计算机组成原理实验报告

学　　号	XXXXXXXXXX
姓　　名	XXXXXX
专　　业	计算机科学与技术
授课老师	XXXXXX

一、实验内容

在本实验中，将使用 verilog HDL 实现 54 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 54条指令的 CPU 的数据通路即控制器。
3. 使用 verilog HDL 设计实现 54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/CLZ/DIV(U)/MUL(TU)		Ext5	Ext16	DMEM		S_Ext16	Ext18	ADD		||		GPR[31]	HI/LO/CP0	CLZ	
				rd	A	B			Addr	Data			A	B	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												
DIV	npc	pc	pc	DIV	rs	rt												HI(q)+LO（r)		
DIVU	npc	pc	pc	DIVU	rs	rt												HI(q)+LO（r)		
MUL	npc	pc	pc	MUL	rs	rt												HI[63:32]+LO[31:0]		
MULTU	npc	pc	pc	MULTU	rs	rt												HI[63:32]+LO[31:0]		
BGEZ	npc	pc	pc	ALU	rs	S_Ext16					imm16									
JALR	||+rs	pc	pc												pc31~28	imem25~0	pc+8			
LBU	npc	pc	pc	Data	rs	S_Ext16			Lbu		offset									
LHU	npc	pc	pc	Data	rs	S_Ext16			Lhu		offset									
LB	npc	pc	pc	Data	rs	S_Ext16			Lb		offset									
LH	npc	pc	pc	Data	rs	S_Ext16			Lh		offset									
SB	npc	pc	pc		rs	S_Ext16			ALU	rt	offset									
SH	npc	pc	pc		rs	S_Ext16			ALU	rt	offset									
BREAK	npc	pc	pc															cp0_reg		
SYSCALL	npc	pc	pc															cp0_reg		
ERET	npc	pc	pc															cp0_reg		
MFHI	npc	pc	pc	HI														HI		
MFLO	npc	pc	pc	LO														LO		
MTHI	npc	pc	pc															rd->HI		
MTLO	npc	pc	pc															rd->LO		
MFC0	npc	pc	pc	cp0_reg																
MTC0	npc	pc	pc															rt->CP0		
CLZ	npc	pc	pc	CLZ															rd	rs
TEQ	npc	pc	pc															cp0_reg		

2. 指令操作时间表（控制信号表）

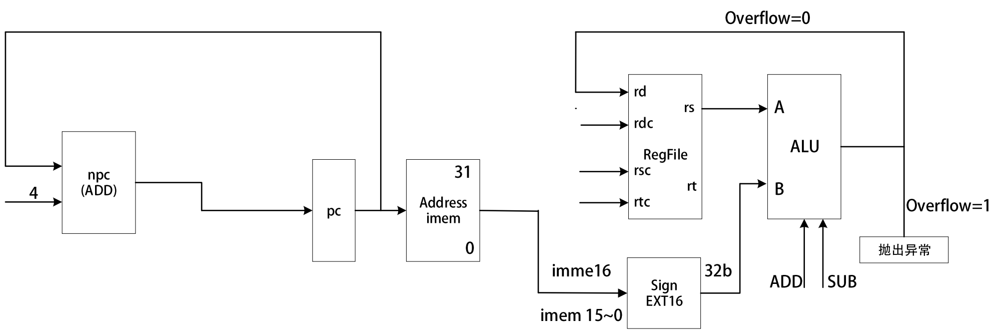
控制信号(微操作)	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR	SLT	SLTU	SLL	SRL	SRA	SLLV	SRLV	SRAV	JR
PC_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21				IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0	IM20-16	IM20-16	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	0		
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
Sw							1							
Lw						1								

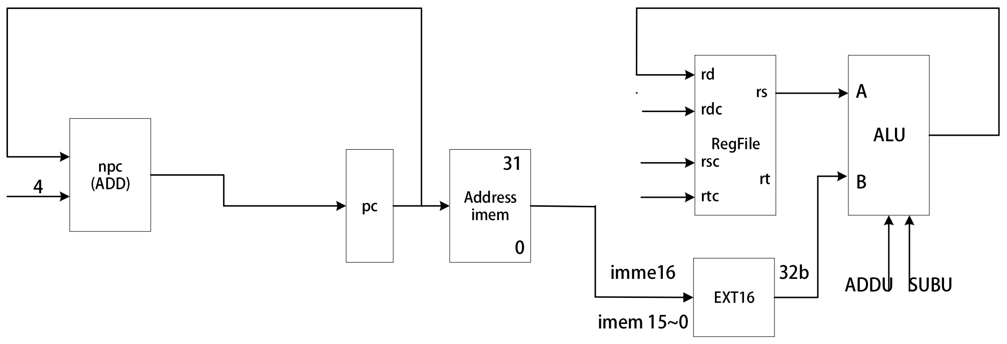
控制信号(微操作)	DIV	DIVU	MUL	MULTU	BGEZ	JALR	LBU	LHU	LB	LH	SB	SH	BREAK	SYSCALL	ERET	MFHI	MFLO	MTHI	MTLO	MFC0	MTC0	CLZ	TEQ
PC_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21												IM25-21	IM25-21			IM25-21	IM25-21
Rtc4-0	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	
M1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	0
M2	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
M3	1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
M4	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
M5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M6	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	
M7											0												
M8											0												
M9											0												
ALUC3					1																		0
ALUC2					0																		0
ALUC1					0																		1
ALUC0					1																		1
RF_W	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	1	1
RF_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
DM_CS	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
DM_R	0	0	0	0	0	0	1	1	1	1	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	1	1	0	0	0	0	0	0	0	0	0	0	0
eret															1								
mtc0																					1		
mfc0																				1			
Sb											1												
Sh												1											
Lb									1														
Lh										1													
Lhu								1															
Lbu							1																

