一、实验内容

在本实验中，将使用 verilog HDL 实现 31 条 MIPS 指令的 CPU 设计，前仿真，后仿真和下板调试运行。

二、实验目标

（1） 深入掌握 CPU 的构成及工作原理。
（2） 设计 31 条指令的 CPU 的数据通路即控制器。
（3） 使用 verilog HDL 设计实现 31 条指令的 CPU 下板运行。

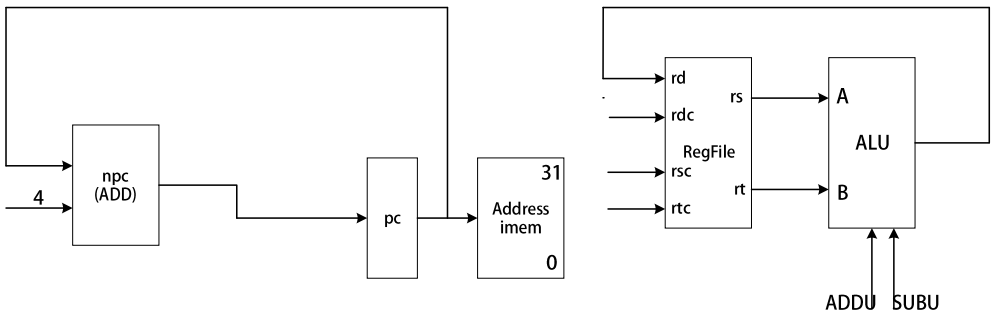
三、实验原理

（1） 根据指令功能，确定每条指令执行中用到的部件，如下表：

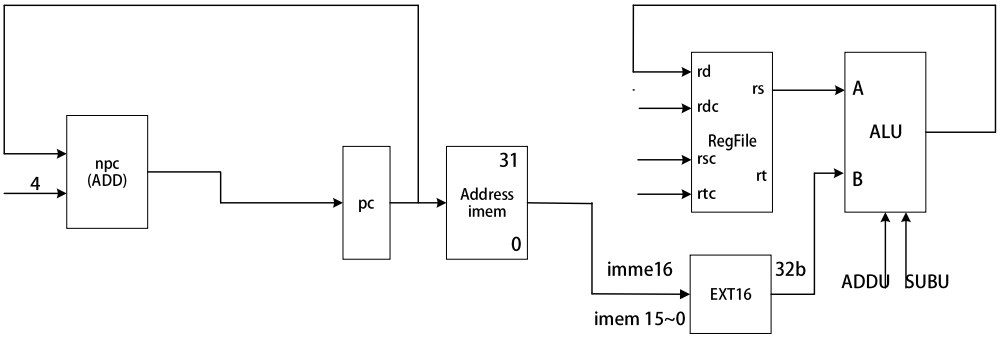
instr	pc	npc	imem	RegFile	ALU		Ext5	Ext16	DMEM		S_Ext16	Ext18	ADD		||		GPR[31]
				rd	A	B			Addr	Data			A	B	A	B	
ADD	npc	pc	pc	ALU	rs	rt											
ADDU	npc	pc	pc	ALU	rs	rt											
ADDI	npc	pc	pc	ALU	rs	S_Ext16					imm16						
ADDIU	npc	pc	pc	ALU	rs	Ext16		imm16									
SUB	npc	pc	pc	ALU	rs	rt											
SUBU	npc	pc	pc	ALU	rs	rt											
SLL	npc	pc	pc	ALU	Ext5	rt	sa										
SRL	npc	pc	pc	ALU	Ext5	rt	sa										
SRA	npc	pc	pc	ALU	Ext5	rt	sa										
SLLV	npc	pc	pc	ALU	rs	rt											
SRLV	npc	pc	pc	ALU	rs	rt											
SRAV	npc	pc	pc	ALU	rs	rt											
ORI	npc	pc	pc	ALU	rs	Ext16		imm16									
XORI	npc	pc	pc	ALU	rs	Ext16		imm16									
NORI	npc	pc	pc	ALU	rs	Ext16		imm16									
ANDI	npc	pc	pc	ALU	rs	Ext16		imm16									
OR	npc	pc	pc	ALU	rs	rt											
XOR	npc	pc	pc	ALU	rs	rt											
AND	npc	pc	pc	ALU	rs	rt											
LW	npc	pc	pc	Data	rs	S_Ext16			ALU		offset						
SW	npc	pc	pc		rs	S_Ext16			ALU	rt	offset						
BEQ	npc	pc	pc		rs	rt						offset	npc	Ext18			
BNE	npc	pc	pc		rs	rt						offset	npc	Ext18			
J	||	pc	pc												pc31~28	imem25~0	
JAL	||	pc	pc											pc31~28	imem25~0	pc+8
JR	rs	pc	pc														
SLT	npc	pc	pc	ALU	rs	rt											
SLTU	npc	pc	pc	ALU	rs	rt											
SLTI	npc	pc	pc	ALU	rs	Ext16		imm16									
SLTIU	npc	pc	pc	ALU	rs	S_Ext16					imm16						
LUI	npc	pc	pc	ALU	rs	Ext16		imm16									

