同济大学计算机系

计算机组成原理实验报告

学　　号 1953072

姓　　名 肖鹏飞

专　　业 计算机科学与技术

授课老师 张冬冬

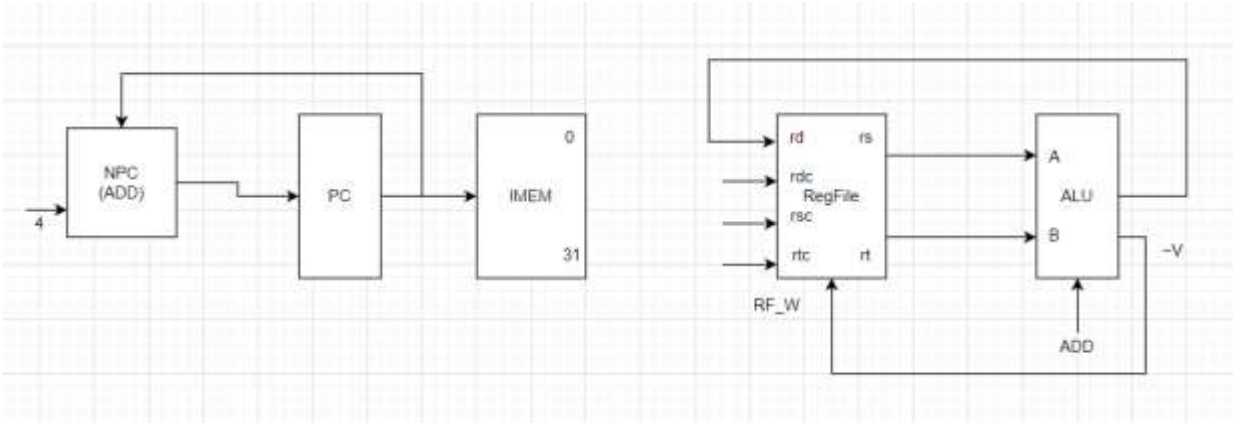
一、实验内容

本次实验内容为：实现 MIPS 架构的 54 条指令单周期 CPU 设计。
具体的指令内容如下：
1.ADD
2.ADDU
3.ADDI
4.ADDIU
5.SUB
6.SUBU
7.AND
8.ANDI
9.OR
10.ORI
11.XOR
12.XORI
13.NOR
14.LUI
15.SLL
16.SLLV
17.SRA
18.SRAV
19.SRL
20.SRLV
21.SLT
22.SLTI
23.SLTU
24.SLTIU
25.BEQ
26.BNE
27.J
28.JAL
29.JR
30.LW
31.SW
32.CLZ
33.DIVU
34.ERET
35.JALR
36.LB
37.LBU
38.LHU
39.SB

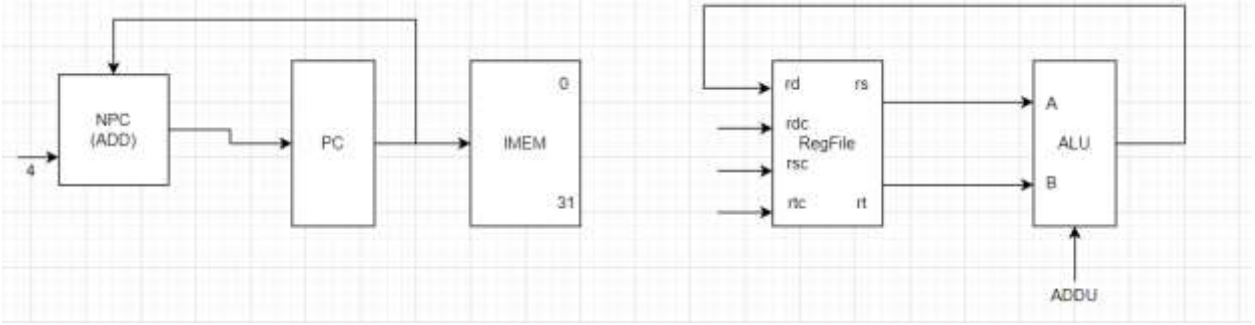
40.SH
41.LH
42.MFC0
43.MFHI
44.MFLO
45.MTC0
46.MTHI
47.MTLO
48.MUL
49.MULTU
50.SYSCALL
51.TEQ
52.BGEZ
53.BREAK
54.DIV