3. 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：

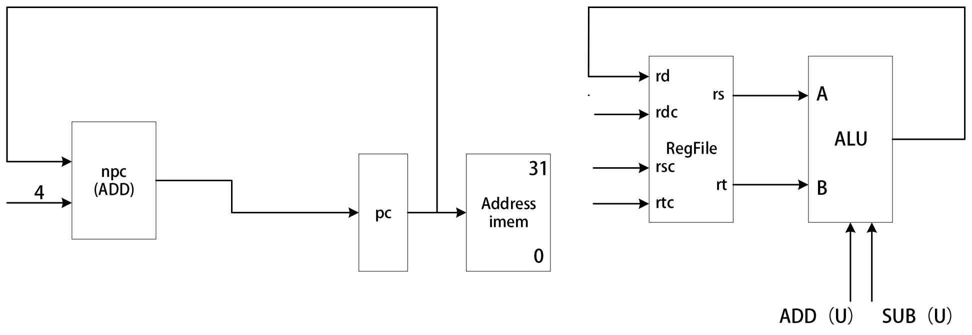
ADDI数据通路



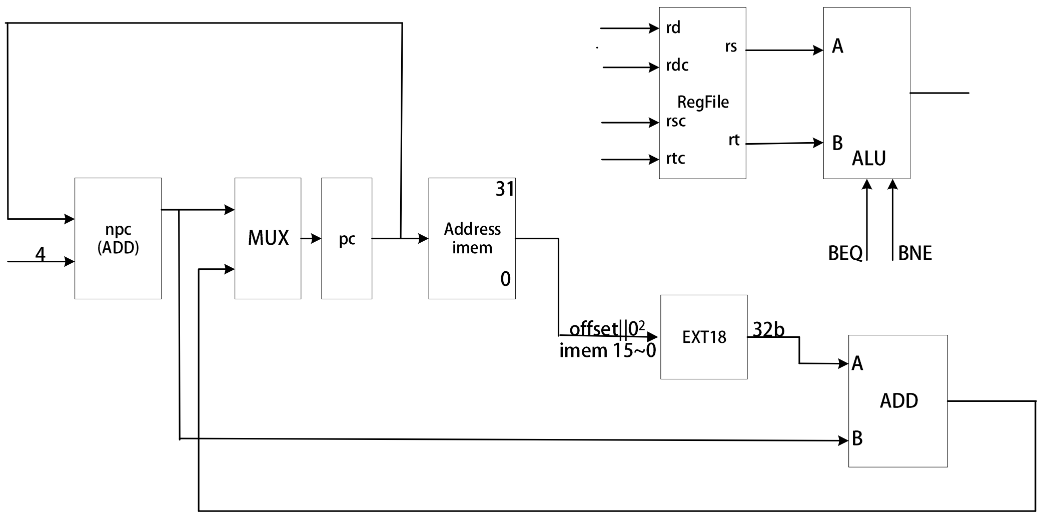
ADDIU数据通路



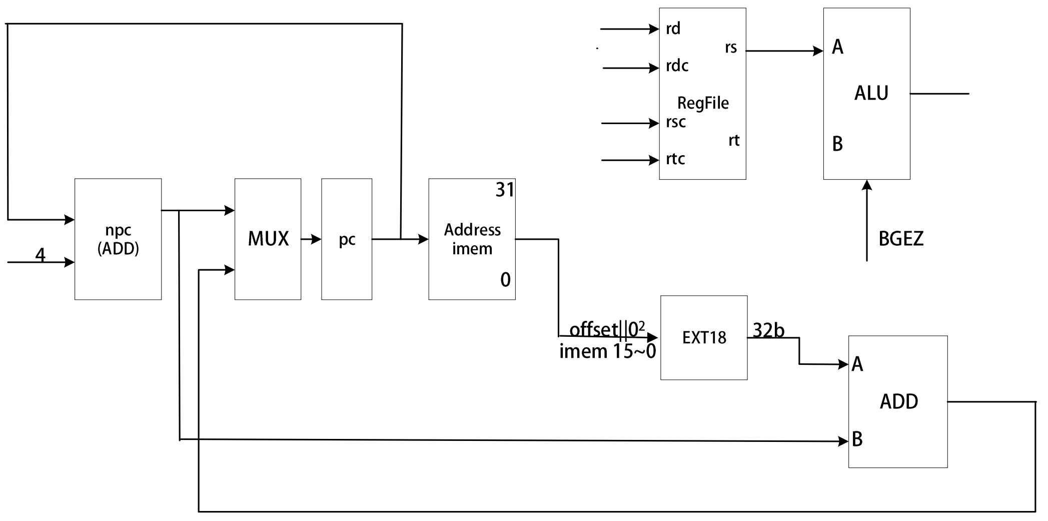
ADDU,ADD,SUBU,SUB数据通路



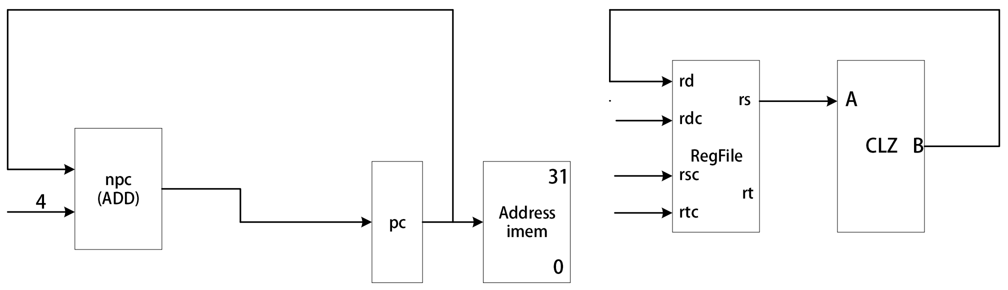
BEQ,BNE数据通路



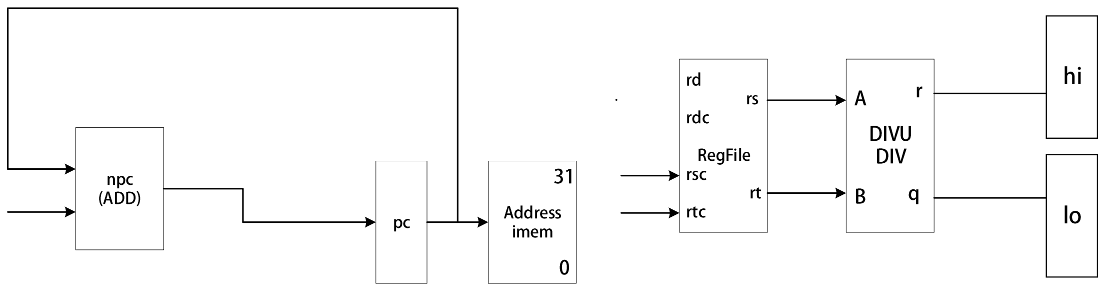
BGEZ数据通路



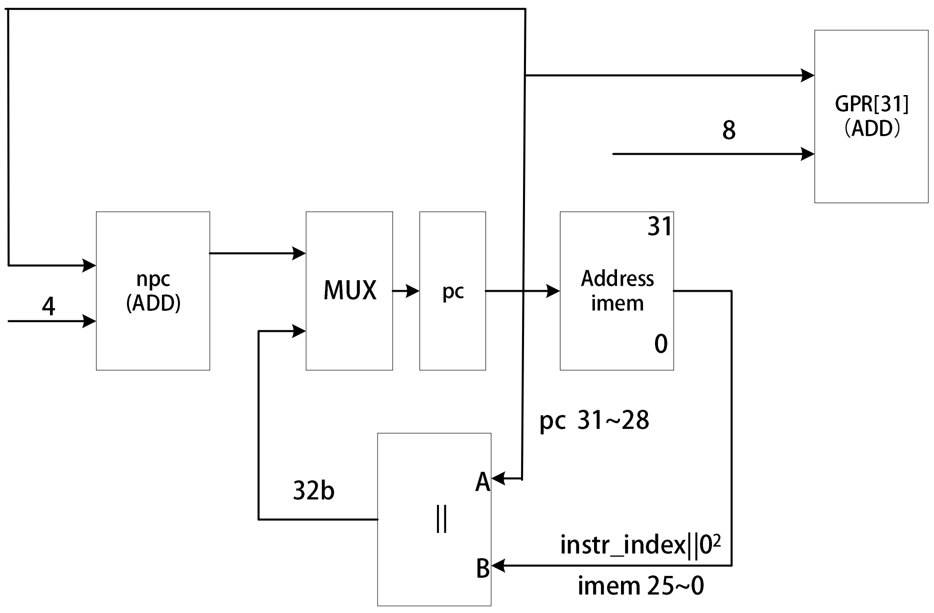
CLZ数据通路



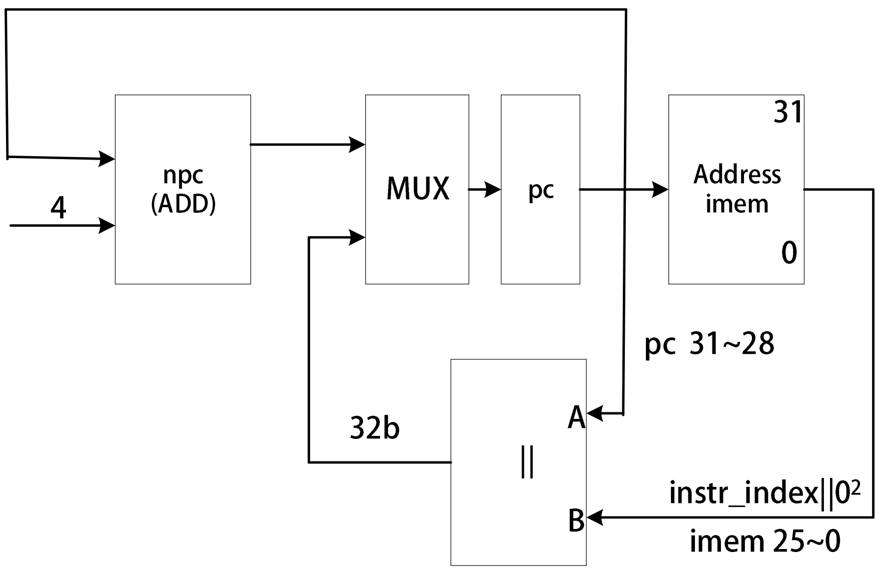
DIVU,DIV数据通路



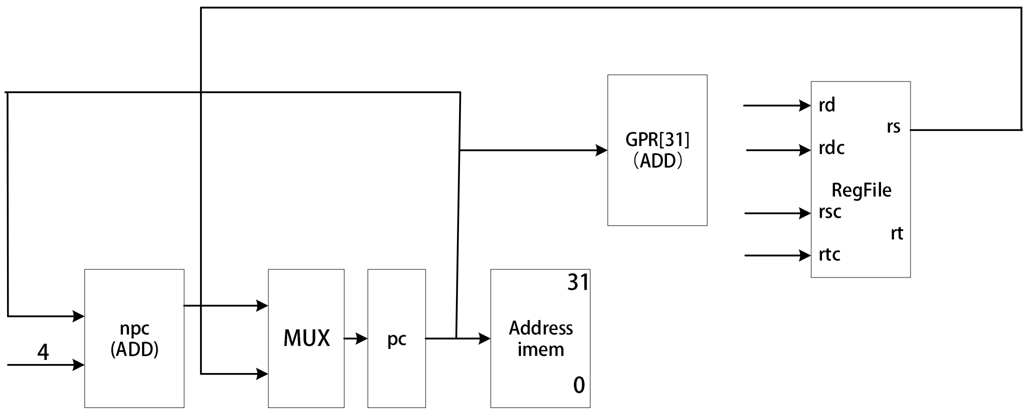
JAL数据通路



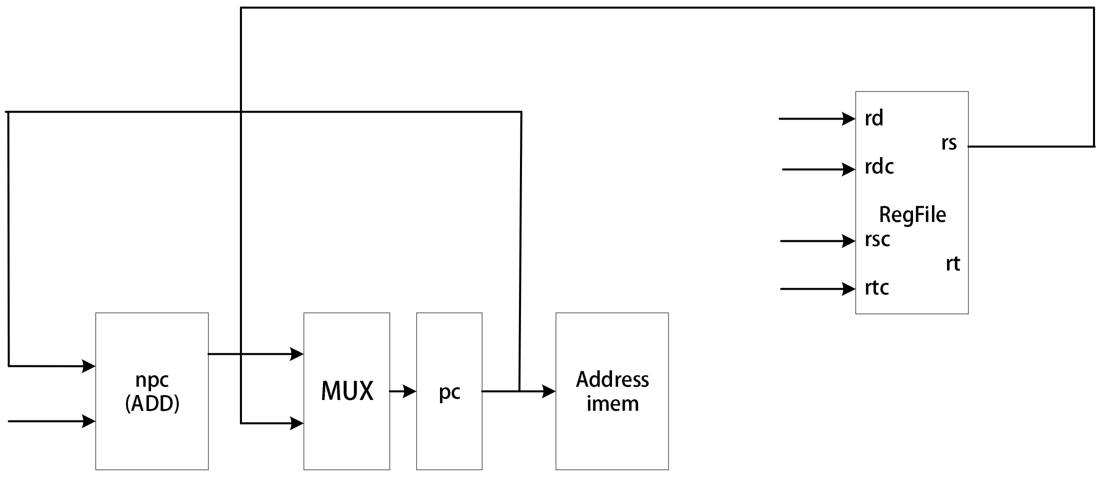
J数据通路



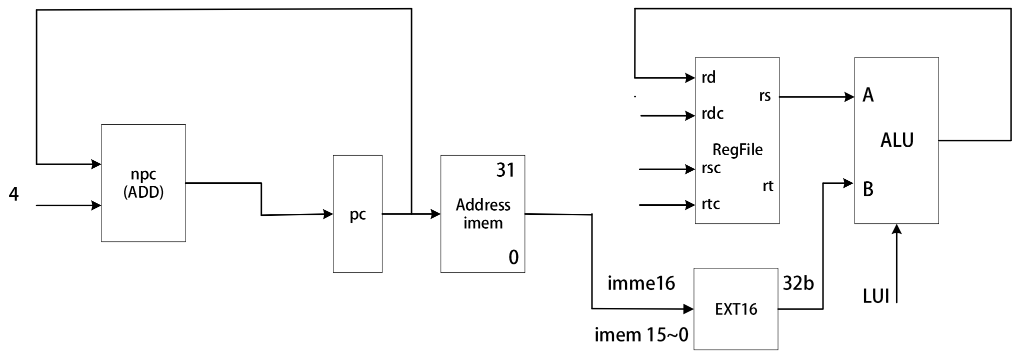
JALR数据通路



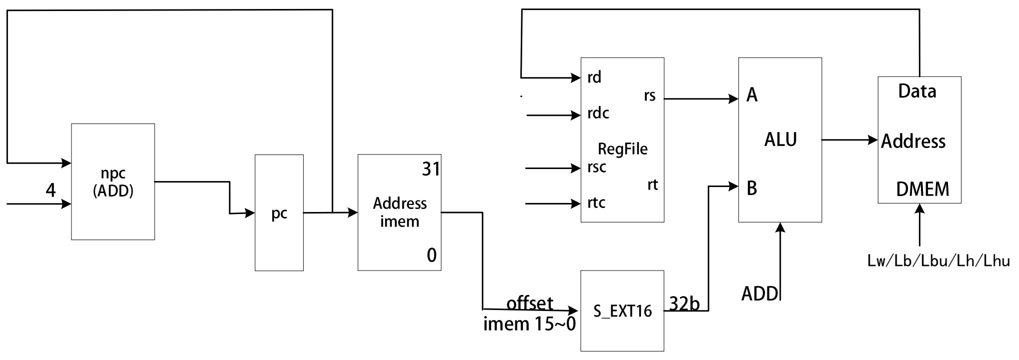
JR数据通路



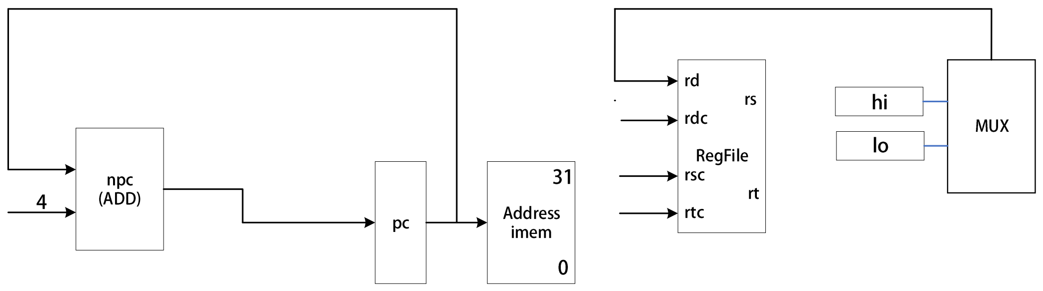
LUI数据通路



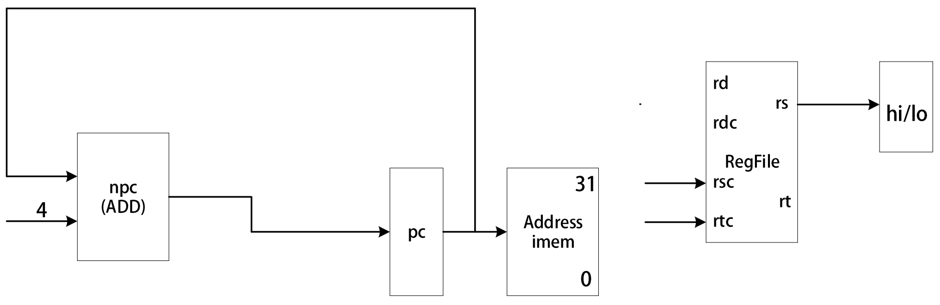
LW/LB/LBU/LH/LHU数据通路



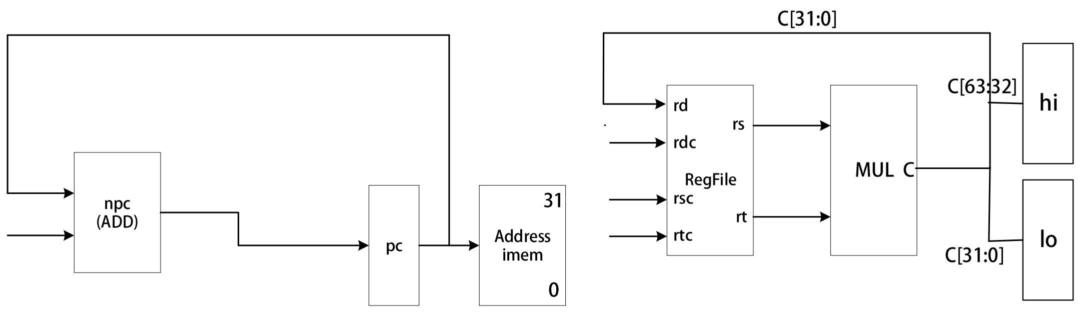
MFHI,MFLO数据通路



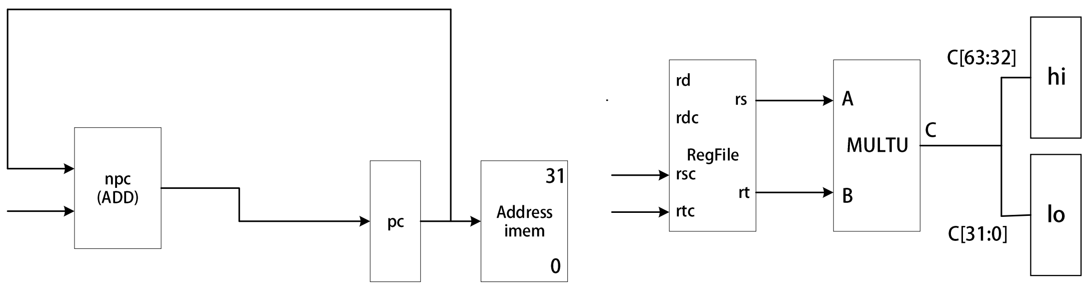
MTHI,MTLO数据通路



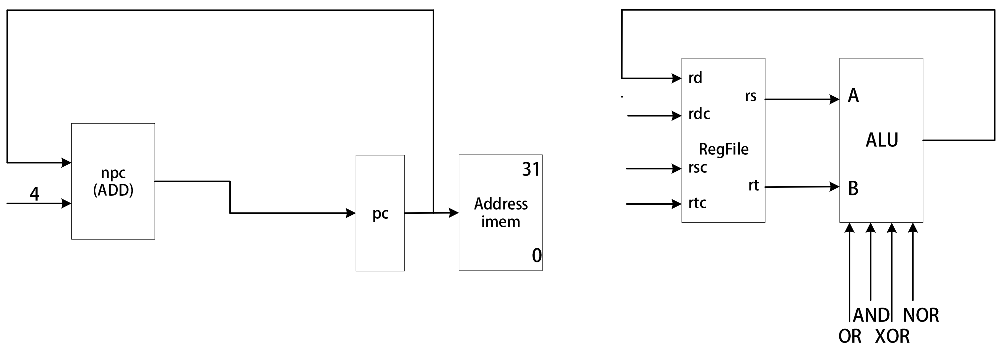
MUL数据通路



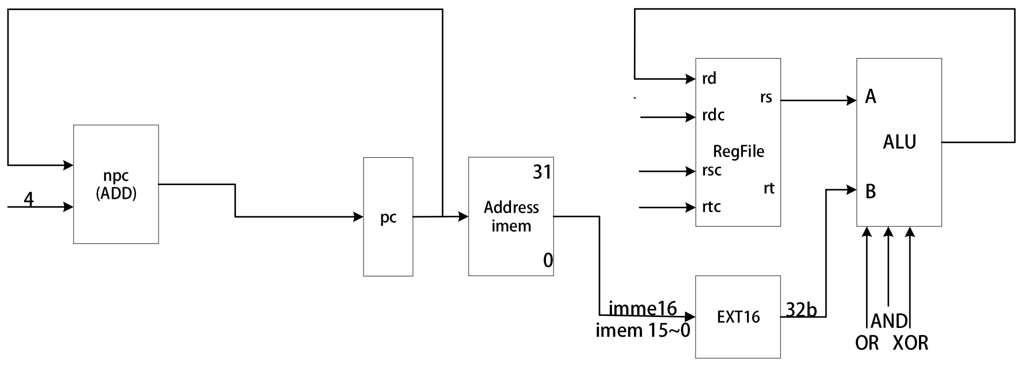
MULTU数据通路



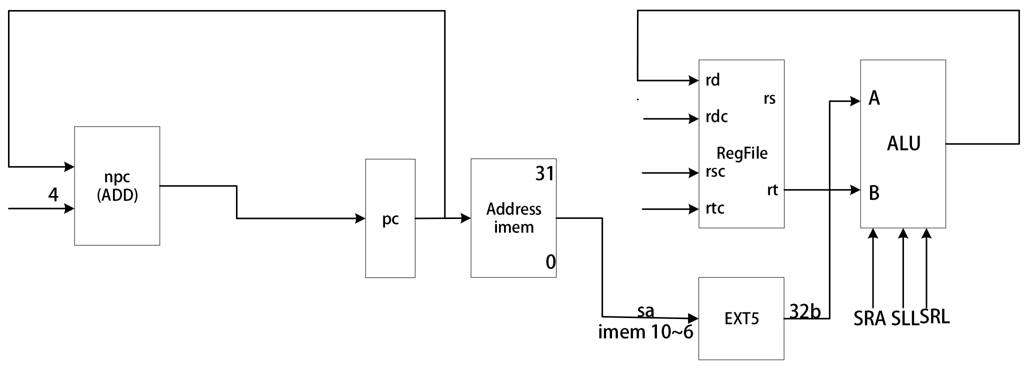
