inst[20:16]; // 指令中的5位rt
assign im_rdc = inst[15:11]; // 指令中的5位rd
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_;
wire _div_, _divu_, _mul_, _mult_, _multu_, _bgez_, _jalr_, _lbu_, _lb_
, _lhu_, _lh_, _sb_, _sh_, _break_, _syscall_, _eret_, _mfhi_, _mflo_,
_mthi_, _mtlo_, _mfc0_, _mtc0_, _clz_, _teq_;

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 _div_ = (op == 6'b000000 && func == 6'b011010) ? 1'b1 : 1'b0;
assign _divu_ = (op == 6'b000000 && func == 6'b011011) ? 1'b1 : 1'b0;
assign _mul_ = (op == 6'b011100 && func == 6'b000010) ? 1'b1 : 1'b0;
assign _mult_ = (op == 6'b000000 && func == 6'b011000) ? 1'b1 : 1'b0;
assign _multu_ = (op == 6'b000000 && func == 6'b011001) ? 1'b1 : 1'b0;
assign _bgez_ = (op == 6'b000001) ? 1'b1 : 1'b0;
assign _jalr_ = (op == 6'b000000 && func == 6'b001001) ? 1'b1 : 1'b0;
assign _lbu_ = (op == 6'b100100) ? 1'b1 : 1'b0;
assign _lhu_ = (op == 6'b100101) ? 1'b1 : 1'b0;
assign _lb_ = (op == 6'b100000) ? 1'b1 : 1'b0;
assign _lh_ = (op == 6'b100001) ? 1'b1 : 1'b0;
assign _sb_ = (op == 6'b101000) ? 1'b1 : 1'b0;
assign _sh_ = (op == 6'b101001) ? 1'b1 : 1'b0;
assign _break_ = (op == 6'b000000 && func == 6'b001101) ? 1'b1 : 1'b0;
assign _syscall_ = (op == 6'b000000 && func == 6'b001100) ? 1'b1 : 1'b0
;
assign _eret_ = (inst == 32'b010000_10000_00000_00000_00000_011000) ? 1
'b1 : 1'b0;
assign _mfhi_ = (op == 6'b000000 && func == 6'b010000) ? 1'b1 : 1'b0;
assign _mflo_ = (op == 6'b000000 && func == 6'b010010) ? 1'b1 : 1'b0;
assign _mthi_ = (op == 6'b000000 && func == 6'b010001) ? 1'b1 : 1'b0;
assign _mtlo_ = (op == 6'b000000 && func == 6'b010011) ? 1'b1 : 1'b0;
assign _mfc0_ = (op == 6'b010000 && rsc == 5'b00000) ? 1'b1 : 1'b0;
assign _mtc0_ = (op == 6'b010000 && rsc == 5'b00100) ? 1'b1 : 1'b0;
assign _clz_ = (op == 6'b011100 && func == 6'b100000) ? 1'b1 : 1'b0;
assign _teq_ = (op == 6'b000000 && func == 6'b110100) ? 1'b1 : 1'b0;

always@(posedge clk)
begin
    if(_div_ && div_start == 0 && !div_busy)
        div_start <= 1;
    else if(_div_ && div_start == 1)
        div_start <= 0;
```

```
    else
        div_start <= 0;
end

always@(posedge clk)
begin
    if(_divu_ && divu_start == 0 && !divu_busy)
        divu_start <= 1;
    else if(_divu_ && divu_start == 1)
        divu_start <= 0;
    else
        divu_start <= 0;
end

always @(posedge clk) begin
    last_rs <= rs;
end

assign exception = _break_ || _syscall_ || (_teq_ && rs == rt);

assign cause = _syscall_ ? 5'b1000 : (_break_ ? 5'b1001 : (_teq_ ? 5'b1
101 : 5'b00000));

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_ || _lbu_ || _lb_ || _lh_ || _lhu_ || _sb_ || _sh_;

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

assign M1[2] = _break_ || _syscall_ || _eret_ || (_teq_ && rs == rt) ||
 (_div_ && div_start) || (_div_ && div_busy) || (_divu_ && divu_start)
|| (_divu_ && divu_busy);

assign M1[1] = _addu_ || _subu_ || _ori_ || _sll_ || _lw_ || _sw_ || _j
_ || _add_ || _addi_ || _addiu_ || _and_ || _andi_ || _jal_ || _lui_ ||
 _nor_ || _or_ || _sllv_ || _slt_ || _slti_ || _sltiu_ || _sltu_ || _sr
a_ || _srav_ || _srl_ || _srlv_ || _sub_ || _xor_ || _xori_ || _mul_ ||
```