（2） 根据每条指令涉及的部件和部件的数据输入来源，画出每条指令的数据通路，下面是数据通路图：

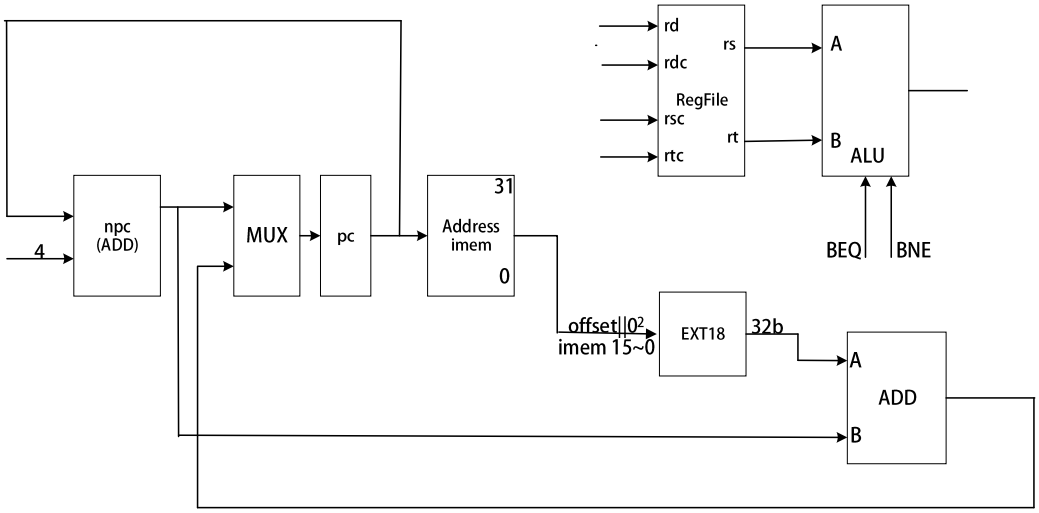
ADDU,ADD,SUBU,SUB数据通路



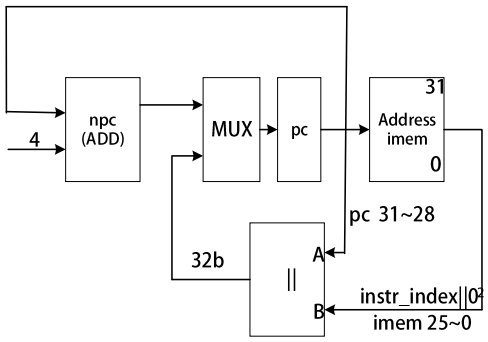
ADDIU,ADDI数据通路



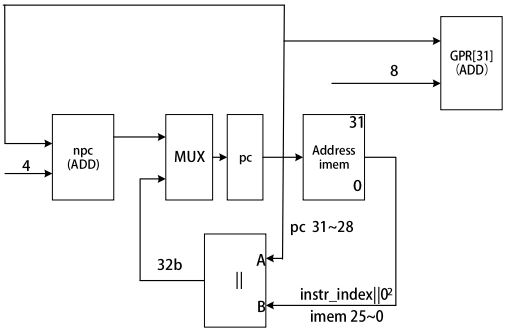
BEQ,BNE数据通路



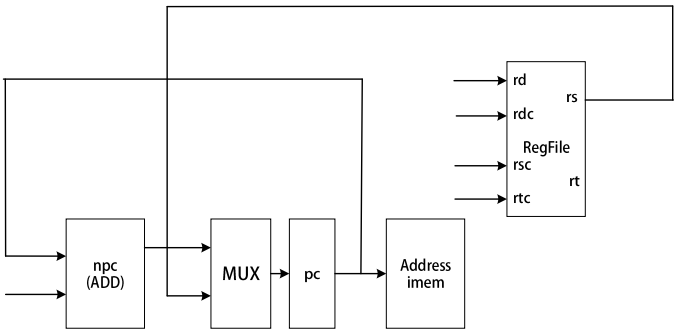
J数据通路



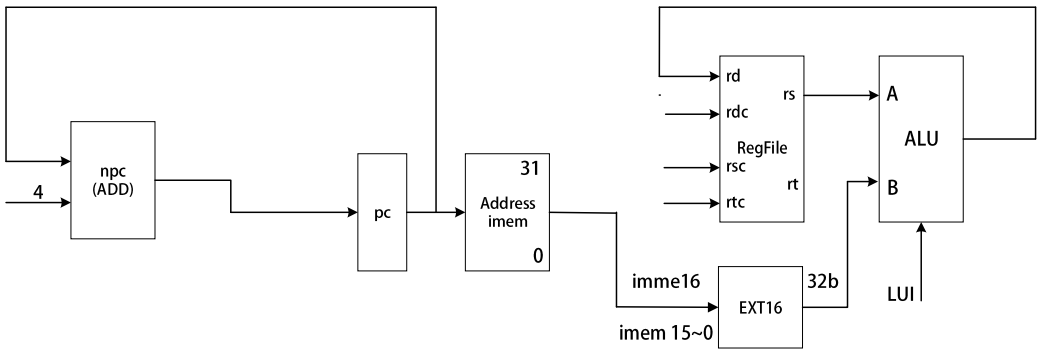
JAL数据通路



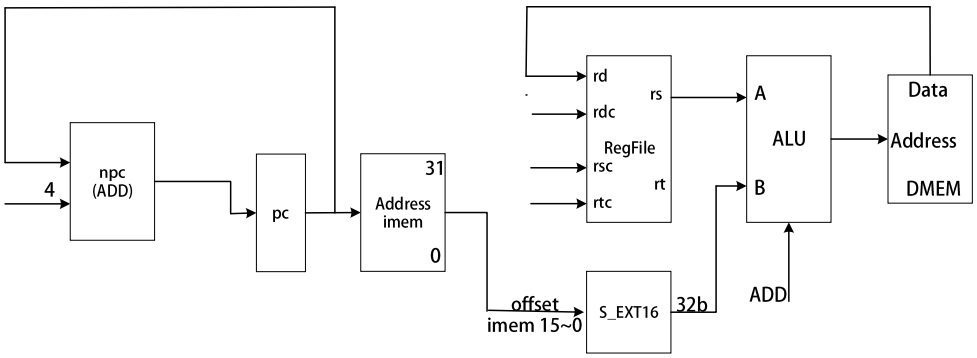
JR数据通路



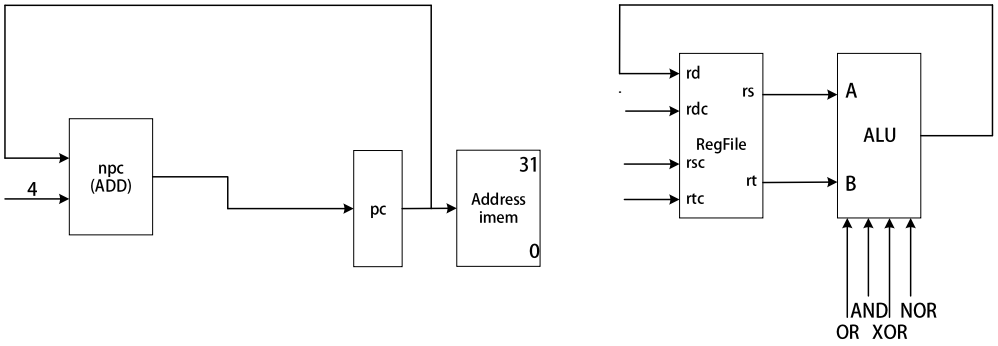
LUI数据通路



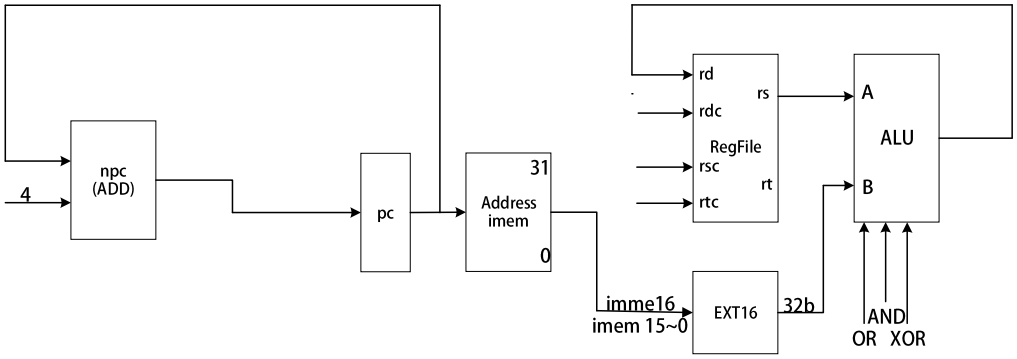
LW数据通路



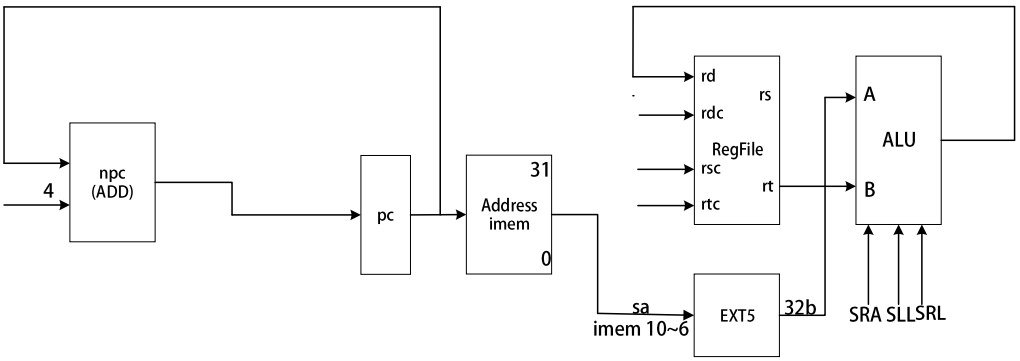
AND,OR,XOR,NOR数据通路



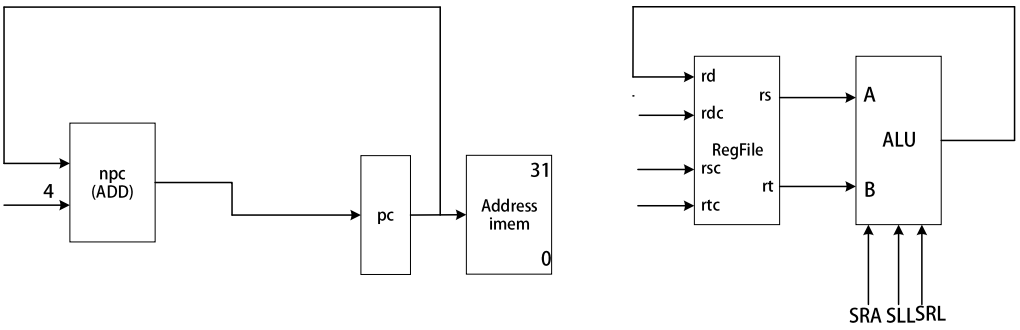
ANDI,ORI,XORI数据通路



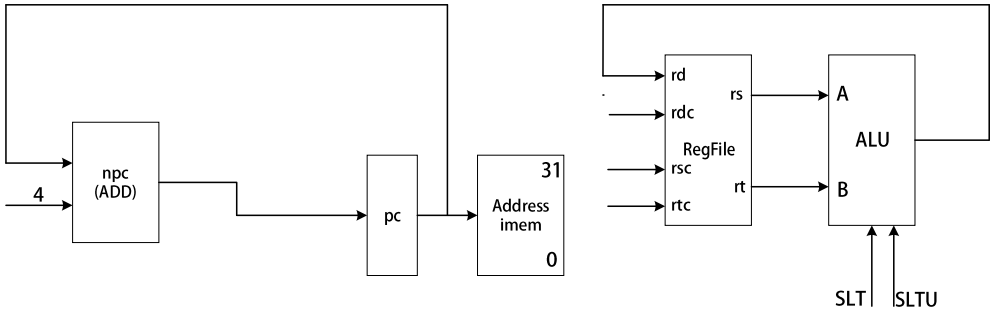
SLL,SRL,SRA数据通路



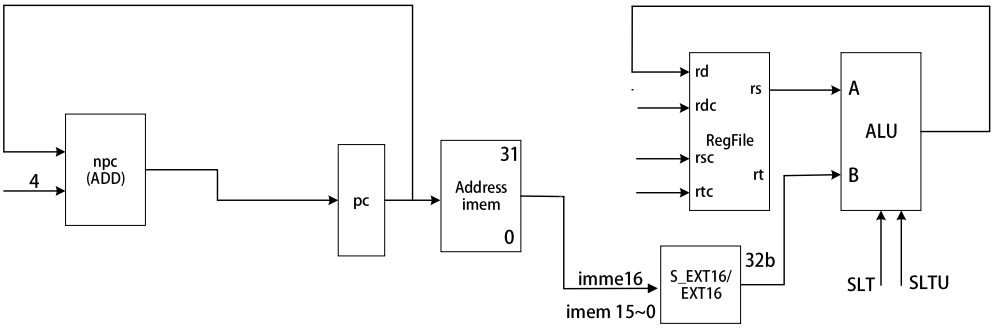
SLLV,SRLV,SRAV数据通路



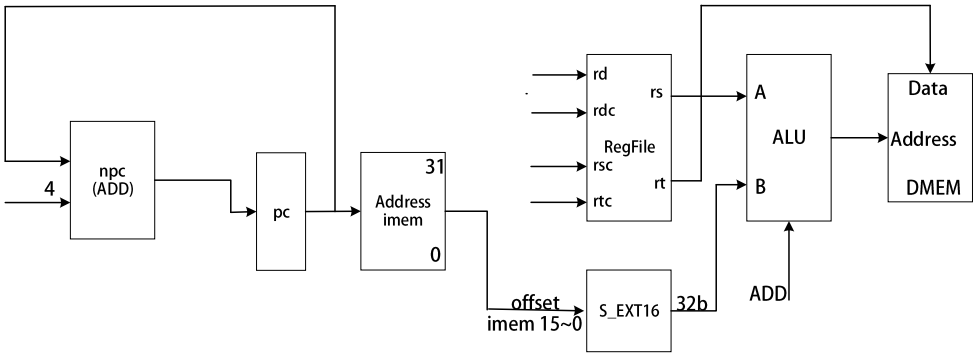
SLT,SLTU数据通路

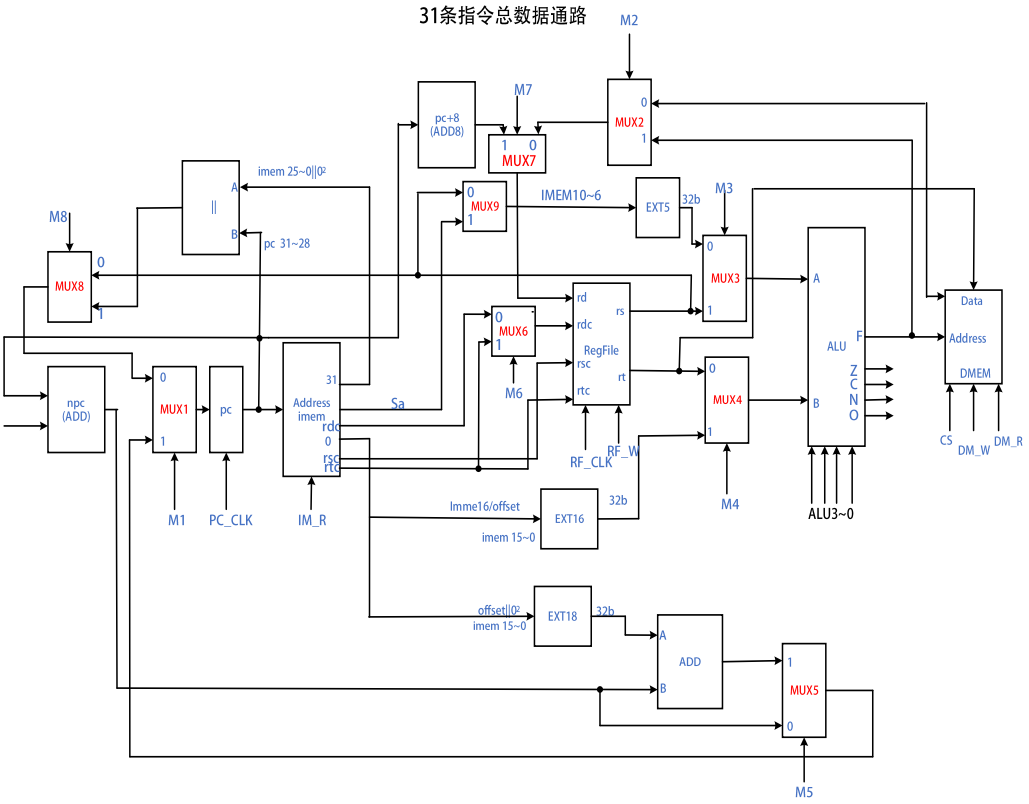


SLTI,SLTIU数据通路



SW数据通路





（3） 指令操作时间表

控制信号(微操作)	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR	SLT	SLTU	SLL	SRL	SRA	SLLV	SRLV	SRAV	JR
PC_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Rsc4-0	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
Rtc4-0	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	IM20-16	
Rdc4-0	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	
M1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
M2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