AND,OR,XOR,NOR数据通路



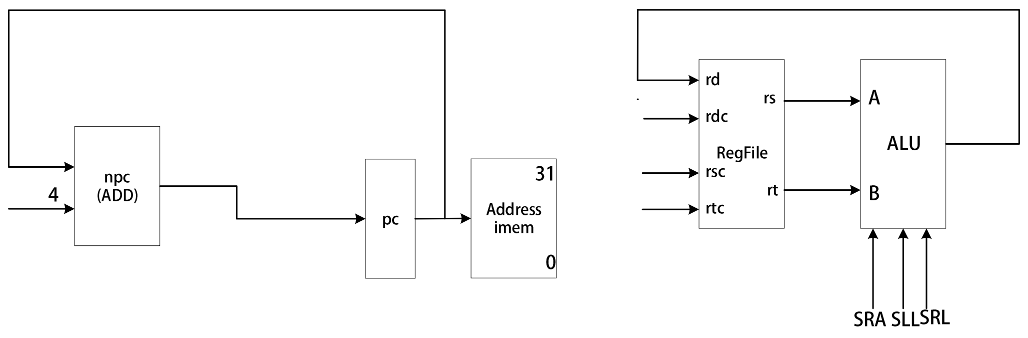
ANDI,ORI,XORI数据通路



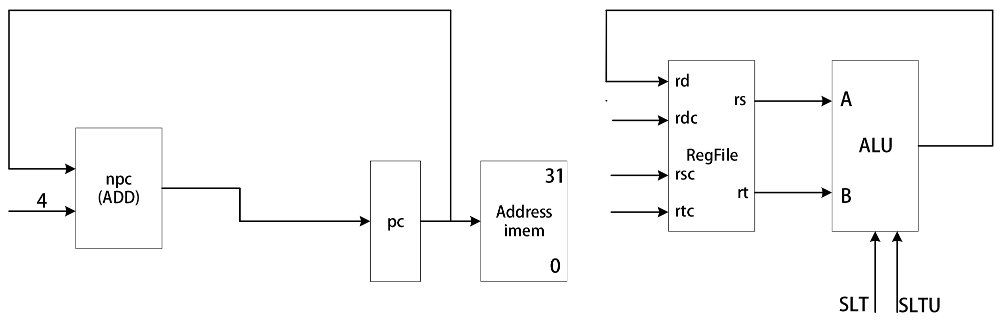
SLL,SRL,SRA数据通路



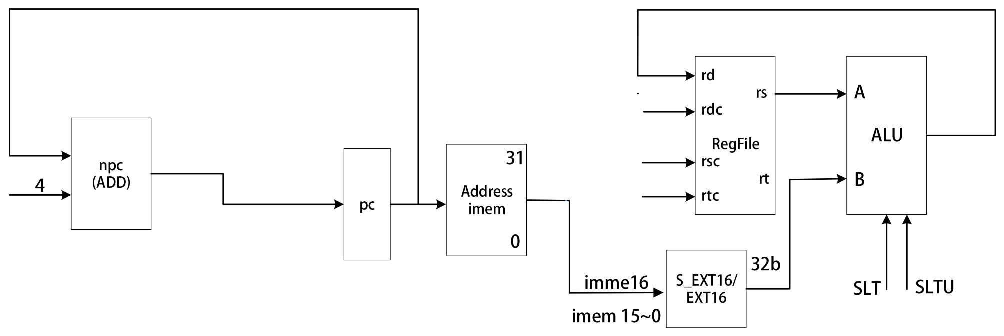
SLLV,SRLV,SRAV数据通路



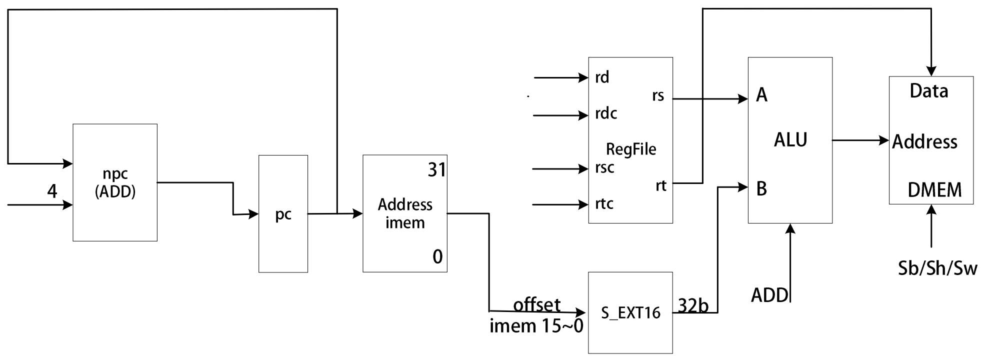
SLT,SLTU数据通路



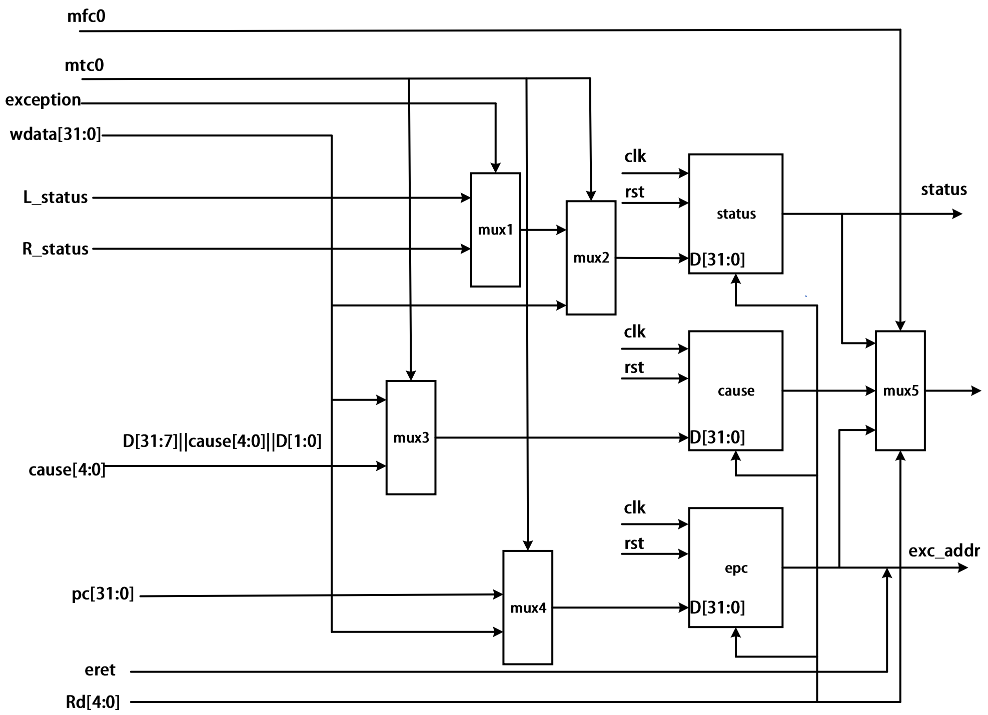
SLTI,SLTIU数据通路



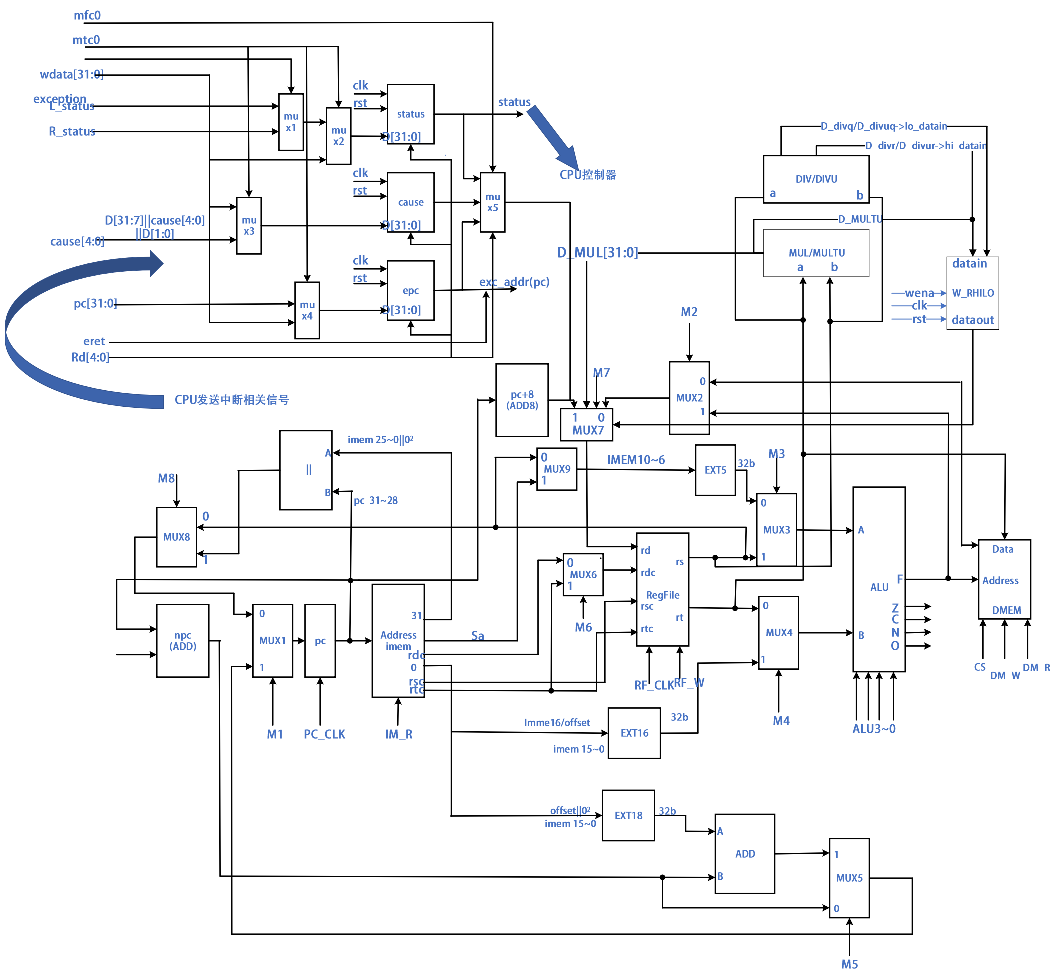
SB/SH/SW数据通路



CP0数据通路



54条单周期CPU总数据通路图



4. 控制信号的逻辑

PC_CLK=clk;

IM_R=1;

M1=~(dec[29]|dec[30]|dec[16]|dec[36]|dec[45]|exc|dec[43]|dec[44]|dec[53]);

M2=dec[22]|dec[37]|dec[38]|dec[39]|dec[40];

M3=~(dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15]|dec[31]|dec[32]|dec[33]|dec[34]|dec[52]|dec[36]);

M4=dec[17]|dec[18]|dec[19]|dec[20]|dec[21]|dec[22]|dec[23]|dec[26]|dec[27]|dec[28]|dec[37]|dec[38]|dec[39]|dec[40]|dec[41]|dec[42];

M5=(dec[24]&z)|(dec[25]&~z)|(dec[35]&~n);

M6=dec[17]|dec[18]|dec[19]|dec[20]|dec[21]|dec[22]|dec[23]|dec[26]|dec[27]|dec[28]|dec[37]|dec[38]|dec[39]|dec[40]|dec[41]|dec[42];

