同济大学
计算机科学与技术系

计算机组成原理课程实验报告

学　　号 **1651459**

姓　　名 宋宇凡

专　　业 计算机科学与技术

授课老师 陈永生

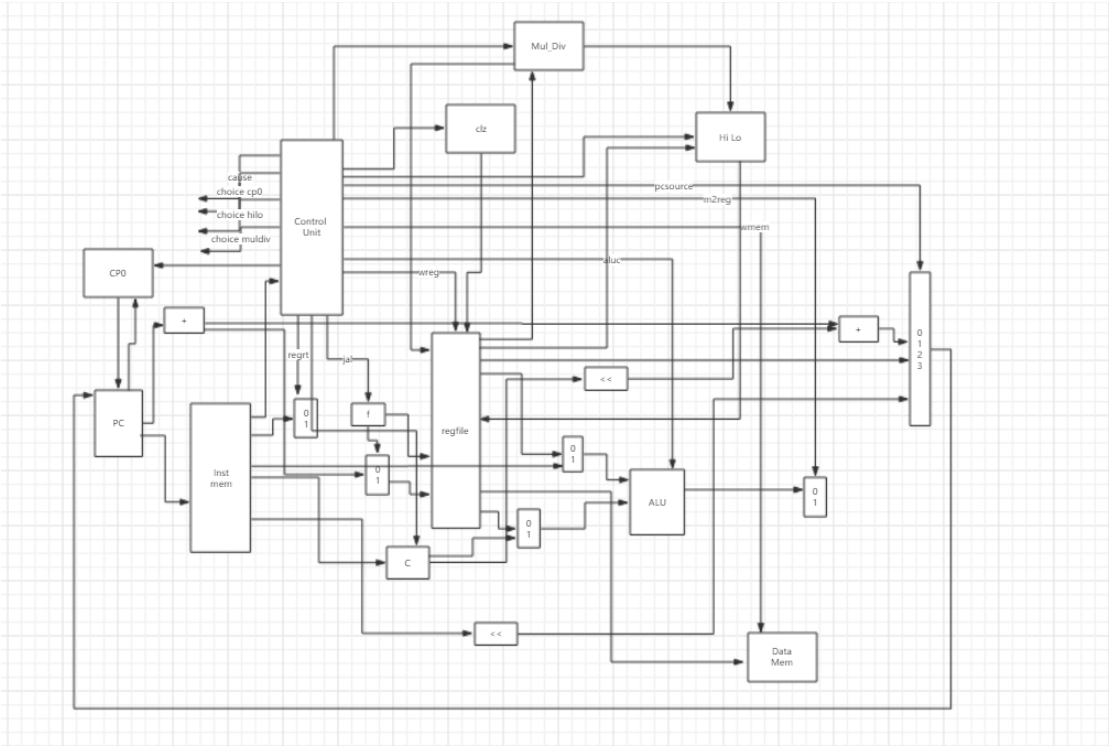
日　　期 2018/6/4

一、实验内容

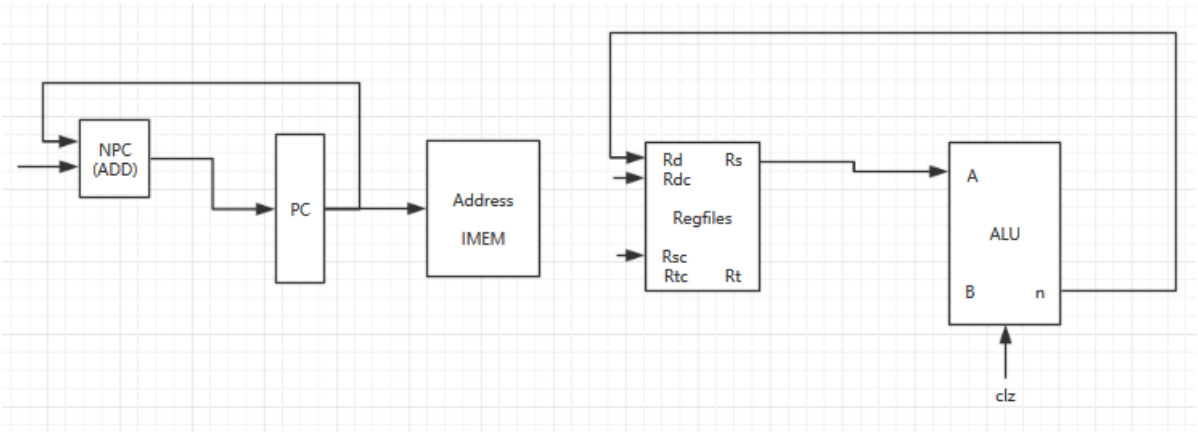
实验内容
1 深入了解 CPU 的原理。
2 画出实现 54 条指令的 CPU 的通路图。
3 学习使用 Verilog HDL 语言设计实现 54 条指令的 CPU。