二、硬件逻辑图

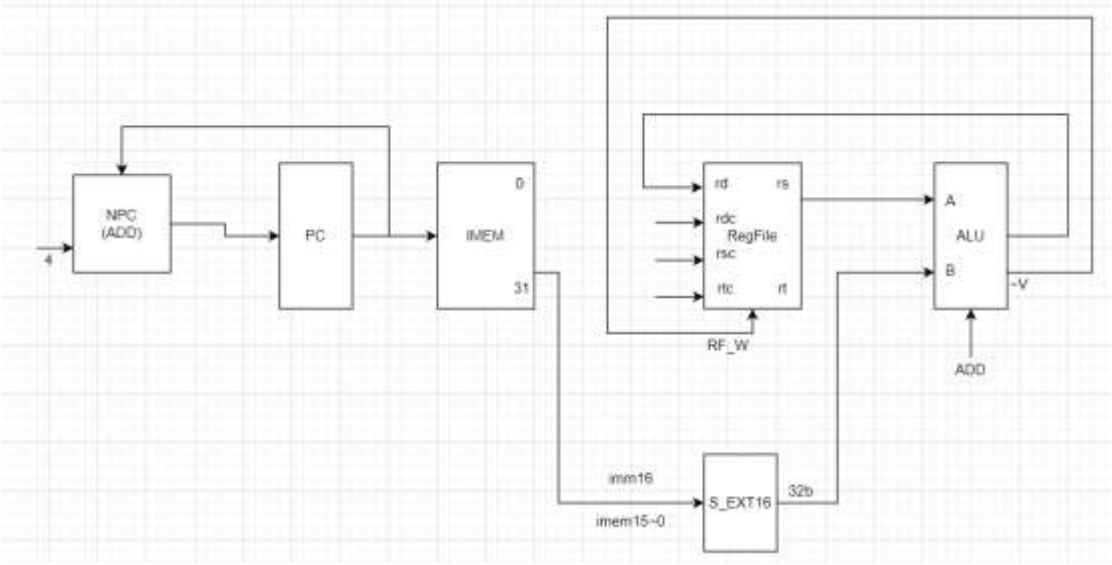
1.ADD



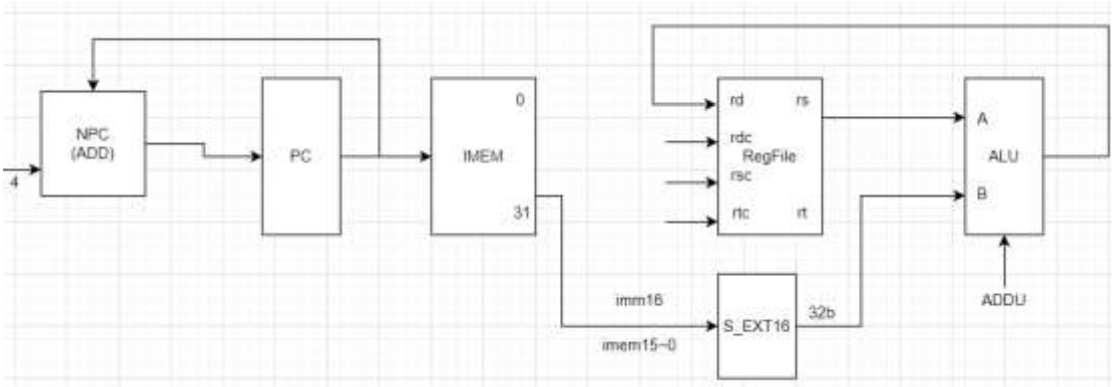
2.ADDU



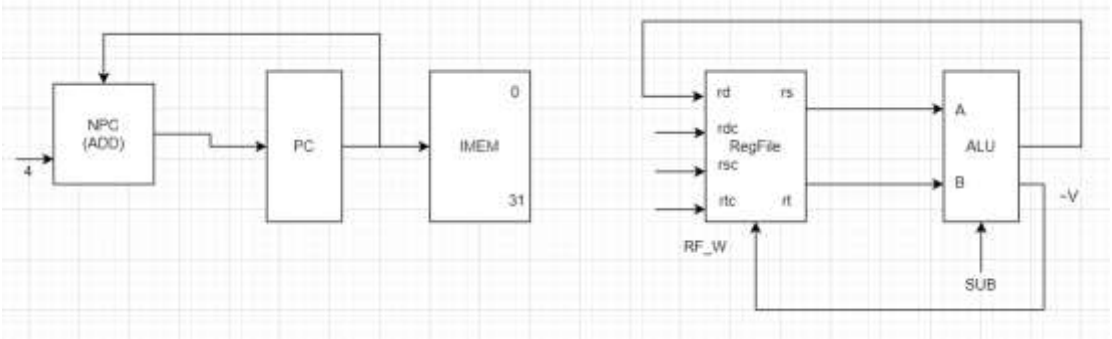
3.ADDI



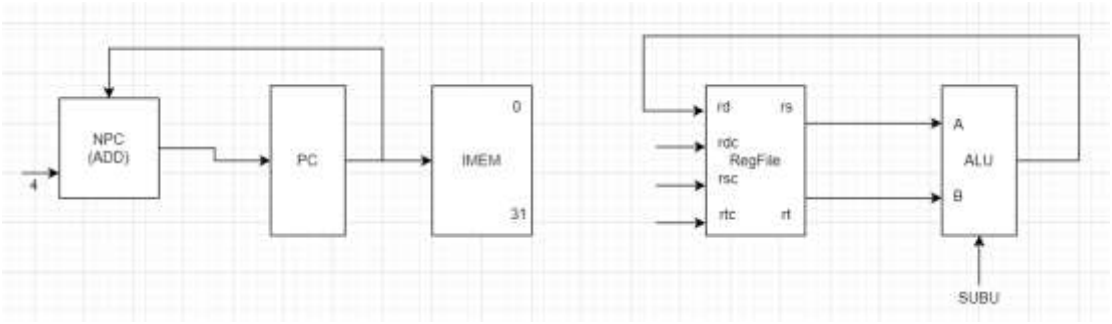
4.ADDIU



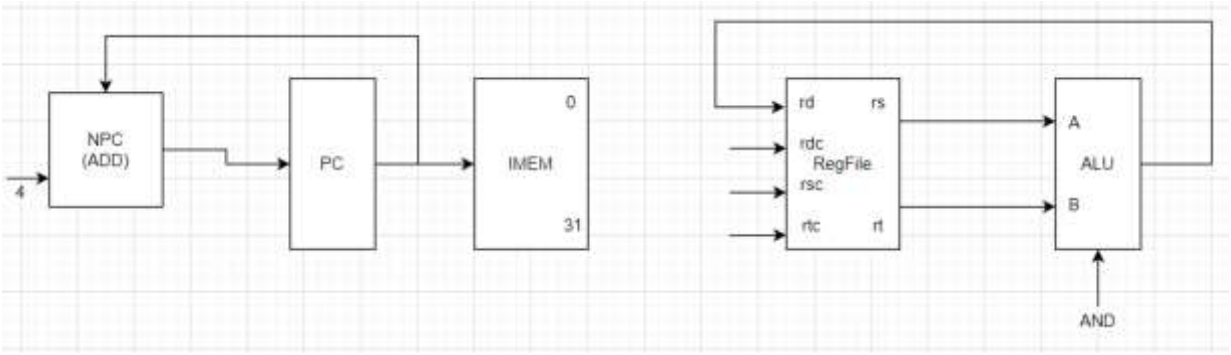
5.SUB



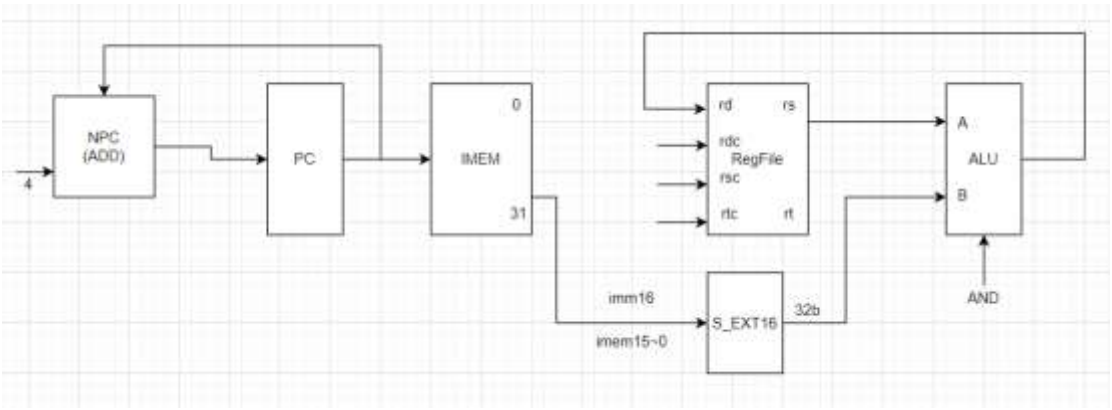
6.SUBU



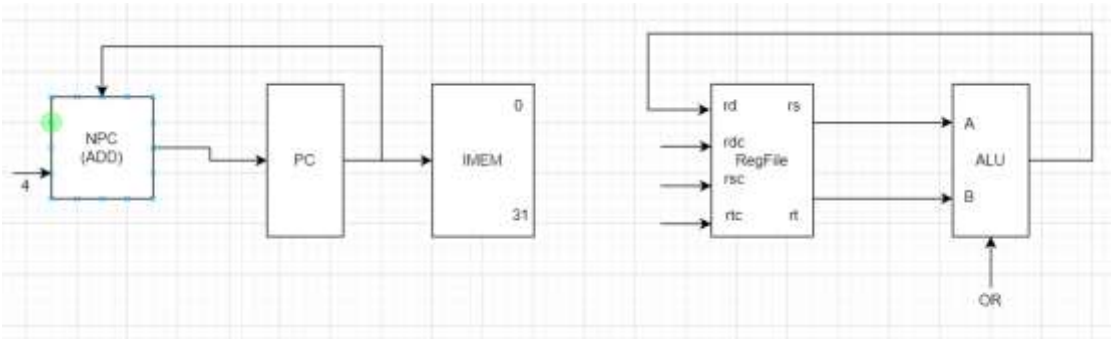
7.AND



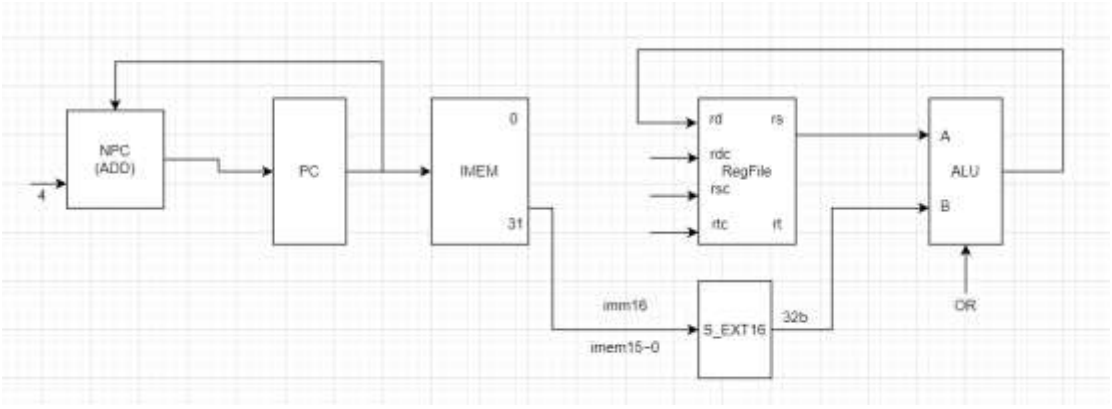
8.ANDI



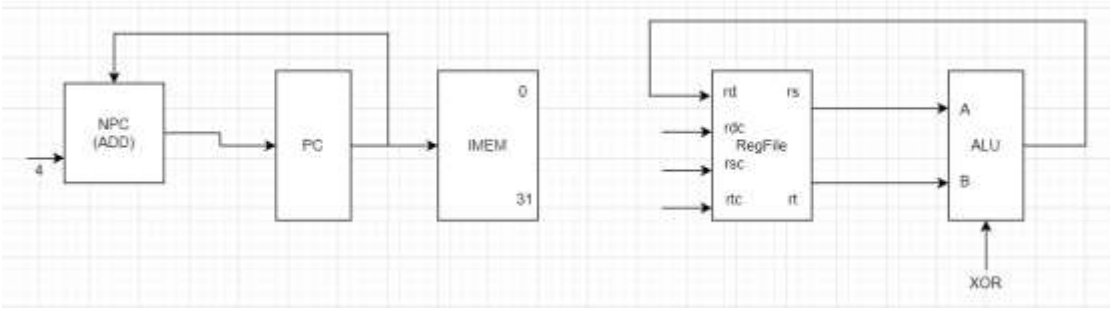
9.OR



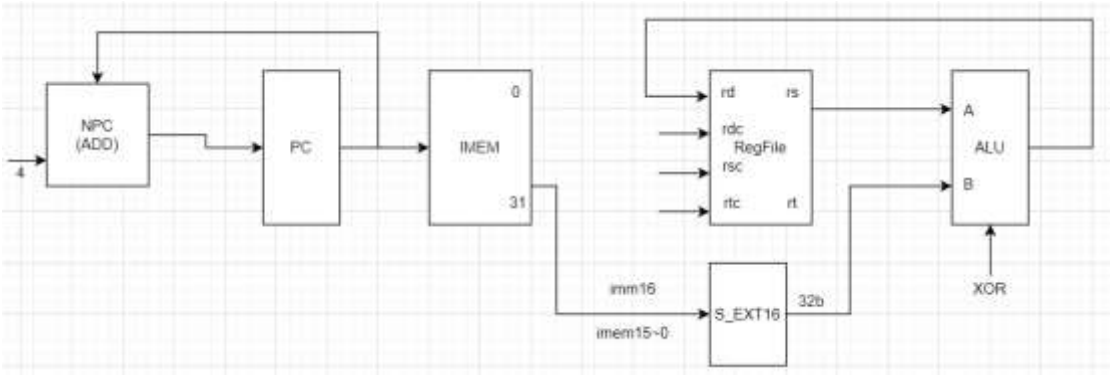
10.ORI



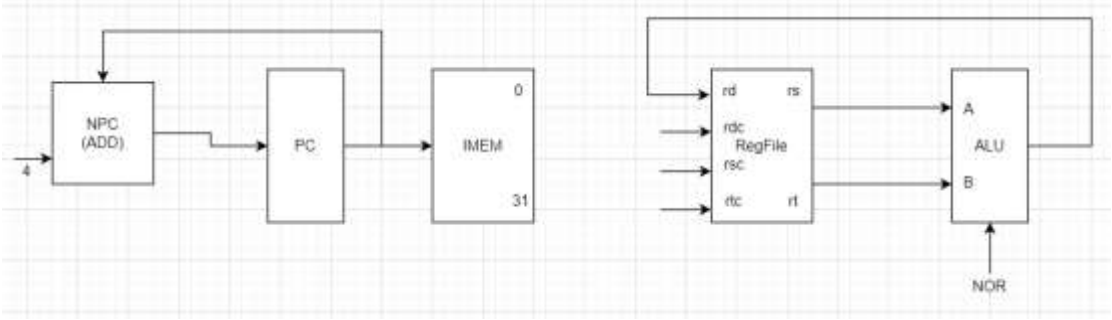
11.XOR



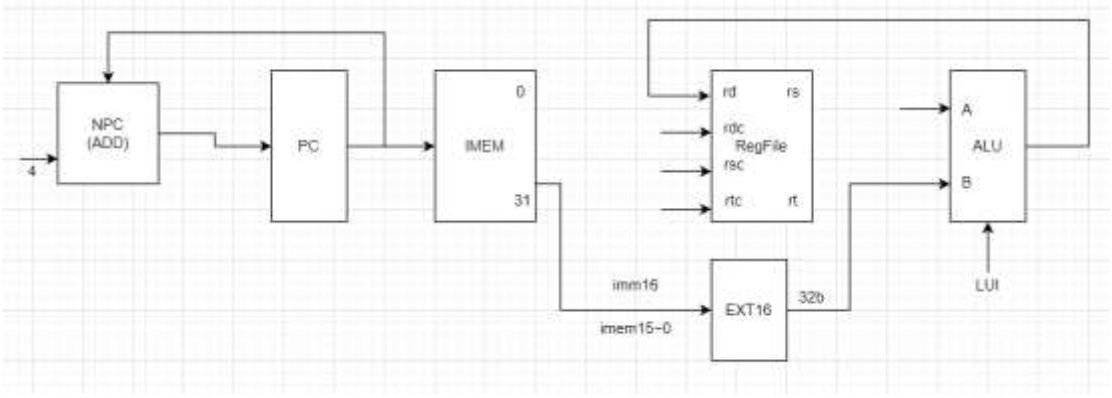
12.XORI



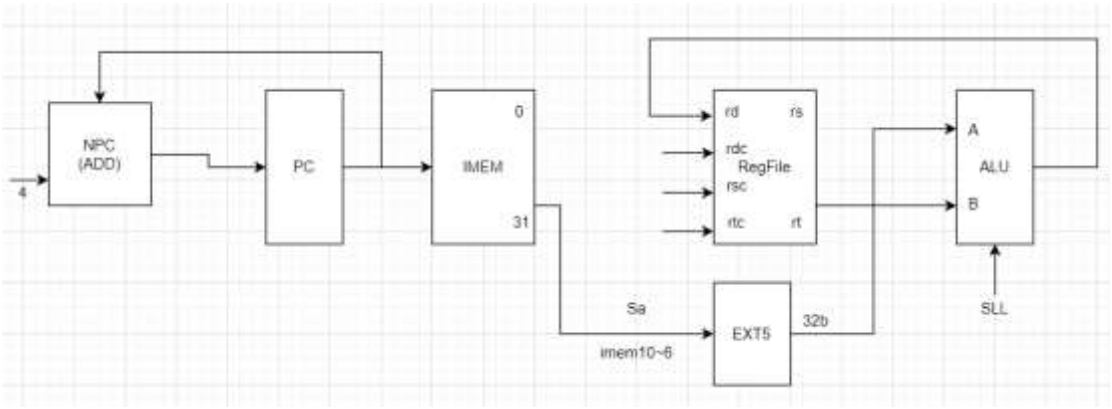
13.NOR



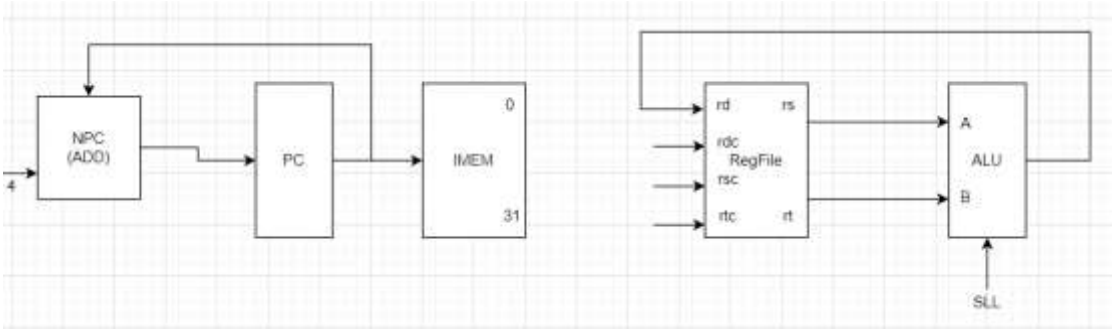
14.LUI



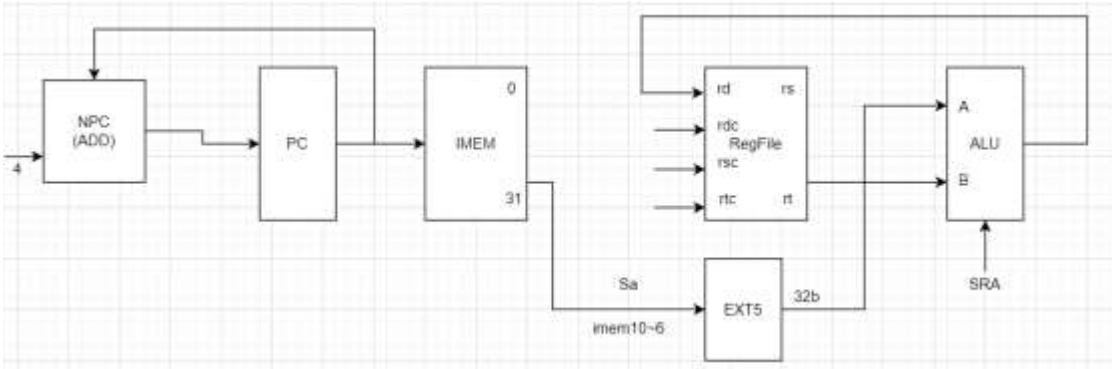
15.SLL



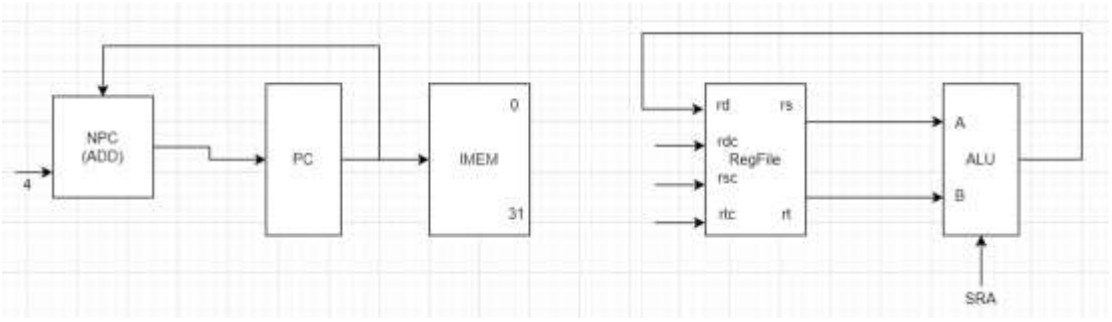
16.SLLV



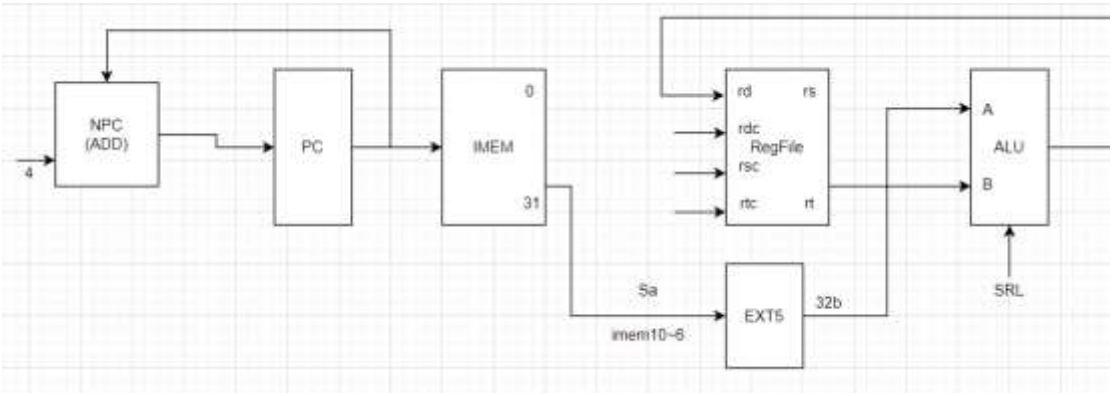
17.SRA



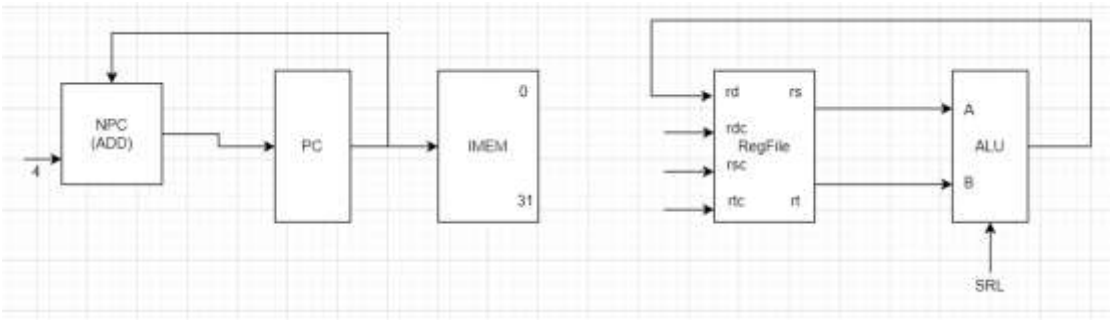
18.SRAV



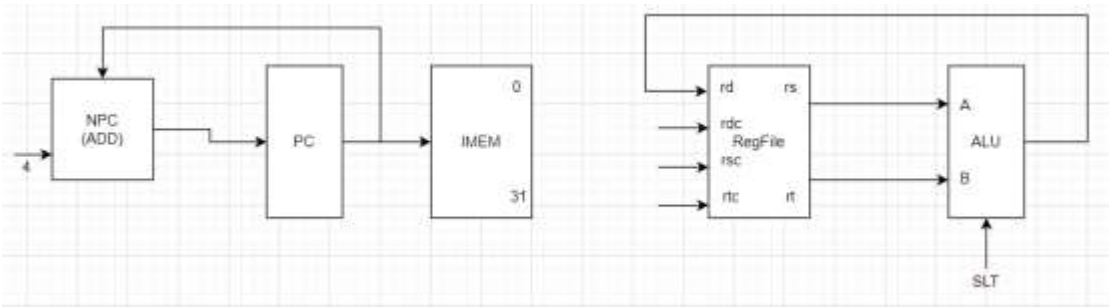
19.SRL



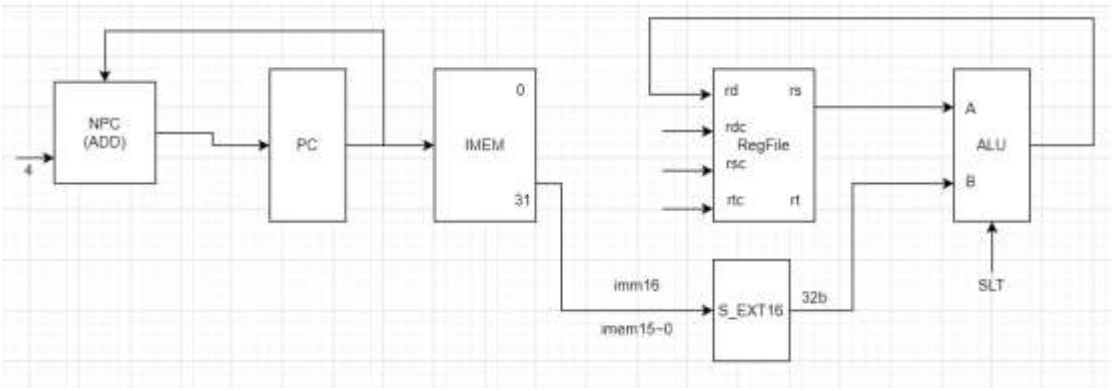
20.SRLV



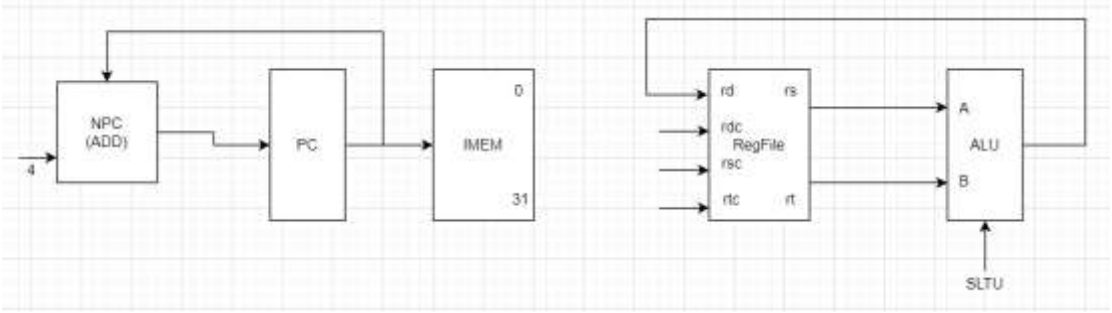
21.SLT



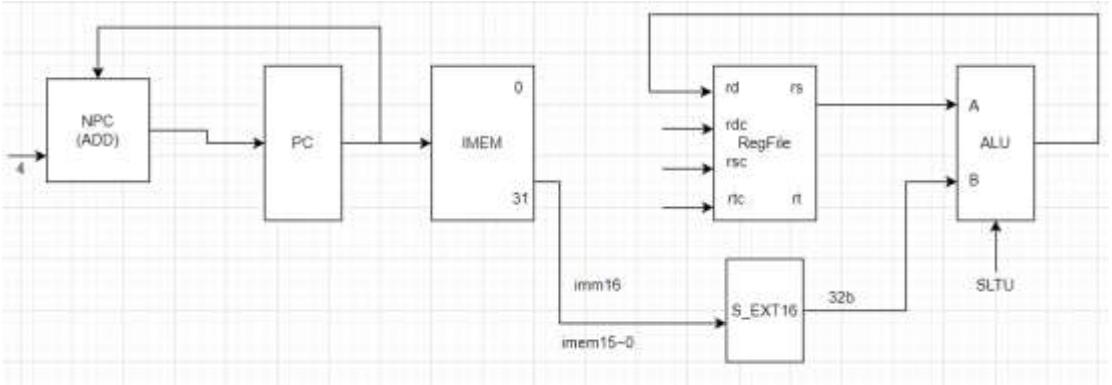
22.SLTI



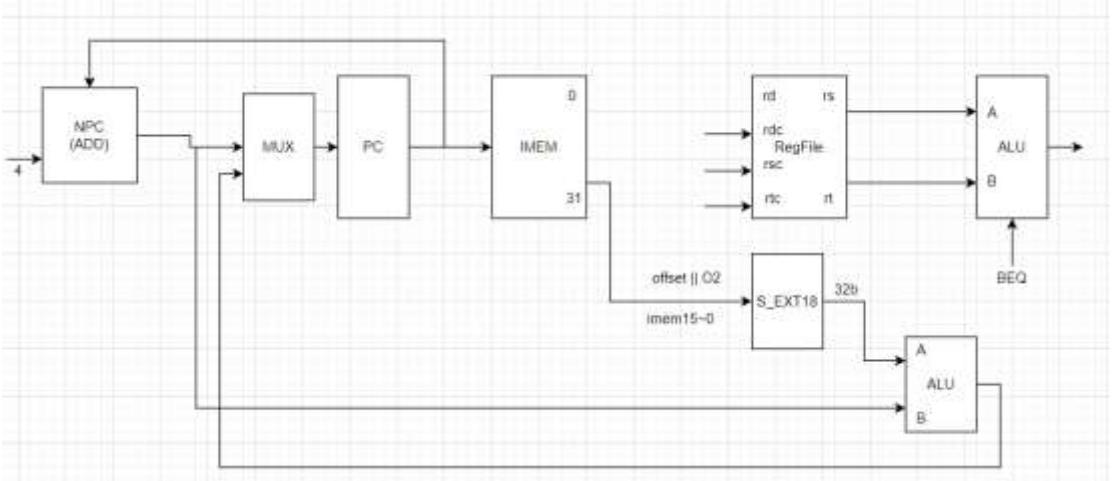
23.SLTU



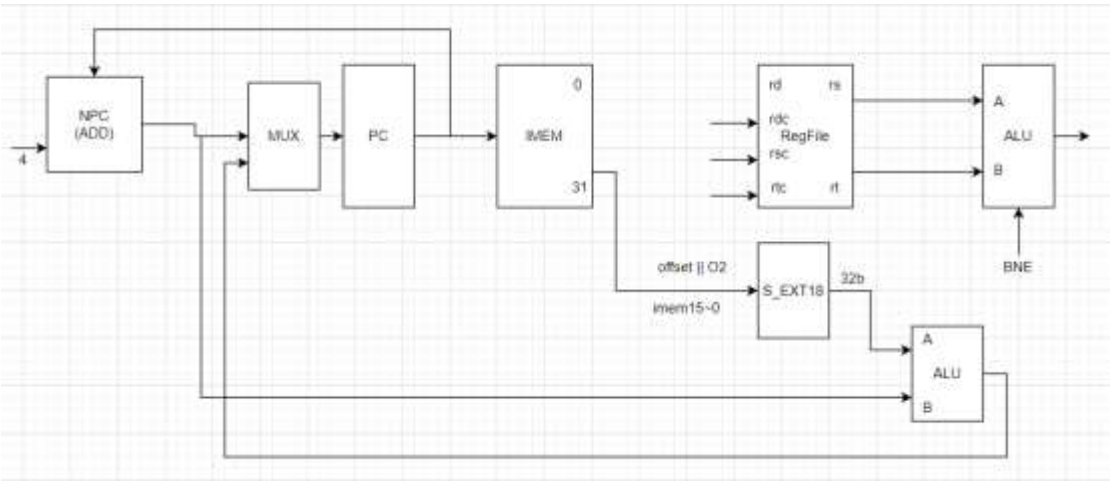
24.SLTIU



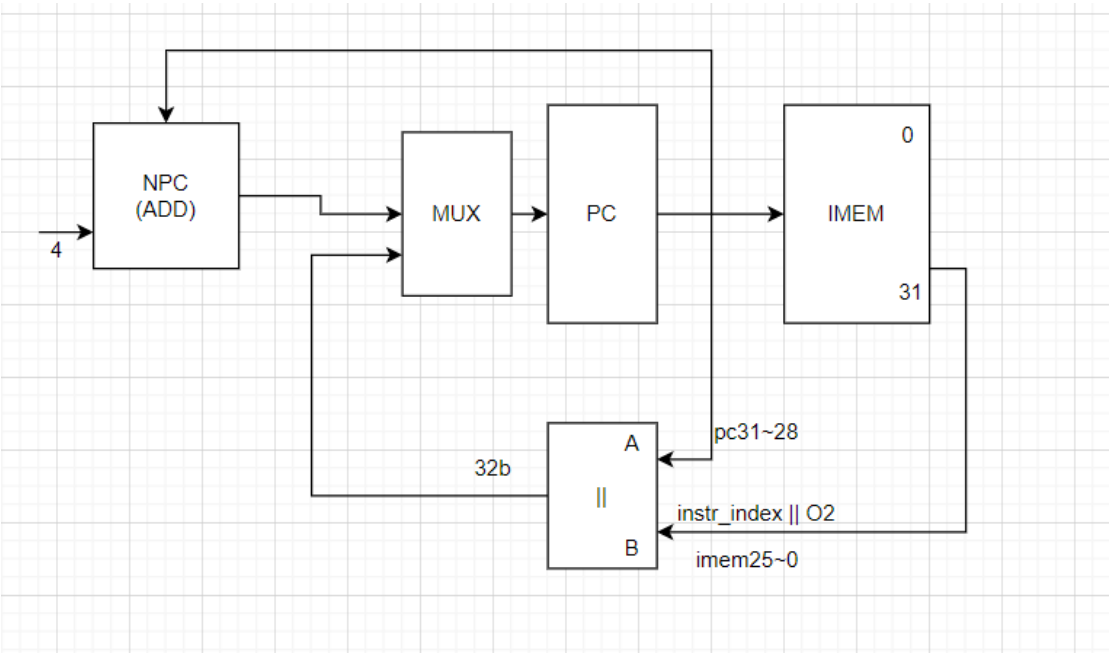
25.BEQ