```
 _mult_ || _multu_ || _lbu_ || _lhu_ || _lb_ || _lh_ || _sb_ || _sh_ ||
 _mfhi_ || _mflo_ || _mthi_ || _mtlo_ || _mfc0_ || _mtc0_ || _clz_ || (
_teq_ && rs != rt) || (_beq_ && zero == 0) || (_bne_ && zero == 1) || (
_div_ && !div_busy && !div_start) || (_divu_ && !divu_busy && !divu_sta
rt) || (_bgez_ && rs[31] == 1);

assign M1[0] = (_beq_ == 1 && zero == 1) || (_j_ == 1) || (_bne_ == 1 &
& zero == 0) || (_jal_ == 1) || (_bgez_ == 1 && rs[31] == 0) || (_div_
&& div_start) || (_div_ && div_busy) || (_divu_ && divu_start) || (_div
u_ && divu_busy);

assign M2 = _addu_ || _subu_ || _ori_ || _lw_ || _sw_ || _beq_ || _add_
 || _addi_ || _addiu_ || _and_ || _andi_ || _bne_ || _nor_ || _or_ || _
sllv_ || _slt_ || _slti_ || _sltiu_ || _sltu_ || _srav_ || _srlv_ || _s
ub_ || _xor_ || _xori_ || _lbu_ || _lhu_ || _lb_ || _lh_ || _sb_ || _sh
_;

assign M3[1] = _lw_ || _sw_ || _addi_ || _addiu_ || _slti_ || _lhu_ ||
_lbu_ || _lh_ || _lb_ || _sb_ || _sh_;

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

assign M4[3] = _mul_ || _mfhi_ || _mflo_ || _mfc0_;

assign M4[2] = _lw_ || _jal_ || _jalr_ || _lhu_ || _lh_;

assign M4[1] = _lw_ || _jal_ || _mul_ || _jalr_ || _lbu_ || _lb_ || _mf
lo_;

assign M4[0] = _addu_ || _subu_ || _ori_ || _sll_ || _add_ || _addi_ ||
 _addiu_ || _and_ || _andi_ || _jal_ || _lui_ || _nor_ || _or_ || _sllv
_ || _slt_ || _slti_ || _sltiu_ || _sltu_ || _sra_ || _srav_ || _srl_ |
| _srlv_ || _sub_ || _xor_ || _xori_ || _mul_ || _lbu_ || _lhu_ || _mfh
i_ || _jalr_;

assign M5[1] = _jal_;

assign M5[0] = _ori_ || _lw_ || _addi_ || _addiu_ || _andi_ || _lui_ ||
 _slti_ || _sltiu_ || _xori_ || _lbu_ || _lhu_ || _lb_ || _lh_ || _mfc0
_;

assign M6[2] = _divu_;

assign M6[1] = _div_ || _multu_;
```

```
assign M6[0] = _div_ ||_mult_;

assign M7[2] = _divu_;

assign M7[1] = _div_ || _multu_;

assign M7[0] = _div_ ||_mult_;

assign RF_W = _addu_ || _subu_ || _ori_ || _sll_ || _lw_ || _add_ || _a
ddi_ || _addiu_ || _and_ || _andi_ || _jal_ || _mul_ || _mfhi_ || _mflo
_ || _mfc0_ || _clz_ || _lui_ || _nor_ || _or_ || _sllv_ || _slt_ || _s
lti_ || _sltiu_ || _sltu_ || _sra_ || _srav_ || _srl_ || _srlv_ || _sub
_ || _xor_ || _xori_ || _jalr_ || _lbu_ || _lhu_ || _lb_ || _lh_;

assign DM_cs = _lw_ || _sw_ || _lb_ || _lbu_ || _lh_ || _lhu_ || _sb_ |
| _sh_;

assign DM_R = _lw_ || _lb_ || _lbu_ || _lh_ || _lhu_;

assign DM_W_W = _sw_;
assign DM_W_H = _sh_;
assign DM_W_B = _sb_;

assign HI_W = _div_ || _divu_ || _mult_ || _multu_ || _mthi_;

assign LO_W = _div_ || _divu_ || _mult_ || _multu_ || _mtlo_;

assign M1_temp1 = (M1[0] == 1) ? add_out : last_rs;
assign M1_temp2 = (M1[0] == 1) ? concat : npc;
assign M1_temp3 = (M1[0] == 1) ? pc : exc_addr;
assign M1_temp4 = (M1[1] == 1) ? M1_temp2 : M1_temp1;
assign newpc = (M1[2] == 1) ? M1_temp3 : M1_temp4;

assign alu_a = M2 ? rs : zext5;

assign M3_temp = M3[0] ? zext16 : rt;
assign alu_b = M3[1] ? sext16 : M3_temp;

assign M4_temp1 = M4[0] ? alu_out : clz_output;
assign M4_temp2 = M4[0] ? DM_data_out_Z_Ext8 : DM_data_out_S_Ext8;
assign M4_temp3 = M4[0] ? DM_data_out_Z_Ext16 : DM_data_out_S_Ext16;
assign M4_temp4 = M4[0] ? npc : DM_data_out;
assign M4_temp5 = M4[0] ? hi_output : cp0_rdata;
```