二、数据通路图

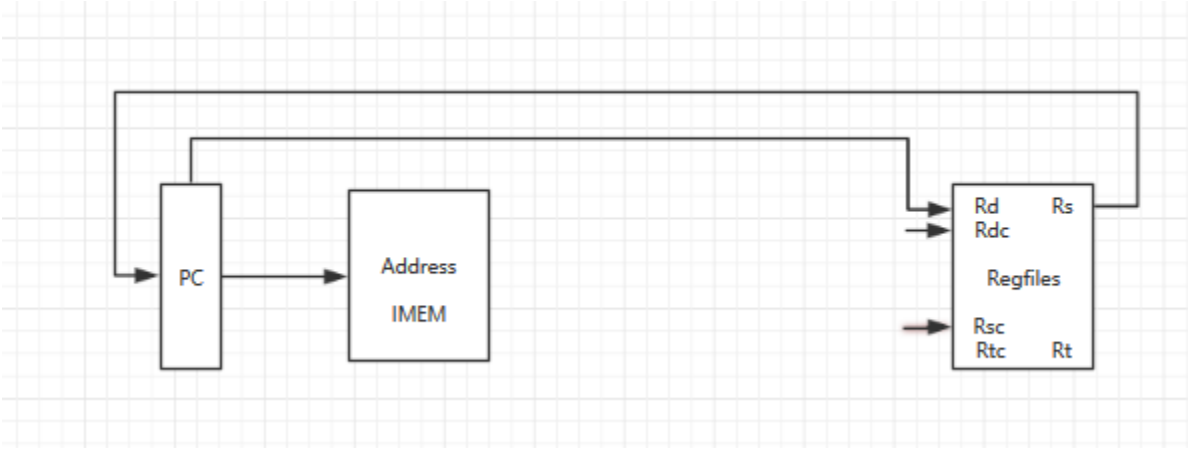
五十四条指令数据通路图



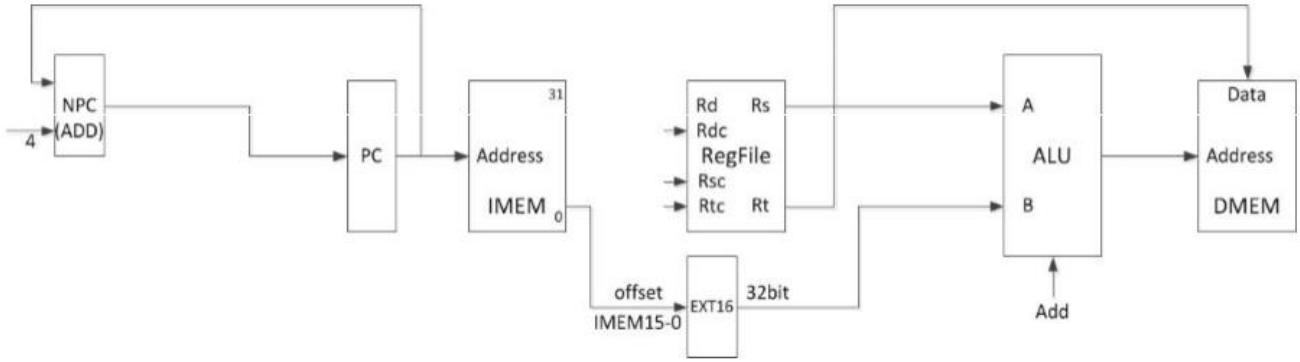
1.CLZ 指令实现



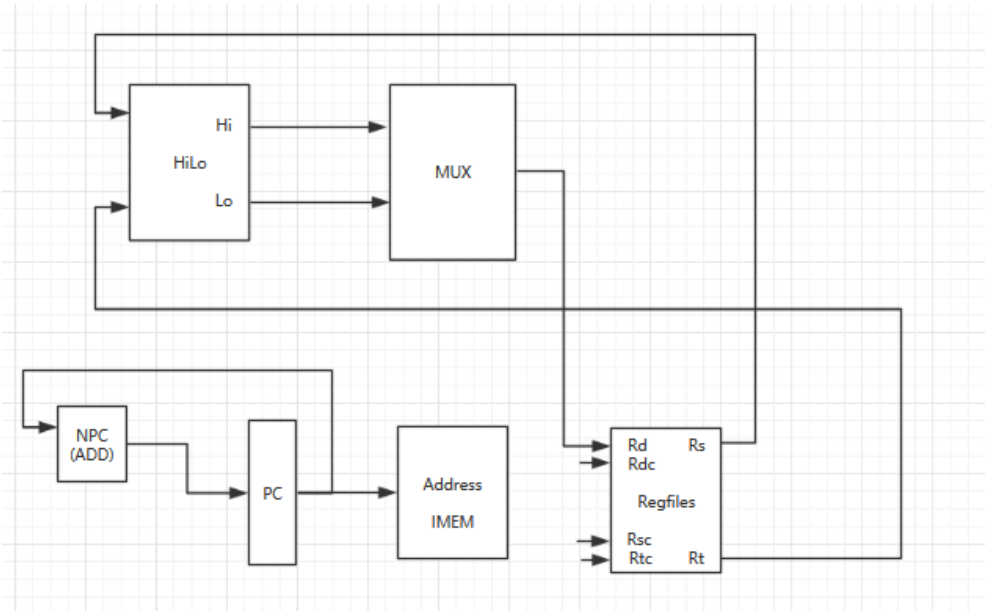
2. JALR 指令实现



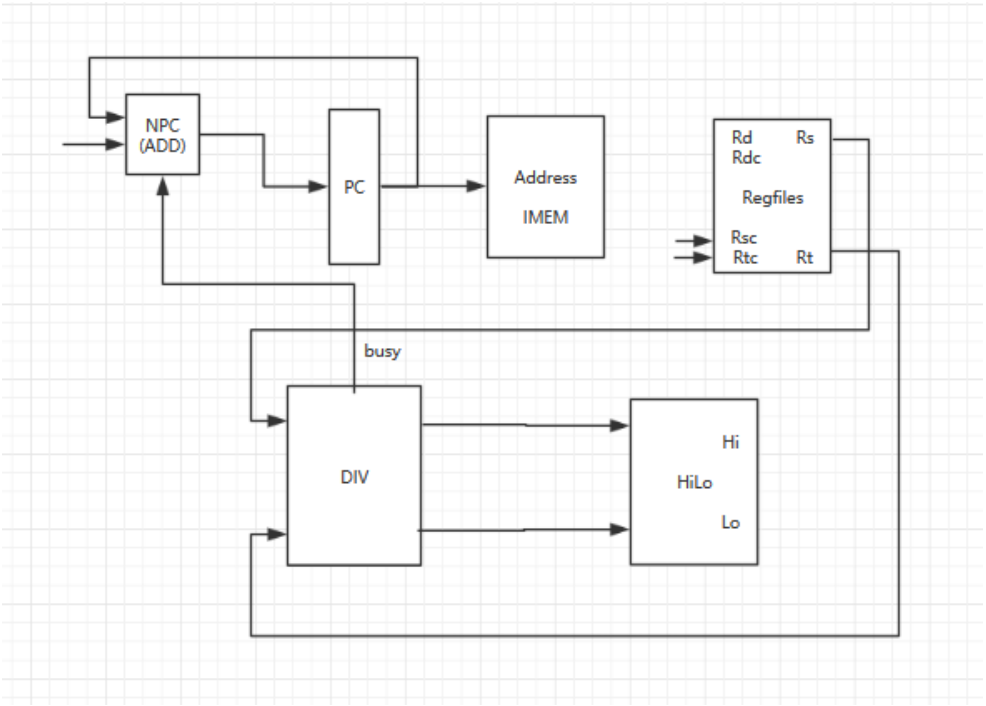
3. LB，LBU，LH，LHU，SB，SH 指令的实现



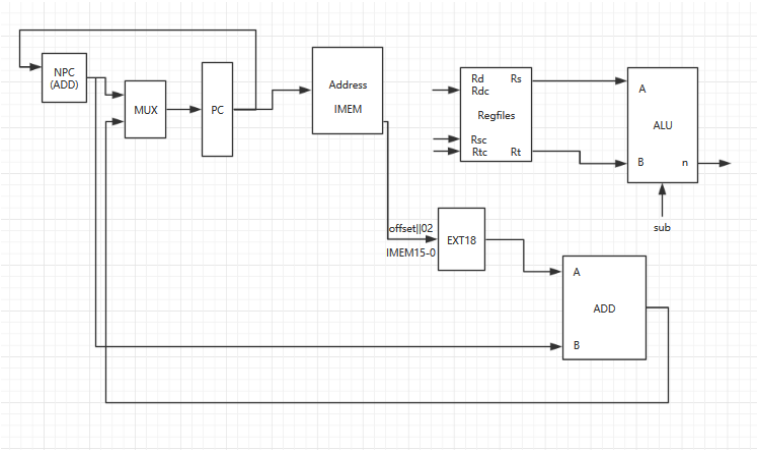
4. MFLO，MTLO，MFHI，MTHI 指令的实现



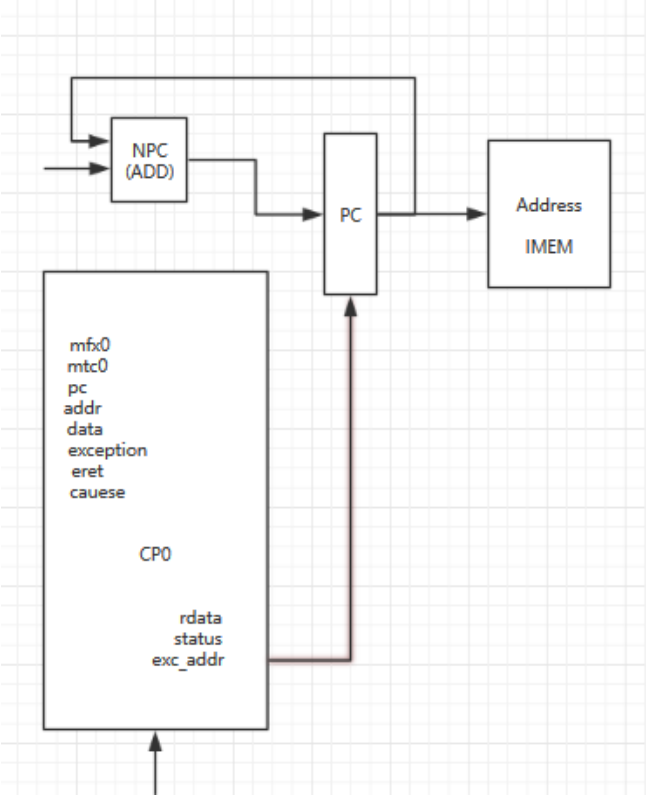
5.DIVU、DIV、MUL、MULTU 指令的实现



6．begz 指令的实现



7. Break teq syscall eret 指令的实现



控制信号

		op	func	wreg	regrt	jal	m2reg	shift
		操作数	功能指令	写寄存器	目的寄存器	子程序调用	存储器数据	ALUa使用
add	x	000000	100000	1	0	0	0	0
addu	**x**	**000000**	**100001**	1	0	0	0	0
sub	x	000000	100010	1	0	0	0	0
subu	**x**	**000000**	**100011**	1	0	0	0	0
and	x	000000	100100	1	0	0	0	0
or	**x**	000000	100101	1	0	0	0	0
xor	x	000000	100110	1	0	0	0	0
nor	**x**	**000000**	**100111**	**1**	**0**	**0**	**0**	**0**