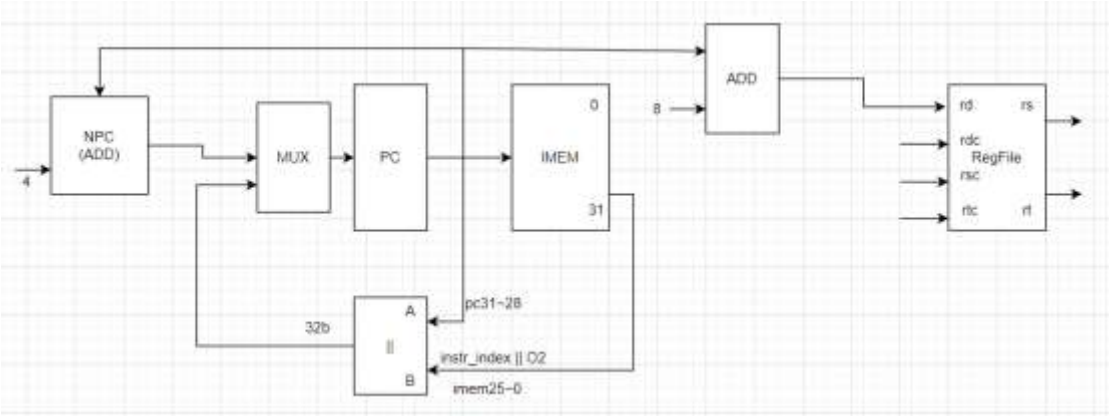
26.BNE



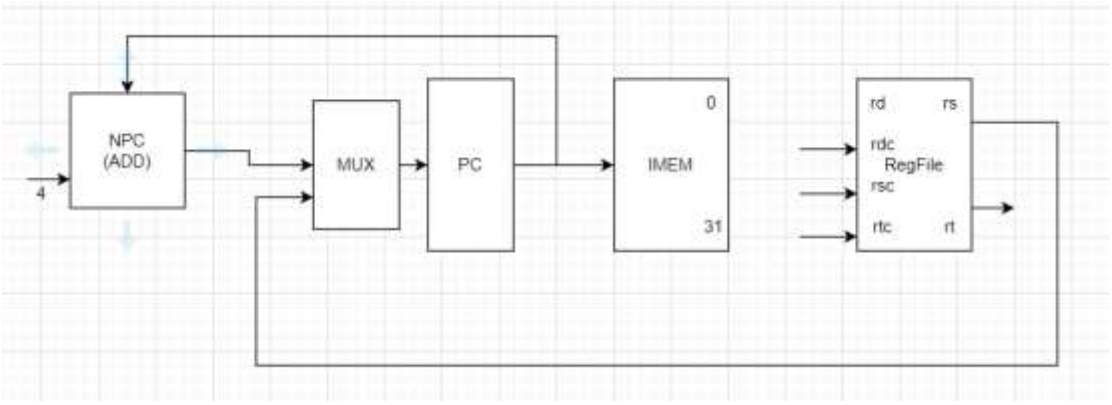
27.J



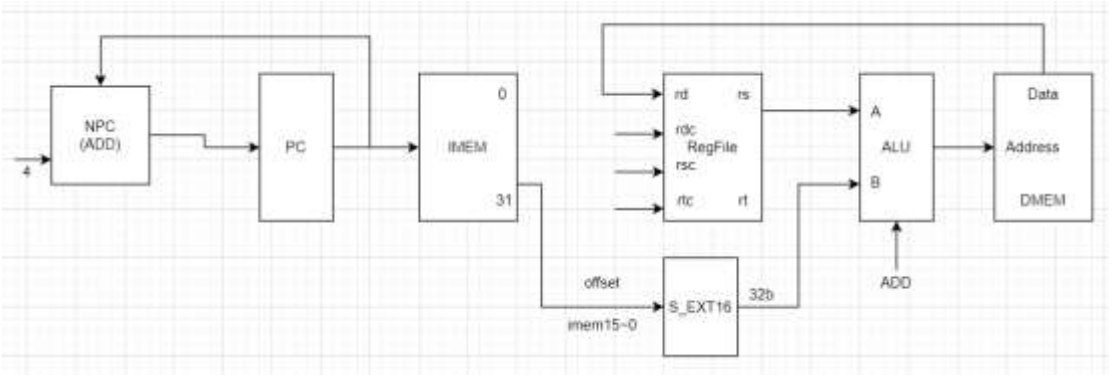
28.JAL



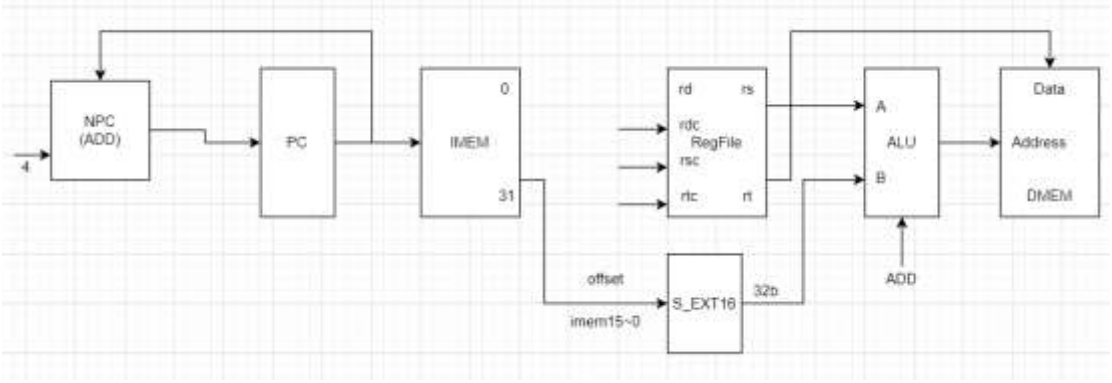
29.JR



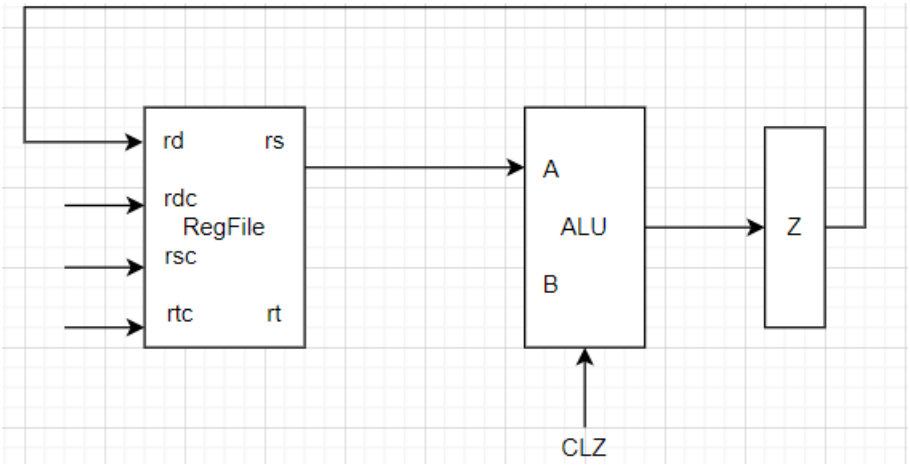
30.LW



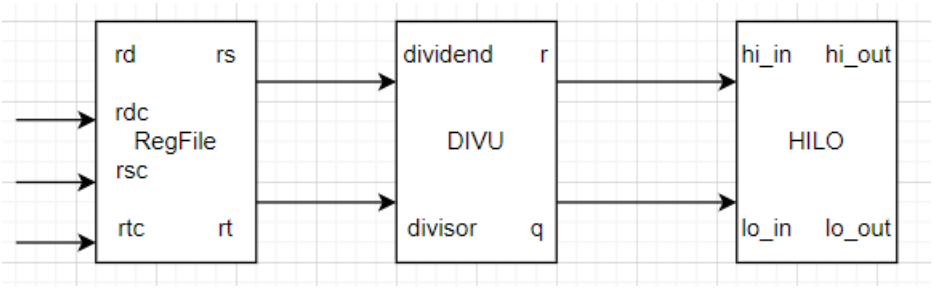
31.SW



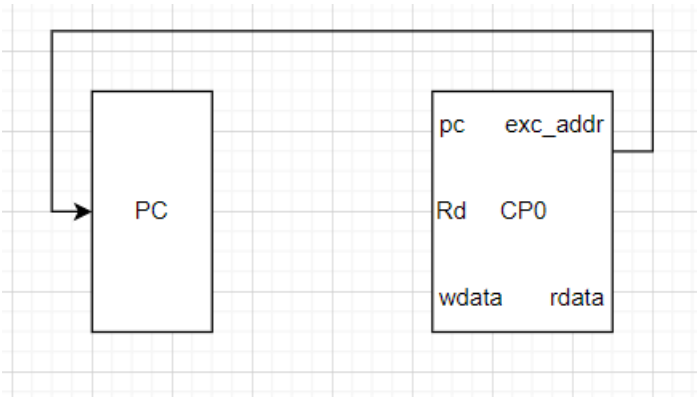
32.CLZ



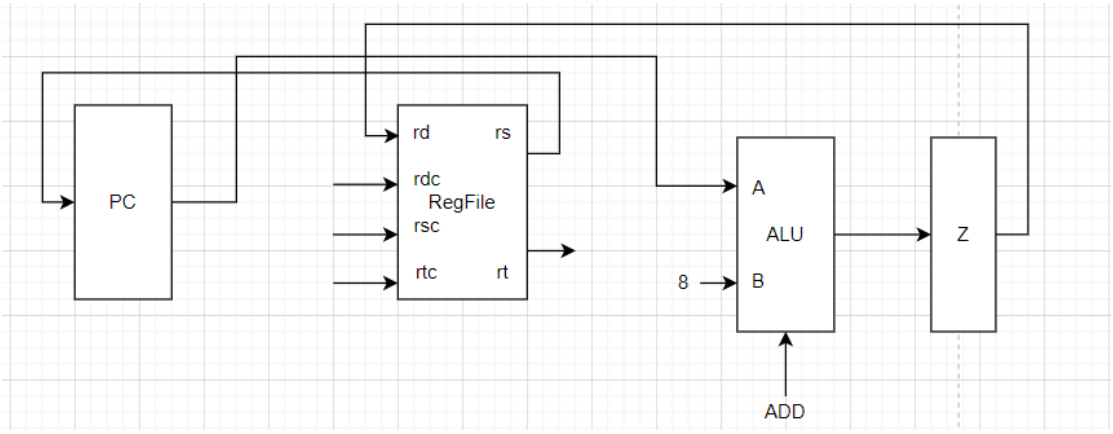
33.DIVU



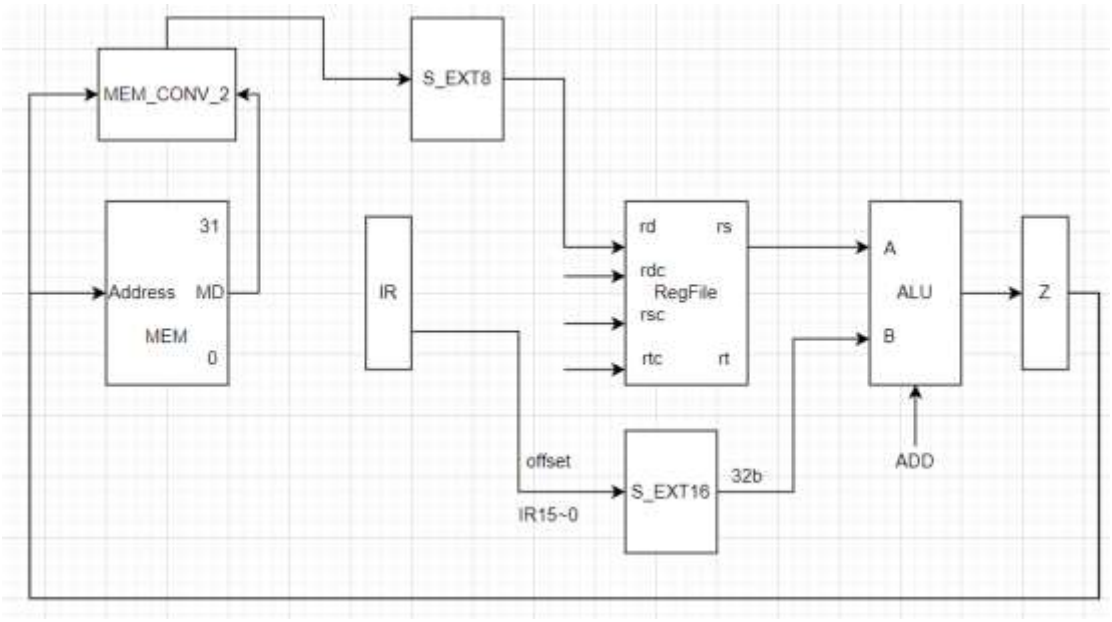
34.ERET



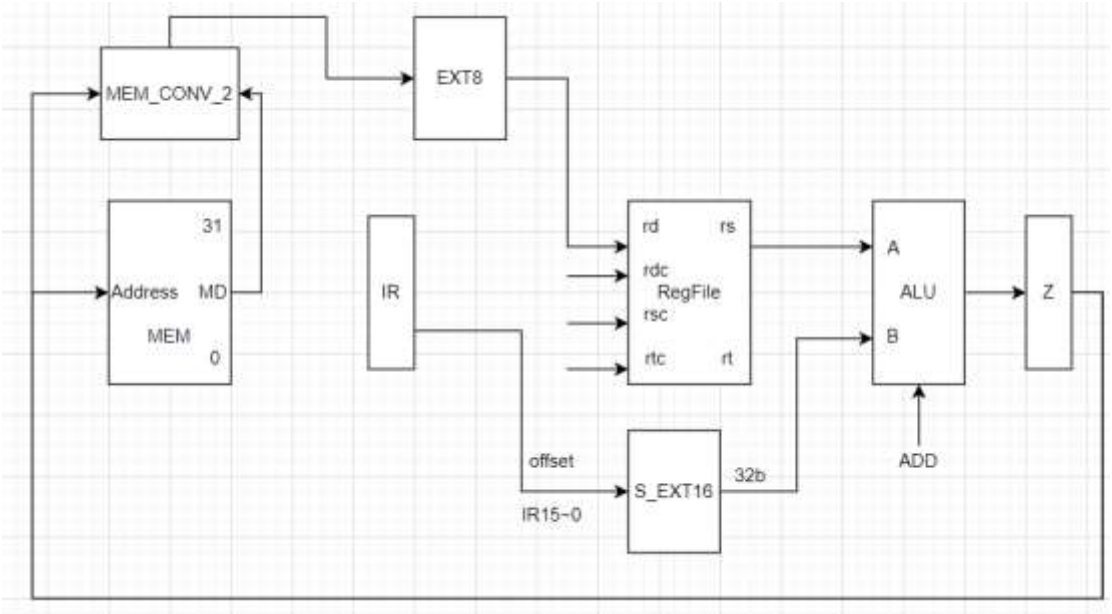
35.JALR



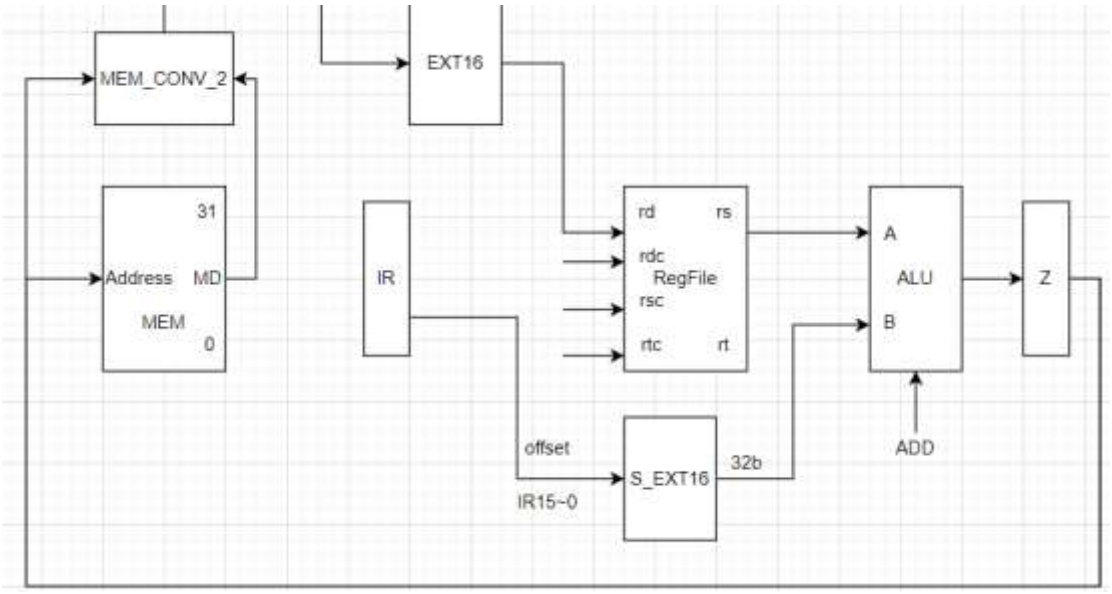
36.LB


MEM_CONV_2
S_EXT8
31
Address
MD
MEM
0
IR
rd
rs
rdc
RegFile
rsc
rtc
rt
A
ALU
B
Z
offset
IR15~0
S_EXT16
32b
ADD

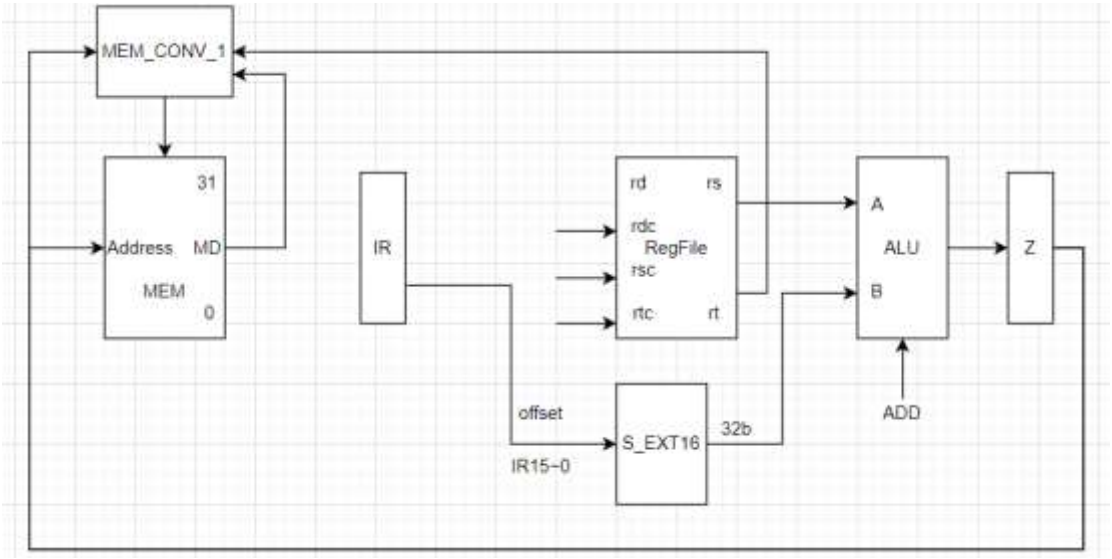
37.LBU


MEM_CONV_2
EXT8
31
Address
MD
MEM
0
IR
rd
rs
rdc
RegFile
rsc
rtc
rt
A
ALU
B
Z
offset
IR15~0
S_EXT16
32b
ADD

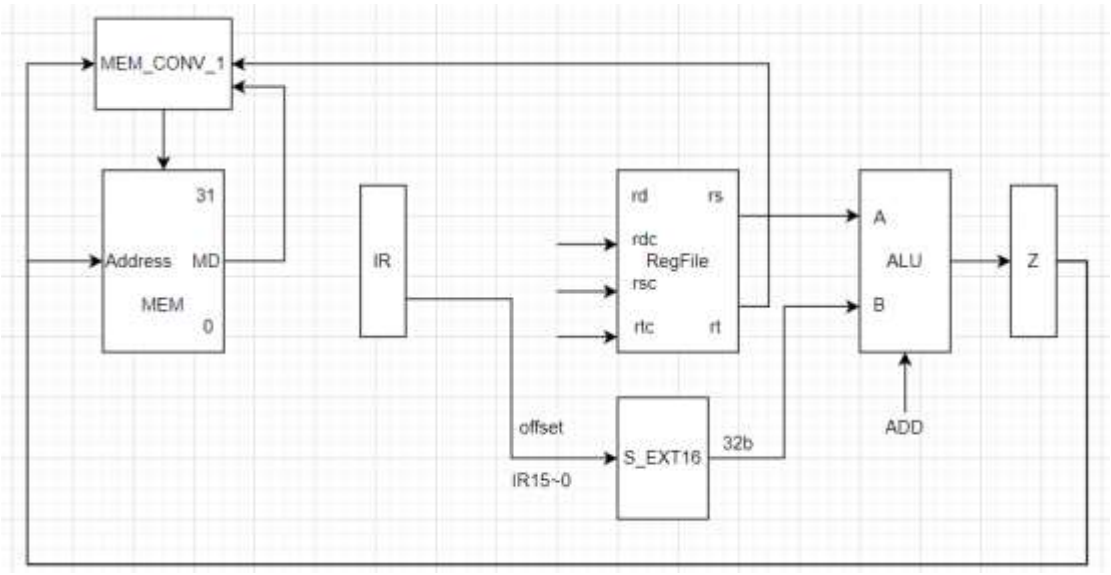
38.LHU


MEM_CONV_2
EXT16
31
Address
MD
MEM
0
IR
rd
rs
rdc
RegFile
rsc
rtc
rt
A
ALU
B
Z
ADD
offset
IR15~0
S_EXT16
32b

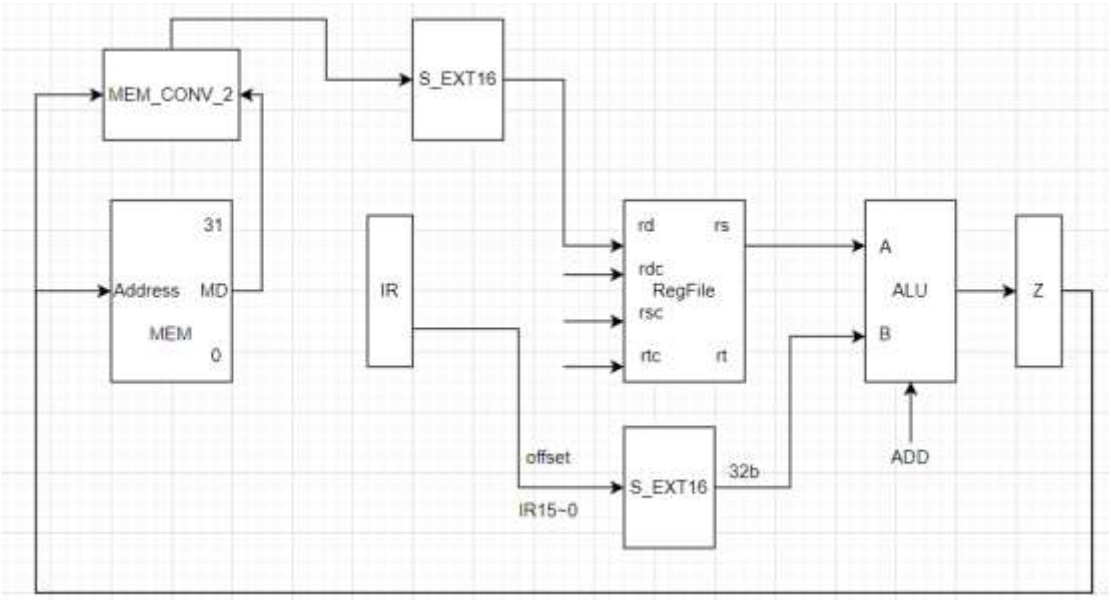
39.SB


MEM_CONV_1
31
Address
MD
MEM
0
IR
rd
rs
rdc
RegFile
rsc
rtc
rt
A
ALU
B
Z
ADD
offset
IR15~0
S_EXT16
32b

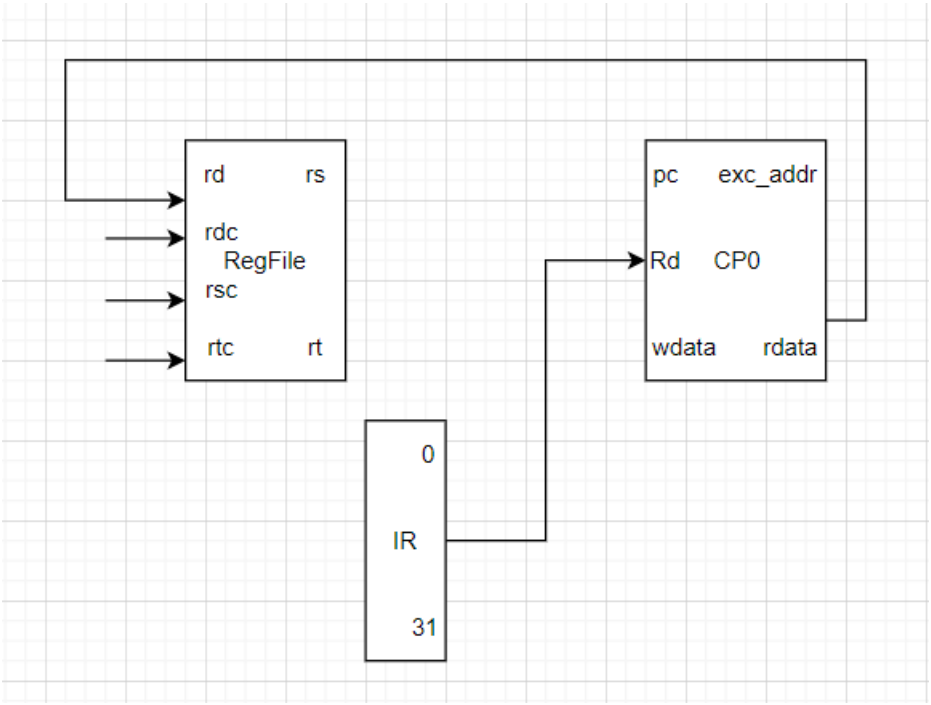
40.SH


MEM_CONV_1
31
Address
MD
MEM
0
IR
rd
rs
rdc
RegFile
rsc
rtc
rt
A
ALU
B
Z
ADD
offset
IR15~0
S_EXT16
32b