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]|dec[35];

```
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]|dec[53]|dec[37]|dec[38]|dec[39]|dec[40]|dec[41]|dec[42];
ALUC[0]=dec[2]|dec[3]|dec[5]|dec[7]|dec[8]|dec[11]|dec[14]|dec[20]|dec[24]|dec[25]|dec[26]|dec[
53]|dec[35];
RF_W=~(dec[16]|dec[23]|dec[24]|dec[25]|dec[29]|dec[51]|dec[41]|dec[42]);
RF_CLK=clk;
DM_R=dec[22]|dec[37]|dec[38]|dec[39]|dec[40];
DM_W=dec[23]|dec[41]|dec[42];
DM_CS=dec[22]|dec[23]|dec[37]|dec[38]|dec[39]|dec[40]|dec[41]|dec[42];
C_ext16=~(dec[19]|dec[20]|dec[21]);
Lbu=dec[37];
Lhu=dec[38];
Lb=dec[39];
Lh=dec[40];
Sb=dec[41];
Sh=dec[42];
Sw=dec[23];
Lw=dec[22];
```

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


    wire Lbu,Lb,Lhu,Lh,Sw,Lw,Sb,Sh;
    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),
    .Lbu(Lbu),.Lhu(Lhu),.Lh(Lh),.Lb(Lb),.Sb(Sb),.Sh(Sh),.Sw(Sw),.Lw(Lw)
     );

    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),
    .Lbu(Lbu),.Lhu(Lhu),.Lh(Lh),.Lb(Lb),.Sb(Sb),.Sh(Sh),.Sw(Sw),.Lw(Lw)
    );