slt	**x**	**000000**	**101010**	**1**	**0**	**0**	**0**	**0**
sltu	**x**	**000000**	**101011**	**1**	**0**	**0**	**0**	**0**
sll	x	000000	000000	1	0	0	0	1
srl	x	000000	000010	1	0	0	0	1
sra	x	000000	000011	1	0	0	0	1
sllv	**x**	**000000**	**000100**	**1**	**0**	**0**	**0**	**0**
srlv	**x**	**000000**	**000110**	**1**	**0**	**0**	**0**	**0**
srav	**x**	**000000**	**000111**	**1**	**0**	**0**	**0**	**0**
jr	**x**	000000	001000	0	x	x	x	x
addi	x	001000	NULL	1	1	0	0	0
addiu	**x**	**001001**	**NULL**	1	1	0	0	0
andi	x	001100	NULL	1	1	0	0	0
ori	x	001101	NULL	1	1	0	0	0
xori	x	001110	NULL	1	1	0	0	0
lw	x	100011	NULL	1	1	0	1	0
sw	x	101011	NULL	0	x	x	x	0
beq	0	000100	NULL	0	x	x	x	0
beq	1	000100	NULL	0	x	x	x	0
bne	0	000101	NULL	0	x	x	x	0
bne	1	000101	NULL	0	x	x	x	0
slti	**x**	**001010**	**NULL**	**1**	**1**	**0**	**0**	**0**
sltiu	**x**	**001011**	**NULL**	**1**	**1**	**0**	**0**	**0**
lui	x	001111	NULL	1	1	0	0	x
j	x	000010	NULL	0	x	x	x	x
jal	x	000011	NULL	1	x	1	x	x
clz		011100	**100000**	1	0			
jalr		000000	**001001**	1	0	0	0	0
bgez		000001		0	0	0?	0	0
lb		100000		1	1	0	1	0
lbu		100100		1	1	0	1	0
lh		100001		1	1	0	1	0
lhu		100101		1	1	0	1	0
sb		101000						
sh		101001						
mthi		000000	**010001**					
mtlo		000000	**010011**					
mfhi		000000	**010000**	1				
mflo		000000	**010010**	1	0	0	0	0
mtc0		010000	**000000**					
mfc0		010000	**000000**	1	1	0	0	0
eret		010000	**011000**					
teq		000000	**110100**	x	x	x	x	
syscall	systemcall	000000	**001100**	x	x	x	x	x
break	break的值	000000	**001101**	x	x	x	x	x
div		000000	**011010**					
divu		000000	**011011**					
mul		011100	**000010**	1				
multu		000000	**011001**					