```
assign M4_temp6 = M4[0] ? MULT_result[31:0] : lo_output;
assign M4_temp7 = M4[1] ? M4_temp2 : M4_temp1;
assign M4_temp8 = M4[1] ? M4_temp4 : M4_temp3;
assign M4_temp9 = M4[1] ? M4_temp6 : M4_temp5;
assign M4_temp10 = M4[2] ? M4_temp8 : M4_temp7;
assign ref_rd = M4[3] ? M4_temp9 : M4_temp10;

assign M5_temp = M5[0] ? rtc : im_rdc;
assign ref_rdc = M5[1] ? 31 : M5_temp;

assign M6_temp1 = M6[0] ? MULT_result[31:0] : rs;
assign M6_temp2 = M6[0] ? div_q : MULTU_result[31:0];
assign M6_temp3 = M6[1] ? M6_temp2 : M6_temp1;
assign lo_input = M6[2] ? divu_q : M6_temp3;

assign M7_temp1 = M7[0] ? MULT_result[63:32] : rs;
assign M7_temp2 = M7[0] ? div_r : MULTU_result[63:32];
assign M7_temp3 = M7[1] ? M7_temp2 : M7_temp1;
assign hi_input = M7[2] ? divu_r : M7_temp3;

assign DM_addr = alu_out;

assign DM_data_in = rt;

assign DM_data_out_Z_Ext8 = {24'b0, DM_data_out[7:0]};
assign DM_data_out_S_Ext8 = {{24{DM_data_out[7]}}, DM_data_out[7:0]};
assign DM_data_out_Z_Ext16 = {16'b0, DM_data_out[15:0]};
assign DM_data_out_S_Ext16 = {{16{DM_data_out[15]}}, DM_data_out[15:0]}
;

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));

cp0 CP0(.clk(clk), .rst(rst), .exception(exception), .mfc0(_mfc0_), .mt
c0(_mtc0_), .eret(_eret_), .pc(pc), .rd(im_rdc), .wdata(rt), .cause(cau
se), .rdata(cp0_rdata), .status(cp0_status), .exc_addr(exc_addr));

clz clz_inst(.input_data(rs), .zero_number(clz_output));
```

```
hi hi_inst(.clk(clk), .rst(rst), .HI_W(HI_W), .hi_input(hi_input), .hi_
output(hi_output));
lo lo_inst(.clk(clk), .rst(rst), .LO_W(LO_W), .lo_input(lo_input), .lo_
output(lo_output));

MULT1 MULT1_inst(.a(rs), .b(rt), .result(MULT_result));
MULTU1 MULTU1_inst(.a(rs), .b(rt), .result(MULTU_result));

DIV DIV_inst(.dividend(rs), .divisor(rt), .start(div_start), .clock(clk
), .reset(rst), .q(div_q), .r(div_r), .busy(div_busy));
DIVU DIVU_inst(.dividend(rs), .divisor(rt), .start(divu_start), .clock(
clk), .reset(rst), .q(divu_q), .r(divu_r), .busy(divu_busy));
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] 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
```

[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
```

[cp0.v]

```
`timescale 1ns / 1ps
module cp0(
    input clk,
    input rst,
    input exception,
    input mfc0,
    input mtc0,
    input eret,
    input [31:0] pc,
    input [4:0] rd,
    input [31:0] wdata,
    input [4:0] cause,
    output [31:0] rdata,
    output [31:0] status,
    output [31:0] exc_addr
);

reg [31:0]cp0reg[31:0];
parameter STATUS = 12,
          CAUSE = 13,
          EPC = 14;
parameter SYSCALL = 5'b1000,
          BREAK = 5'b1001,
          TEQ = 5'b1101;
parameter STATUS_SYSCALL = 8,
          STATUS_BREAK = 9,
          STATUS_TEQ = 10;
// 到底0屏蔽还是1屏蔽？？？
wire legalException = exception;
assign exc_addr = eret ? cp0reg[EPC] : 32'h00400004;
assign status = cp0reg[STATUS];
assign rdata = mfc0 ? cp0reg[rd] : 32'b0;
// assign rdata = mfc0 ? 32'b1 : 32'b0;
always @(posedge clk or posedge rst) begin
    if(rst)
    begin
        cp0reg[0]<=32'b0;
        cp0reg[1]<=32'b0;
        cp0reg[2]<=32'b0;
        cp0reg[3]<=32'b0;
        cp0reg[4]<=32'b0;
        cp0reg[5]<=32'b0;
        cp0reg[6]<=32'b0;
```

```
        cp0reg[7]<=32'b0;
        cp0reg[8]<=32'b0;
        cp0reg[9]<=32'b0;
        cp0reg[10]<=32'b0;
        cp0reg[11]<=32'b0;
        cp0reg[12]<=32'b0;
        cp0reg[13]<=32'b0;
        cp0reg[14]<=32'b0;
        cp0reg[15]<=32'b0;
        cp0reg[16]<=32'b0;
        cp0reg[17]<=32'b0;
        cp0reg[18]<=32'b0;
        cp0reg[19]<=32'b0;
        cp0reg[20]<=32'b0;
        cp0reg[21]<=32'b0;
        cp0reg[22]<=32'b0;
        cp0reg[23]<=32'b0;
        cp0reg[24]<=32'b0;
        cp0reg[25]<=32'b0;
        cp0reg[26]<=32'b0;
        cp0reg[27]<=32'b0;
        cp0reg[28]<=32'b0;
        cp0reg[29]<=32'b0;
        cp0reg[30]<=32'b0;
        cp0reg[31]<=32'b0;
    end
    else
    begin
        if(mtc0)
            cp0reg[rd] <= wdata;
        if(legalException)
        begin
            cp0reg[STATUS] <= cp0reg[STATUS] << 5;
            cp0reg[CAUSE][6:2] <= cause;
            cp0reg[EPC] <= pc;
        end
        if(eret)
            cp0reg[STATUS] <= cp0reg[STATUS] >> 5;
    end
end

endmodule
```