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 Lw,Sw,Sb,Lb,Lbu,Sh,Lh,Lhu,
        input [31:0] addr,
        input [31:0] datain,
        output reg[31:0] dataout
    );

  reg [31:0]mem[2047:0];
  wire [10:0]Addr=addr[10:0];
```

```
    always @(*)begin
        if(ena&&rena)begin
            if(Lb) dataout<={{24{mem[Addr][7]}},mem[Addr][7:0]};
            if(Lbu)dataout<={24'b0,mem[Addr][7:0]};
            if(Lw) dataout<=mem[Addr];
            if(Lh) dataout<={{16{mem[Addr][15]}},mem[Addr][15:0]};
            if(Lhu) dataout<={16'b0,mem[Addr][15:0]};
        end
        else
            dataout<=32'bx;
      end
      always@(posedge clk)begin
      if(ena)begin
          if(wena&&ena)begin
          if(Sw)mem[Addr]<=datain;
          if(Sb)mem[Addr][7:0]<=datain[7:0];
          if(Sh)mem[Addr][15:0]<=datain[15:0];
      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
    output Sw,
    output Lw,
    output Lbu,
    output Lhu,
    output Lh,
    output Lb,
    output Sh,
    output Sb
);
```

```
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 [63:0] D_mul;
wire [63:0] D_multu;
wire add_ov;
//cp0
wire [4:0]Rd;
wire [31:0]cp0_rdata;
wire [4:0]cause;
wire [31:0]exc_addr;

//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
    wire [31:0]status;
    wire exc;
    wire eret;
    wire mtc0;
    wire mfc0;
    wire div_busy;
    wire divu_busy;
    wire [31:0] D_divr;
     wire [31:0] D_divr1;
    wire [31:0] D_divq;
    wire [31:0] D_divq1;
    wire [31:0] D_divur;
    wire [31:0] D_divur1;
    wire [31:0] D_divuq;
    wire [31:0] D_divuq1;

    assign PC_ENA=1;
    assign pc=D_pc;
    assign addr=D_alu;    //dram read/write address
    assign wdata=D_rt;
    //cp0
    assign  Rd=instr[15:11];//control  cp0_reg  output  mfc0  rt,rd  (rt<-rd)  [20:16]<-
[15:11];
    wire [63:0] dec;//decoded instruction

    reg [31:0] hi;
    reg [31:0] lo;
    wire [31:0]D_hi;
    wire[31:0]D_lo;
    wire [31:0] D_clz;
    wire wena_hi=(dec[31]|dec[32]|dec[48]|dec[34]|dec[33])?1'b1:1'b0;
    wire wena_lo=(dec[31]|dec[32]|dec[49]|dec[34]|dec[33])?1'b1:1'b0;
    wire[5:0]opt={dec[48],dec[49],dec[33],dec[34],dec[31],dec[32]};
    assign          divu_q=divu_busy?32'bz:D_divuq1;
    assign          divu_r=divu_busy?32'bz:D_divur1;
    assign          div_q=div_busy?32'bz:D_divq1;
    assign          div_r=div_busy?32'bz:D_divr1;
```

```
always@(*)begin
case(opt)
6'b100000:hi<=D_rs;
6'b010000:lo<=D_rs;
6'b001000: begin hi<=D_mul[63:32];
                  lo<=D_mul[31:0];end
6'b000100:begin   hi<=D_multu[63:32];
                  lo<=D_multu[31:0]; end
6'b000010: begin   hi<=D_divr;
                   lo<=D_divq; end
6'b000001:begin   hi<=D_divur;
                  lo<=D_divuq;   end
 endcase
end

 instr_decode cpu_dec(//instruction decode module
     .instr(instr),
     .dec(dec)
 );
 cp0 cpu_cp0(
     .clk(clk),.rst(rst),.mfc0(mfc0),.mtc0(mtc0),
     .pc(pc),.Rd(Rd),.wdata(D_rt),.exception(exc),.eret(eret),.cause(cause),
     .rdata(cp0_rdata),.status(status),.exc_addr(exc_addr)
 );
 ctrl cpu_ctrl(// control module
     .clk(clk),.rst(rst),.z(Z),.n(N),.dec(dec),.M1(M1),.M2(M2),.M3(M3),
     .M4(M4),.M5(M5),.M6(M6),.M7(M7),.M8(M8),.M9(M9),.PC_CLK(PC_C
LK),
     .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),

     .status(status),.exc(exc),.eret(eret),.mtc0(mtc0),.mfc0(mfc0),.cause(cause),
    .Lbu(Lbu),.Lhu(Lhu),.Lh(Lh),.Lb(Lb),.Sb(Sb),.Sh(Sh),.Sw(Sw),.Lw(Lw)
 );
 wire busy=divu_busy|div_busy?1'b1:1'b0;
 pcreg   cpu_pc(
     .clk(clk),.rst(rst),
     .ena(PC_ENA),.data_in(D_M1),.wena(!busy),
     .data_out(D_pc)
 );

 regfile cpu_ref(
     .clk(clk),.rst(rst),.ena(1'b1),.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)
 );

WandR_HILO cpu_hi(
            .clk(clk),.rst(rst),.data_in(hi),.data_out(D_hi),.wena(wena_hi)
);

WandR_HILO cpu_lo(
            .clk(clk),.rst(rst),.data_in(lo),.data_out(D_lo),.wena(wena_lo)
);

 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_4     cpu_mux1(
     .a(D_M8),.b(D_M5),.c(D_rs),.d(exc_addr),.opt({dec[45],exc,dec[36],M1}),
.e(D_M1)//
 );
 mux32       cpu_mux2(
     .a(D_alu),.b(rdata),.opt(M2),.c(D_M2)///
 );
 mux32_2     cpu_mux3(
     .a(D_ext5),.b(D_rs),.opt({dec[31],dec[32],dec[36],dec[52],dec[33],dec[34],
M3}),.c(D_M3)//
 );
 mux32_3     cpu_mux4(
     .a(D_rt),.b(D_ext16),.opt({dec[31],dec[32],dec[33],dec[34],M4}),.c(D_M4)
//
 );
 mux32       cpu_mux5(
     .a(D_npc),.b(D_add),.opt(M5),.c(D_M5)//
 );
 mux5_1      cpu_mux6(//dec[30]=M7 : jal
     .a(instr[15:11]),.b(instr[20:16]),
     .opt({dec[33],dec[36],dec[52],dec[46],dec[47],mfc0,M7,M6}),.c(D_M6)
 );//rd      ,rt
 mux32_1     cpu_mux7(
     .a(D_M2),.b(D_add8),.c(cp0_rdata),.d(D_hi),.e(D_lo),.f(D_clz),.g(D_mul[3
1:0]),
     .opt({dec[33],dec[36],dec[52],dec[46],dec[47],mfc0,M7}),.h(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)
 );

 MUL    cpu_mul(.clk(clk),.reset(rst),
 .a(D_M3),.b(D_M4),.z(D_mul)
 );

 MULTU    cpu_multu(.clk(clk),.reset(rst),
 .a(D_M3),.b(D_M4),.z(D_multu)
 );

 DIVU      cpu_divu(.dividend(D_M3),.divisor(D_M4),
 .start(dec[32]),.clk(clk),.reset(rst),
 .q(D_divuq1),.r(D_divur1),.busy(divu_busy)
 );

 DIV      cpu_div(.dividend(D_M3),.divisor(D_M4),
 .start(dec[31]),.clk(clk),.reset(rst),
 .q(D_divq1),.r(D_divr1),.busy(div_busy)
 );

 clz    cpu_clz(.a(D_M3),.opt(dec[52]),.b(D_clz)
 );
endmodule
```

4. 读写HILO寄存器模块

```
module WandR_HILO(
input clk,
input rst,
input wena,
input [31:0]data_in,
output[31:0]data_out
);
reg       [31:0]data;
always @(posedge clk or posedge rst)
        if (rst)
                data <= 32'h00000000;
        else if ( wena)
                data <= data_in;
assign data_out = data;
endmodule
```

5. 计算前导零模块CLZ

```
module clz(
        input [31:0]a,
        input opt,
        output [31:0] b
        );
    assign    b    =a[31]==1?    32'h00000000:a[30]==1?    32'h00000001:a[29]==1?
32'h00000002:a[28]==1? 32'h00000003:a[27]==1? 32'h00000004:
                        a[26]==1?       32'h00000005:a[25]==1?       32'h00000006:a[24]==1?
32'h00000007:a[23]==1? 32'h00000008:a[22]==1? 32'h00000009:
                        a[21]==1?       32'h0000000a:a[20]==1?       32'h0000000b:a[19]==1?
32'h0000000c:a[18]==1? 32'h0000000d:a[17]==1? 32'h0000000e:
                        a[16]==1?       32'h0000000f:a[15]==1?       32'h00000010:a[14]==1?
32'h00000011:a[13]==1? 32'h00000012:a[12]==1? 32'h00000013:
                        a[11]==1?       32'h00000014:a[10]==1?       32'h00000015:a[9]==1?
32'h00000016:a[8]==1? 32'h00000017:a[7]==1? 32'h00000018:
                        a[6]==1?        32'h00000019:a[5]==1?        32'h0000001a:a[4]==1?
32'h0000001b:a[3]==1? 32'h0000001c:a[2]==1? 32'h0000001d:
                        a[1]==1? 32'h0000001e:a[0]==1? 32'h0000001f:32'h00000020;
endmodule
```

6. CP0模块

```
module cp0(
        input clk,
        input rst,
        input mfc0,//mfc0 rt,rd (rt<-rd) [20:16]<-[15:11]
        input mtc0,

        input [31:0]pc,//异常发生时异常指令的 pc

        input [4:0]Rd,//控制 cp0 寄存器选择输出  //mfc0 rt,rd (rt<-rd) [20:16]<-[15:11]

        input [31:0]wdata,//写 cp0 数据,D_rt

        input exception,//异常发生信号,在 cpu 中必须用 status 寄存器控制

        //exception=status[0]&status[3/2/1]&cause[4:0](=syscall or teq or break)

        input eret,//异常返回信号

        input [4:0]cause,//异常发生原因

        output[31:0]rdata,//读 cp0 寄存器数据=>cp0_out

        output[31:0]status,//异常发生，改变后的 status 数据,做控制器[3:1]0 屏蔽中断，

1 禁止屏蔽中断

        output reg [31:0]exc_addr //返回异常发生时的 pc 地址=epc_out

        );
        reg [31:0] cp0_reg [0:31]; //32 cp0 registers
        assign    status = cp0_reg [12];

        always @(posedge clk or posedge rst)begin
        if(rst)begin
                cp0_reg [0] <=0 ;
                cp0_reg [1] <=0 ;
                cp0_reg [2] <=0 ;
                cp0_reg [3] <=0 ;
                cp0_reg [4] <=0 ;
                cp0_reg [5] <=0 ;
                cp0_reg [6] <=0 ;
                cp0_reg [7] <=0 ;
                cp0_reg [8] <=0 ;
                cp0_reg [9] <=0 ;
                cp0_reg [10] <=0 ;
                cp0_reg [11] <=0 ;
                cp0_reg [12] <=0 ;
                cp0_reg [13] <=0 ;
                cp0_reg [14] <=0 ;
                cp0_reg [15] <=0 ;
                cp0_reg [16] <=0 ;
                cp0_reg [17] <=0 ;
                cp0_reg [18] <=0 ;
                cp0_reg [19] <=0 ;
                cp0_reg [20] <=0 ;
                cp0_reg [21] <=0 ;
                cp0_reg [22] <=0 ;
                cp0_reg [23] <=0 ;
                cp0_reg [24] <=0 ;
                cp0_reg [25] <=0 ;
                cp0_reg [26] <=0 ;
                cp0_reg [27] <=0 ;
                cp0_reg [28] <=0 ;
                cp0_reg [29] <=0 ;
                cp0_reg [30] <=0 ;
                cp0_reg [31] <=0 ;

        end
        else begin
        if(mtc0)begin
          cp0_reg [Rd] <= wdata;
        end
        else if(exception)begin
            cp0_reg [14] <=     pc; //epc;
            exc_addr<=32'h00400004;
            cp0_reg [13] <={24'b0,cause,2'b0}; //cause
        end

        else if(eret)begin//处在异常并且返回时

            exc_addr   <=cp0_reg [14];
        end
    end
  end
    assign rdata = mfc0? cp0_reg [Rd]:32'hz;
endmodule
```

7. 控制器模块