三、模块建模

1. sccomp_dataflow

以下是 Verilog HDL 代码：

```
module sccomp_dataflow(
input clk_in,
input reset,
output clk_return,
//input mem_clk,
output [31:0] inst,
output [31:0] pc,
output [31:0] addr,
output [31:0] mem_out
      );
wire [31:0] alu_out;
wire [31:0] data;
wire              wmem;
wire [5:0] choice_mem;
assign addr=alu_out;
//reg    [31:0] pc_show_temp;
wire [31:0] pc_0;
assign pc=pc_0+32'h400000;

assign clk_return=clk_in;


Data_Mem
dmem(
.clk(clk_in),
.d_ram_rena(d_ram_rena),
.d_ram_wena(d_ram_wena),
.DAddr(alu_out),
.DataIn(data),
.choice(choice_mem),
.Data_out(mem_out)
);
```

```
cpu
sccpu(
.clk(clk_in),
.rst(reset),
.inst(inst),
.mem(mem_out),
.pc(pc_0),
.alu(alu_out),
.data(data),
.d_ram_rena(d_ram_rena),
.d_ram_wena(d_ram_wena),
.choice_mem(choice_mem)
);

//    scinstmem
//    scinstmem_inst(
//    .pc(pc_0),
//    .inst(inst)
//    );
imem
imem_inst(pc_0[12:2],inst
);




endmodule
```