[clz.v]

```
module clz (
    input [31:0] input_data,
    output [31:0] zero_number
);
assign zero_number =
(input_data[31:0] == 32'b0) ? 32 :
(input_data[31:1] == 31'b0) ? 31 :
(input_data[31:2] == 30'b0) ? 30 :
(input_data[31:3] == 29'b0) ? 29 :
(input_data[31:4] == 28'b0) ? 28 :
(input_data[31:5] == 27'b0) ? 27 :
(input_data[31:6] == 26'b0) ? 26 :
(input_data[31:7] == 25'b0) ? 25 :
(input_data[31:8] == 24'b0) ? 24 :
(input_data[31:9] == 23'b0) ? 23 :
(input_data[31:10] == 22'b0) ? 22 :
(input_data[31:11] == 21'b0) ? 21 :
(input_data[31:12] == 20'b0) ? 20 :
(input_data[31:13] == 19'b0) ? 19 :
(input_data[31:14] == 18'b0) ? 18 :
(input_data[31:15] == 17'b0) ? 17 :
(input_data[31:16] == 16'b0) ? 16 :
(input_data[31:17] == 15'b0) ? 15 :
(input_data[31:18] == 14'b0) ? 14 :
(input_data[31:19] == 13'b0) ? 13 :
(input_data[31:20] == 12'b0) ? 12 :
(input_data[31:21] == 11'b0) ? 11 :
(input_data[31:22] == 10'b0) ? 10 :
(input_data[31:23] == 9'b0) ? 9 :
(input_data[31:24] == 8'b0) ? 8 :
(input_data[31:25] == 7'b0) ? 7 :
(input_data[31:26] == 6'b0) ? 6 :
(input_data[31:27] == 5'b0) ? 5 :
(input_data[31:28] == 4'b0) ? 4 :
(input_data[31:29] == 3'b0) ? 3 :
(input_data[31:30] == 2'b0) ? 2 :
(input_data[31] == 1'b0) ? 1 : 0;

endmodule
```

[hi.v]

```
module hi (
```

```
    input clk,
    input rst,
    input HI_W,
    input [31:0] hi_input,
    output [31:0] hi_output
);

reg [31:0] hi_reg;
assign hi_output = hi_reg;
always @(posedge clk or posedge rst) begin
    if(rst)
        hi_reg <= 32'b0;
    else
    begin
        if(HI_W)
            hi_reg <= hi_input;
    end
end

endmodule
```

[lo.v]

```
module lo (
    input clk,
    input rst,
    input LO_W,
    input [31:0] lo_input,
    output [31:0] lo_output
);

reg [31:0] lo_reg;
assign lo_output = lo_reg;
always @(posedge clk or posedge rst) begin
    if(rst)
        lo_reg <= 32'b0;
    else
    begin
        if(LO_W)
            lo_reg <= lo_input;
    end
end
```

```
endmodule
```