```
module ctrl(
        input clk,
        input rst,
        input [63:0]dec,
        input z,//zero
        input n,//neg
        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,
        //cp0
        input [31:0]status,
        output mtc0,
        output mfc0,
        output eret,
        output exc,
        output [4:0]cause,
       output Lbu,
       output Lhu,
       output Lb,
       output Lh,
       output Sb,
       output Sh,
       output Sw,
       output Lw
         );
PC_CLK=clk;
IM_R=1;
M1=~(dec[29]|dec[30]|dec[16]|dec[36]|dec[45]|exc|dec[43]|dec[44]|dec[53]);
M2=dec[22]|dec[37]|dec[38]|dec[39]|dec[40];
M3=~(dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15]|dec[31]|dec[32]|dec[33]|dec[3
4]|dec[52]|dec[36]);
M4=dec[17]|dec[18]|dec[19]|dec[20]|dec[21]|dec[22]|dec[23]|dec[26]|dec[27]|dec[28]|
dec[37]|dec[38]|dec[39]|dec[40]|dec[41]|dec[42];
M5=(dec[24]&z)|(dec[25]&~z)|(dec[35]&~n);
M6=dec[17]|dec[18]|dec[19]|dec[20]|dec[21]|dec[22]|dec[23]|dec[26]|dec[27]|dec[28]|
dec[37]|dec[38]|dec[39]|dec[40]|dec[41]|dec[42];
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]|dec[35];

ALUC[2]=dec[4]|dec[5]|dec[6]|dec[7]|dec[10]|dec[11]|dec[12]|dec[13]|dec[14]|dec[15
]|dec[19]|dec[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]|d
ec[22]|dec[23]|dec[24]|dec[25]|dec[26]|dec[27]|dec[53]|dec[37]|dec[38]|dec[39]|dec[4
0]|dec[41]|dec[42];

ALUC[0]=dec[2]|dec[3]|dec[5]|dec[7]|dec[8]|dec[11]|dec[14]|dec[20]|dec[24]|dec[25]|
dec[26]|dec[53]|dec[35];
F_W=~(dec[16]|dec[23]|dec[24]|dec[25]|dec[29]|dec[51]|dec[41]|dec[42]);
F_CLK=clk;
M_R=dec[22]|dec[37]|dec[38]|dec[39]|dec[40];
M_W=dec[23]|dec[41]|dec[42];
M_CS=dec[22]|dec[23]|dec[37]|dec[38]|dec[39]|dec[40]|dec[41]|dec[42];
C_ext16=~(dec[19]|dec[20]|dec[21]);
Lbu=dec[37];
Lhu=dec[38];
Lb=dec[39];
Lh=dec[40];
Sb=dec[41];
Sh=dec[42];
Sw=dec[23];
Lw=dec[22];
//cp0 ctrl
parameter Break=5'b01001;
parameter Syscall=5'b01000;
parameter Teq=5'b01101;

assign cause = dec[43]?Break:(dec[44]?Syscall:(dec[53]?Teq:5'b00000));
        assign eret=dec[45];
        assign mfc0=dec[50];
        assign mtc0=dec[51];
        assign
exc=(status[0]&&((status[3]&&cause==Teq&&dec[53]&z)||(status[1]&&cause==Sys
call)||(status[2]&&cause==Break)))?1'b1:1'b0; //1 表示中断允许，0 表示中断禁止
endmodule
```

8. 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??????

`define DIV      12'b000000011010
`define DIVU    12'b000000011011
```

```
`define MUL      12'b011100000010
`define MULTU 12'b000000011001

`define BGEZ    12'b000001??????

`define JALR    12'b000000001001
`define LBU      12'b100100??????
`define LHU      12'b100101??????
`define LB        12'b100000??????
`define LH        12'b100001??????
`define SB        12'b101000??????
`define SH        12'b101001??????
`define BREAK 12'b000000001101
`define SYSCALL 12'b000000001100
`define ERET    12'b010000011000
`define MFHI    12'b000000010000
`define MFLO    12'b000000010010
`define MTHI    12'b000000010001
`define MTLO    12'b000000010011
`define MFC0    12'b010000000000
`define MTC0    12'b010000100000
`define CLZ      12'b011100100000
`define TEQ      12'b000000110100

module instr_decode(
        input [31:0]instr,
        output reg [63:0] dec
    );
always @(*)begin
        if(instr[31:26]!=6'b010000)
          casez ({instr[31:26],instr[5:0]})
                `ADD:dec<=64'h00000000_00000001;
                `ADDU:dec<=64'h00000000_00000002;
                `SUB:dec<=64'h00000000_00000004;
                `SUBU:dec<=64'h00000000_00000008;

                `AND:dec<=64'h00000000_00000010;
                `OR:dec<=64'h00000000_00000020;
                `XOR:dec<=64'h00000000_00000040;
                `NOR:dec<=64'h00000000_00000080;

                `SLT:dec<=64'h00000000_00000100;
                `SLTU:dec<=64'h00000000_00000200;
                `SLL:dec<=64'h00000000_00000400;
```

```
  `SRL:dec<=64'h00000000_00000800;

  `SRA:dec<=64'h00000000_00001000;
  `SLLV:dec<=64'h00000000_00002000;
  `SRLV:dec<=64'h00000000_00004000;
  `SRAV:dec<=64'h00000000_00008000;

  `JR:dec<=64'h00000000_00010000;

  `ADDI:dec<=64'h00000000_00020000;
  `ADDIU:dec<=64'h00000000_00040000;
  `ANDI:dec<=64'h00000000_00080000;

  `ORI:dec<=64'h00000000_00100000;
  `XORI:dec<=64'h00000000_00200000;
  `LW:dec<=64'h00000000_00400000;
  `SW:dec<=64'h00000000_00800000;

  `BEQ:dec<=64'h00000000_01000000;
  `BNE:dec<=64'h00000000_02000000;
  `SLTI:dec<=64'h00000000_04000000;
  `SLTIU:dec<=64'h00000000_08000000;
  `LUI:dec<=64'h00000000_10000000;


  `J:dec<=64'h00000000_20000000;
  `JAL:dec<=64'h00000000_40000000;

  `DIV:dec<=64'h00000000_80000000;
  `DIVU:dec<=64'h00000001_00000000;
  `MUL:dec<=64'h00000002_00000000;
`MULTU:dec<=64'h00000004_00000000;
`BGEZ:dec<=64'h00000008_00000000;

`JALR:dec<=64'h00000010_00000000;
`LBU:dec<=64'h00000020_00000000;
`LHU:dec<=64'h00000040_00000000;
`LB:dec<=64'h00000080_00000000;
`LH:dec<=64'h00000100_00000000;
`SB:dec<=64'h00000200_00000000;
`SH:dec<=64'h00000400_00000000;
`BREAK:dec<=64'h00000800_00000000;
`SYSCALL:dec<=64'h00001000_00000000;
`MFHI:dec<=64'h00004000_00000000;
```

```
           `MFLO:dec<=64'h00008000_00000000;
           `MTHI:dec<=64'h00010000_00000000;
           `MTLO:dec<=64'h00020000_00000000;
           `CLZ:dec<=64'h00100000_00000000;
           `TEQ:dec<=64'h00200000_00000000;
            default:dec<=64'bx;
           endcase
         else begin
            if({instr[31:26],instr[5:0]}==`ERET)
                dec<=64'h00002000_00000000;
            else
            casez({instr[31:26],instr[23],instr[4:0]})
            `MFC0:dec<=64'h00040000_00000000;
           `MTC0:dec<=64'h00080000_00000000;
              default:dec<=64'bx;
            endcase
         end
      end

endmodule
```

9. PC寄存器模块

```
module pcreg(
     input clk,
     input rst,
     input ena,
     input wena,
     input [31:0]data_in,
     output[31:0]data_out
     );
     reg    [31:0]data;
     always @(negedge clk or posedge rst)
          if (rst)
               data <= 32'h00400000;
          else if (ena & wena)
               data <= data_in;
     assign data_out = data;
endmodule
```

10. Regfile模块

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

reg [31:0]array_reg[31:0];
assign rdata1 = ena?array_reg[raddr1]:32'bz;
assign rdata2 = ena?array_reg[raddr2]:32'bz;
        always @(negedge clk or posedge rst)
                begin
                if (rst) begin
                array_reg[0] <= 32'h0;
                array_reg[1] <= 32'h0;
                array_reg[2] <= 32'h0;
                array_reg[3] <= 32'h0;
                array_reg[4] <= 32'h0;
                array_reg[5] <= 32'h0;
                array_reg[6] <= 32'h0;
                array_reg[7] <= 32'h0;
                array_reg[8] <= 32'h0;
                array_reg[9] <= 32'h0;
                array_reg[10] <= 32'h0;
                array_reg[11] <= 32'h0;
                array_reg[12] <= 32'h0;
                array_reg[13] <= 32'h0;
                array_reg[14] <= 32'h0;
                array_reg[15] <= 32'h0;
                array_reg[16] <= 32'h0;
                array_reg[17] <= 32'h0;
                array_reg[18] <= 32'h0;
                array_reg[19] <= 32'h0;
                array_reg[20] <= 32'h0;
                array_reg[21] <= 32'h0;
                array_reg[22] <= 32'h0;
                array_reg[23] <= 32'h0;
                array_reg[24] <= 32'h0;
                array_reg[25] <= 32'h0;
                array_reg[26] <= 32'h0;
                array_reg[27] <= 32'h0;
                array_reg[28] <= 32'h0;
```