2. cpu

以下是 Verilog HDL 代码:

```
`timescale 1ns / 1ps

module cpu(
input clk,
input rst,
input [31:0] inst,
input [31:0] mem,
```

```
input intr,
output inta,
//output wmem,
output [31:0] pc,
output [31:0] alu,
output [31:0] data,
output d_ram_rena,
output d_ram_wena,
output [5:0] choice_mem
      );

parameter EXC_BASE = 32'h00000004; //base

wire [31:0] pc_show;////
wire [31:0] p4,bpc,npc,adr,ra,alua,alub,res,alu_mem;
wire [3:0]    aluc;
wire [4:0]    reg_dest,wn;
wire [1:0]    pcsource;
wire
zero,wmem,wreg,regrt,m2reg,shift,aluimm,jal,sext;//d_ram_wena,d_ram_rena;
wire [31:0] sa = {27'b0, inst[10:6]};
//////////////////cpu54
wire [3:0] choice_md;
wire [4:0] choice_hilo;
//wire [5:0] choice_mem;
wire [3:0] choice_cp0;
wire buzy;
wire [31:0] q;
wire [31:0] r;
wire [ 3:0] cause;
wire [31:0] exc_addr;
wire [31:0] data_rc;
wire i_clz , i_jalr , i_bgez;
wire [31:0] clz_num;

wire              e = sext & inst[15];
wire [15:0] imm = {16{e}};
wire [31:0] immediate = {imm,inst[15:0]};
wire [31:0] offset = {imm[13:0],inst[15:0],2'b00};
wire [31:0] jpc ={p4[31:28],inst[25:0],2'b00};
wire [31:0] Hi_o , Lo_o;
```

```
////////////////////////////////

Mul_32 alu_b (aluimm,data,immediate,alub);
Mul_32 alu_a (shift,ra,sa,alua);
Mul_32 result (m2reg,alu,mem,alu_mem);
Mul_32 link (jal,alu_mem,p4,res);
Mul_5 reg_wn (regrt,inst[15:11],inst[20:16],reg_dest);

assign wn = reg_dest| {5{jal}} ;
wire [31:0] res_all = choice_cp0[0] ? data_rc :
                      (i_clz? clz_num :
                      (i_jalr?    pc + 4 + 32'h400000:
                      (choice_hilo[4] ? Hi_o :
                      (choice_hilo[3] ? Lo_o    :
                      ( choice_md[1] ?    r : res)))));
// wire [31:0] res_all = choice_cp0[0] ? data_rc :
//                       (i_clz? clz_num :
//                       (i_jalr?    pc + 4 + 32'h400000:res));



PC
PC_inst(
.clk(clk),
.rst(rst),
.jal(jal),
.jpc(jpc),
.pcsource(pcsource),
.ra(ra),
.offset(offset),
.cause(cause),
.buzy(buzy),
.i_jalr(i_jalr),
.i_bgez(i_bgez),
.i_eret(choice_cp0[2]),
.inst(inst),
.exc_addr(exc_addr),
.pc(pc),
.p4(p4)
);

Con_Unit
Con_Unit_inst (
.op(inst[31:26]),
.func(inst[5:0]),
```

```
.z(zero),
.wreg(wreg),
.regrt(regrt),
.jal(jal),
.m2reg(m2reg),
.shift(shift),
.aluimm(aluimm),
.sext(sext),
.wmem(wmem),
.aluc(aluc),
.pcsource(pcsource),
.d_ram_wena(d_ram_wena),
.d_ram_rena(d_ram_rena),
.rd(inst[25:21]),
.cause(cause),
.i_clz(i_clz),
.i_jalr(i_jalr),
.i_bgez(i_bgez),
.choice_md(choice_md),
.choice_hilo(choice_hilo),
.choice_mem(choice_mem),
.choice_cp0(choice_cp0)
);

ALU
alu_unit(
.a(alua),
.b(alub),
.aluc(aluc),
.r(alu),
.zero(zero)
// .carry(carry),
// .negative(negative),
// .overflow(overflow)
);

Clz
Clz_inst(
.clk(clk),
.data(ra),
.num(clz_num)
);
```

```
CP0
CP0_inst(
.clk(clk),
.rst(rst),
.choice(choice_cp0),
.pc(pc),
.cause(cause),
.addr(inst[15:11]),
.wdata(data),
.rdata(data_rc),
.exc_addr(exc_addr)
);



Hi_Lo
Hi_Lo_inst(
.clk(clk),
.rst(rst),
.choice(choice_hilo),
.Hi_i(ra),
.Lo_i(ra),
.mul_h(q),
.mul_l(r),
.Hi_o(Hi_o),
.Lo_o(Lo_o)
);

Mul_Div
Mul_Div_inst(
.choice(choice_md),
.a(ra),
.b(data),
.clk(clk),
.rst(rst),
.q(q),
.r(r)
);



regfile
```

```
cpu_ref(
.clk(clk),
.clrn(rst),
.we(wreg),
.rna(inst[25:21]),
.rnb(inst[20:16]),
.wn(wn),
.d(res_all),
.qa(ra),
.qb(data)
);

// wire carry,negative,overflow;

endmodule



//////////////////////////////////pc module
// reg [31:0]pc_show_temp=0;
// reg [31:0]p4_temp=0;
// reg [31:0]adr_temp=0;
// reg [31:0]pc_temp=0;
// reg [31:0] jpc_temp=0;
// always @(posedge clk)
// begin
//     if (rst==1)begin
//         pc_temp=0;
//         pc_show_temp=pc_temp+32'h400000;
//     end
//     else
//     begin
//         jpc_temp = jpc-32'h400000;
//         p4_temp = pc_temp +4;
//             // p4_temp = (choice_cp0[3] | choice_cp0[1] | choice_cp0[0]) ?
EXC_BASE : //cp0
//         //                  ( buzy ? p4_temp :
//         //                  ( i_jalr? ra :
//          //                     ((i_bgez & ra[31]==1) ? {2'b1 , inst[15:0] , 2'b00} :
pc_temp+4)));// div mul
//         adr_temp = p4_temp+offset;
//         pc_show_temp=pc_temp+32'h400000;
//         case (pcsource)
//             2'b00: pc_temp=p4_temp;
```

```
//             2'b01: pc_temp=adr_temp;
//             2'b10: pc_temp=ra-32'h400000;
//             2'b11: pc_temp=jpc_temp;
//             default: pc_temp=0;
//         endcase
//     end
//         // if (pc==32'h100)
//         //     pc_temp=pc_temp;
// end

// assign p4=jal?p4_temp+32'h400004:p4_temp;
// assign adr=adr_temp;
// assign pc=pc_temp;
// assign pc_show=pc_temp+32'h400000;
//////////////////////////////////////////////////////
```