[MULT1.v]

```
module MULT1 (
    input [31:0] a,
    input [31:0] b,
    output [63:0] result
);

wire sign;
assign sign = (a[31] == 0) ? ((b[31] == 0) ? 1'b0 : 1'b1) : ((b[31] ==
0) ? 1'b1 : 1'b0);
wire [31:0] posa;
wire [31:0] posb;
assign posa = a[31] ? ~a + 1 : a;
assign posb = b[31] ? ~b + 1 : b;
wire [63:0] adder[62:0];
assign adder[0] = posb[0] ? {32'b0, posa} : 64'b0;
assign adder[1] = posb[1] ? {31'b0, posa, 1'b0} : 64'b0;
assign adder[2] = posb[2] ? {30'b0, posa, 2'b0} : 64'b0;
assign adder[3] = posb[3] ? {29'b0, posa, 3'b0} : 64'b0;
assign adder[4] = posb[4] ? {28'b0, posa, 4'b0} : 64'b0;
assign adder[5] = posb[5] ? {27'b0, posa, 5'b0} : 64'b0;
assign adder[6] = posb[6] ? {26'b0, posa, 6'b0} : 64'b0;
assign adder[7] = posb[7] ? {25'b0, posa, 7'b0} : 64'b0;
assign adder[8] = posb[8] ? {24'b0, posa, 8'b0} : 64'b0;
assign adder[9] = posb[9] ? {23'b0, posa, 9'b0} : 64'b0;
assign adder[10] = posb[10] ? {22'b0, posa, 10'b0} : 64'b0;
assign adder[11] = posb[11] ? {21'b0, posa, 11'b0} : 64'b0;
assign adder[12] = posb[12] ? {20'b0, posa, 12'b0} : 64'b0;
assign adder[13] = posb[13] ? {19'b0, posa, 13'b0} : 64'b0;
assign adder[14] = posb[14] ? {18'b0, posa, 14'b0} : 64'b0;
assign adder[15] = posb[15] ? {17'b0, posa, 15'b0} : 64'b0;
assign adder[16] = posb[16] ? {16'b0, posa, 16'b0} : 64'b0;
assign adder[17] = posb[17] ? {15'b0, posa, 17'b0} : 64'b0;
assign adder[18] = posb[18] ? {14'b0, posa, 18'b0} : 64'b0;
assign adder[19] = posb[19] ? {13'b0, posa, 19'b0} : 64'b0;
assign adder[20] = posb[20] ? {12'b0, posa, 20'b0} : 64'b0;
assign adder[21] = posb[21] ? {11'b0, posa, 21'b0} : 64'b0;
assign adder[22] = posb[22] ? {10'b0, posa, 22'b0} : 64'b0;
assign adder[23] = posb[23] ? {9'b0, posa, 23'b0} : 64'b0;
assign adder[24] = posb[24] ? {8'b0, posa, 24'b0} : 64'b0;
```

```
assign adder[25] = posb[25] ? {7'b0, posa, 25'b0} : 64'b0;
assign adder[26] = posb[26] ? {6'b0, posa, 26'b0} : 64'b0;
assign adder[27] = posb[27] ? {5'b0, posa, 27'b0} : 64'b0;
assign adder[28] = posb[28] ? {4'b0, posa, 28'b0} : 64'b0;
assign adder[29] = posb[29] ? {3'b0, posa, 29'b0} : 64'b0;
assign adder[30] = posb[30] ? {2'b0, posa, 30'b0} : 64'b0;
assign adder[31] = posb[31] ? {1'b0, posa, 31'b0} : 64'b0;

assign adder[32] = adder[0] + adder[1];
assign adder[33] = adder[2] + adder[3];
assign adder[34] = adder[4] + adder[5];
assign adder[35] = adder[6] + adder[7];
assign adder[36] = adder[8] + adder[9];
assign adder[37] = adder[10] + adder[11];
assign adder[38] = adder[12] + adder[13];
assign adder[39] = adder[14] + adder[15];
assign adder[40] = adder[16] + adder[17];
assign adder[41] = adder[18] + adder[19];
assign adder[42] = adder[20] + adder[21];
assign adder[43] = adder[22] + adder[23];
assign adder[44] = adder[24] + adder[25];
assign adder[45] = adder[26] + adder[27];
assign adder[46] = adder[28] + adder[29];
assign adder[47] = adder[30] + adder[31];

assign adder[48] = adder[32] + adder[33];
assign adder[49] = adder[34] + adder[35];
assign adder[50] = adder[36] + adder[37];
assign adder[51] = adder[38] + adder[39];
assign adder[52] = adder[40] + adder[41];
assign adder[53] = adder[42] + adder[43];
assign adder[54] = adder[44] + adder[45];
assign adder[55] = adder[46] + adder[47];

assign adder[56] = adder[48] + adder[49];
assign adder[57] = adder[50] + adder[51];
assign adder[58] = adder[52] + adder[53];
assign adder[59] = adder[54] + adder[55];

assign adder[60] = adder[56] + adder[57];
assign adder[61] = adder[58] + adder[59];

assign adder[62] = adder[60] + adder[61];
```

```
assign result = sign ? ~adder[62] + 1 : adder[62];

endmodule
```