```
            array_reg[29] <= 32'h0;
            array_reg[30] <= 32'h0;
            array_reg[31] <= 32'h0;
            end
            else if ((ena&we) &&(waddr != 0))
                array_reg[waddr] <= wdata;
        end
endmodule
```

11. ALU模块

```
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 Lui    =    4'b1000;    //r={b[15:0],16'b0}
        parameter Bgez    =    4'b1001;    //r=a;
        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
                Lui: result = {b[15:0], 16'b0};
                Bgez: result = a;
        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
```

12. 乘法器模块

```
module    MUL(
    input clk,
    input reset,
    input     [31:0] a,//rs
    input     [31:0] b,//rt
    output    [63:0] z
);
    reg    [31:0]a1;
    reg    [31:0]b1;
    reg      [63:0] temp;
    reg      [63:0] s0;
    reg      [63:0] s1;
    reg      [63:0] s2;
    reg      [63:0] s3;
    reg      [63:0] s4;
    reg      [63:0] s5;
    reg      [63:0] s6;
    reg      [63:0] s7;
    reg      [63:0] s8;
    reg      [63:0] s9;
    reg      [63:0] s10;
    reg      [63:0] s11;
    reg      [63:0] s12;
    reg      [63:0] s13;
    reg      [63:0] s14;
    reg      [63:0] s15;
    reg      [63:0] s16;
    reg      [63:0] s17;
reg      [63:0] s18;
reg      [63:0] s19;
reg      [63:0] s20;
reg      [63:0] s21;
reg      [63:0] s22;
reg      [63:0] s23;
reg      [63:0] s24;
reg      [63:0] s25;
reg      [63:0] s26;
reg      [63:0] s27;
 reg      [63:0] s28;
reg      [63:0] s29;
reg      [63:0] s30;
reg      [63:0] s31;

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] add01_23;
reg      [63:0] add45_67;
 reg      [63:0] add89_1011;
reg      [63:0] add1213_1415;
reg      [63:0] add1617_1819;
reg      [63:0] add2021_2223;
 reg      [63:0] add2425_2627;
reg      [63:0] add2829_3031;

 reg      [63:0] add0t3_4t7;
reg      [63:0] add8t11_12t15;
reg      [63:0] add16t19_20t23;
reg     [63:0] add24t27_28t31;

reg     [63:0] add0t7_8t15;
reg     [63:0] add16t23_24t31;
reg     sign;
integer i;
always@(*)
begin
if(reset)begin
  temp<=0;
  s0<=0;
  s1<=0;
  s2<=0;
  s3<=0;
  s4<=0;
  s5<=0;
  s6<=0;
  s7<=0;
  s8<=0;
  s9<=0;
  s10<=0;
  s11<=0;
  s12<=0;
  s13<=0;
  s14<=0;
  s15<=0;
  s16<=0;
  s17<=0;
  s18<=0;
  s19<=0;
  s20<=0;
  s21<=0;
  s22<=0;
  s23<=0;
  s24<=0;
  s25<=0;
  s26<=0;
  s27<=0;
  s28<=0;
  s29<=0;
  s30<=0;
  s31<=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;
add01_23<=0;
 add45_67<=0;
 add89_1011<=0;
add1213_1415<=0;
add1617_1819<=0;
add2021_2223<=0;
 add2425_2627<=0;
add2829_3031<=0;

 add0t3_4t7<=0;
add8t11_12t15<=0;
add16t19_20t23<=0;
add24t27_28t31<=0;

add0t7_8t15<=0;
add16t23_24t31<=0;
end
else begin
  sign<=a[31]^b[31];
   if(a[31]==1'b0)
      a1<=a;
   else
        a1=~a+1'b1;
   if(b[31]==1'b0)
      b1<=b;
     else
      b1<=~b+1'b1;
 for(i=1;i<=32;i=i+1)begin
  s0=b1[0]?{32'b0,a1}:64'b0;
```

```
s1=b1[1]?{31'b0,a1,1'b0}:64'b0;
s2=b1[2]?{30'b0,a1,2'b0}:64'b0;
s3=b1[3]?{29'b0,a1,3'b0}:64'b0;
s4=b1[4]?{28'b0,a1,4'b0}:64'b0;
s5=b1[5]?{27'b0,a1,5'b0}:64'b0;
s6=b1[6]?{26'b0,a1,6'b0}:64'b0;
s7=b1[7]?{25'b0,a1,7'b0}:64'b0;
s8=b1[8]?{24'b0,a1,8'b0}:64'b0;
 s9=b1[9]?{23'b0,a1,9'b0}:64'b0;
 s10=b1[10]?{22'b0,a1,10'b0}:64'b0;
 s11=b1[11]?{21'b0,a1,11'b0}:64'b0;
s12<=b1[12]?{20'b0,a1,12'b0}:64'b0;
s13<=b1[13]?{19'b0,a1,13'b0}:64'b0;
s14<=b1[14]?{18'b0,a1,14'b0}:64'b0;
s15<=b1[15]?{17'b0,a1,15'b0}:64'b0;
s16<=b1[16]?{16'b0,a1,16'b0}:64'b0;
s17<=b1[17]?{15'b0,a1,17'b0}:64'b0;
s18<=b1[18]?{14'b0,a1,18'b0}:64'b0;
s19<=b1[19]?{13'b0,a1,19'b0}:64'b0;
s20<=b1[20]?{12'b0,a1,20'b0}:64'b0;
s21<=b1[21]?{11'b0,a1,21'b0}:64'b0;
s22<=b1[22]?{10'b0,a1,22'b0}:64'b0;
 s23<=b1[23]?{9'b0,a1,23'b0}:64'b0;
s24<=b1[24]?{8'b0,a1,24'b0}:64'b0;
s25<=b1[25]?{7'b0,a1,25'b0}:64'b0;
s26<=b1[26]?{6'b0,a1,26'b0}:64'b0;
s27<=b1[27]?{5'b0,a1,27'b0}:64'b0;
s28<=b1[28]?{4'b0,a1,28'b0}:64'b0;
s29<=b1[29]?{3'b0,a1,29'b0}:64'b0;
s30<=b1[30]?{2'b0,a1,30'b0}:64'b0;
s31<=b1[31]?{1'b0,a1,31'b0}:64'b0;
add01<=s0+s1;
add23<=s3+s2;
add45<=s4+s5;
add67<=s6+s7;
add89<=s8+s9;
add1011<=s10+s11;
add1213<=s12+s13;
add1415<=s14+s15;
 add1617<=s16+s17;
add1819<=s18+s19;
add2021<=s20+s21;
add2223<=s22+s23;
add2425<=s24+s25;
```

```
         add2627<=s26+s27;
         add2829<=s28+s29;
         add3031<=s30+s31;

         add01_23<=add01+add23;
          add45_67<=add45+add67;
          add89_1011<=add89+add1011;
         add1213_1415<=add1213+add1415;
         add1617_1819<=add1617+add1819;
         add2021_2223<=add2021+add2223;
          add2425_2627<=add2425+add2627;
         add2829_3031<=add2829+add3031;

          add0t3_4t7<=add01_23+add45_67;
         add8t11_12t15<=add89_1011+add1213_1415;
         add16t19_20t23<=add1617_1819+add2021_2223;
         add24t27_28t31<=add2425_2627+add2829_3031;

         add0t7_8t15<=add0t3_4t7+add8t11_12t15;
         add16t23_24t31<=add16t19_20t23+add24t27_28t31;
         if(sign==1'b1)
         temp<=~(add0t7_8t15+add16t23_24t31)+1'b1;
         else
         temp<=add0t7_8t15+add16t23_24t31;
         end
         end
       end
       assign z=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] s0;
        reg [63:0] s1;
       reg [63:0] s2;
       reg [63:0] s3;
reg [63:0] s4;
reg [63:0] s5;
reg [63:0] s6;
reg [63:0] s7;
reg [63:0] s8;
reg [63:0] s9;
reg [63:0] s10;
reg [63:0] s11;
 reg [63:0] s12;
reg [63:0] s13;
reg [63:0] s14;
reg [63:0] s15;
reg [63:0] s16;
 reg [63:0] s17;
reg [63:0] s18;
reg [63:0] s19;
reg [63:0] s20;
reg [63:0] s21;
reg [63:0] s22;
reg [63:0] s23;
reg [63:0] s24;
reg [63:0] s25;
reg [63:0] s26;
reg [63:0] s27;
 reg [63:0] s28;
reg [63:0] s29;
reg [63:0] s30;
reg [63:0] s31;
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] add01_23;
reg [63:0] add45_67;
 reg [63:0] add89_1011;
reg [63:0] add1213_1415;
reg [63:0] add1617_1819;
reg [63:0] add2021_2223;
 reg [63:0] add2425_2627;
reg [63:0] add2829_3031;
 reg [63:0] add0t3_4t7;
reg [63:0] add8t11_12t15;
reg [63:0] add16t19_20t23;
reg [63:0] add24t27_28t31;
reg [63:0] add0t7_8t15;
reg [63:0] add16t23_24t31;
always@(posedge clk or negedge reset)
begin
if(reset)begin
  temp<=0;
  s0<=0;
  s1<=0;
  s2<=0;
  s3<=0;
  s4<=0;
  s5<=0;
  s6<=0;
  s7<=0;
  s8<=0;
  s9<=0;
  s10<=0;
  s11<=0;
  s12<=0;
  s13<=0;
  s14<=0;
  s15<=0;
  s16<=0;
  s17<=0;
  s18<=0;
  s19<=0;
  s20<=0;
  s21<=0;
  s22<=0;
  s23<=0;
  s24<=0;
  s25<=0;
```

```
s26<=0;
s27<=0;
s28<=0;
s29<=0;
s30<=0;
s31<=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;
add01_23<=0;
 add45_67<=0;
 add89_1011<=0;
add1213_1415<=0;
add1617_1819<=0;
add2021_2223<=0;
 add2425_2627<=0;
add2829_3031<=0;
 add0t3_4t7<=0;
add8t11_12t15<=0;
add16t19_20t23<=0;
add24t27_28t31<=0;
add0t7_8t15<=0;
add16t23_24t31<=0;
end
else begin
s0<=b[0]?{32'b0,a}:64'b0;
s1<=b[1]?{31'b0,a,1'b0}:64'b0;
s2<=b[2]?{30'b0,a,2'b0}:64'b0;
s3<=b[3]?{29'b0,a,3'b0}:64'b0;
s4<=b[4]?{28'b0,a,4'b0}:64'b0;
s5<=b[5]?{27'b0,a,5'b0}:64'b0;
```