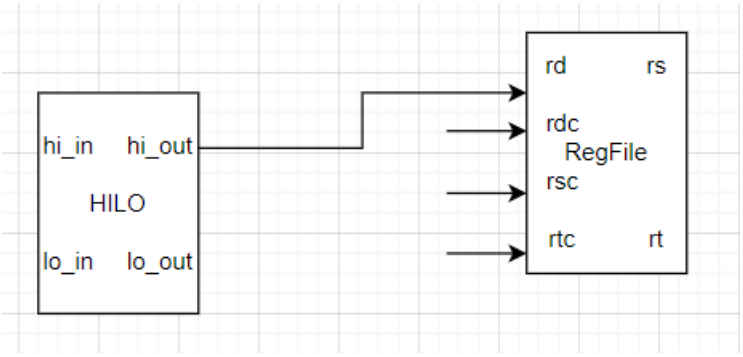
41.LH


MEM_CONV_2
S_EXT16
31
Address
MD
MEM
0
IR
rd
rs
rdc
RegFile
rsc
rtc
rt
A
ALU
B
Z
ADD
offset
IR15~0
S_EXT16
32b

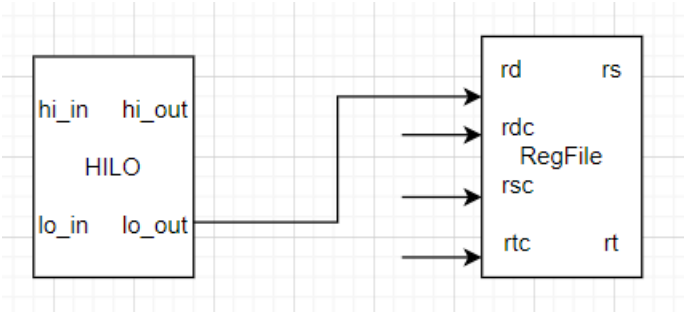
42.MFC0



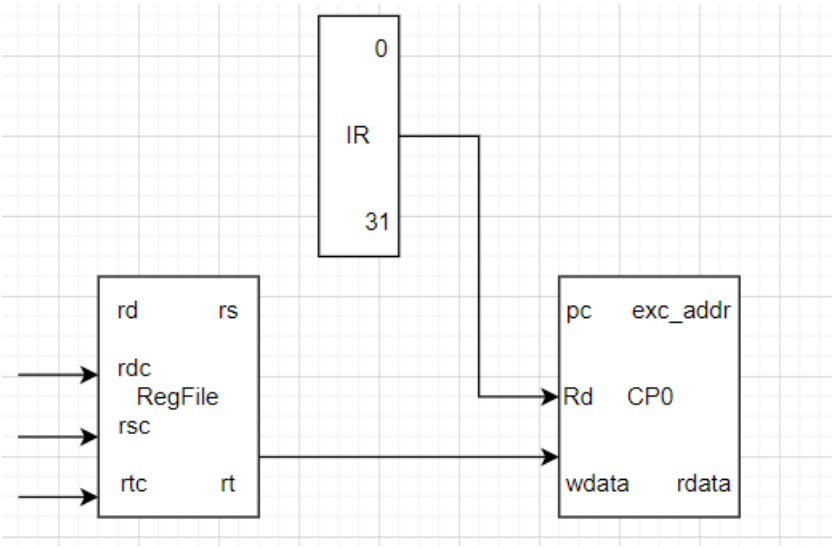
43.MFHI



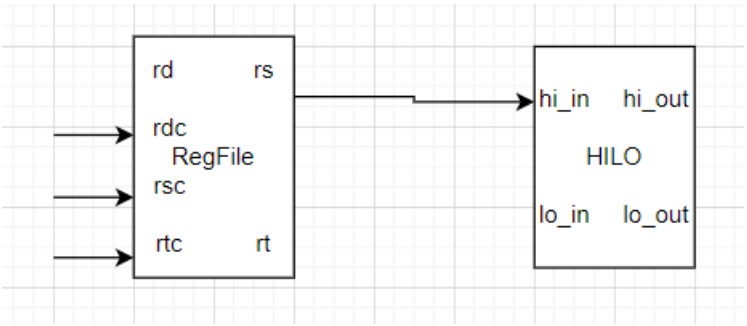
44.MFLO



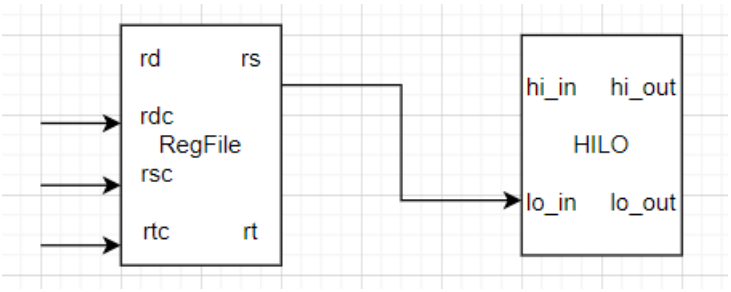
45.MTC0



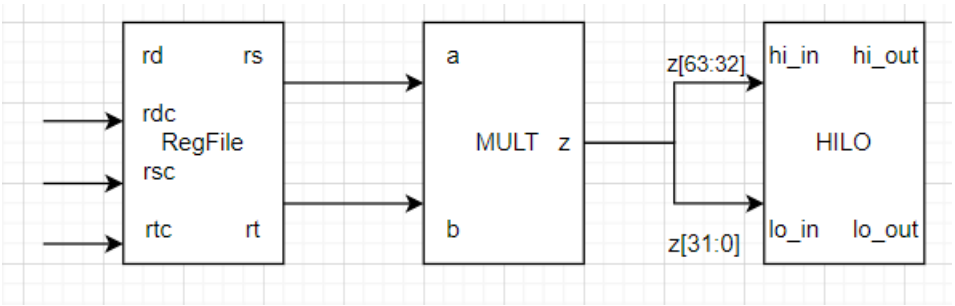
46.MTHI



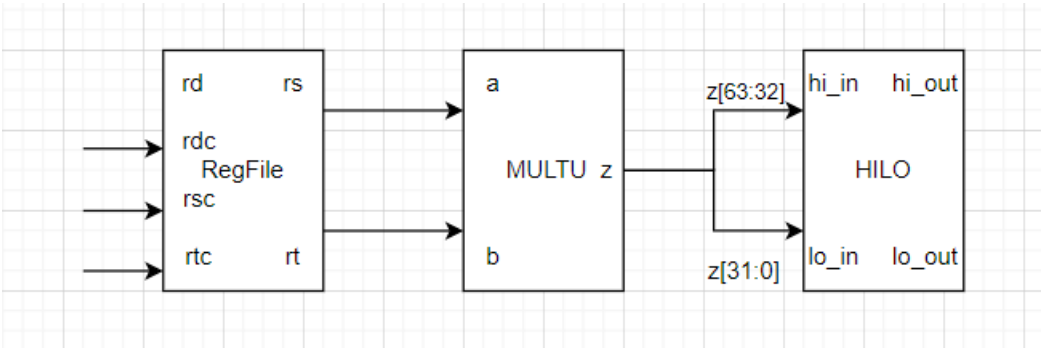
47.MTLO



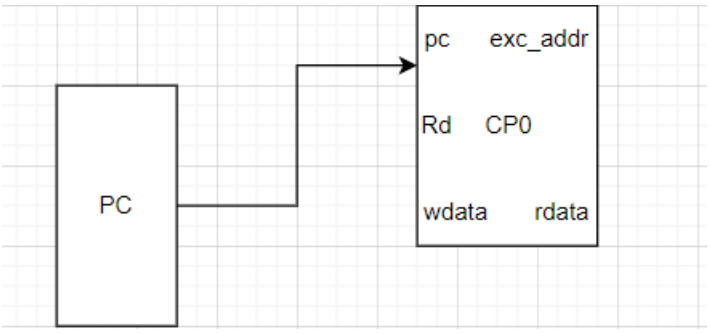
48.MUL



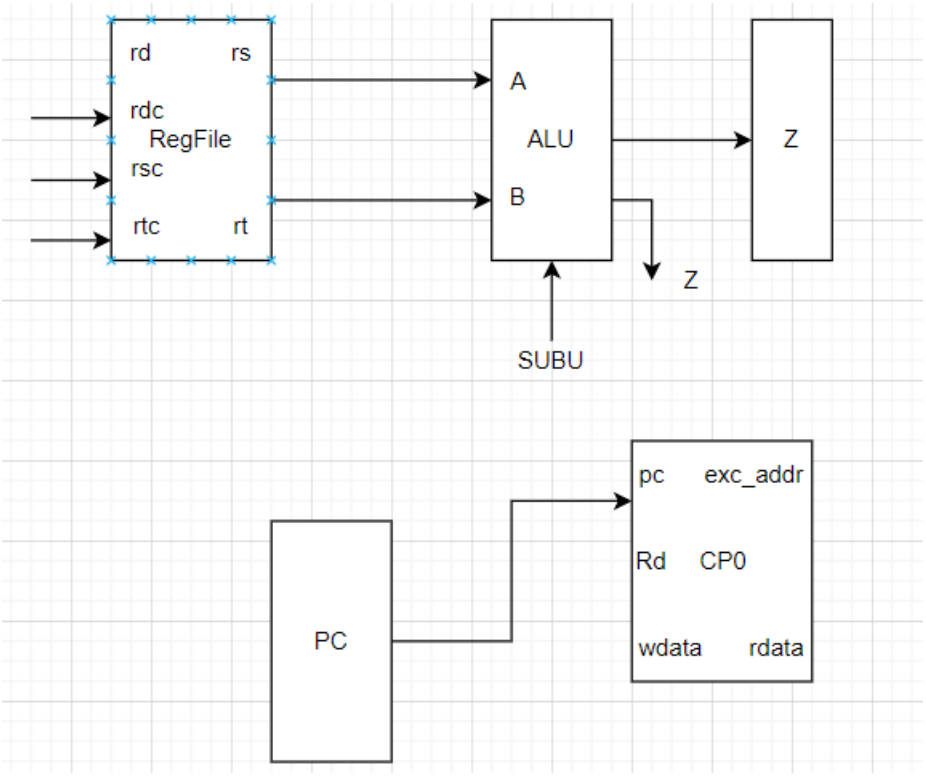
49.MULTU



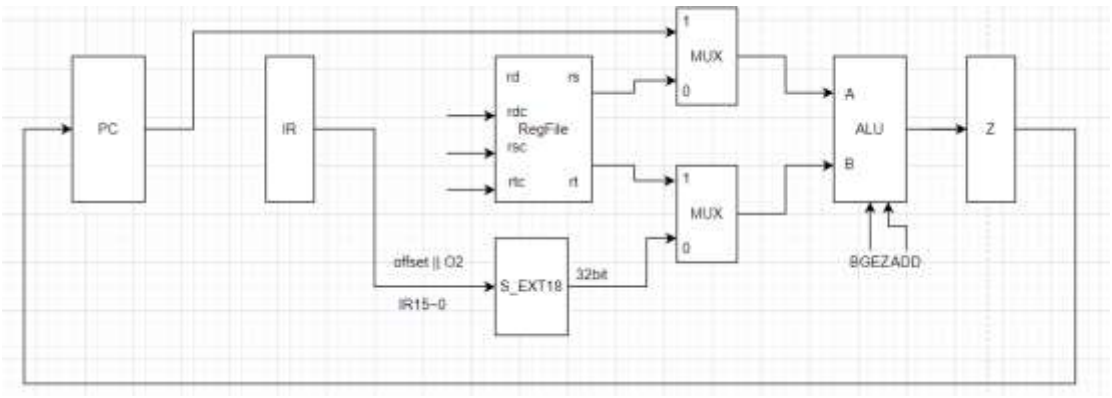
50.SYSCALL



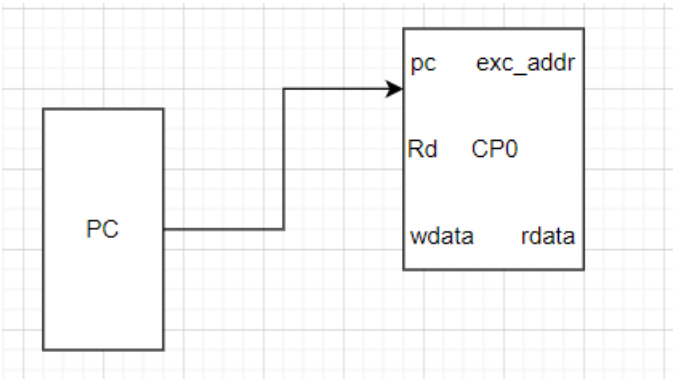
51.TEQ



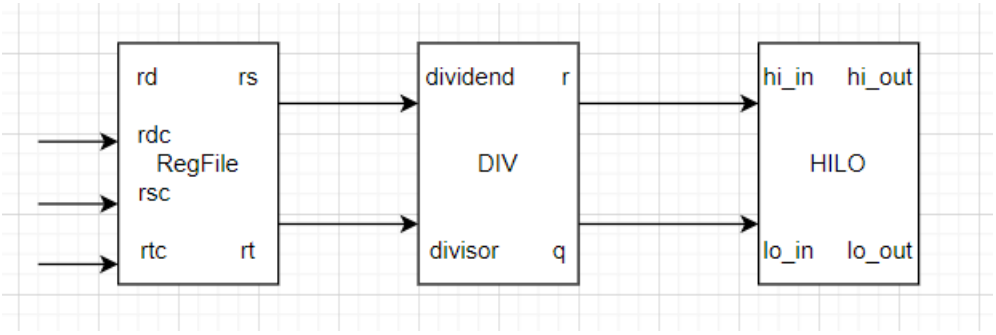
52.BGEZ



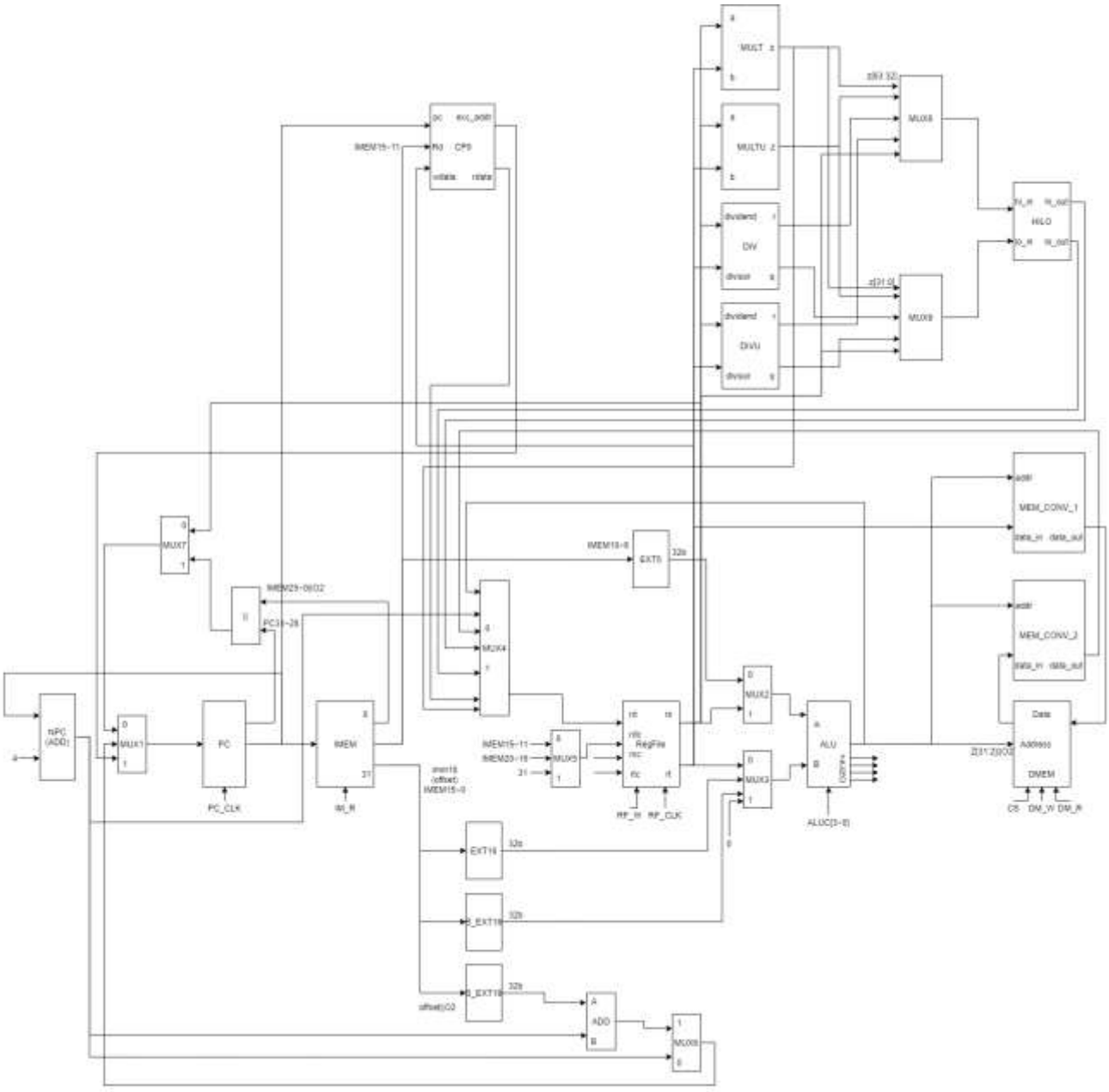
53.BREAK



54.DIV



54 条指令单周期 CPU 设计:



组成部件：

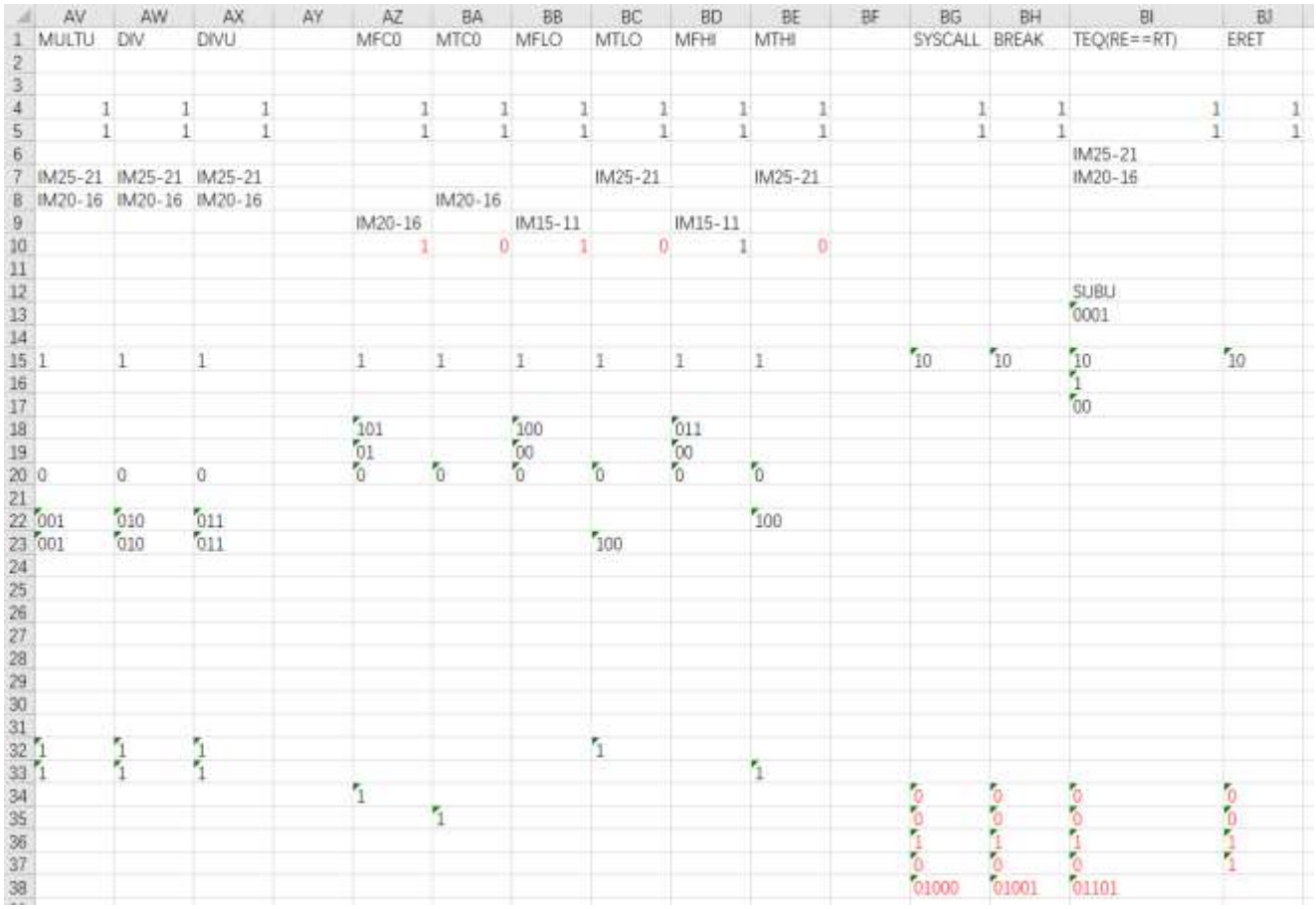
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	指令	PC	NPC	IMEM	Regfile		ALU			Ext5	Ext16	DMEM		S_Ext16	S_Ext18	ADD				MEM_CONV_1	MEM_CONV_2
2					rd	RF_W	A	B	ALUC			Addr	Data			A	B	A	B		
3	ADD	npc	pc	ps	ALU	-V	rs	rt	ADD												
4	ADDU	npc	pc	ps	ALU		rs	rt	ADDU												
5	ADDI	npc	pc	ps	ALU	-V	rs	S_Ext16	ADD					imm16							
6	ADDIU	npc	pc	ps	ALU		rs	S_Ext16	ADDU					imm16							
7	SUB	npc	pc	ps	ALU	-V	rs	rt	SUB												
8	SUBU	npc	pc	ps	ALU		rs	rt	SUBU												
9	AND	npc	pc	ps	ALU		rs	rt	AND												
10	ANDI	npc	pc	ps	ALU		rs	Ext16	AND		imm16										
11	OR	npc	pc	ps	ALU		rs	rt	OR												
12	ORI	npc	pc	ps	ALU		rs	Ext16	OR		imm16										
13	XOR	npc	pc	ps	ALU		rs	rt	XOR												
14	XORI	npc	pc	ps	ALU		rs	Ext16	XOR		imm16										
15	NOR	npc	pc	ps	ALU		rs	rt	NOR												
16	LUI	npc	pc	ps	ALU			Ext16	LUI		imm16										
17	SLL	npc	pc	ps	ALU		Ext5	rt	SLL	5a											
18	SLLV	npc	pc	ps	ALU		rs	rt	SLL												
19	SRA	npc	pc	ps	ALU		Ext5	rt	SRA	5a											
20	SRAV	npc	pc	ps	ALU		rs	rt	SRA												
21	SRL	npc	pc	ps	ALU		Ext5	rt	SRL	5a											
22	SRLV	npc	pc	ps	ALU		rs	rt	SRL												
23	SLT	npc	pc	ps	ALU		rs	rt	SLT												
24	SLTI	npc	pc	ps	ALU		rs	S_Ext16	SLT					imm16							
25	SLTU	npc	pc	ps	ALU		rs	rt	SLTU												
26	SLTIU	npc	pc	ps	ALU		rs	S_Ext16	SLTU					imm16							
27	CLZ	npc	pc	ps	ALU		rs	rt	CLZ												
28																					
29	BEQ	add	pc	ps			rs	rt	SUBU						offset@02	NPC	S_Ext18				
30	BNE	add	pc	ps			rs	rt	SUBU						offset@02	NPC	S_Ext18				