[MULTU1.v]

```
module MULTU1 (
    input [31:0] a,
    input [31:0] b,
    output [63:0] result
);
wire [63:0] adder[62:0];
assign adder[0] = b[0] ? {32'b0, a} : 64'b0;
assign adder[1] = b[1] ? {31'b0, a, 1'b0} : 64'b0;
assign adder[2] = b[2] ? {30'b0, a, 2'b0} : 64'b0;
assign adder[3] = b[3] ? {29'b0, a, 3'b0} : 64'b0;
assign adder[4] = b[4] ? {28'b0, a, 4'b0} : 64'b0;
assign adder[5] = b[5] ? {27'b0, a, 5'b0} : 64'b0;
assign adder[6] = b[6] ? {26'b0, a, 6'b0} : 64'b0;
assign adder[7] = b[7] ? {25'b0, a, 7'b0} : 64'b0;
assign adder[8] = b[8] ? {24'b0, a, 8'b0} : 64'b0;
assign adder[9] = b[9] ? {23'b0, a, 9'b0} : 64'b0;
assign adder[10] = b[10] ? {22'b0, a, 10'b0} : 64'b0;
assign adder[11] = b[11] ? {21'b0, a, 11'b0} : 64'b0;
assign adder[12] = b[12] ? {20'b0, a, 12'b0} : 64'b0;
assign adder[13] = b[13] ? {19'b0, a, 13'b0} : 64'b0;
assign adder[14] = b[14] ? {18'b0, a, 14'b0} : 64'b0;
assign adder[15] = b[15] ? {17'b0, a, 15'b0} : 64'b0;
assign adder[16] = b[16] ? {16'b0, a, 16'b0} : 64'b0;
assign adder[17] = b[17] ? {15'b0, a, 17'b0} : 64'b0;
assign adder[18] = b[18] ? {14'b0, a, 18'b0} : 64'b0;
assign adder[19] = b[19] ? {13'b0, a, 19'b0} : 64'b0;
assign adder[20] = b[20] ? {12'b0, a, 20'b0} : 64'b0;
assign adder[21] = b[21] ? {11'b0, a, 21'b0} : 64'b0;
assign adder[22] = b[22] ? {10'b0, a, 22'b0} : 64'b0;
assign adder[23] = b[23] ? {9'b0, a, 23'b0} : 64'b0;
assign adder[24] = b[24] ? {8'b0, a, 24'b0} : 64'b0;
assign adder[25] = b[25] ? {7'b0, a, 25'b0} : 64'b0;
assign adder[26] = b[26] ? {6'b0, a, 26'b0} : 64'b0;
assign adder[27] = b[27] ? {5'b0, a, 27'b0} : 64'b0;
assign adder[28] = b[28] ? {4'b0, a, 28'b0} : 64'b0;
assign adder[29] = b[29] ? {3'b0, a, 29'b0} : 64'b0;
assign adder[30] = b[30] ? {2'b0, a, 30'b0} : 64'b0;
```

```
assign adder[31] = b[31] ? {1'b0, a, 31'b0} : 64'b0;

assign adder[32] = adder[0] + adder[1];
assign adder[33] = adder[2] + adder[3];
assign adder[34] = adder[4] + adder[5];
assign adder[35] = adder[6] + adder[7];
assign adder[36] = adder[8] + adder[9];
assign adder[37] = adder[10] + adder[11];
assign adder[38] = adder[12] + adder[13];
assign adder[39] = adder[14] + adder[15];
assign adder[40] = adder[16] + adder[17];
assign adder[41] = adder[18] + adder[19];
assign adder[42] = adder[20] + adder[21];
assign adder[43] = adder[22] + adder[23];
assign adder[44] = adder[24] + adder[25];
assign adder[45] = adder[26] + adder[27];
assign adder[46] = adder[28] + adder[29];
assign adder[47] = adder[30] + adder[31];

assign adder[48] = adder[32] + adder[33];
assign adder[49] = adder[34] + adder[35];
assign adder[50] = adder[36] + adder[37];
assign adder[51] = adder[38] + adder[39];
assign adder[52] = adder[40] + adder[41];
assign adder[53] = adder[42] + adder[43];
assign adder[54] = adder[44] + adder[45];
assign adder[55] = adder[46] + adder[47];

assign adder[56] = adder[48] + adder[49];
assign adder[57] = adder[50] + adder[51];
assign adder[58] = adder[52] + adder[53];
assign adder[59] = adder[54] + adder[55];

assign adder[60] = adder[56] + adder[57];
assign adder[61] = adder[58] + adder[59];

assign adder[62] = adder[60] + adder[61];

assign result = adder[62];

endmodule
```