```
s6<=b[6]?{26'b0,a,6'b0}:64'b0;
s7<=b[7]?{25'b0,a,7'b0}:64'b0;
s8<=b[8]?{24'b0,a,8'b0}:64'b0;
 s9<=b[9]?{23'b0,a,9'b0}:64'b0;
 s10<=b[10]?{22'b0,a,10'b0}:64'b0;
 s11<=b[11]?{21'b0,a,11'b0}:64'b0;
s12<=b[12]?{20'b0,a,12'b0}:64'b0;
s13<=b[13]?{19'b0,a,13'b0}:64'b0;
s14<=b[14]?{18'b0,a,14'b0}:64'b0;
s15<=b[15]?{17'b0,a,15'b0}:64'b0;
s16<=b[16]?{16'b0,a,16'b0}:64'b0;
s17<=b[17]?{15'b0,a,17'b0}:64'b0;
s18<=b[18]?{14'b0,a,18'b0}:64'b0;
s19<=b[19]?{13'b0,a,19'b0}:64'b0;
s20<=b[20]?{12'b0,a,20'b0}:64'b0;
s21<=b[21]?{11'b0,a,21'b0}:64'b0;
s22<=b[22]?{10'b0,a,22'b0}:64'b0;
 s23<=b[23]?{9'b0,a,23'b0}:64'b0;
s24<=b[24]?{8'b0,a,24'b0}:64'b0;
s25<=b[25]?{7'b0,a,25'b0}:64'b0;
s26<=b[26]?{6'b0,a,26'b0}:64'b0;
s27<=b[27]?{5'b0,a,27'b0}:64'b0;
s28<=b[28]?{4'b0,a,28'b0}:64'b0;
s29<=b[29]?{3'b0,a,29'b0}:64'b0;
s30<=b[30]?{2'b0,a,30'b0}:64'b0;
s31<=b[31]?{1'b0,a,31'b0}:64'b0;
  add01<=s0+s1;
add23<=s3+s2;
add45<=s4+s5;
add67<=s6+s7;
add89<=s8+s9;
add1011<=s10+s11;
add1213<=s12+s13;
add1415<=s14+s15;
 add1617<=s16+s17;
add1819<=s18+s19;
add2021<=s20+s21;
add2223<=s22+s23;
add2425<=s24+s25;
add2627<=s26+s27;
add2829<=s28+s29;
add3031<=s30+s31;
add01_23<=add01+add23;
 add45_67<=add45+add67;
```

```
             add89_1011<=add89+add1011;
            add1213_1415<=add1213+add1415;
            add1617_1819<=add1617+add1819;
            add2021_2223<=add2021+add2223;
             add2425_2627<=add2425+add2627;
            add2829_3031<=add2829+add3031;
             add0t3_4t7<=add01_23+add45_67;
            add8t11_12t15<=add89_1011+add1213_1415;
            add16t19_20t23<=add1617_1819+add2021_2223;
            add24t27_28t31<=add2425_2627+add2829_3031;
            add0t7_8t15<=add0t3_4t7+add8t11_12t15;
            add16t23_24t31<=add16t19_20t23+add24t27_28t31;
            temp<=add0t7_8t15+add16t23_24t31;
            end
        end
        assign z=temp;
endmodule
```

13. 除法器模块

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

        reg [4:0]count;
        reg [31:0] reg_q;
        reg [31:0] reg_r;
        reg [31:0] reg_b;
        reg r_sign;
        wire  [32:0]  sub_add  =  r_sign?({reg_r,q[31]}  +  {1'b0,reg_b}):({reg_r,q[31]}  -
{1'b0,reg_b}); //加、减法器
        assign r = r_sign? reg_r + reg_b : reg_r;
        assign q = reg_q;
        always @ (negedge clk or posedge reset)begin
        if (reset == 1) begin //重置
        count <=5'b0;
        busy <= 0;
        end
        else begin
        if (start&&(!busy)) begin //开始除法运算，初始化
        reg_r <= 32'b0;
        r_sign <= 0;
        reg_q <= dividend;
        reg_b <= divisor;
        count <= 5'b0;
        busy <= 1'b1;
        end else if (busy)
        begin //循环操作
        reg_r <= sub_add[31:0]; //部分余数
        r_sign <= sub_add[32]; //如果为负，下次相加
        reg_q <= {reg_q[30:0],~sub_add[32]};
        count <= count +5'b1; //计数器加一
        if (count == 5'b11111) busy <= 0; //结束除法运算
        end
        end
        end
endmodule


module DIV(
    input [31:0]dividend,
    input [31:0]divisor,
    input start,
    input clk,
    input reset,
    output [31:0]q,
    output [31:0]r,
    output reg busy
    );
    reg [4:0]count;
    reg [31:0]reg_q;
    reg [31:0]reg_r;
    reg [31:0]reg_b;
    reg q_sign;
```

```
   reg r_sign;
   reg a_sign;
   wire    [32:0]sub_add=r_sign?{reg_r,reg_q[31]}+{1'b0,reg_b}:{reg_r,reg_q[31]}-
{1'b0,reg_b};
   assign r=a_sign?(-(r_sign?reg_r+reg_b:reg_r)):(r_sign?reg_r+reg_b:reg_r);
   assign q=q_sign?-reg_q:reg_q;
   always@(negedge clk or posedge reset)
   begin
   if(reset)
   begin count<=0;busy<=0;end
   else if(start&&(!busy))begin
   count<=0;reg_q<=dividend[31]?-dividend:dividend;
   reg_r<=0;
   reg_b<=divisor[31]?-divisor:divisor;
   r_sign<=0;busy<=1'b1;
   q_sign<=dividend[31]^divisor[31];
   a_sign<=dividend[31];
   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+1;
   if(count==31)busy<=0;
   end
   end
   endmodule
```

14. 其他模块

```
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
```

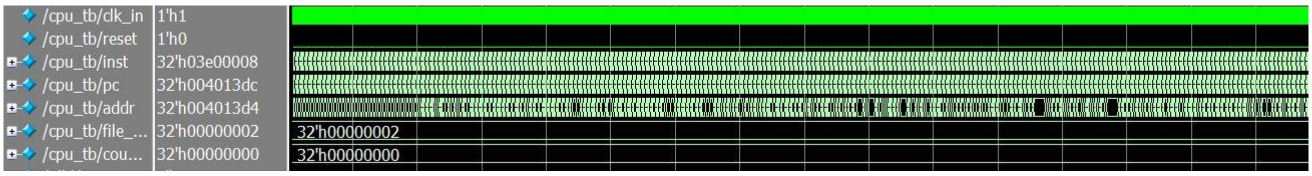
```
module ext18#(parameter WIDTH =18)(
    input [15:0] a,
    output [31:0] b
    );
    assign b={{(32-WIDTH){a[15]}},a,2'b00};
endmodule

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 mux32_1(//deselect 32bits data
     input [31:0]a,
     input [31:0]b,
     input [31:0]c,
     input [31:0]d,//hi
     input [31:0]e,//lo
     input [31:0]f,
     input [31:0]g,
     input [6:0]opt,
     output reg[31:0]h
     );
     always@(*)begin
     case(opt)
     7'b0000000:h<=a;
     7'b0000001,7'b0100000:h<=b;
     7'b0000010:h<=c;
     7'b0001000:h<=d;
     7'b0000100:h<=e;
     7'b0010000:h<=f;
     7'b1000000:h<=g;
     endcase
  end
endmodule
```

```
module mux32_2(//deselect 32bits data
	input [31:0]a,
	input [31:0]b,
	input [6:0]opt,
	output reg[31:0]c
	);
	always @(*) begin
	case(opt)
	7'b0000000:c<=a;
	7'b0000001,7'b0000010,7'b0000100,7'b0001000,7'b0010000,7'b1000000,7'b0100000:c<=b;
	endcase
	end
endmodule

module mux32_3(//deselect 32bits data
	input [31:0]a,
	input [31:0]b,
	input [4:0]opt,
	output reg[31:0]c
	);
	always @(*) begin
	case(opt)
	5'b00000,5'b00100,5'b00010,5'b10000,5'b01000:c<=a;
	5'b00001:c<=b;
	endcase
	end
endmodule

module mux32_4(//deselect 32bits data
	input [31:0]a,
	input [31:0]b,
	input [31:0]c,
	input [31:0]d,
	input [3:0]opt,
	output reg[31:0]e
	);
	always @(*) begin
	case(opt)
	4'b0000:e<=a;
	4'b0001:e<=b;
	4'b0010:e<=c;
	4'b1000,4'b0100:e<=d;
	endcase
```

```
        end
endmodule

module mux5_1(//de-select 5bits address
        input [4:0]a,
        input [4:0]b,
        input [7:0]opt,
        output reg[4:0]c
        );
        always @(*)begin
        case (opt)

8'b00000000,8'b00010000,8'b00001000,8'b00100000,8'b01000000,8'b10000000:c<=a;//rd
            8'b00000001,8'b00000100:c<=b;//rt
            8'b00000010,8'b00000011: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
```

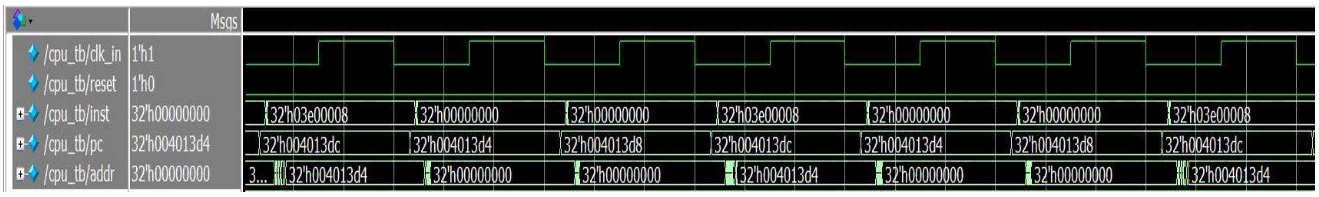
五、 实验结果截图

1. 前仿真结果截图



```
/cpu_tb/clk_in     1'h1
/cpu_tb/reset      1'h0
/cpu_tb/inst       32'h03e00008
/cpu_tb/pc         32'h004013dc
/cpu_tb/addr       32'h004013d4
/cpu_tb/file_...   32'h00000002     32'h00000002
/cpu_tb/cou...     32'h00000000     32'h00000000
```

2. 后仿真结果截图



```
                   Msgs
/cpu_tb/clk_in     1'h1
/cpu_tb/reset      1'h0
/cpu_tb/inst       32'h00000000   32'h03e00008  32'h00000000  32'h00000000  32'h03e00008  32'h00000000  32'h00000000  32'h03e00008
/cpu_tb/pc         32'h004013d4   32'h004013dc  32'h004013d4  32'h004013d8  32'h004013dc  32'h004013d4  32'h004013d8  32'h004013dc
/cpu_tb/addr       32'h00000000   3...  32'h004013d4  32'h00000000  32'h00000000  32'h004013d4  32'h00000000  32'h00000000  32'h004013d4
```

3. 下板结果截图