BGEZ	add	pc	pc			rs	rt	SUBU								other[32	NPC	S_Ext18							
J	||	pc	pc																	pc31-28	imem25-0				
JAL	||	pc	pc	pc+4[31号]																pc31-28	imem25-0				
JR	rs	pc	pc																						
JALR	rs	pc	pc	pc+4[31号]																					
LW	npc	pc	pc	MEM_CONV_2		rs	S_Ext18	ADD			ALU		offset												
LH	npc	pc	pc	MEM_CONV_2		rs	S_Ext16	ADD			ALU		offset												
LHU	npc	pc	pc	MEM_CONV_2		rs	S_Ext16	ADD			ALU		offset												
LB	npc	pc	pc	MEM_CONV_2		rs	S_Ext16	ADD			ALU		offset												
LBU	npc	pc	pc	MEM_CONV_2		rs	S_Ext16	ADD			ALU		offset												
SW	npc	pc	pc			rs	S_Ext16	ADD			ALU	MEM_CONV_1	offset												
SH	npc	pc	pc			rs	S_Ext16	ADD			ALU	MEM_CONV_1	offset												
SB	npc	pc	pc			rs	S_Ext16	ADD			ALU	MEM_CONV_1	offset												
MULT	npc	pc	pc																						
MULTU	npc	pc	pc																						
DIV	npc	pc	pc																						
DIVU	npc	pc	pc																						
MFOU	npc	pc	pc	data																					
MTOI	npc	pc	pc																						
MFLO	npc	pc	pc	hi_out																					
MTLO	npc	pc	pc																						
MFHI	npc	pc	pc	hi_out																					
MTHI	npc	pc	pc																						
SYSCALL	npc	pc	pc																						
BREAK	npc	pc	pc																						
TEQ	npc	pc	pc																						
ERET	exc_addr	pc	pc																						

控制信号：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1		ADD	ADDU	ADDI	ADDIU	SUB	SUBU	AND	ANDI	OR	ORI	XOR	XORI	NOR	LUI	SLL	SLLV	SRA	SRAV	SRL	SRLV	SLT	SLTI
2																							
3	BUSY																						
4	PC_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	IM_R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6																							
7	rs:4~0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21			IM25-21		IM25-21			IM25-21	IM25-21	IM25-21
8	rt:4~0	IM20-16	IM20-16			IM20-16	IM20-16	IM20-16		IM20-16		IM20-16		IM20-16		IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	
9	rd:4~0	IM15-11	IM15-11	IM20-16	IM20-16	IM15-11	IM15-11	IM25-11	IM20-16	IM15-11	IM20-16	IM15-11	IM20-16	IM15-11	IM20-16	IM25-11	IM25-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM20-16
10	RF_W	~V		1→V		1→V		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11																							
12	ALUC3~0	ADD	ADDU	ADD	ADDU	SUB	SUBU	AND	AND	OR	OR	XOR	XOR	NOR	LUI	SLL	SLL	SRA	SRA	SRL	SRL	SLT	SLT
13	ALUC3~0	0010	0000	0010	0000	0011	0001	0100	0100	0101	0101	0110	0110	0111	1000	111X	111X	1100	1100	1101	1101	0111	0111
14																							
15	M1[1~0]	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	M2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	1	1
17	M3[1~0]	0-0	0-0	1-0	1-0	0-0	0-0	0-0	0-1	0-0	0-1	0-0	0-1	0-0	0-1	0-0	0-0	0-0	0-0	0-0	0-0	0-0	1-0
18	M4[2~0]	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000
19	M5[1~0]	0	0	1	1	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	1
20	M6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	M7																						
22	M8[2~0]																						
23	M9[2~0]																						
24																							
25	CS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	DM_R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	DM_W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28																							
29	conv1type																						
30	conv2type																						
31																							
32	hin	0																					
33	hin	0																					
34	Mtc0																						
35	Mtc0																						
36	exception																						
37	exit																						
38	cause																						
39	start																						

	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU
1	SLTU	CLZ		BEQ	BNE	BGEZ		J	JAL	JR	JALR		LW	LH	LHU	LB	LBU	SW	SH	SB		MUL	MULT
2																							
3																							
4		1	1		1	1	1		1	1	1	1		1	1	1	1	1	1	1	1		1	1
5		1	1		1	1	1		1	1	1	1		1	1	1	1	1	1	1	1		1	1
6																							
7	IM25-21	IM25-21		IM25-21	IM25-21	IM25-21					IM25-21		IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21		IM25-21	IM25-21
8		IM20-16																				IM20-16	IM20-16
9	IM20-16	IM15-11		IM20-16	IM20-16	IM20-16					IM15-11		IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16		IM15-11	
10	1	1		0	0	0		0	1	0	1		1	1	1	1	1	0	0	0		1	
11																							
12	SLTU	CLZ		SUBU	SUBU	SUB							ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD			
13	1010	1101[1]????		0001	0001	0011							0010	0010	0010	0010	0010	0010	0010	0010			
14																							
15	1	1		1	1	1		0	0	0	0		1	1	1	1	1	1	1	1		1	1
16	1	1		1	1	1					1												
17	1-0	0-0		0-0	0-0	11		0-0	0-0	0-0	00		1-0	1-0	1-0	1-0	1-0	1-0	1-0	1-0			
18	000	000							001		001		010	010	010	010	010					110	
19	1	0		1	1	1			10		00		1	1	1	1	1	1	1	1		0	
20	0	0		1	1	1							0	0	0	0	0	0	0	0		0	0
21								1	1	0	0												
22																						000	000
23																						000	000
24																							
25	0	0		0	0	0		0	0	0	0		1	1	1	1	1	1					
26	0	0		0	0	0		0	0	0	0		1	1	1	1	1	0					
27	0	0		0	0	0		0	0	0	0		0	0	0	0	0	1					
28																							
29																		SW	SH	SB			
30													LW	LH	LHU	LB	LBU						
31																							
32																						1	1
33																						1	1
34																							



	AV	AW	AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ
1	MULTU	DIV	DIVU		MFC0	MTC0	MFLO	MTLO	MFHI	MTHI		SYSCALL	BREAK	TEQ(RE==RT)	ERET
2															
3															
4	1	1	1		1	1	1	1	1	1		1	1	1	1
5	1	1	1		1	1	1	1	1	1		1	1	1	1
6														IM25-21	
7	IM25-21	IM25-21	IM25-21					IM25-21		IM25-21				IM20-16	
8	IM20-16	IM20-16	IM20-16			IM20-16									
9					IM20-16		IM15-11		IM15-11						
10					1	0	1	0	1	0					
11															
12														SUBU	
13														0001	
14															
15	1	1	1		1	1	1	1	1	1		10	10	10	10
16														1	
17														00	
18					101		100		011						
19					01		00		00						
20	0	0	0		0	0	0	0	0	0					
21															
22	001	010	011							100					
23	001	010	011					100							
24															
25															
26															
27															
28															
29															
30															
31															
32	1	1	1					1							
33	1	1	1							1					
34					1							0	0	0	0
35						1						0	0	0	0
36												1	1	1	1
37												0	0	0	1
38												01000	01001	01101	

三、模块建模

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

顶层模块：sccomp_dataflow

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

                                    //控制信号
/*------------------------*/
wire Z;
wire C;
wire N;
wire O;
//wire busy_mult;
//wire busy_multu;
wire busy_div;
wire busy_divu;
//wire over_mult;
//wire over_multu;
wire over_div;
wire over_divu;
```

```
//wire start_mult;
//wire start_multu;
wire start_div;
wire start_divu;
wire pc_no_add;
// wire    PC_CLK;
//    wire    IM_R;
wire    RF_W;
// wire    RF_CLK;
wire    [3:0]ALUC;

wire    [1:0]M1;
wire    M2;
wire    [1:0]M3;
wire    [2:0]M4;
wire    [1:0]M5;
wire    M6;
wire    M7;
wire    [2:0]M8;
wire    [2:0]M9;

wire    DM_W;
wire [1:0]DMEM_in_type;
wire [1:0]conv1type;//sw:00,sh:01,sb:10
wire [2:0]conv2type;//lw:000,lh:001,lhu:010,lb:011,lbu:100
wire loin;
wire hiin;
wire mfc0_c;
wire mtc0_c;
wire execption;
wire eret_c;
wire [4:0]cause;

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

wire [31:0] exc_addr_out;
wire [31:0] rdata_out;

wire [31:0] conv1_data_out32;
wire [15:0] conv1_data_out16;
wire [7:0] conv1_data_out8;
wire [31:0] conv2_data_out;

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

wire [63:0] mult_z_out;
wire [63:0] multu_z_out;
wire [31:0] div_r_out;
wire [31:0] div_q_out;
wire [31:0] divu_r_out;
wire [31:0] divu_q_out;

wire [31:0] hilo_hi_out;
wire [31:0] hilo_lo_out;


wire intr;
wire status;
wire timer_int;

//wire [31:0]pcx;

    //assign pc=pcx- 32'h00400000;

    assign im_addr =    pc- 32'h00400000;

    assign dm_addr =    alu_out- 32'h10010000;

    IMEM_ip imemory(im_addr[12:2],inst);
    DMEM
dmemory(clk_in,DMEM_in_type,dm_addr[1:0],DM_W,dm_addr[12:2],conv1_data_out32,conv1_data_out16,conv1_data_out8,dmem_o
ut);

    //注意 pcx 和 pc 的转换关系
    cpu sccpu(clk_in,reset,pc,inst,dmem_out,rt_out,alu_out,exc_addr_out,rdata_out,
```

```
        conv1_data_out32,conv1_data_out16,conv1_data_out8,conv2_data_out,
          mult_z_out,multu_z_out,div_r_out,div_q_out,divu_r_out,divu_q_out,
             hilo_hi_out,hilo_lo_out,
        Z,C,N,O,
        busy_div,busy_divu,
        over_div,over_divu,
        start_div,start_divu,pc_no_add,
        RF_W,ALUC,
        M1,M2,M3,M4,M5,M6,M7,M8,M9,
        conv1type,conv2type,loin,hiin);


        instruction_to_control itc(
        inst,
        Z,C,N,O,
        busy_div,busy_divu,
        over_div,over_divu,
        start_div,start_divu,pc_no_add,
        RF_W,ALUC,
        M1,M2,M3,M4,M5,M6,M7,M8,M9,
        DM_W,DMEM_in_type,conv1type,conv2type,loin,hiin,
        mfc0_c,mtc0_c,execption,eret_c,cause);

        CP0 cp0_uut(clk_in,reset,mfc0_c,mtc0_c,pc,inst[15:11],rt_out,exception,eret,cause,intr,rdata_out,status,timer_int,exc_addr_out);


Endmodule



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

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

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

          always @(posedge clk or posedge rst)
          begin
          if(rst==1'b1)
          begin
          array_reg[0]=32'b0;
          array_reg[1]=32'b0;
          array_reg[2]=32'b0;
          array_reg[3]=32'b0;
          array_reg[4]=32'b0;
          array_reg[5]=32'b0;
          array_reg[6]=32'b0;
          array_reg[7]=32'b0;
          array_reg[8]=32'b0;
          array_reg[9]=32'b0;
          array_reg[10]=32'b0;
          array_reg[11]=32'b0;
          array_reg[12]=32'b0;
          array_reg[13]=32'b0;
          array_reg[14]=32'b0;
          array_reg[15]=32'b0;
          array_reg[16]=32'b0;
          array_reg[17]=32'b0;
          array_reg[18]=32'b0;
          array_reg[19]=32'b0;
          array_reg[20]=32'b0;
          array_reg[21]=32'b0;
          array_reg[22]=32'b0;
          array_reg[23]=32'b0;
          array_reg[24]=32'b0;
          array_reg[25]=32'b0;
          array_reg[26]=32'b0;
          array_reg[27]=32'b0;
          array_reg[28]=32'b0;
          array_reg[29]=32'b0;
          array_reg[30]=32'b0;
```

```
array_reg[31]=32'b0;


end
else
begin
    if(rf_w==1)
    begin
    if (waddr == 0)
    array_reg[0] = 0;
    else if (waddr == 1)
    array_reg[1] = wdata;
    else if (waddr == 2)
    array_reg[2] = wdata;
    else if (waddr == 3)
    array_reg[3] = wdata;
    else if (waddr == 4)
    array_reg[4] = wdata;
    else if (waddr == 5)
    array_reg[5] = wdata;
    else if (waddr == 6)
    array_reg[6] = wdata;
    else if (waddr == 7)
    array_reg[7] = wdata;
    else if (waddr == 8)
    array_reg[8] = wdata;
    else if (waddr == 9)
    array_reg[9] = wdata;
    else if (waddr == 10)
    array_reg[10] = wdata;
    else if (waddr == 11)
    array_reg[11] = wdata;
    else if (waddr == 12)
    array_reg[12] = wdata;
    else if (waddr == 13)
    array_reg[13] = wdata;
    else if (waddr == 14)
    array_reg[14] = wdata;
    else if (waddr == 15)
    array_reg[15] = wdata;
    else if (waddr == 16)
    array_reg[16] = wdata;
    else if (waddr == 17)
    array_reg[17] = wdata;
    else if (waddr == 18)
    array_reg[18] = wdata;
    else if (waddr == 19)
    array_reg[19] = wdata;
    else if (waddr == 20)
    array_reg[20] = wdata;
    else if (waddr == 21)
    array_reg[21] = wdata;
    else if (waddr == 22)
    array_reg[22] = wdata;
    else if (waddr == 23)
    array_reg[23] = wdata;
    else if (waddr == 24)
    array_reg[24] = wdata;
    else if (waddr == 25)
    array_reg[25] = wdata;
    else if (waddr == 26)
    array_reg[26] = wdata;
    else if (waddr == 27)
    array_reg[27] = wdata;
    else if (waddr == 28)
    array_reg[28] = wdata;
    else if (waddr == 29)
    array_reg[29] = wdata;
    else if (waddr == 30)
    array_reg[30] = wdata;
    else if (waddr == 31)
    array_reg[31] = wdata;
    else
    begin

    end

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

```
    end



end
end

always @(*)
begin
if(raddr1==0)
rdata1=array_reg[0];
else if(raddr1==1)
rdata1=array_reg[1];
else if(raddr1==2)
rdata1=array_reg[2];
else if(raddr1==3)
rdata1=array_reg[3];
else if(raddr1==4)
rdata1=array_reg[4];
else if(raddr1==5)
rdata1=array_reg[5];
else if(raddr1==6)
rdata1=array_reg[6];
else if(raddr1==7)
rdata1=array_reg[7];
else if(raddr1==8)
rdata1=array_reg[8];
else if(raddr1==9)
rdata1=array_reg[9];
else if(raddr1==10)
rdata1=array_reg[10];
else if(raddr1==11)
rdata1=array_reg[11];
else if(raddr1==12)
rdata1=array_reg[12];
else if(raddr1==13)
rdata1=array_reg[13];
else if(raddr1==14)
rdata1=array_reg[14];
else if(raddr1==15)
rdata1=array_reg[15];
else if(raddr1==16)
rdata1=array_reg[16];
else if(raddr1==17)
rdata1=array_reg[17];
else if(raddr1==18)
rdata1=array_reg[18];
else if(raddr1==19)
rdata1=array_reg[19];
else if(raddr1==20)
rdata1=array_reg[20];
else if(raddr1==21)
rdata1=array_reg[21];
else if(raddr1==22)
rdata1=array_reg[22];
else if(raddr1==23)
rdata1=array_reg[23];
else if(raddr1==24)
rdata1=array_reg[24];
else if(raddr1==25)
rdata1=array_reg[25];
else if(raddr1==26)
rdata1=array_reg[26];
else if(raddr1==27)
rdata1=array_reg[27];
else if(raddr1==28)
rdata1=array_reg[28];
else if(raddr1==29)
rdata1=array_reg[29];
else if(raddr1==30)
rdata1=array_reg[30];
else if(raddr1==31)
rdata1=array_reg[31];
else
begin

end
```

```
        if(raddr2==0)
        rdata2=array_reg[0];
        else if(raddr2==1)
        rdata2=array_reg[1];
        else if(raddr2==2)
        rdata2=array_reg[2];
        else if(raddr2==3)
        rdata2=array_reg[3];
        else if(raddr2==4)
        rdata2=array_reg[4];
        else if(raddr2==5)
        rdata2=array_reg[5];
        else if(raddr2==6)
        rdata2=array_reg[6];
        else if(raddr2==7)
        rdata2=array_reg[7];
        else if(raddr2==8)
        rdata2=array_reg[8];
        else if(raddr2==9)
        rdata2=array_reg[9];
        else if(raddr2==10)
        rdata2=array_reg[10];
        else if(raddr2==11)
        rdata2=array_reg[11];
        else if(raddr2==12)
        rdata2=array_reg[12];
        else if(raddr2==13)
        rdata2=array_reg[13];
        else if(raddr2==14)
        rdata2=array_reg[14];
        else if(raddr2==15)
        rdata2=array_reg[15];
        else if(raddr2==16)
        rdata2=array_reg[16];
        else if(raddr2==17)
        rdata2=array_reg[17];
        else if(raddr2==18)
        rdata2=array_reg[18];
        else if(raddr2==19)
        rdata2=array_reg[19];
        else if(raddr2==20)
        rdata2=array_reg[20];
        else if(raddr2==21)
        rdata2=array_reg[21];
        else if(raddr2==22)
        rdata2=array_reg[22];
        else if(raddr2==23)
        rdata2=array_reg[23];
        else if(raddr2==24)
        rdata2=array_reg[24];
        else if(raddr2==25)
        rdata2=array_reg[25];
        else if(raddr2==26)
        rdata2=array_reg[26];
        else if(raddr2==27)
        rdata2=array_reg[27];
        else if(raddr2==28)
        rdata2=array_reg[28];
        else if(raddr2==29)
        rdata2=array_reg[29];
        else if(raddr2==30)
        rdata2=array_reg[30];
        else if(raddr2==31)
        rdata2=array_reg[31];
        else
        begin

        end

        end

endmodule


module pc(
input PC_CLK,
input reset,
input pc_in_control,
input [31:0] pc_in,
```

```
output reg [31:0] pc_out
    );

    always @(negedge PC_CLK or posedge reset)//上升沿  or  下降沿？
    begin
        if(reset==1'b1)
            pc_out<=32'h00400000;
        else
        begin
            if(pc_in_control==1'b0)
            pc_out<=pc_in;
            else
            begin

            end
        end
    end

endmodule




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

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

    end

endmodule

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

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

    end

endmodule

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

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

        end

endmodule

module selector81(
input [31:0] iC0,
input [31:0] iC1,
input [31:0] iC2,
input [31:0] iC3,
input [31:0] iC4,
input [31:0] iC5,
input [31:0] iC6,
input [31:0] iC7,
input [2:0]iS,
output reg [31:0] oZ
        );
        always @(*)
        begin

        if(iS==0)
        oZ=iC0;
        else if(iS==1)
        oZ=iC1;
        else if(iS==2)
        oZ=iC2;
        else if(iS==3)
        oZ=iC3;
        else if(iS==4)
        oZ=iC4;
        else if(iS==5)
        oZ=iC5;
        else if(iS==6)
        oZ=iC6;
        else
        oZ=iC7;


        end

endmodule


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

        assign joint_out={pc_in,imem_in};

endmodule


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

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


module i_to_controller(
input [5:0]op,
input [4:0]rs,
input [5:0]func,
input Z,
input C,
input N,
input O,


input busy_div,
input busy_divu,
```

```
input over_div,
input over_divu,

output start_div,
output start_divu,
output pc_no_add,

output    RF_W,
output    [3:0]ALUC,

output    [1:0]M1,
output    M2,
output    [1:0]M3,
output    [2:0]M4,
output    [1:0]M5,
output    M6,
output    M7,
output    [2:0]M8,
output    [2:0]M9,

output    DM_W,
output [1:0]DMEM_in_type,
output [1:0]conv1type,//sw:00,sh:01,sb:10
output [2:0]conv2type,//lw:000,lh:001,lhu:010,lb:011,lbu:100
output loin,
output hiin,
output mfc0_c,
output mtc0_c,
output execption,
output eret_c,
output [4:0]cause
    );

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

    //23-extend
    wire i_clz;
    wire i_div,i_divu,i_mul,i_multu;
    wire i_bgez;
    wire i_jalr;
    wire i_lh,i_lhu,i_lb,i_lbu,i_sh,i_sb;
    wire i_mfc0,i_mtc0,i_mfhi,i_mthi,i_mflo,i_mtlo;
    wire i_syscall,i_break,i_teq,i_eret;

    //R-TYPE
    assign
i_add=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(~func[2])&(~func[1])&(~func[0
]);
    assign
i_addu=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(~func[2])&(~func[1])&(func[0
]);
    assign
i_sub=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(~func[2])&(func[1])&(~func[0])
;
    assign
i_subu=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(~func[2])&(func[1])&(func[0])
;
    assign
i_and=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(func[2])&(~func[1])&(~func[0]
);
    assign
i_or=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(func[2])&(~func[1])&(func[0]);
    assign
i_xor=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(func[2])&(func[1])&(~func[0]);
    assign
i_nor=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(func[2])&(func[1])&(func[0]);
    assign
i_slt=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(func[3])&(~func[2])&(func[1])&(~func[0]);
    assign
i_sltu=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(func[3])&(~func[2])&(func[1])&(func[0]);

    assign
```