[DIV.v]

```
module DIV(
    input [31:0] dividend,
    input [31:0] divisor,
    input start,
    input clock,
    input reset,
    output [31:0] q,
    output [31:0] r,
    output busy
);

wire [31:0] posdividend;
wire [31:0] posdivisor;
reg [63:0] temp;
reg [6:0] count;
reg regbusy;

assign posdividend = dividend[31] ? ~dividend + 1 : dividend;
assign posdivisor = divisor[31] ? ~divisor + 1 : divisor;

always@(clock or posedge reset)
begin
    if(reset)
    begin
        temp <= 0;
        count <= 32;
        regbusy <= 0;
    end
    else if(clock == 0) // 时钟下降沿
    begin
        if(start)
        begin
            regbusy <= 1;
            temp <= {32'b0, posdividend};
            count <= 32;
        end
        else
        begin
            if(count)
            begin
                temp = {temp[62:0], 1'b0};
                if(temp[63:32] >= posdivisor)
                begin
```

```
                    temp[63:32] = temp[63:32] - posdivisor;
                    temp = temp + 1;
                end
                count = count - 1;
            end
        end
    end
    else
    begin
        if(count == 0)
            regbusy = 0;
    end
end

assign q = (dividend[31] ^ divisor[31]) ? ~temp[31:0] + 1 : temp[31:0];

assign r = dividend[31] ? ~temp[63:32] + 1 : temp[63:32];
assign busy = regbusy;

endmodule
```

[DIVU.v]

```
module DIVU(
    input [31:0] dividend,
    input [31:0] divisor,
    input start,
    input clock,
    input reset,
    output [31:0] q,
    output [31:0] r,
    output busy
);

reg [63:0] temp;
reg [6:0] count;
reg regbusy;

always@(clock or posedge reset)
begin
    if(reset)
    begin
        temp <= 0;
```

```
        count <= 32;
        regbusy <= 0;
    end
    else if(clock == 0) // 时钟下降沿
    begin
        if(start)
        begin
            regbusy <= 1;
            temp <= {32'b0, dividend};
            count <= 32;
        end
        else
        begin
            if(count)
            begin
                temp = {temp[62:0], 1'b0};
                if(temp[63:32] >= divisor)
                begin
                    temp[63:32] = temp[63:32] - divisor;
                    temp = temp + 1;
                end
                count = count - 1;
            end
        end
    end
    else
    begin
        if(count == 0)
            regbusy = 0;
    end
end

assign q = temp[31:0];
assign r = temp[63:32];
assign busy = regbusy;

endmodule
```

四、测试模块建模

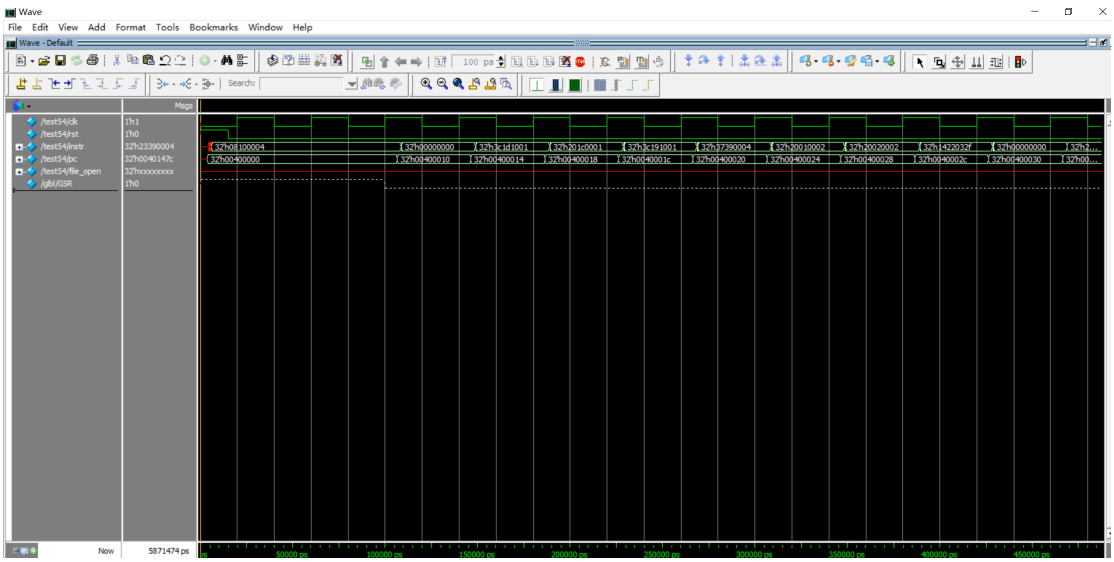
（要求列写各建模模块的 test bench 模块代码）
[cpu_tb.v]

```
`timescale 1ns / 1ps
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/资料
/54 条 CPUtest_2019_5_23/CPU 测试 COE 文件/result_mine.txt");
    clk = 0;
    rst = 1;
    #225 rst = 0;
end

always
begin
    #50 clk = ~clk;
end

always@(posedge clk)
begin
    if(!rst)
    begin
    $fdisplay(file_output, "pc: %h", pc);
    $fdisplay(file_output, "instr: %h", inst);
    $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[
```