M3	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	
M4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
M5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
M7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
M8																	0
M9											1	1	1	0	0	0	
ALUC3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
ALUC2	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	
ALUC1	1	0	1	0	0	0	1	1	1	1	1	0	0	1	0	0	
ALUC0	0	0	1	1	0	1	0	1	1	0	x	1	0	x	1	0	
RF_W	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
RF_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
DM_CS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DM_R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DM_W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

控制信号(微操作)	ADDI	ADDIU	ANDI	ORI	XORI	LW	SW	BEQ	BNE	SLTI	SLTIU	LUI	J	JAL
PC_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21			
Rtc4-0							IM20-16	IM20-16	IM20-16					
Rdc4-0	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16				IM20-16	IM20-16	IM20-16		
M1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
M2	1	1	1	1	1	0	1	1	1	1	1	1		
M3	1	1	1	1	1	1	1	1	1	1	1			
M4	1	1	1	1	1	1	1	0	0	1	1	1		
M5	0	0	0	0	0	0	0	1	1	0	0	0	0	0
M6	0	0	0	0	0	1		0	0	0	0			0
M7	0	0	0	0	0	0		0	0	0	0	0		1
M8													1	1
M9														
ALUC3	0	0	0	0	0	0	0	0	0	1	1	1		
ALUC2	0	0	1	1	1	0	0	0	0	0	0	0		
ALUC1	1	0	0	0	1	1	1	1	1	1	1	0		
ALUC0	0	0	0	1	0	0	0	1	1	1	0	x		
RF_W	1	1	1	1	1	1	0	0	0	1	1	1	0	1
RF_CLK	1	1	1	1	1	1	0	0	0	1	1	1	0	1
DM_CS	0	0	0	0	0	1	1	0	0	0	0	0	0	0
DM_R	0	0	0	0	0	1	0	0	0	0	0	0	0	0
DM_W	0	0	0	0	0	0	1	0	0	0	0	0	0	0

（4） 控制信号的逻辑表达式

```
PC_CLK=~clk;
IM_R=1;
M1=~(dec[29]|dec[30]|dec[16]);
M2=dec[22];
M3=~(dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15]);
M4=dec[17]|dec[18]|dec[19]|dec[20]|dec[21]|dec[22]|dec[23]|dec[26]|dec[27]|dec[28];
M5=(dec[24]&z)|(dec[25]&~z);
M6=dec[17]|dec[18]|dec[19]|dec[20]|dec[21]|dec[22]|dec[23]|dec[26]|dec[27]|dec[28];
M7=dec[30];
M8=dec[16];//~(dec[29]|dec[30])
M9=dec[13]|dec[14]|dec[15];
ALUC[3]=dec[8]|dec[9]|dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15]|dec[26]|dec[27]|dec[28];
ALUC[2]=dec[4]|dec[5]|dec[6]|dec[7]|dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15]|dec[19]|de
c[20]|dec[21];
ALUC[1]=dec[0]|dec[2]|dec[6]|dec[7]|dec[8]|dec[9]|dec[10]|dec[13]|dec[17]|dec[21]|dec[22]|dec[2
3]|dec[24]|dec[25]|dec[26]|dec[27];
ALUC[0]=dec[2]|dec[3]|dec[5]|dec[7]|dec[8]|dec[11]|dec[14]|dec[20]|dec[24]|dec[25]|dec[26];
RF_W=~(dec[16]|dec[23]|dec[24]|dec[25]|dec[29]);
RF_CLK=~clk;
DM_R=dec[22];
DM_W=dec[23];
DM_CS=dec[22]|dec[23];
C_ext16=~(dec[19]|dec[20]|dec[21]);
```