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

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

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

    //23 条扩展指令
    assign
i_clz=(~op[5])&(op[4])&(op[3])&(op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(~func[2])&(~func[1])&(~func[0]);

    assign
i_div=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(func[4])&(func[3])&(~func[2])&(func[1])&(~func[0]);
    assign
i_divu=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(func[4])&(func[3])&(~func[2])&(func[1])&(func[0]);
    assign
i_mul=(~op[5])&(op[4])&(op[3])&(op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(~func[3])&(~func[2])&(func[1])&(~func[0]);
    //assign
i_mult=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(func[4])&(func[3])&(~func[2])&(~func[1])&(~func[0
]);
    assign
i_multu=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(func[4])&(func[3])&(~func[2])&(~func[1])&(func[0
]);

    assign i_bgez=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(op[0]);

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

    assign i_lh=(op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(op[0]);
    assign i_lhu=(op[5])&(~op[4])&(~op[3])&(op[2])&(~op[1])&(op[0]);
    assign i_lb=(op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0]);
    assign i_lbu=(op[5])&(~op[4])&(~op[3])&(op[2])&(~op[1])&(~op[0]);
    assign i_sb=(op[5])&(~op[4])&(op[3])&(~op[2])&(~op[1])&(~op[0]);
    assign i_sh=(op[5])&(~op[4])&(op[3])&(~op[2])&(~op[1])&(op[0]);

    assign i_mfc0=(~op[5])&(op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~rs[4])&(~rs[3])&(~rs[2])&(~rs[1])&(~rs[0]);
    assign i_mtc0=(~op[5])&(op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~rs[4])&(~rs[3])&(rs[2])&(~rs[1])&(~rs[0]);
    assign
i_mfhi=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(func[4])&(~func[3])&(~func[2])&(~func[1])&(~func[
0]);
    assign
i_mthi=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(func[4])&(~func[3])&(~func[2])&(~func[1])&(func[0
]);
    assign
i_mflo=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(func[4])&(~func[3])&(~func[2])&(func[1])&(~func[0
]);
    assign
```

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

	assign
i_syscall=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(func[3])&(func[2])&(~func[1])&(~func
[0]);
	assign
i_break=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(func[3])&(func[2])&(~func[1])&(func[0]
);
	assign
i_teq=(~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(func[4])&(~func[3])&(func[2])&(~func[1])&(~func[0]);
	assign
i_eret=(~op[5])&(op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(rs[4])&(~rs[3])&(~rs[2])&(~rs[1])&(~rs[0])&(~func[5])&(func[4])&
(func[3])&(~func[2])&(~func[1])&(~func[0]);




	//实例化，产生控制信号
	controller con(i_add, i_addu, i_addi, i_addiu, i_sub, i_subu,i_and, i_andi, i_or, i_ori, i_xor, i_xori, i_nor,
		i_lui,i_sll, i_sllv, i_sra, i_srav, i_srl, i_srlv,i_slt, i_slti, i_sltu, i_sltiu,i_beq, i_bne,
		i_j, i_jal, i_jr,i_lw, i_sw,
		i_clz,
		i_div,i_divu,i_mul,i_multu,
		i_bgez,
		i_jalr,
		i_lh,i_lhu, i_lb,i_lbu,i_sh,i_sb,
		i_mfc0,i_mtc0,i_mfhi,i_mthi, i_mflo,i_mtlo,
		i_syscall, i_break,i_teq,i_eret,

		Z,C,N,O,
		busy_div,busy_divu,
		over_div,over_divu,
		start_div,start_divu,pc_no_add,
		RF_W,ALUC,
		M1,M2,M3,M4,M5,M6,M7,M8,M9,
		DM_W,DMEM_in_type,conv1type,conv2type,loin,hiin,
		mfc0_c,mtc0_c,execption,eret_c,cause);

endmodule


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

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

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

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

module EXT_8(
input [7:0]a,
output [31:0]b
);
assign b={{24{1'b0}},a};
endmodule

module S_EXT_8(
input [7:0]a,
```

```
output [31:0]b
);
assign b={{24{a[7]}},a};
endmodule

module instruction_to_control(
input [31:0]instru,
input Z,
input C,
input N,
input O,


input busy_div,
input busy_divu,

input over_div,
input over_divu,

output start_div,
output start_divu,
output pc_no_add,

output    RF_W,
output    [3:0]ALUC,

output    [1:0]M1,
output    M2,
output    [1:0]M3,
output    [2:0]M4,
output    [1:0]M5,
output    M6,
output    M7,
output    [2:0]M8,
output    [2:0]M9,

output    DM_W,
output [1:0]DMEM_in_type,
output [1:0]conv1type,//sw:00,sh:01,sb:10
output [2:0]conv2type,//lw:000,lh:001,lhu:010,lb:011,lbu:100
output loin,
output hiin,
output mfc0_c,
output mtc0_c,
output execption,
output eret_c,
output [4:0]cause
      );

            i_to_controller i_to_con(instru[31:26],instru[25:21],instru[5:0],
            Z,C,N,O,
            busy_div,busy_divu,
            over_div,over_divu,
            start_div,start_divu,pc_no_add,
            RF_W,ALUC,
            M1,M2,M3,M4,M5,M6,M7,M8,M9,
            DM_W,DMEM_in_type,conv1type,conv2type,loin,hiin,
            mfc0_c,mtc0_c,execption,eret_c,cause);

endmodule

module DMEM(
input clk,
//input ena,
input [1:0]in_type,//00:32,01:16,10:8
input [1:0]addr_tail,//地址尾部
input wena,//高电平写入有效，修改：高低电平读均有效
input [10:0] addr,
input [31:0] data_in32,
input [15:0] data_in16,
input [7:0] data_in8,
output reg [31:0] data_out
      );

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

     // assign data_out= memory[addr];

      always @(*)
      begin
```

```
if(addr==0)
data_out=memory[0];
else if(addr==1)
data_out=memory[1];
else if(addr==2)
data_out=memory[2];
else if(addr==3)
data_out=memory[3];
else if(addr==4)
data_out=memory[4];
else if(addr==5)
data_out=memory[5];
else if(addr==6)
data_out=memory[6];
else if(addr==7)
data_out=memory[7];
else if(addr==8)
data_out=memory[8];
else if(addr==9)
data_out=memory[9];
else if(addr==10)
data_out=memory[10];
else if(addr==11)
data_out=memory[11];
else if(addr==12)
data_out=memory[12];
else if(addr==13)
data_out=memory[13];
else if(addr==14)
data_out=memory[14];
else if(addr==15)
data_out=memory[15];
else if(addr==16)
data_out=memory[16];
else if(addr==17)
data_out=memory[17];
else if(addr==18)
data_out=memory[18];
else if(addr==19)
data_out=memory[19];
else if(addr==20)
data_out=memory[20];
else if(addr==21)
data_out=memory[21];
else if(addr==22)
data_out=memory[22];
else if(addr==23)
data_out=memory[23];
else if(addr==24)
data_out=memory[24];
else if(addr==25)
data_out=memory[25];
else if(addr==26)
data_out=memory[26];
else if(addr==27)
data_out=memory[27];
else if(addr==28)
data_out=memory[28];
else if(addr==29)
data_out=memory[29];
else if(addr==30)
data_out=memory[30];
else
data_out=memory[31];



end

always @(posedge clk )
begin


if(wena==1)//写有效
begin

if(in_type==2'b00)
begin
if(addr==0)
```

```
memory[0]=data_in32;
else if(addr==1)
memory[1]=data_in32;
else if(addr==2)
memory[2]=data_in32;
else if(addr==3)
memory[3]=data_in32;
else if(addr==4)
memory[4]=data_in32;
else if(addr==5)
memory[5]=data_in32;
else if(addr==6)
memory[6]=data_in32;
else if(addr==7)
memory[7]=data_in32;
else if(addr==8)
memory[8]=data_in32;
else if(addr==9)
memory[9]=data_in32;
else if(addr==10)
memory[10]=data_in32;
else if(addr==11)
memory[11]=data_in32;
else if(addr==12)
memory[12]=data_in32;
else if(addr==13)
memory[13]=data_in32;
else if(addr==14)
memory[14]=data_in32;
else if(addr==15)
memory[15]=data_in32;
else if(addr==16)
memory[16]=data_in32;
else if(addr==17)
memory[17]=data_in32;
else if(addr==18)
memory[18]=data_in32;
else if(addr==19)
memory[19]=data_in32;
else if(addr==20)
memory[20]=data_in32;
else if(addr==21)
memory[21]=data_in32;
else if(addr==22)
memory[22]=data_in32;
else if(addr==23)
memory[23]=data_in32;
else if(addr==24)
memory[24]=data_in32;
else if(addr==25)
memory[25]=data_in32;
else if(addr==26)
memory[26]=data_in32;
else if(addr==27)
memory[27]=data_in32;
else if(addr==28)
memory[28]=data_in32;
else if(addr==29)
memory[29]=data_in32;
else if(addr==30)
memory[30]=data_in32;
else
memory[31]=data_in32;
end

else if(in_type==2'b01)
begin
if(addr_tail[1]==1'b0)
memory[addr][15:0]=data_in16;
else
memory[addr][31:16]=data_in16;
end


else if(in_type==2'b10)
begin
if(addr_tail==2'b00)
memory[addr][7:0]=data_in8;
else if(addr_tail==2'b01)
memory[addr][15:8]=data_in8;
```

```
        else if(addr_tail==2'b10)
        memory[addr][23:16]=data_in8;
        else
        memory[addr][31:24]=data_in8;
        end


      // memory[addr]=data_in;
        end
        else//读有效
        begin
    //    data_out=memory[addr];
        end

        end



endmodule


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

output [31:0] exc_addr_out,
output [31:0] rdata_out,

output [31:0] conv1_data_out32,
output [15:0] conv1_data_out16,
output [7:0] conv1_data_out8,
output [31:0] conv2_data_out,

        output [63:0] mult_z_out,
        output [63:0] multu_z_out,
        output [31:0] div_r_out,
        output [31:0] div_q_out,
        output [31:0] divu_r_out,
        output [31:0] divu_q_out,

        output [31:0] hilo_hi_out,
        output [31:0] hilo_lo_out,

output Z,
output C,
output N,
output O,

//output busy_mult,
//output busy_multu,
output busy_div,
output busy_divu,
//output over_mult,
//output over_multu,
output over_div,
output over_divu,

//input start_mult,
//input start_multu,
input start_div,
input start_divu,
input pc_no_add,
//input    PC_CLK,
//input    IM_R,
input    RF_W,
//input    RF_CLK,
input    [3:0]ALUC,

input    [1:0]M1,
input    M2,
input    [1:0]M3,
input    [2:0]M4,
input    [1:0]M5,
```

```
input    M6,
input    M7,
input    [2:0]M8,
input    [2:0]M9,

//input    DM_W,
input [1:0]conv1type,//sw:00,sh:01,sb:10
input [2:0]conv2type,//lw:000,lh:001,lhu:010,lb:011,lbu:100
input loin,
input hiin

/*input mfc0_c,
input mtc0_c,
input execption,
input eret_c,
input [4:0]cause*/


    );



    //各个部件的输出
    /*------------------------*/
    wire [31:0] mux1_out;
    wire [31:0] mux2_out;
    wire [31:0] mux3_out;
    wire [31:0] mux4_out;
    wire [4:0] mux5_out;
    wire [31:0] mux6_out;
    wire [31:0] mux7_out;
    wire [31:0] mux8_out;
    wire [31:0] mux9_out;

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

    wire [31:0] join_out;
   // wire [31:0] pc_out;
    wire [31:0] npc_out;
    //wire [31:0] imem_out;
   // wire [31:0] dmem_out;
   // wire [31:0] dmem_data_out;
    wire [31:0] add_out;
    wire [31:0] rs_out;
   // wire [31:0] rt_out;
    //wire [31:0] alu_out;




    /*wire [63:0] mult_z_out;
    wire [63:0] multu_z_out;
    wire [31:0] div_r_out;
    wire [31:0] div_q_out;
    wire [31:0] divu_r_out;
    wire [31:0] divu_q_out;

    wire [31:0] hilo_hi_out;
    wire [31:0] hilo_lo_out;*/



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


    //选择器部分
    selector41 mux1(mux7_out,mux6_out,exc_addr_out,32'b0,M1,mux1_out);
    selector21 mux2(ext5_out,rs_out,M2,mux2_out);
    selector41 mux3(rt_out,ext16_out,s_ext16_out,32'b0,M3,mux3_out);
    selector81 mux4(alu_out,npc_out,conv2_data_out,hilo_hi_out,hilo_lo_out,rdata_out,mult_z_out[31:0],32'b0,M4,mux4_out);
    selector41_5 mux5(imem_out[15:11],imem_out[20:16],5'b11111,5'b0,M5,mux5_out);
    selector21 mux6(npc_out,add_out,M6,mux6_out);
    selector21 mux7(rs_out,join_out,M7,mux7_out);
    selector81 mux8(mult_z_out[63:32],multu_z_out[63:32],div_r_out,divu_r_out,rs_out,32'b0,32'b0,32'b0,M8,mux8_out);
    selector81 mux9(mult_z_out[31:0],multu_z_out[31:0],div_q_out,divu_q_out,rs_out,32'b0,32'b0,32'b0,M9,mux9_out);
```

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

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


    Regfiles cpu_ref(clk_in,reset,RF_W,imem_out[25:21],imem_out[20:16],mux5_out,mux4_out,rs_out,rt_out);
    alu ALunit(mux2_out,mux3_out,ALUC,reset,alu_out,Z,C,N,O);

    MULT mu1(clk_in,reset,rs_out,rt_out,mult_z_out);
    MULTU mu2(clk_in,reset,rs_out,rt_out,multu_z_out);
    DIV di1(rs_out,rt_out,start_div,clk_in,reset,div_q_out,div_r_out,busy_div,over_div);
    DIVU di2(rs_out,rt_out,start_divu,clk_in,reset,divu_q_out,divu_r_out,busy_divu,over_divu);

    HILO hl(clk_in,reset,mux8_out,mux9_out,hiin,loin,hilo_hi_out,hilo_lo_out);

    MEM_CONV1 mc1(conv1type,alu_out[1:0],rt_out,conv1_data_out32,conv1_data_out16,conv1_data_out8);
    MEM_CONV2 mc2(conv2type,alu_out[1:0],dmem_out,conv2_data_out);


endmodule

module controller(

//input：输入的指令种类
input i_add,input i_addu,input i_addi,input i_addiu,input i_sub,input i_subu,
    input i_and,input i_andi,input i_or,input i_ori,input i_xor,input i_xori,input i_nor,
    input i_lui,
    input i_sll,input i_sllv,input i_sra,input i_srav,input i_srl,input i_srlv,
    input i_slt,input i_slti,input i_sltu,input i_sltiu,
    input i_beq,input i_bne,
    input i_j,input i_jal,input i_jr,
    input i_lw,input i_sw,
    input i_clz,
    input i_div,input i_divu,input i_mult,input i_multu,
    input i_bgez,
    input i_jalr,
    input i_lh,input i_lhu,input i_lb,input i_lbu,input i_sh,input i_sb,
    input i_mfc0,input i_mtc0,input i_mfhi,input i_mthi,input i_mflo,input i_mtlo,
    input i_syscall,input i_break,input i_teq,input i_eret,

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


input busy_div,
input busy_divu,

input over_div,
input over_divu,

output start_div,
output start_divu,
output pc_no_add,

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

output    [1:0]M1,
output    M2,
output    [1:0]M3,
output    [2:0]M4,
output    [1:0]M5,
output    M6,
```

```
output    M7,
output    [2:0]M8,
output    [2:0]M9,

//output    CS,
//output    DM_R,
output    DM_W,
output [1:0]DMEM_in_type,
output [1:0]conv1type,//sw:00,sh:01,sb:10
output [2:0]conv2type,//lw:000,lh:001,lhu:010,lb:011,lbu:100
output loin,
output hiin,
output mfc0_c,
output mtc0_c,
output execption,
output eret_c,
output [4:0]cause
//output start

      );


    // assign start_mult=i_mult&(~busy_mult)&(~over_mult);
//      assign start_multu=i_multu&(~busy_multu)&(~over_multu);
      assign start_div=i_div&(~busy_div)&(~over_div);
      assign start_divu=i_divu&(~busy_divu)&(~over_divu);
      assign pc_no_add=busy_div|busy_divu|start_div|start_divu;

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

      //由此可见 regfle 的 rd 在下降沿写入，根据运算结果决定
      assign RF_W=((~O)&(i_add|i_addi|i_sub))|(i_addu)|(i_addiu)|(i_subu)|(i_and)|(i_andi)|(i_or)|(i_ori)|
      (i_xor)|(i_xori)|(i_nor)|(i_lui)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl)|(i_srlv)|
      (i_slt)|(i_slti)|(i_sltu)|(i_sltiu)|(i_jal)|(i_lw)|i_clz|i_jalr|i_lh|i_lhu|i_lb|i_lbu|i_mfc0|i_mflo|i_mfhi|i_mult;

      //assign RF_CLK=

     // assign ALUC=2;
      assign ALUC[3]=(i_lui)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl)|(i_srlv)|(i_slt)|(i_slti)|(i_sltu)|(i_sltiu)|i_clz;
      assign ALUC[2]=(i_and)|(i_andi)|(i_or)|(i_ori)|(i_xor)|(i_xori)|(i_nor)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl)|(i_srlv);
      assign
ALUC[1]=(i_add)|(i_addi)|(i_sub)|(i_xor)|(i_xori)|(i_nor)|(i_sll)|(i_sllv)|(i_slt)|(i_slti)|(i_sltu)|(i_sltiu)|(i_lw)|(i_sw)|i_lh|i_lhu|i_lb|i_lbu|i_s
h|i_sb|i_bgez;
      assign ALUC[0]=(i_sub)|(i_subu)|(i_or)|(i_ori)|(i_nor)|(i_srl)|(i_srlv)|(i_slt)|(i_slti)|(i_beq)|(i_bne)|i_clz|i_bgez|i_teq;



      assign M1[0]=(i_add)|(i_addu)|(i_addi)|(i_addiu)|(i_sub)|(i_subu)|(i_and)|(i_andi)|(i_or)|(i_ori)|
      (i_xor)|(i_xori)|(i_nor)|(i_lui)|(i_sll)|(i_sllv)|(i_sra)|(i_srav)|(i_srl)|(i_srlv)|
      (i_slt)|(i_slti)|(i_sltu)|(i_sltiu)|(i_beq)|(i_bne)|(i_lw)|(i_sw)|i_bgez|i_lh|i_lhu|i_lb|i_lbu|i_sh|i_sb|
      i_mult|i_multu|i_div|i_divu|i_mfc0|i_mtc0|i_mflo|i_mtlo|i_mfhi|i_mthi|i_clz|(~Z&i_teq);
      assign M1[1]=i_syscall|i_break|(Z&i_teq)|i_eret;

      assign M2=(i_add)|(i_addu)|(i_addi)|(i_addiu)|(i_sub)|(i_subu)|(i_and)|(i_andi)|(i_or)|(i_ori)|
      (i_xor)|(i_xori)|(i_nor)|(i_lui)|(i_sllv)|(i_srav)|(i_srlv)|
      (i_slt)|(i_slti)|(i_sltu)|(i_sltiu)|i_clz|i_teq|(i_beq)|(i_bne)|i_bgez|i_jalr;

      assign M3[0]=i_andi|i_ori|i_xori|i_lui|i_bgez;
      assign M3[1]=i_addi|i_addiu|i_slti|i_sltiu|i_lw|i_lh|i_lhu|i_lb|i_lbu|i_sw|i_sh|i_sb|i_bgez;

      assign M4[0]=i_jal|i_jalr|i_mfc0|i_mfhi;
      assign M4[1]=i_lw|i_lh|i_lhu|i_lb|i_lbu|i_mfhi|i_mult;
      assign M4[2]=i_mfc0|i_mflo|i_mult;


      assign M5[0]=i_addi|i_addiu|i_andi|i_ori|i_xori|i_lui|i_slti|i_sltiu|i_beq|i_bne|i_bgez|i_lw|i_lh|i_lhu|i_lb|i_lbu|i_sw|i_sh|i_sb|i_mfc0;
      assign M5[1]=i_jal;

      //assign M6=i_beq|i_bne|i_bgez|i_mfc0|i_mtc0|i_mflo|i_mtlo|i_mfhi|i_mthi;
      assign M6=(Z&i_beq)|(~Z&i_bne)|(~N&i_bgez);

      assign M7=i_j|i_jal;

      assign M8[0]=i_multu|i_divu;
      assign M8[1]=i_div|i_divu;
      assign M8[2]=i_mthi;

      assign M9[0]=i_multu|i_divu;
      assign M9[1]=i_div|i_divu;
```

```
	assign M9[2]=i_mtlo;