3. Data_Mem

以下是 Verilog HDL 代码:

```
`timescale 1ns / 1ps
module Data_Mem(
input clk,
input d_ram_rena,//Êý¾Ý´æ´¢Æ÷¶ÁÊ¹ÄÜ,1¿É¶Á?è¯?
input d_ram_wena,//Êý¾Ý´æ´¢Æ÷Ð´Ê¹ÄÜ£¬1¿ÉÐ´
input [31:0] DAddr,
input [31:0] DataIn,
input [5:0] choice,
output [31:0] Data_out
    );
reg [31:0] data_out=0;
reg [31:0] memory [0:800];
reg [31:0] DAddr_temp;
reg [31:0] load_word;
parameter LB   = 6'b100000,
          LBU = 6'b010000,
          LH   = 6'b001000,
          LHU = 6'b000100,
          SB   = 6'b000010,
          SH   = 6'b000001;
integer i;
initial
```

```
begin
   for (i=0;i<800;i=i+1)
         memory[i]<=0;
end

always @(negedge clk)
begin
      DAddr_temp=DAddr-32'h10010000;
   if(d_ram_rena==1 && d_ram_wena==1 )
      data_out<=data_out;
   else if (d_ram_wena)
      memory[DAddr_temp] <= DataIn;
   // else if (d_ram_rena)
   //     data_out = memory[DAddr_temp];
   else begin
      load_word = memory[DAddr_temp];
      case (choice)
         LB : data_out = {{24{load_word[7]}} , load_word[7:0]};
         LBU: data_out = { 24'b0                    , load_word[7:0]};
         LH : data_out = {{16{load_word[15]}}, load_word[15:0]};
         LHU: data_out = { 16'b0                    , load_word[15:0]};
         SB : memory[DAddr_temp] = {24'b0 , DataIn[7:0] };
         SH : memory[DAddr_temp] = {16'b0 , DataIn[15:0]};
         default:
            ;
      endcase
   end
end

assign       Data_out = d_ram_rena ? memory[DAddr_temp] :
                        (choice == LB)?   {{24{ memory[DAddr_temp][7]}} ,
memory[DAddr_temp][7:0]} :
                        ((choice  ==  LBU)?     {  24'b0                    ,
memory[DAddr_temp][7:0]} :
                        ((choice                    ==                    LH)?
{{16{ memory[DAddr_temp][15]}}, memory[DAddr_temp][15:0]}    :
                        (  (choice  ==  LHU)?  {  16'b0                    ,
memory[DAddr_temp][15:0]}    : 0 )));

endmodule
```

4. scinstmem

模块 2

以下是 Verilog HDL 代码:

```
module scinstmem(pc,inst);
	input [31:0] pc;	//æŒ‡ä»¤åœ°å€
	output [31:0] inst; //è¾“å‡ºæŒ‡ä»¤

	reg [31:0] ram [0:1000]; //æŒ‡ä»¤å-˜å¨™?

	assign inst = ram[pc[31:2]];

	initial
	begin
		$readmemh("C:\\Users\\Ordinary\\Desktop\\test\\cp0.hex.txt",	ram);
//è¯»å–æµ‹è¯•æ–‡ä¡£-çš„æŒ‡ä»¤
	end

endmodule
```

5. ALU

以下是 Verilog HDL 代码:

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

```
reg signed [31:0] a_sign;
reg signed [31:0] b_sign;
reg signed [31:0] r_sign;

always@(*)begin
a_sign = $signed(a);
b_sign = $signed(b);
case(aluc)
4'b0000:begin    //addu
            r=a+b;
            carry = (r<a || r<b)?1:0;
      end
4'b0010:begin     //add
            r_sign=a_sign+b_sign;
            if(a_sign>0 && b_sign>0)
                  overflow = (r_sign<0)? 1:0;
            else if(a_sign<0 && b_sign<0)
                  overflow = (r_sign>0)? 1:0;
            else
                  overflow = 0 ;
            r=$unsigned(r_sign);
      end
4'b0001:begin    //subu
            r=a-b;
            carry=(r>a)? 1: 0;
      end
4'b0011:begin     //sub
            r_sign=a_sign - b_sign;
            if(a_sign>0 && b_sign<0)
                  overflow = (r_sign<0)? 1:0;
            else if(a_sign<0 && b_sign>0)
                  overflow = (r_sign>0)? 1:0;
            else
                  overflow = 0 ;
            r=$unsigned(r_sign);
      end
4'b0100:begin     //and
            r= a&b;
      end
4'b0101:begin     //or
            r= a|b;
      end
4'b0110:begin     //xor
            r = a^b;
```

```
        end
    4'b0111:begin      //nor
                r=~(a|b);
        end
    4'b1000,4'b1001:begin      //lui
                r={b[15:0],16'b0};
        end
    4'b1011:begin      //slt
                r= (a_sign<b_sign)?1:0;
        end
    4'b1010:begin      //sltu
                r = (a<b)?1:0;
        end
    4'b1100:begin      //sta
                if(a_sign!=0)begin
                   r_sign = b_sign>>>(a_sign-1);
                   carry = r_sign[0];
                   r_sign = r_sign>>>1;
                end
                else
                      r_sign=b_sign;
                r = $unsigned(r_sign);
        end
    4'b1110,4'b1111:begin      //sll/slr
                if(a != 0)begin
                      r = b<<(a-1);
                      carry = r[31];
                      r = r<<1;
                end
                else
                      r = b;
        end
    4'b1101:begin      //srl
              if(a != 0)
              begin
                    r = b>>(a-1);
                    carry = r[0];
                    r = r>>1;
              end
              else
                    r = b;
                end
    endcase
    if(aluc == 4'b1010 || aluc == 4'b1011)
```

```
        zero = (a == b)? 1:0;
    else
        zero = (r == 0)? 1:0;
    negative = r[31];
    end
    endmodule
```