四、模块建模

(1) 顶层模块

```
module    sccomp_dataflow(
    input clk_in,
    input reset,
    output [31:0]inst,//instruction from imem
    output [31:0]pc,//program counter
    output [31:0]addr
    );
    wire IM_R;//imem read ->1
    wire DM_W;//dmem write ena -wena
    wire DM_R;//dmem read ena    -rena
    wire DM_CS;//dram- dmem ena -ena
    wire [31:0]rdata;//data read from dram
    //wire [31:0]addr;// dmem write/read address
    wire [31:0]wdata;//data write in dram

    cpu sccpu(
```

```
    .clk(clk_in),
    .rst(reset),
    .IM_R(IM_R),
    .DM_W(DM_W),
    .DM_R(DM_R),
    .DM_CS(DM_CS),
    .rdata(rdata),
    .addr(addr),
    .wdata(wdata),
    .instr(inst),
    .pc(pc)
     );

    dist_mem_gen_0 imem(//get instruction from instr_mem    ->    IMEM
    .a(pc[12:2]),
    .spo(inst)
    );

    dram dmem(
    .clk(clk_in),
    .ena(DM_CS),
    .rena(DM_R),
    .wena(DM_W),
    .addr(addr),
    .datain(wdata),
    .dataout(rdata)
    );
Endmodule
```

(2)　dram 模块

```
module dram(
        input clk,
        input ena,// DM_CS enable
        input wena,//DM_W write enable
        inout rena,//DM_R read enable
        input [31:0] addr,
        input [31:0] datain,
        output[31:0] dataout
      );
        reg [31:0]mem[2047:0];
        assign dataout=(ena&&rena)? mem[addr[10:0]>>2]:32'bz;
        always@(posedge clk)
        begin
        if(ena)begin
            if(wena)begin
```

```
                mem[addr[10:0]>>2]=datain;
            end
        end
    end
endmodule
```

(3) cpu 模块

```
module cpu(
    input clk,
    input rst,
    input [31:0]instr,
    input [31:0]rdata,//data read from dram
    output IM_R,
    output DM_W,
    output DM_R,
    output DM_CS,
    output [31:0]pc,
    output [31:0]addr,//dram write address
    output [31:0]wdata //data write in dram
);

    wire [31:0]D_M1;
    wire [31:0]D_M2;
    wire [31:0]D_M3;
    wire [31:0]D_M4;
    wire [31:0]D_M5;//rd
    wire [4:0]D_M6;//rdc <=rdc(instr[15:11])or rtc(instr[20:16])
    wire [31:0]D_M7;
    wire [31:0]D_M8;
    wire [4:0]D_M9;//instr[10:6] or rs[4:0]

    wire [31:0] D_ext16;//output of ext16
    wire [31:0] D_ext18;//output of ext18
    wire [31:0] D_ext5;//output of ext5
    wire [31:0] D_pc;//output of pc
    wire [31:0] D_npc;//output of npc
    wire [31:0] D_add8;//output of add8
    wire [31:0] D_add;//output of add
    wire [31:0] D_rs;
    wire [31:0] D_rt;
    wire [31:0] D_alu;//output of alu
    wire [31:0] D_join;
    wire add_ov;

    //control instruction
```

```
wire M1;
wire M2;
wire M3;
wire M4;
wire M5;
wire M6;
wire M7;
wire M8;
wire M9;
wire PC_CLK;
wire PC_ENA;
wire RF_CLK;
wire RF_W;
wire C_ext16;
wire [3:0]ALUC;
wire Z;//zero
wire C;//carry
wire N;//negative
wire O;//overflow


assign PC_ENA=1;
assign pc=D_pc;
assign addr=D_alu;    //dram read/write address
assign wdata=D_rt;

wire [31:0] de_instr;//decoded instruction

 instr_decode cpu_dec(//instruction decode module
        .instr(instr),
        .dec(de_instr)
 );
 ctrl cpu_ctrl(// control module
        .clk(clk),.z(Z),.dec(de_instr),.M1(M1),.M2(M2),.M3(M3),
        .M4(M4),.M5(M5),.M6(M6),.M7(M7),.M8(M8),.M9(M9),.PC_CLK(PC_CLK),
        .IM_R(IM_R),.ALUC(ALUC),.RF_CLK(RF_CLK),.RF_W(RF_W),.DM_W(DM_W),
        .DM_R(DM_R),.DM_CS(DM_CS),.C_ext16(C_ext16)
 );
 pcreg    cpu_pc(
        .clk(PC_CLK),.rst(rst),
        .ena(PC_ENA),.data_in(D_M1),
        .data_out(D_pc)
 );
 regfile cpu_ref(
```

```
        .clk(RF_CLK),.rst(rst),.we(RF_W),
        .raddr1(instr[25:21]),.raddr2(instr[20:16]),
        .waddr(D_M6),.wdata(D_M7),.rdata1(D_rs),.rdata2(D_rt)
);
alu         cpu_alu(
        .a(D_M3),.b(D_M4),.aluc(ALUC),
        .r(D_alu),.zero(Z),.carry(C),.negative(N),.overflow(O)
);
ext5        cpu_ext5(
        .a(D_M9),.b(D_ext5)
);
ext16       cpu_ext16(
        .a(instr[15:0]),.b(D_ext16),.s_ext(C_ext16)
);
ext18       cpu_ext18(
        .a(instr[15:0]),.b(D_ext18)
);
mux32       cpu_mux1(
        .a(D_M8),.b(D_M5),.opt(M1),.c(D_M1)//
);
mux32       cpu_mux2(
        .a(D_alu),.b(rdata),.opt(M2),.c(D_M2)///
);
mux32       cpu_mux3(
        .a(D_ext5),.b(D_rs),.opt(M3),.c(D_M3)//
);
mux32       cpu_mux4(
        .a(D_rt),.b(D_ext16),.opt(M4),.c(D_M4)//
);
mux32       cpu_mux5(
        .a(D_npc),.b(D_add),.opt(M5),.c(D_M5)//
);
mux5_1      cpu_mux6(//de_inttr[30]=M7 : jal
        .a(instr[15:11]),.b(instr[20:16]),.opt({M7,M6}),.c(D_M6)//
);//rd      ,rt
mux32       cpu_mux7(
        .a(D_M2),.b(D_add8),.opt(M7),.c(D_M7)//
);
mux32       cpu_mux8(
        .a(D_join),.b(D_rs),.opt(M8),.c(D_M8)///
);
mux5_2      cpu_mux9(
        .a(instr[10:6]),.b(D_rs[4:0]),.opt(M9),.c(D_M9)///
);
```

```
    add         cpu_add(
            .a(D_ext18),.b(D_npc),.c(D_add),.ov(add_ov)
    );
    add8        cpu_add8(
            .a(D_pc),.c(D_add8)
    );
    join_instr    cpu_join(
            .part1(D_pc[31:28]),.part2(instr[25:0]),.r(D_join)
    );

    npc      cpu_npc(
            .a(D_pc),.rst(rst),.c(D_npc)
    );
endmodule
```