	//assign CS=(i_lw)|(i_sw);
	//assign DM_R=(i_lw);
	assign DM_W=(i_sw)|i_sh|i_sb;

	assign DMEM_in_type[0]=i_sh;
	assign DMEM_in_type[1]=i_sb;

	assign conv1type[0]=i_sh;
	assign conv1type[1]=i_sb;

	assign conv2type[0]=i_lh|i_lb;
	assign conv2type[1]=i_lhu|i_lb;
	assign conv2type[2]=i_lbu;

	assign loin=i_mult|i_multu|i_div|i_divu|i_mtlo;
	assign hiin=i_mult|i_multu|i_div|i_divu|i_mthi;

	assign mfc0_c=i_mfc0;
	assign mtc0_c=i_mtc0;

	assign execption=i_syscall|i_break|(Z&i_teq)|i_eret;
	assign eret_c=i_eret;

	assign cause[0]=i_break|i_teq;
	assign cause[1]=0;
	assign cause[2]=i_teq;
	assign cause[3]=1;
	assign cause[4]=0;



endmodule

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

	reg [31:0]kkk;

	always @(*)
	begin

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

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

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

```
negative=0;

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

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


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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

end

//clz:计算前导 0 个数
else if (aluc == 4'b1001)
begin

if(a[31]==1)
	r=0;
else if(a[30]==1)
	r=1;
else if(a[29]==1)
	r=2;
else if(a[28]==1)
	r=3;
else if(a[27]==1)
	r=4;
else if(a[26]==1)
	r=5;
else if(a[25]==1)
	r=6;
else if(a[24]==1)
	r=7;
else if(a[23]==1)
	r=8;
else if(a[22]==1)
	r=9;
else if(a[21]==1)
	r=10;
else if(a[20]==1)
	r=11;
else if(a[19]==1)
	r=12;
```

```
  else if(a[18]==1)
        r=13;
  else if(a[17]==1)
        r=14;
  else if(a[16]==1)
        r=15;
  else if(a[15]==1)
        r=16;
  else if(a[14]==1)
        r=17;
  else if(a[13]==1)
        r=18;
  else if(a[12]==1)
        r=19;
  else if(a[11]==1)
        r=20;
  else if(a[10]==1)
        r=21;
  else if(a[9]==1)
        r=22;
  else if(a[8]==1)
        r=23;
  else if(a[7]==1)
        r=24;
  else if(a[6]==1)
        r=25;
  else if(a[5]==1)
        r=26;
  else if(a[4]==1)
        r=27;
  else if(a[3]==1)
        r=28;
  else if(a[2]==1)
        r=29;
  else if(a[1]==1)
        r=30;
  else if(a[0]==1)
        r=31;
  else
        r=32;


  end


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

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



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

  kkk=a-b;

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

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

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

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

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

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


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

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


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

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

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

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

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

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


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

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

```
        negative=0;

        end
        else
        begin

        end
        end
        end
endmodule


module CP0(
input clk,
input rst,
input mfc0,//读 cp0 寄存器
input mtc0,//写 cp0 寄存器
input [31:0]pc,
input [4:0]Rd,//写 cp0 寄存器地址
input [31:0]wdata,//写 cp0 寄存器值
input exception,
input eret,//eret 异常返回
input [4:0]cause,//中断序号，5'b01000(syscall)  5'b01001(break)  5'b01101(teq)
input intr,//外部中断，没用
output [31:0]rdata,//读到的 cp0 寄存器值
output [31:0]status,
output reg timer_int,//定时产生外部中断，没用
output reg [31:0]exc_addr
        );

        reg [31:0] reg_cp0[0:31];
        /*status：12 号寄存器，cause：13 号寄存器，epc：14 号寄存器*/
        /*cause[6:2]为异常类型号，5'b01000(syscall)  5'b01001(break)  5'b01101(teq)*/

        assign rdata = mfc0 ? reg_cp0[Rd] : 0;
        assign status = reg_cp0[12];


        always @(negedge clk or posedge rst)
        begin
        if(rst==1'b1)
        begin
        reg_cp0[12]<=32'h0000000f;
        reg_cp0[13]<=32'h00000000;
        reg_cp0[14]<=32'h00000000;
        exc_addr<=32'h00000000;
        end
        else if(mtc0==1'b1)
        begin
        reg_cp0[Rd]<=wdata;
        end
        else if(exception==1'b1)
        begin
            if(eret==1'b1)
              begin
              reg_cp0[12]<=reg_cp0[12]>>5;//关闭异常，右移 5 位，开中断
              exc_addr<=reg_cp0[14];
              end
              else if(cause==5'b01000)//syscall
              begin
                    if(reg_cp0[12][1]==1'b1)
                    begin
                    exc_addr<=4;//异常地址入口为 0x4
                    reg_cp0[12]<=reg_cp0[12]<<5;
                    reg_cp0[13][6:2]<=5'b01000;
                    reg_cp0[14]<=pc;
                    end
                    else
                    begin
                    exc_addr<=pc+4;
                    end
              end
              else if(cause==5'b01001)//break
              begin
                    if(reg_cp0[12][2]==1'b1)
                    begin
                    exc_addr<=4;
                    reg_cp0[12]<=reg_cp0[12]<<5;
                    reg_cp0[13][6:2]<=5'b01001;
```

```
                reg_cp0[14]<=pc;
                end
                else
                begin
                exc_addr<=pc+4;
                end
            end
            else if(cause==5'b01101)//teq
            begin
                if(reg_cp0[12][2]==1'b1)
                begin
                exc_addr<=4;
                reg_cp0[12]<=reg_cp0[12]<<5;
                reg_cp0[13][6:2]<=5'b01101;
                reg_cp0[14]<=pc;
                end
                else
                begin
                exc_addr<=pc+4;
                end
            end
            else
            begin
            //其他指令，未实现
            end


        end


        end



endmodule


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 reg busy,
output reg over
        );

        wire ready;
        reg [4:0]count;
        reg [31:0]reg_q;
        reg [31:0]reg_r;
        reg [31:0]reg_b;
        reg busy2,r_sign;

        wire [31:0] dividend_yuan= dividend[31]?(~(dividend-1)):dividend;
        wire [31:0] divisor_yuan= divisor[31]?(~(divisor-1)):divisor;
        wire dividend_sign= dividend[31];
        wire divisor_sign= divisor[31];
        wire [31:0] rr = r_sign ? reg_r + reg_b : reg_r;
        wire [31:0] qq = reg_q;


        assign ready = ~busy & busy2;

        wire [32:0] sub_add = r_sign ? ({reg_r,qq[31]}+{1'b0,reg_b}) : ({reg_r,qq[31]}-{1'b0,reg_b}) ;

        assign r= dividend_sign?((~rr)+1):rr;
        assign q= (dividend_sign+divisor_sign)?((~qq)+1):qq;


        always @(posedge clock or posedge reset)
        begin
        if(reset==1)
        begin
            count<=5'b0;
            busy<=0;
            busy2<=0;
            over<=0;
```

```
        end
        else
        begin
            busy2<=busy;
            if(over==1)
            begin
            over<=0;
            end
            else if(start)
            begin
                reg_r<=32'b0;
                r_sign<=0;
                reg_q<=dividend_yuan;
                reg_b<=divisor_yuan;
                count<=5'b0;
                busy<=1'b1;
                over<=0;
            end
            else if(busy)
            begin
                reg_r<=sub_add[31:0];
                r_sign<=sub_add[32];
                reg_q<={reg_q[30:0],~sub_add[32]};
                count<=count+5'b1;
                if(count==5'b11111)
                begin
                    busy<=0;
                    over<=1;
                end
            end
        end
        end
endmodule


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 reg busy,
output reg over
        );
        wire ready;
        reg [4:0]count;
        reg [31:0]reg_q;
        reg [31:0]reg_r;
        reg [31:0]reg_b;
        reg busy2,r_sign;
        assign ready = ~busy & busy2;

        wire [32:0] sub_add = r_sign ? ({reg_r,q[31]}+{1'b0,reg_b}) : ({reg_r,q[31]}-{1'b0,reg_b}) ;
        assign r = r_sign ? reg_r + reg_b : reg_r;
        assign q = reg_q;

        always @(posedge clock or posedge reset)
        begin
        if(reset==1)
        begin
            count<=5'b0;
            busy<=0;
            busy2<=0;
            over<=0;
        end
        else
        begin
            busy2<=busy;
            if(over==1)
            begin
            over<=0;
            end
            else if(start)
            begin
                reg_r<=32'b0;
                r_sign<=0;
                reg_q<=dividend;
                reg_b<=divisor;
```

```
                count<=5'b0;
                busy<=1'b1;
                over<=0;
            end
            else if(busy)
            begin
                reg_r<=sub_add[31:0];
                r_sign<=sub_add[32];
                reg_q<={reg_q[30:0],~sub_add[32]};
                count<=count+5'b1;
                if(count==5'b11111)
                begin
                    busy<=0;
                    over<=1;
                    end
            end
        end
        end
endmodule


module MULT(
input clk,
input reset,
input [31:0]a,
input [31:0]b,
output [63:0]z
        );
        reg [63:0] temp;
        reg [63:0] stored0;
        reg [63:0] stored1;
        reg [63:0] stored2;
        reg [63:0] stored3;
        reg [63:0] stored4;
        reg [63:0] stored5;
        reg [63:0] stored6;
        reg [63:0] stored7;
        reg [63:0] stored8;
        reg [63:0] stored9;
        reg [63:0] stored10;
        reg [63:0] stored11;
        reg [63:0] stored12;
        reg [63:0] stored13;
        reg [63:0] stored14;
        reg [63:0] stored15;
        reg [63:0] stored16;
        reg [63:0] stored17;
        reg [63:0] stored18;
        reg [63:0] stored19;
        reg [63:0] stored20;
        reg [63:0] stored21;
        reg [63:0] stored22;
        reg [63:0] stored23;
        reg [63:0] stored24;
        reg [63:0] stored25;
        reg [63:0] stored26;
        reg [63:0] stored27;
        reg [63:0] stored28;
        reg [63:0] stored29;
        reg [63:0] stored30;
        reg [63:0] stored31;
        /*-------------------------*/
        reg [63:0] add01;
        reg [63:0] add23;
        reg [63:0] add45;
        reg [63:0] add67;
        reg [63:0] add89;
        reg [63:0] add1011;
        reg [63:0] add1213;
        reg [63:0] add1415;
        reg [63:0] add1617;
        reg [63:0] add1819;
        reg [63:0] add2021;
        reg [63:0] add2223;
        reg [63:0] add2425;
        reg [63:0] add2627;
        reg [63:0] add2829;
        reg [63:0] add3031;
        /*-------------------------*/
        reg [63:0] add0123;
```

```
  reg [63:0] add4567;
  reg [63:0] add891011;
  reg [63:0] add12131415;
  reg [63:0] add16171819;
  reg [63:0] add20212223;
  reg [63:0] add24252627;
  reg [63:0] add28293031;
  /*------------------------------*/
  reg [63:0] add01234567;
  reg [63:0] add89101112131415;
  reg [63:0] add1617181920212223;
  reg [63:0] add2425262728293031;
  /*------------------------------*/
  reg [63:0] add0_15;
  reg [63:0] add16_31;
  /*------------------------------*/
// reg [63:0] add0_31;



wire [31:0] aa;
wire [31:0] bb;

assign aa=a[31]?(~(a-1)) :a;
assign bb=b[31]?(~(b-1)) :b;

always @(*)
begin
if(reset)
begin
temp=0;
stored0 = 0;
stored1 = 0;
stored2 = 0;
stored3 = 0;
stored4 = 0;
stored5 = 0;
stored6 = 0;
stored7 = 0;
stored8 = 0;
stored9 = 0;
stored10 = 0;
stored11 = 0;
stored12 = 0;
stored13 = 0;
stored14 = 0;
stored15 = 0;
stored16 = 0;
stored17 = 0;
stored18 = 0;
stored19 = 0;
stored20 = 0;
stored21 = 0;
stored22 = 0;
stored23 = 0;
stored24 = 0;
stored25 = 0;
stored26 = 0;
stored27 = 0;
stored28 = 0;
stored29 = 0;
stored30 = 0;
stored31 = 0;
add01 = 0;
add23 = 0;
add45 = 0;
add67 = 0;
add89 = 0;
add1011 = 0;
add1213 = 0;
add1415 = 0;
add1617 = 0;
add1819 = 0;
add2021 = 0;
add2223 = 0;
add2425 = 0;
add2627 = 0;
add2829 = 0;
add3031 = 0;
add0123 = 0;
```

```
add4567 = 0;
add891011 = 0;
add12131415 = 0;
add16171819 = 0;
add20212223 = 0;
add24252627 = 0;
add28293031 = 0;
add01234567 = 0;
add89101112131415 = 0;
add1617181920212223 = 0;
add2425262728293031 = 0;
add0_15 = 0;
add16_31 = 0;
//add0_31 = 0;

end
else
begin


stored0 = bb[0] ? {32'b0 , aa} : 64'b0;
stored1 = bb[1] ? {31'b0 , aa , 1'b0} : 64'b0;
stored2 = bb[2] ? {30'b0 , aa , 2'b0} : 64'b0;
stored3 = bb[3] ? {29'b0 , aa , 3'b0} : 64'b0;
stored4 = bb[4] ? {28'b0 , aa , 4'b0} : 64'b0;
stored5 = bb[5] ? {27'b0 , aa , 5'b0} : 64'b0;
stored6 = bb[6] ? {26'b0 , aa , 6'b0} : 64'b0;
stored7 = bb[7] ? {25'b0 , aa , 7'b0} : 64'b0;
stored8 = bb[8] ? {24'b0 , aa , 8'b0} : 64'b0;
stored9 = bb[9] ? {23'b0 , aa , 9'b0} : 64'b0;
stored10 = bb[10] ? {22'b0 , aa , 10'b0} : 64'b0;
stored11 = bb[11] ? {21'b0 , aa , 11'b0} : 64'b0;
stored12 = bb[12] ? {20'b0 , aa , 12'b0} : 64'b0;
stored13 = bb[13] ? {19'b0 , aa , 13'b0} : 64'b0;
stored14 = bb[14] ? {18'b0 , aa , 14'b0} : 64'b0;
stored15 = bb[15] ? {17'b0 , aa , 15'b0} : 64'b0;
stored16 = bb[16] ? {16'b0 , aa , 16'b0} : 64'b0;
stored17 = bb[17] ? {15'b0 , aa , 17'b0} : 64'b0;
stored18 = bb[18] ? {14'b0 , aa , 18'b0} : 64'b0;
stored19 = bb[19] ? {13'b0 , aa , 19'b0} : 64'b0;
stored20 = bb[20] ? {12'b0 , aa , 20'b0} : 64'b0;
stored21 = bb[21] ? {11'b0 , aa , 21'b0} : 64'b0;
stored22 = bb[22] ? {10'b0 , aa , 22'b0} : 64'b0;
stored23 = bb[23] ? {9'b0 , aa , 23'b0} : 64'b0;
stored24 = bb[24] ? {8'b0 , aa , 24'b0} : 64'b0;
stored25 = bb[25] ? {7'b0 , aa , 25'b0} : 64'b0;
stored26 = bb[26] ? {6'b0 , aa , 26'b0} : 64'b0;
stored27 = bb[27] ? {5'b0 , aa , 27'b0} : 64'b0;
stored28 = bb[28] ? {4'b0 , aa , 28'b0} : 64'b0;
stored29 = bb[29] ? {3'b0 , aa , 29'b0} : 64'b0;
stored30 = bb[30] ? {2'b0 , aa , 30'b0} : 64'b0;
stored31 = bb[31] ? {1'b0 , aa , 31'b0} : 64'b0;

add01 = stored0 + stored1;
add23 = stored2 + stored3;
add45 = stored4 + stored5;
add67 = stored6 + stored7;
add89 = stored8 + stored9;
add1011 = stored10 + stored11;
add1213 = stored12 + stored13;
add1415 = stored14 + stored15;
add1617 = stored16 + stored17;
add1819 = stored18 + stored19;
add2021 = stored20 + stored21;
add2223 = stored22 + stored23;
add2425 = stored24 + stored25;
add2627 = stored26 + stored27;
add2829 = stored28 + stored29;
add3031 = stored30 + stored31;

add0123 = add01 + add23;
add4567 = add45 + add67;
add891011 = add89 + add1011;
add12131415 = add1213 + add1415;
add16171819 = add1617 + add1819;
add20212223 = add2021 + add2223;
add24252627 = add2425 + add2627;
add28293031 = add2829 + add3031;

add01234567 = add0123 + add4567;
```

```
    add89101112131415 = add891011 + add12131415;
    add1617181920212223 = add16171819 + add20212223;
    add2425262728293031 = add24252627 + add28293031;

    add0_15 = add01234567 + add89101112131415;
    add16_31 = add1617181920212223 + add2425262728293031;

   // add0_31 = add0_15 + add16_31;
    temp = add0_15 + add16_31;


    end
    end
    assign z= (a[31]^b[31]) ?((~temp)+1): temp;

endmodule


module MULTU(
input clk,
input reset,
input [31:0]a,
input [31:0]b,
output [63:0]z
      );
      reg [63:0] temp;
      reg [63:0] stored0;
      reg [63:0] stored1;
      reg [63:0] stored2;
      reg [63:0] stored3;
      reg [63:0] stored4;
      reg [63:0] stored5;
      reg [63:0] stored6;
      reg [63:0] stored7;
      reg [63:0] stored8;
      reg [63:0] stored9;
      reg [63:0] stored10;
      reg [63:0] stored11;
      reg [63:0] stored12;
      reg [63:0] stored13;
      reg [63:0] stored14;
      reg [63:0] stored15;
      reg [63:0] stored16;
      reg [63:0] stored17;
      reg [63:0] stored18;
      reg [63:0] stored19;
      reg [63:0] stored20;
      reg [63:0] stored21;
      reg [63:0] stored22;
      reg [63:0] stored23;
      reg [63:0] stored24;
      reg [63:0] stored25;
      reg [63:0] stored26;
      reg [63:0] stored27;
      reg [63:0] stored28;
      reg [63:0] stored29;
      reg [63:0] stored30;
      reg [63:0] stored31;
      /*--------------------------*/
      reg [63:0] add01;
      reg [63:0] add23;
      reg [63:0] add45;
      reg [63:0] add67;
      reg [63:0] add89;
      reg [63:0] add1011;
      reg [63:0] add1213;
      reg [63:0] add1415;
      reg [63:0] add1617;
      reg [63:0] add1819;
      reg [63:0] add2021;
      reg [63:0] add2223;
      reg [63:0] add2425;
      reg [63:0] add2627;
      reg [63:0] add2829;
      reg [63:0] add3031;
      /*--------------------------*/
      reg [63:0] add0123;
      reg [63:0] add4567;
      reg [63:0] add891011;
      reg [63:0] add12131415;
```

```
  reg [63:0] add16171819;
  reg [63:0] add20212223;
  reg [63:0] add24252627;
  reg [63:0] add28293031;
  /*------------------------------*/
  reg [63:0] add01234567;
  reg [63:0] add89101112131415;
  reg [63:0] add1617181920212223;
  reg [63:0] add2425262728293031;
  /*------------------------------*/
  reg [63:0] add0_15;
  reg [63:0] add16_31;
  /*------------------------------*/
// reg [63:0] add0_31;



  always @(*)
  begin
  if(reset)
  begin
  temp=0;
  stored0 = 0;
  stored1 = 0;
  stored2 = 0;
  stored3 = 0;
  stored4 = 0;
  stored5 = 0;
  stored6 = 0;
  stored7 = 0;
  stored8 = 0;
  stored9 = 0;
  stored10 = 0;
  stored11 = 0;
  stored12 = 0;
  stored13 = 0;
  stored14 = 0;
  stored15 = 0;
  stored16 = 0;
  stored17 = 0;
  stored18 = 0;
  stored19 = 0;
  stored20 = 0;
  stored21 = 0;
  stored22 = 0;
  stored23 = 0;
  stored24 = 0;
  stored25 = 0;
  stored26 = 0;
  stored27 = 0;
  stored28 = 0;
  stored29 = 0;
  stored30 = 0;
  stored31 = 0;
  add01 = 0;
  add23 = 0;
  add45 = 0;
  add67 = 0;
  add89 = 0;
  add1011 = 0;
  add1213 = 0;
  add1415 = 0;
  add1617 = 0;
  add1819 = 0;
  add2021 = 0;
  add2223 = 0;
  add2425 = 0;
  add2627 = 0;
  add2829 = 0;
  add3031 = 0;
  add0123 = 0;
  add4567 = 0;
  add891011 = 0;
  add12131415 = 0;
  add16171819 = 0;
  add20212223 = 0;
  add24252627 = 0;
  add28293031 = 0;
  add01234567 = 0;
  add89101112131415 = 0;
```

```
add1617181920212223 = 0;
add2425262728293031 = 0;
add0_15 = 0;
add16_31 = 0;


//add0_31 = 0;
end
else
begin


stored0 = b[0] ? {32'b0 , a} : 64'b0;
stored1 = b[1] ? {31'b0 , a , 1'b0} : 64'b0;
stored2 = b[2] ? {30'b0 , a , 2'b0} : 64'b0;
stored3 = b[3] ? {29'b0 , a , 3'b0} : 64'b0;
stored4 = b[4] ? {28'b0 , a , 4'b0} : 64'b0;
stored5 = b[5] ? {27'b0 , a , 5'b0} : 64'b0;
stored6 = b[6] ? {26'b0 , a , 6'b0} : 64'b0;
stored7 = b[7] ? {25'b0 , a , 7'b0} : 64'b0;
stored8 = b[8] ? {24'b0 , a , 8'b0} : 64'b0;
stored9 = b[9] ? {23'b0 , a , 9'b0} : 64'b0;
stored10 = b[10] ? {22'b0 , a , 10'b0} : 64'b0;
stored11 = b[11] ? {21'b0 , a , 11'b0} : 64'b0;
stored12 = b[12] ? {20'b0 , a , 12'b0} : 64'b0;
stored13 = b[13] ? {19'b0 , a , 13'b0} : 64'b0;
stored14 = b[14] ? {18'b0 , a , 14'b0} : 64'b0;
stored15 = b[15] ? {17'b0 , a , 15'b0} : 64'b0;
stored16 = b[16] ? {16'b0 , a , 16'b0} : 64'b0;
stored17 = b[17] ? {15'b0 , a , 17'b0} : 64'b0;
stored18 = b[18] ? {14'b0 , a , 18'b0} : 64'b0;
stored19 = b[19] ? {13'b0 , a , 19'b0} : 64'b0;
stored20 = b[20] ? {12'b0 , a , 20'b0} : 64'b0;
stored21 = b[21] ? {11'b0 , a , 21'b0} : 64'b0;
stored22 = b[22] ? {10'b0 , a , 22'b0} : 64'b0;
stored23 = b[23] ? {9'b0 , a , 23'b0} : 64'b0;
stored24 = b[24] ? {8'b0 , a , 24'b0} : 64'b0;
stored25 = b[25] ? {7'b0 , a , 25'b0} : 64'b0;
stored26 = b[26] ? {6'b0 , a , 26'b0} : 64'b0;
stored27 = b[27] ? {5'b0 , a , 27'b0} : 64'b0;
stored28 = b[28] ? {4'b0 , a , 28'b0} : 64'b0;
stored29 = b[29] ? {3'b0 , a , 29'b0} : 64'b0;
stored30 = b[30] ? {2'b0 , a , 30'b0} : 64'b0;
stored31 = b[31] ? {1'b0 , a , 31'b0} : 64'b0;

add01 = stored0 + stored1;
add23 = stored2 + stored3;
add45 = stored4 + stored5;
add67 = stored6 + stored7;
add89 = stored8 + stored9;
add1011 = stored10 + stored11;
add1213 = stored12 + stored13;
add1415 = stored14 + stored15;
add1617 = stored16 + stored17;
add1819 = stored18 + stored19;
add2021 = stored20 + stored21;
add2223 = stored22 + stored23;
add2425 = stored24 + stored25;
add2627 = stored26 + stored27;
add2829 = stored28 + stored29;
add3031 = stored30 + stored31;

add0123 = add01 + add23;
add4567 = add45 + add67;
add891011 = add89 + add1011;
add12131415 = add1213 + add1415;
add16171819 = add1617 + add1819;
add20212223 = add2021 + add2223;
add24252627 = add2425 + add2627;
add28293031 = add2829 + add3031;

add01234567 = add0123 + add4567;
add89101112131415 = add891011 + add12131415;
add1617181920212223 = add16171819 + add20212223;
add2425262728293031 = add24252627 + add28293031;

add0_15 = add01234567 + add89101112131415;
add16_31 = add1617181920212223 + add2425262728293031;