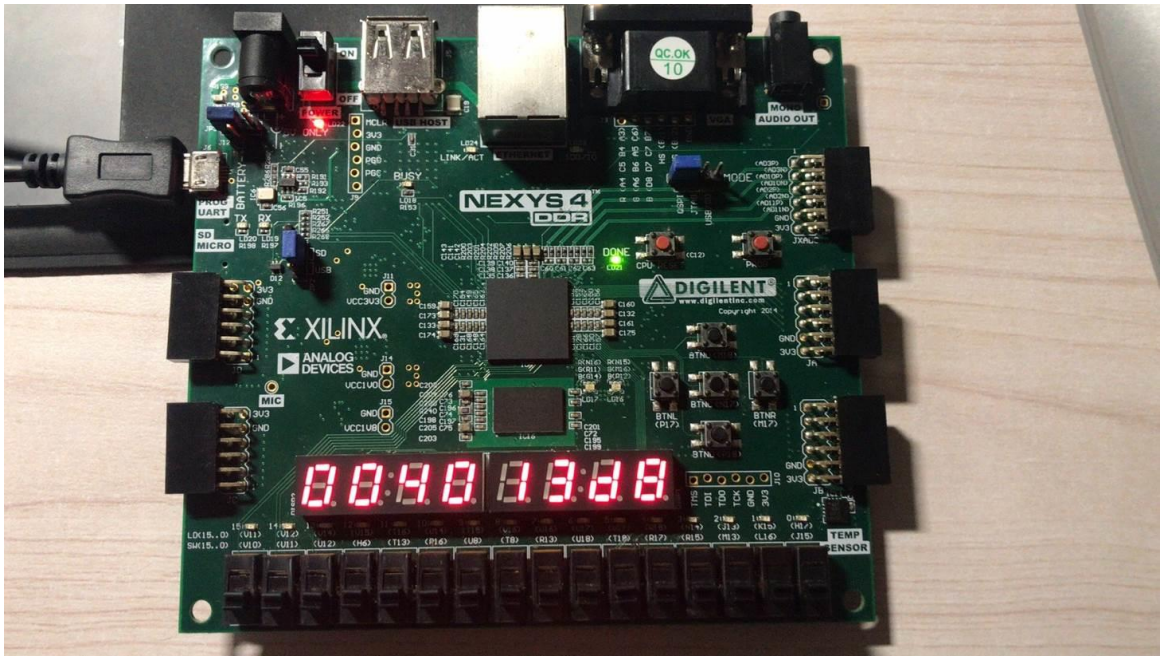
```
7]);
    $fdisplay(file_output, "regfile8: %h", uut.sccpu.cpu_ref.array_reg[
8]);
    $fdisplay(file_output, "regfile9: %h", uut.sccpu.cpu_ref.array_reg[
9]);
    $fdisplay(file_output, "regfile10: %h", uut.sccpu.cpu_ref.array_reg
[10]);
    $fdisplay(file_output, "regfile11: %h", uut.sccpu.cpu_ref.array_reg
[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
```

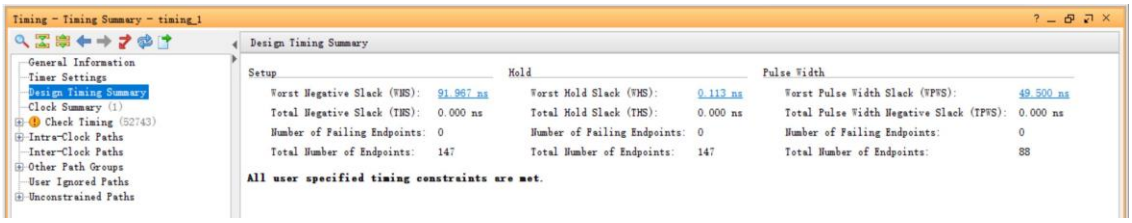
五、后仿真结果展示



六、下板结果展示


NEXYS 4
DDR
DIGILENT
XILINX
ANALOG DEVICES
AUDIO OUT
MIC
TEMP SENSOR

七、时序分析报告


Timing - Timing Summary - timing_1
General Information
Timer Settings
Design Timing Summary
Clock Summary (1)
Check Timing (52743)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths
Design Timing Summary
Setup
Worst Negative Slack (WNS): 91.967 ns
Total Negative Slack (TNS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 147
Hold
Worst Hold Slack (WHS): 0.113 ns
Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 147
Pulse Width
Worst Pulse Width Slack (WPWS): 49.500 ns
Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 88
All user specified timing constraints are met.