(4) control 控制器模块

```
module ctrl(
    input clk,
    input [31:0]dec,
    input z,//zero
    output M1,
    output M2,
    output M3,
    output M4,
    output M5,
    output M6,
    output M7,
    output M8,
    output M9,
    output PC_CLK,
    output IM_R,//imem(ram1) read
    output [3:0]ALUC,
    output RF_CLK,
    output RF_W,//regfile write
    output DM_W,//dmem(ram2)
    output DM_R,//dmem read
    output DM_CS,//dmem ena
    output C_ext16
);
    assign PC_CLK=~clk;
    assign IM_R=1;
    assign M1=~(dec[29]|dec[30]|dec[16]);
    assign M2=dec[22];
    assign M3=~(dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15]);
    assign M4=dec[17]|dec[18]|dec[19]|dec[20]|dec[21]|dec[22]|dec[23]|dec[26]|dec[27]|dec[28];
```

```
    assign M5=(dec[24]&z)|(dec[25]&~z);
    assign M6=dec[17]|dec[18]|dec[19]|dec[20]|dec[21]|dec[22]|dec[23]|dec[26]|dec[27]|dec[28];
    assign M7=dec[30];
    assign M8=dec[16];//~(dec[29]|dec[30])
    assign M9=dec[13]|dec[14]|dec[15];
    assign
ALUC[3]=dec[8]|dec[9]|dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15]|dec[26]|dec[27]|dec[28];
    assign
ALUC[2]=dec[4]|dec[5]|dec[6]|dec[7]|dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15]|dec[19]|de
c[20]|dec[21];
    assign
ALUC[1]=dec[0]|dec[2]|dec[6]|dec[7]|dec[8]|dec[9]|dec[10]|dec[13]|dec[17]|dec[21]|dec[22]|dec[2
3]|dec[24]|dec[25]|dec[26]|dec[27];
    assign
ALUC[0]=dec[2]|dec[3]|dec[5]|dec[7]|dec[8]|dec[11]|dec[14]|dec[20]|dec[24]|dec[25]|dec[26];
    assign RF_W=~(dec[16]|dec[23]|dec[24]|dec[25]|dec[29]);
    assign RF_CLK=~clk;////
    assign DM_R=dec[22];
    assign DM_W=dec[23];
    assign DM_CS=dec[22]|dec[23];
    assign C_ext16=~(dec[19]|dec[20]|dec[21]);
endmodule
```

(5) 译码器模块

```
`define ADD    12'b000000100000
`define ADDU 12'b000000100001
`define SUB    12'b000000100010
`define SUBU 12'b000000100011
`define AND    12'b000000100100
`define OR     12'b000000100101
`define XOR    12'b000000100110
`define NOR    12'b000000100111
`define SLT    12'b000000101010
`define SLTU 12'b000000101011
`define SLL    12'b000000000000
`define SRL    12'b000000000010
`define SRA    12'b000000000011
`define SLLV 12'b000000000100
`define SRLV 12'b000000000110
`define SRAV 12'b000000000111
`define JR     12'b000000001000

`define ADDI    12'b001000??????
`define ADDIU 12'b001001??????
`define ANDI 12'b001100??????
```

```
`define ORI    12'b001101??????
`define XORI 12'b001110??????
`define LW      12'b100011??????
`define SW      12'b101011??????
`define BEQ    12'b000100??????
`define BNE    12'b000101??????
`define SLTI 12'b001010??????
`define SLTIU 12'b001011??????
`define LUI    12'b001111??????

`define J        12'b000010??????
`define JAL    12'b000011??????

module instr_decode(
        input [31:0]instr,
        output reg [31:0] dec
);
always @(*)begin
        casez ({instr[31:26],instr[5:0]})
                `ADD:dec<=32'h00000001;
                `ADDU:dec<=32'h00000002;
                `SUB:dec<=32'h00000004;
                `SUBU:dec<=32'h00000008;

                `AND:dec<=32'h00000010;
                `OR:dec<=32'h00000020;
                `XOR:dec<=32'h00000040;
                `NOR:dec<=32'h00000080;

                `SLT:dec<=32'h00000100;
                `SLTU:dec<=32'h00000200;
                `SLL:dec<=32'h00000400;
                `SRL:dec<=32'h00000800;

                `SRA:dec<=32'h00001000;
                `SLLV:dec<=32'h00002000;
                `SRLV:dec<=32'h00004000;
                `SRAV:dec<=32'h00008000;

                `JR:dec<=32'h00010000;

                `ADDI:dec<=32'h00020000;
                `ADDIU:dec<=32'h00040000;
                `ANDI:dec<=32'h00080000;
```

```
            `ORI:dec<=32'h00100000;
            `XORI:dec<=32'h00200000;
            `LW:dec<=32'h00400000;
            `SW:dec<=32'h00800000;

            `BEQ:dec<=32'h01000000;
            `BNE:dec<=32'h02000000;
            `SLTI:dec<=32'h04000000;
            `SLTIU:dec<=32'h08000000;
            `LUI:dec<=32'h10000000;

            `J:dec<=32'h20000000;
            `JAL:dec<=32'h40000000;
            default:dec<=32'bx;
          endcase
end
endmodule
```