// add0_31 = add0_15 + add16_31;
```

```
        temp = add0_15 + add16_31;
        end


        end
        assign z=temp;

endmodule


module HILO(
input clk,
input rst,
input [31:0]hi_in,
input [31:0]lo_in,
input hi_w,
input lo_w,
output [31:0]hi_out,
output [31:0]lo_out
        );
        reg [31:0]hi;
        reg [31:0]lo;

        assign hi_out=hi;
        assign lo_out=lo;

        always @(negedge clk or posedge rst)
        begin
        if(rst==1'b1)
        begin
        hi<=32'b0;
        lo<=32'b0;
        end
        else
        begin
                if(hi_w==1)
                begin
                hi<=hi_in;
                end
                else
                begin
                end

                if(lo_w==1)
                begin
                lo<=lo_in;
                end
                else
                begin
                end

        end


        end

endmodule
```

四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

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

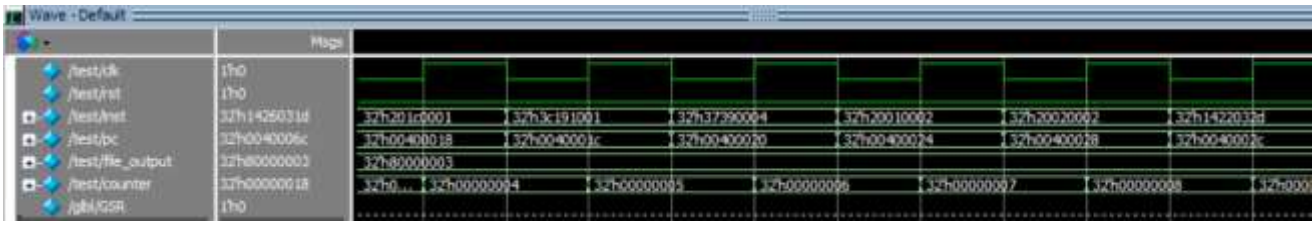
```
integer file_output;
integer counter=0 ;
initial begin
	file_output = $fopen("result.txt", "a");
	clk = 1'b0;
	rst = 1'b1;
	#50 rst=1'b0;
end

always begin
	#20 clk = !clk;
end

always @(posedge clk) begin
	if(counter>=1500)
	begin
	$fclose(file_output);
	end
	else if(clk==1'b1&&rst==1'b0)
	begin
	counter=counter+1;
	$fdisplay(file_output, "pc: %h", pc);
	$fdisplay(file_output, "instr: %h", inst);
	$fdisplay(file_output, "regfile0: %h", test.uut.sccpu.cpu_ref.array_reg[0]);
	$fdisplay(file_output, "regfile1: %h", test.uut.sccpu.cpu_ref.array_reg[1]);
	$fdisplay(file_output, "regfile2: %h", test.uut.sccpu.cpu_ref.array_reg[2]);
	$fdisplay(file_output, "regfile3: %h", test.uut.sccpu.cpu_ref.array_reg[3]);
	$fdisplay(file_output, "regfile4: %h", test.uut.sccpu.cpu_ref.array_reg[4]);
	$fdisplay(file_output, "regfile5: %h", test.uut.sccpu.cpu_ref.array_reg[5]);
	$fdisplay(file_output, "regfile6: %h", test.uut.sccpu.cpu_ref.array_reg[6]);
	$fdisplay(file_output, "regfile7: %h", test.uut.sccpu.cpu_ref.array_reg[7]);
	$fdisplay(file_output, "regfile8: %h", test.uut.sccpu.cpu_ref.array_reg[8]);
	$fdisplay(file_output, "regfile9: %h", test.uut.sccpu.cpu_ref.array_reg[9]);
	$fdisplay(file_output, "regfile10: %h", test.uut.sccpu.cpu_ref.array_reg[10]);
	$fdisplay(file_output, "regfile11: %h", test.uut.sccpu.cpu_ref.array_reg[11]);
	$fdisplay(file_output, "regfile12: %h", test.uut.sccpu.cpu_ref.array_reg[12]);
	$fdisplay(file_output, "regfile13: %h", test.uut.sccpu.cpu_ref.array_reg[13]);
	$fdisplay(file_output, "regfile14: %h", test.uut.sccpu.cpu_ref.array_reg[14]);
	$fdisplay(file_output, "regfile15: %h", test.uut.sccpu.cpu_ref.array_reg[15]);
	$fdisplay(file_output, "regfile16: %h", test.uut.sccpu.cpu_ref.array_reg[16]);
	$fdisplay(file_output, "regfile17: %h", test.uut.sccpu.cpu_ref.array_reg[17]);
```

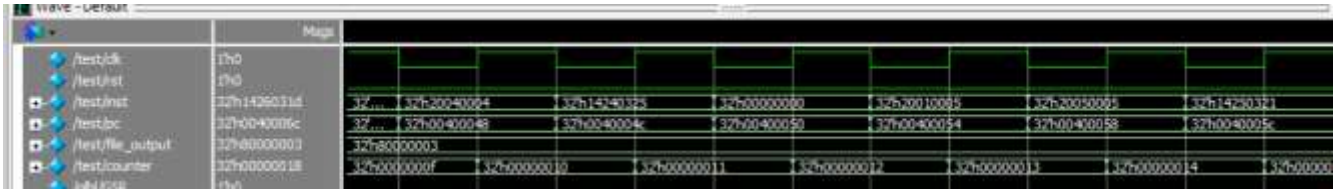
```
            $fdisplay(file_output, "regfile18: %h", test.uut.sccpu.cpu_ref.array_reg[18]);
            $fdisplay(file_output, "regfile19: %h", test.uut.sccpu.cpu_ref.array_reg[19]);
            $fdisplay(file_output, "regfile20: %h", test.uut.sccpu.cpu_ref.array_reg[20]);
            $fdisplay(file_output, "regfile21: %h", test.uut.sccpu.cpu_ref.array_reg[21]);
            $fdisplay(file_output, "regfile22: %h", test.uut.sccpu.cpu_ref.array_reg[22]);
            $fdisplay(file_output, "regfile23: %h", test.uut.sccpu.cpu_ref.array_reg[23]);
            $fdisplay(file_output, "regfile24: %h", test.uut.sccpu.cpu_ref.array_reg[24]);
            $fdisplay(file_output, "regfile25: %h", test.uut.sccpu.cpu_ref.array_reg[25]);
            $fdisplay(file_output, "regfile26: %h", test.uut.sccpu.cpu_ref.array_reg[26]);
            $fdisplay(file_output, "regfile27: %h", test.uut.sccpu.cpu_ref.array_reg[27]);
            $fdisplay(file_output, "regfile28: %h", test.uut.sccpu.cpu_ref.array_reg[28]);
            $fdisplay(file_output, "regfile29: %h", test.uut.sccpu.cpu_ref.array_reg[29]);
            $fdisplay(file_output, "regfile30: %h", test.uut.sccpu.cpu_ref.array_reg[30]);
            $fdisplay(file_output, "regfile31: %h", test.uut.sccpu.cpu_ref.array_reg[31]);
            end
            else
            begin

            end

    end

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


endmodule
```

五、实验结果

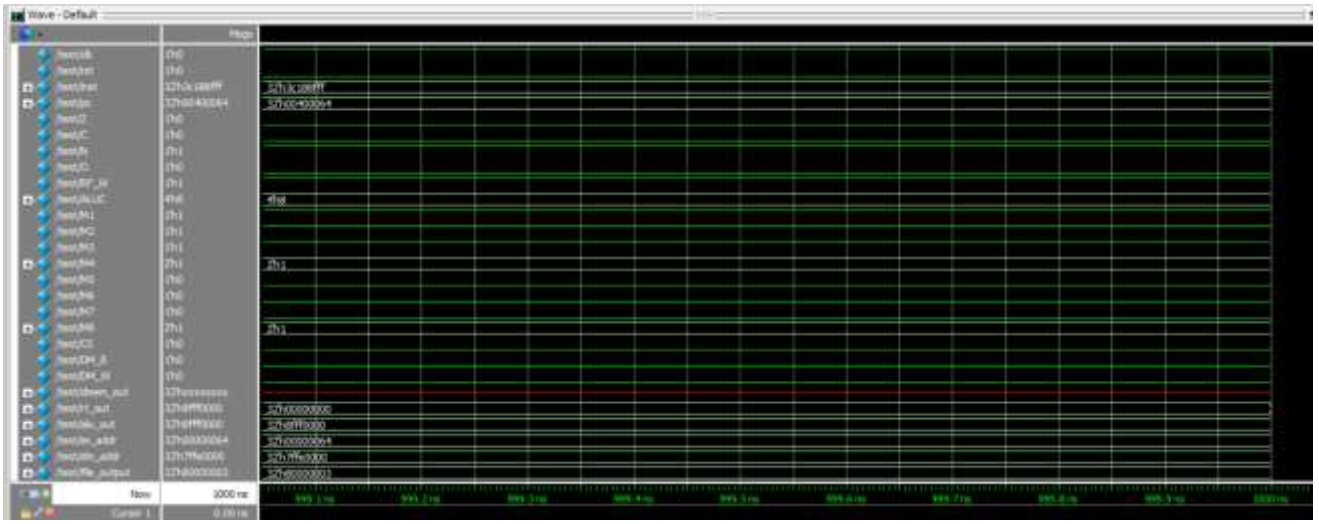
（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

（1）.前仿真
顶层模块对外接口波形图：

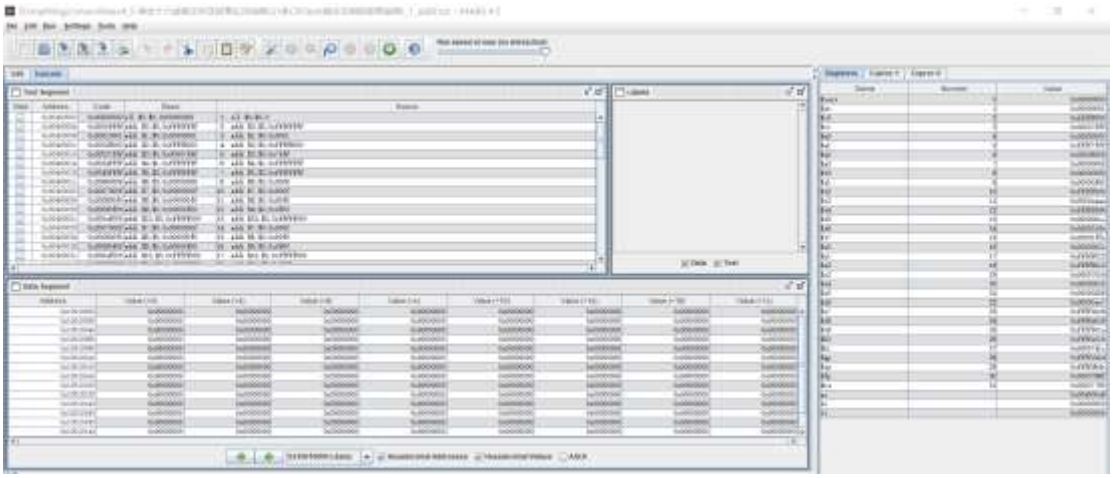




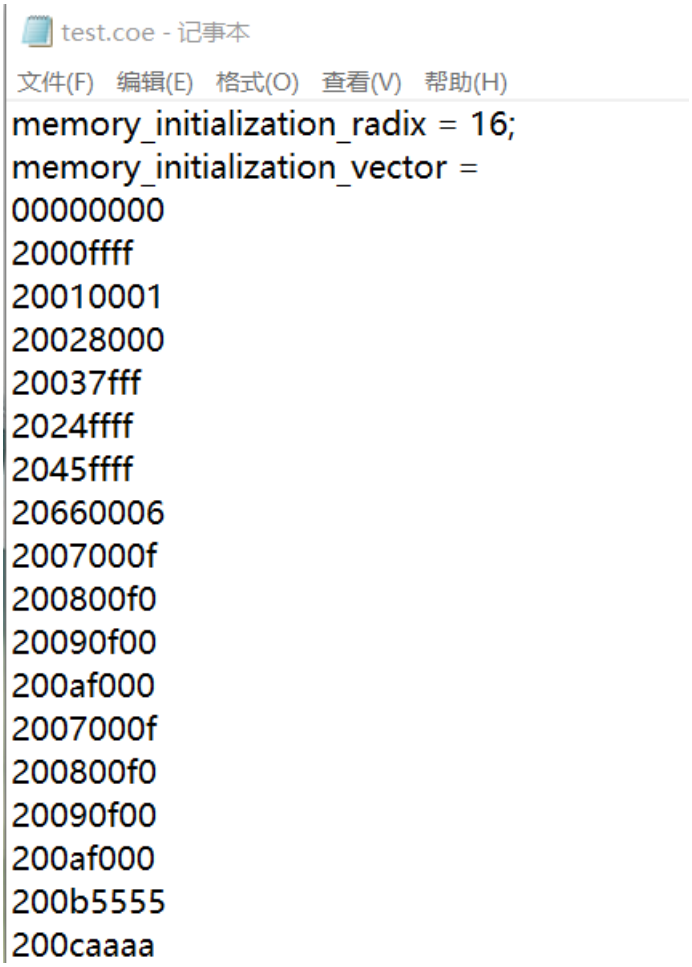
包含控制信号的波形图：



从 Mars 中获取 coe 文件和 result 文件：



Mars 导出的 coe 文件：



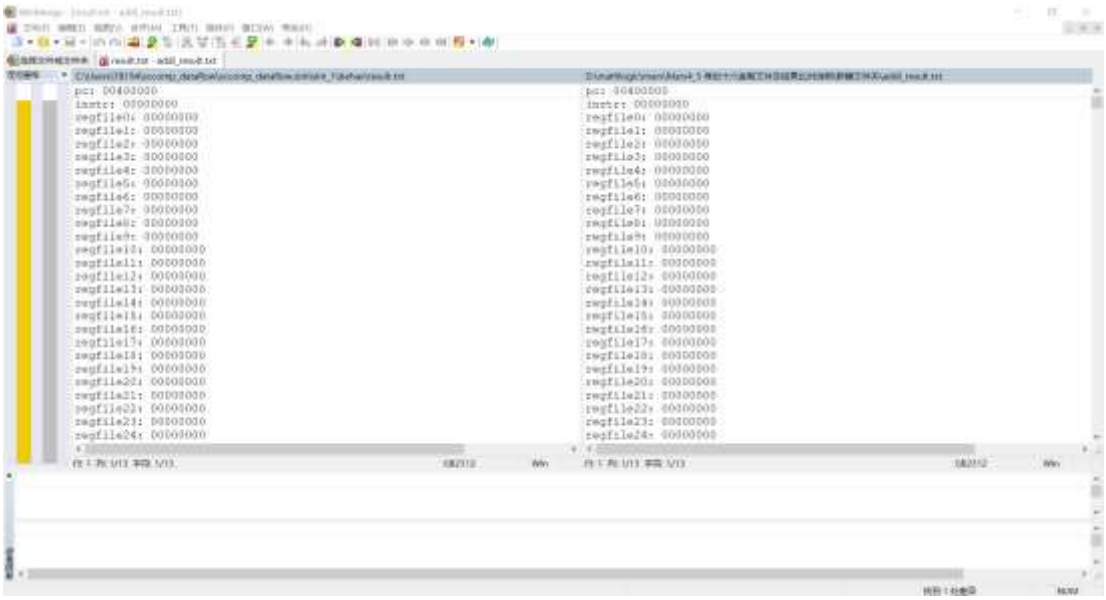
过程中打印的 result.txt 文件：

result.txt - 记事本

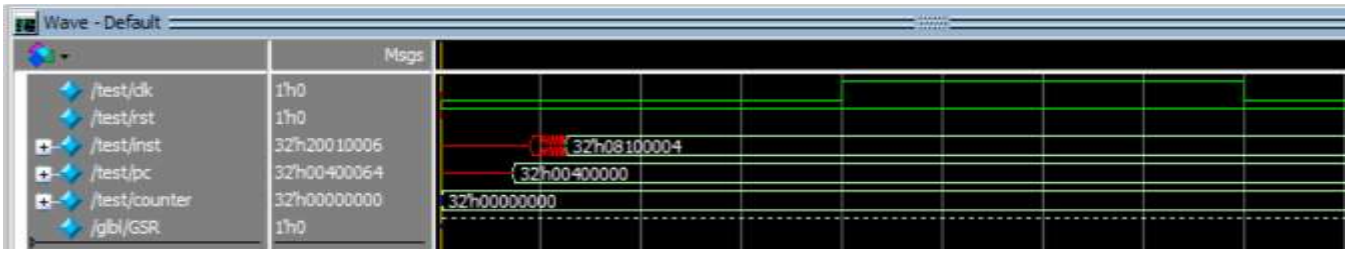
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

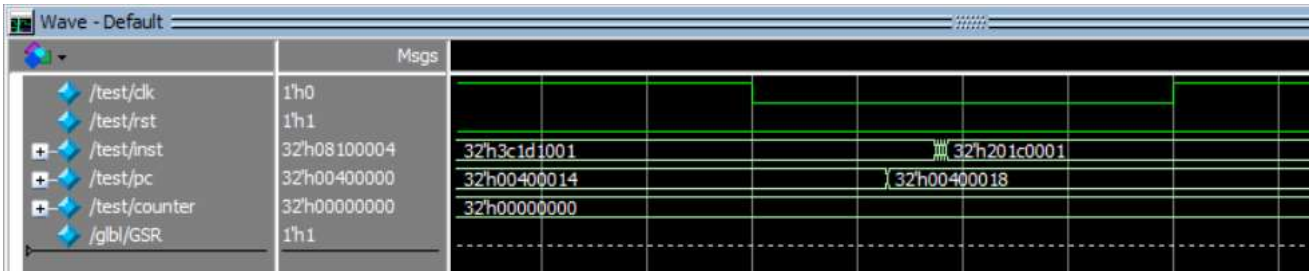
```
pc: 00400000
instr: 3c1d1001
regfile0: 00000000
regfile1: 00000000
regfile2: 00000000
regfile3: 00000000
regfile4: 00000000
regfile5: 00000000
regfile6: 00000000
regfile7: 00000000
regfile8: 00000000
regfile9: 00000000
regfile10: 00000000
regfile11: 00000000
regfile12: 00000000
regfile13: 00000000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

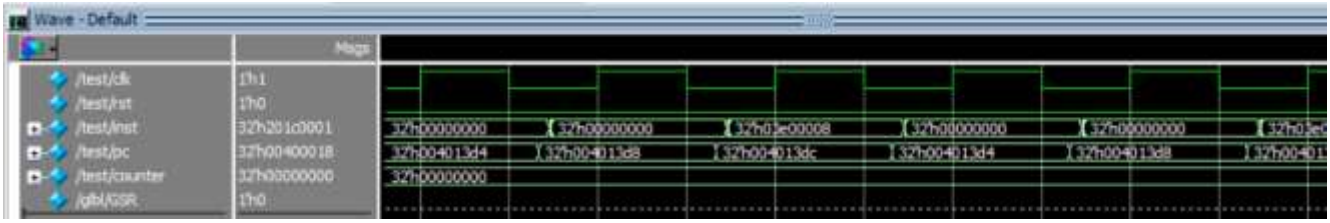
生成的 result 文件与 Mars 的执行结果在文本比较器中比较：



（2）.后仿真


Wave - Default
Msgs
/test/clk
/test/rst
/test/inst
/test/pc
/test/counter
/glbl/GSR
1'h0
1'h0
32'h20010006
32'h00400064
32'h00000000
1'h0
32'h08100004
32'h00400000
32'h00000000


Wave - Default
Msgs
/test/clk
/test/rst
/test/inst
/test/pc
/test/counter
/glbl/GSR
1'h0
1'h1
32'h08100004
32'h00400000
32'h00000000
1'h1
32'h3c1d1001
32'h201c0001
32'h00400014
32'h00400018
32'h00000000


Wave - Default
Msgs
/test/clk
/test/rst
/test/inst
/test/pc
/test/counter
/glbl/GSR
1'h1
1'h0
32'h201c0001
32'h00400018
32'h00000000
1'h0

（3）.下板


NEXYS 4
DDR
XILINX
DIGILENT
ANALOG DEVICES


NEXYS 4
DDR
DIGILENT
XILINX
ANALOG DEVICES
ETHERNET
USB HOST
VGA
MONO AUDIO OUT
POWER
CPU RESET
PROG
DONE