(6) 其他模块

```
module pcreg(
    input clk,rst,ena,
    input [31:0] data_in,
    output [31:0] data_out
);

    D_FF d31(data_in[31],ena,clk,rst,data_out[31]);
    D_FF d30(data_in[30],ena,clk,rst,data_out[30]);
    D_FF d29(data_in[29],ena,clk,rst,data_out[29]);
    D_FF d28(data_in[28],ena,clk,rst,data_out[28]);
    D_FF d27(data_in[27],ena,clk,rst,data_out[27]);
    D_FF d26(data_in[26],ena,clk,rst,data_out[26]);
    D_FF d25(data_in[25],ena,clk,rst,data_out[25]);
    D_FF d24(data_in[24],ena,clk,rst,data_out[24]);
    D_FF d23(data_in[23],ena,clk,rst,data_out[23]);
    D_FF0 d22(data_in[22],ena,clk,rst,data_out[22]);//DFF0      00400000
    D_FF d21(data_in[21],ena,clk,rst,data_out[21]);
    D_FF d20(data_in[20],ena,clk,rst,data_out[20]);
    D_FF d19(data_in[19],ena,clk,rst,data_out[19]);
    D_FF d18(data_in[18],ena,clk,rst,data_out[18]);
    D_FF d17(data_in[17],ena,clk,rst,data_out[17]);
    D_FF d16(data_in[16],ena,clk,rst,data_out[16]);
    D_FF d15(data_in[15],ena,clk,rst,data_out[15]);
    D_FF d14(data_in[14],ena,clk,rst,data_out[14]);
    D_FF d13(data_in[13],ena,clk,rst,data_out[13]);
```

```
        D_FF d12(data_in[12],ena,clk,rst,data_out[12]);
        D_FF d11(data_in[11],ena,clk,rst,data_out[11]);
        D_FF d10(data_in[10],ena,clk,rst,data_out[10]);
        D_FF d9(data_in[9],ena,clk,rst,data_out[9]);
        D_FF d8(data_in[8],ena,clk,rst,data_out[8]);
        D_FF d7(data_in[7],ena,clk,rst,data_out[7]);
        D_FF d6(data_in[6],ena,clk,rst,data_out[6]);
        D_FF d5(data_in[5],ena,clk,rst,data_out[5]);
        D_FF d4(data_in[4],ena,clk,rst,data_out[4]);
        D_FF d3(data_in[3],ena,clk,rst,data_out[3]);
        D_FF d2(data_in[2],ena,clk,rst,data_out[2]);
        D_FF d1(data_in[1],ena,clk,rst,data_out[1]);
        D_FF d0(data_in[0],ena,clk,rst,data_out[0]);
endmodule

module pcreg0(
        input clk,rst,ena,
        input [31:0] data_in,
        output [31:0] data_out
);
        D_FF d31(data_in[31],ena,clk,rst,data_out[31]);
        D_FF d30(data_in[30],ena,clk,rst,data_out[30]);
        D_FF d29(data_in[29],ena,clk,rst,data_out[29]);
        D_FF d28(data_in[28],ena,clk,rst,data_out[28]);
        D_FF d27(data_in[27],ena,clk,rst,data_out[27]);
        D_FF d26(data_in[26],ena,clk,rst,data_out[26]);
        D_FF d25(data_in[25],ena,clk,rst,data_out[25]);
        D_FF d24(data_in[24],ena,clk,rst,data_out[24]);
        D_FF d23(data_in[23],ena,clk,rst,data_out[23]);
        D_FF d22(data_in[22],ena,clk,rst,data_out[22]);
        D_FF d21(data_in[21],ena,clk,rst,data_out[21]);
        D_FF d20(data_in[20],ena,clk,rst,data_out[20]);
        D_FF d19(data_in[19],ena,clk,rst,data_out[19]);
        D_FF d18(data_in[18],ena,clk,rst,data_out[18]);
        D_FF d17(data_in[17],ena,clk,rst,data_out[17]);
        D_FF d16(data_in[16],ena,clk,rst,data_out[16]);
        D_FF d15(data_in[15],ena,clk,rst,data_out[15]);
        D_FF d14(data_in[14],ena,clk,rst,data_out[14]);
        D_FF d13(data_in[13],ena,clk,rst,data_out[13]);
        D_FF d12(data_in[12],ena,clk,rst,data_out[12]);
        D_FF d11(data_in[11],ena,clk,rst,data_out[11]);
        D_FF d10(data_in[10],ena,clk,rst,data_out[10]);
        D_FF d9(data_in[9],ena,clk,rst,data_out[9]);
        D_FF d8(data_in[8],ena,clk,rst,data_out[8]);
```

```
        D_FF d7(data_in[7],ena,clk,rst,data_out[7]);
        D_FF d6(data_in[6],ena,clk,rst,data_out[6]);
        D_FF d5(data_in[5],ena,clk,rst,data_out[5]);
        D_FF d4(data_in[4],ena,clk,rst,data_out[4]);
        D_FF d3(data_in[3],ena,clk,rst,data_out[3]);
        D_FF d2(data_in[2],ena,clk,rst,data_out[2]);
        D_FF d1(data_in[1],ena,clk,rst,data_out[1]);
        D_FF d0(data_in[0],ena,clk,rst,data_out[0]);
endmodule

module D_FF(
        input datain,
        input ena,
        input clk,
        input rst,
        output reg dataout
);
        always@(posedge clk or posedge rst)
        begin
           if(rst)
                dataout<=0;
           else
           if(ena)
                dataout<=datain;
        end
  endmodule

module D_FF0(
         input datain,
         input ena,
         input clk,
         input rst,
         output reg dataout
  );
         always@(posedge clk or posedge rst)
         begin
            if(rst)
                 dataout<=1;
            else
            if(ena)
                 dataout<=datain;
         end
    endmodule
```

```
module regfile(
        input clk,
        input rst,
        input we,//write ena
        input [4:0]raddr1,
        input [4:0]raddr2,
        input [4:0]waddr,
        input [31:0]wdata,
        output [31:0] rdata1,
        output [31:0] rdata2
  );
          wire [31:0]oData1;
          wire [31:0]array_reg[31:0];

            decoder dec(waddr,we,oData1);
            assign array_reg[0] = 0;
            pcreg0 reg1 (clk,rst,oData1[1],wdata,array_reg[1]);
            pcreg0 reg2 (clk,rst,oData1[2],wdata,array_reg[2]);
            pcreg0 reg3 (clk,rst,oData1[3],wdata,array_reg[3]);
            pcreg0 reg4 (clk,rst,oData1[4],wdata,array_reg[4]);
            pcreg0 reg5 (clk,rst,oData1[5],wdata,array_reg[5]);
            pcreg0 reg6 (clk,rst,oData1[6],wdata,array_reg[6]);
            pcreg0 reg7 (clk,rst,oData1[7],wdata,array_reg[7]);
            pcreg0 reg8 (clk,rst,oData1[8],wdata,array_reg[8]);
            pcreg0 reg9 (clk,rst,oData1[9],wdata,array_reg[9]);
            pcreg0 reg10 (clk,rst,oData1[10],wdata,array_reg[10]);
            pcreg0 reg11 (clk,rst,oData1[11],wdata,array_reg[11]);
            pcreg0 reg12 (clk,rst,oData1[12],wdata,array_reg[12]);
            pcreg0 reg13 (clk,rst,oData1[13],wdata,array_reg[13]);
            pcreg0 reg14 (clk,rst,oData1[14],wdata,array_reg[14]);
            pcreg0 reg15 (clk,rst,oData1[15],wdata,array_reg[15]);
            pcreg0 reg16 (clk,rst,oData1[16],wdata,array_reg[16]);
            pcreg0 reg17 (clk,rst,oData1[17],wdata,array_reg[17]);
            pcreg0 reg18 (clk,rst,oData1[18],wdata,array_reg[18]);
            pcreg0 reg19 (clk,rst,oData1[19],wdata,array_reg[19]);
            pcreg0 reg20 (clk,rst,oData1[20],wdata,array_reg[20]);
            pcreg0 reg21 (clk,rst,oData1[21],wdata,array_reg[21]);
            pcreg0 reg22 (clk,rst,oData1[22],wdata,array_reg[22]);
            pcreg0 reg23 (clk,rst,oData1[23],wdata,array_reg[23]);
            pcreg0 reg24 (clk,rst,oData1[24],wdata,array_reg[24]);
            pcreg0 reg25 (clk,rst,oData1[25],wdata,array_reg[25]);
            pcreg0 reg26 (clk,rst,oData1[26],wdata,array_reg[26]);
            pcreg0 reg27 (clk,rst,oData1[27],wdata,array_reg[27]);
            pcreg0 reg28 (clk,rst,oData1[28],wdata,array_reg[28]);
```

```
        pcreg0 reg29 (clk,rst,oData1[29],wdata,array_reg[29]);
        pcreg0 reg30 (clk,rst,oData1[30],wdata,array_reg[30]);
        pcreg0 reg31 (clk,rst,oData1[31],wdata,array_reg[31]);
        assign rdata1 = array_reg[raddr1];
        assign rdata2 = array_reg[raddr2];
endmodule

module decoder(
        input [4:0] iData,
        input    iEna,
        output[31:0]oData
);
        assign oData=(iEna==1)?(32'd1<<iData):32'bx;
endmodule

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

      parameter Addu     =      4'b0000;      //r=a+b unsigned
      parameter Add      =      4'b0010;      //r=a+b signed
      parameter Subu     =      4'b0001;      //r=a-b unsigned
      parameter Sub      =      4'b0011;      //r=a-b signed
      parameter And      =      4'b0100;      //r=a&b
      parameter Or       =      4'b0101;      //r=a|b
      parameter Xor      =      4'b0110;      //r=a^b
      parameter Nor      =      4'b0111;      //r=~(a|b)
      parameter Lui1     =      4'b1000;      //r={b[15:0],16'b0}
      parameter Lui2     =      4'b1001;      //r={b[15:0],16'b0}
      parameter Slt      =      4'b1011;      //r=(a-b<0)?1:0 signed
      parameter Sltu     =      4'b1010;      //r=(a-b<0)?1:0 unsigned
      parameter Sra      =      4'b1100;      //r=b>>>a
      parameter Sll      =      4'b1110;      //r=b<<a
      parameter Srl      =      4'b1101;      //r=b>>a
      parameter Slr      =      4'b1111;      //r=b<<a

      parameter bits=31;
```

```
parameter ENABLE=1,DISABLE=0;

reg signed [32:0] result;
reg [33:0] sresult;
wire signed [31:0] sa=a,sb=b;

always@(*)begin
	case(aluc)
		Addu: begin
			result=a+b;
			sresult={sa[31],sa}+{sb[31],sb};
		end
		Subu: begin
			result=a-b;
			sresult={sa[31],sa}-{sb[31],sb};
		end
		Add: begin
			result=sa+sb;
		end
		Sub: begin
			result=sa-sb;
		end
		Sra: begin
			if(a==0) {result[31:0],result[32]}={b,1'b0};
			else {result[31:0],result[32]}=sb>>>(a-1);
		end
		Srl: begin
			if(a==0) {result[31:0],result[32]}={b,1'b0};
			else {result[31:0],result[32]}=b>>(a-1);
		end
		Sll,Slr: begin
			result=b<<a;
		end
		And: begin
			result=a&b;
		end
		Or: begin
			result=a|b;
		end
		Xor: begin
			result=a^b;
		end
		Nor: begin
			result=~(a|b);
```

```
                end
                Sltu: begin
                    result=a<b?1:0;
                end
                Slt: begin
                    result=sa<sb?1:0;
                end
                Lui1,Lui2: result = {b[15:0], 16'b0};
                default:
                    result=a+b;
            endcase
        end

        assign r=result[31:0];
        assign carry =
        (aluc==Addu|aluc==Subu|aluc==Sltu|aluc==Sra|aluc==Srl|aluc==Sll)?result[32]:1'bz;
        assign zero=(r==32'b0)?1:0;
        assign negative=result[31];
        assign overflow=(aluc==Add|aluc==Sub)?(sresult[32]^sresult[33]):1'bz;
endmodule

module ext5#(parameter WIDTH =5)(
    input [WIDTH - 1:0] a,
    output [31:0] b
    );
    assign b={{(32-WIDTH){1'b0}},a};
endmodule

module ext16#(parameter WIDTH =16)(
    input [WIDTH - 1:0] a,
    input s_ext,
    output [31:0] b
    );
    assign b=s_ext?{ { (32-WIDTH){a[WIDTH-1]}},a}:{16'b0,a};
endmodule

module ext18#(parameter WIDTH =18)(
    input [15:0] a,
    output [31:0] b
    );
    assign b={{(32-WIDTH){a[15]}},a,2'b00};
endmodule

module mux32(//deselect 32bits data
```

```
        input [31:0]a,
        input [31:0]b,
        input opt,
        output reg[31:0]c
);
always @(*) begin
        case(opt)
        1'b0:c<=a;
        1'b1:c<=b;
        endcase
end
endmodule

module mux5_1(//de-select 5bits address
        input [4:0]a,
        input [4:0]b,
        input [1:0]opt,
        output reg[4:0]c
);
always @(*)begin
        case (opt)
            2'b00:c<=a;
            2'b01:c<=b;
            2'b10,2'b11:c<=5'b11111;//store address in 31th reg
        endcase
end
endmodule

module mux5_2(//de-select 5bits address
        input [4:0]a,
        input [4:0]b,
        input opt,
        output reg[4:0]c
);
always @(*)begin
        case (opt)
            1'b0:c<=a;
            1'b1:c<=b;
        endcase
end
endmodule

module npc(
     input [31:0]a,
```

```
    input rst,
    output [31:0]c
);
    assign c=(rst)?a:a+4;
endmodule

module join_instr(
    input [3:0]part1,
    input [25:0]part2,
    output [31:0]r
    );
    assign r={part1,part2,2'b0};
endmodule

module add8(
      input [31:0]a,
      output [31:0]c
      );
assign c=a+32'd4;
endmodule

module add(
      input [31:0]a,
      input [31:0]b,
      output [31:0]c,
      output ov
      );
assign c=a+b;
assign ov=(a[31]==b[31]&&c[31]!=b[31])?1:0;
endmodule
```

五、实验结果