6. Con_Unit

以下是 Verilog HDL 代码:

```
    `timescale 1ns / 1ps

    module Con_Unit(
    //cpu31
    input [5:0] op,
    input [5:0] func,
    input z,
    output wreg,
    output regrt,
    output jal,
    output m2reg,
    output shift,
    output aluimm,
    output sext,
    output wmem,
    output [3:0] aluc,
    output [1:0] pcsource,
    output d_ram_wena,
    output d_ram_rena,
    //cp0
    input [4:0] rd,
    output [4:0] cause,
    //others
    output i_clz,
    output i_jalr,
    output i_bgez,
    //choice
    output [3:0] choice_md,
    output [4:0] choice_hilo,
    output [5:0] choice_mem,
    output [3:0] choice_cp0
```

```
        );

wire r_type=~op[5] & ~op[4] &~op[3] &~op[2] &~op[1] &~op[0];
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_jalr= r_type & ~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];

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

wire i_mfhi = r_type & ~func[5] &   func[4] & ~func[3] & ~func[2] & ~func[1]
& ~func[0];
wire i_mflo = r_type & ~func[5] &   func[4] & ~func[3] & ~func[2] &   func[1]
& ~func[0];
wire i_mthi = r_type & ~func[5] &   func[4] & ~func[3] & ~func[2] & ~func[1]
&   func[0];
wire i_mtlo = r_type & ~func[5] &   func[4] & ~func[3] & ~func[2] & func[1]
&   func[0];

wire 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];
wire i_multu= r_type & ~func[5] &   func[4] &   func[3] & ~func[2] & ~func[1]
&   func[0];
wire i_div= ~func[5] & func[4] & func[3] & ~func[2] & func[1] & ~func[0];
wire i_divu= r_type & ~func[5] & func[4] & func[3] & ~func[2] & func[1] &
func[0];

wire i_syscall=r_type& ~func[5] &   ~func[4]&   func[3] &   func[2] &
~func[1] & ~func[0];
wire i_teq=     r_type &   func[5] &   func[4] & ~func[3] &   func[2] & ~func[1]
& ~func[0];
wire i_break=   r_type& ~func[5] &   ~func[4]&   func[3] &   func[2] &
~func[1] &   func[0];
wire i_mtc0 = ~op[5] & op[4] & ~op[3] & ~op[2] & ~op[1] & ~op[0]
                        & ~func[5] & ~func[4] & ~func[3] & ~func[2] &
```

```
~func[1] & ~func[0] &    rd[2];
wire i_mfc0= ~op[5] & op[4] & ~op[3] & ~op[2] & ~op[1] & ~op[0]
                              & ~func[5] & ~func[4] & ~func[3] & ~func[2] &
~func[1] & ~func[0] & ~rd[2];
wire i_eret= ~op[5] & op[4] & ~op[3] & ~op[2] & ~op[1] & ~op[0]
                & ~func[5] & func[4] & func[3] & ~func[2] & ~func[1] &
~func[0];



//choice
assign choice_hilo = { i_mfhi, i_mflo , i_mthi, i_mtlo , (i_div | i_divu | i_multu)};
assign choice_md     = { i_div , i_divu , i_mul , i_multu};
assign choice_mem    = { i_lb , i_lbu , i_lh , i_lhu , i_sb , i_sh};
assign choice_cp0    = { i_teq , i_eret , i_mtc0 , i_mfc0};

parameter      SYSCALL = 4'b1000,
                BREAK     = 4'b1001,
                TEQ         = 4'b1101;
assign cause = i_syscall ? SYSCALL :
                        ( i_break ? BREAK :
                        ( i_teq ? TEQ : 4'b0000 ));
////////////////////////////////////////////////////////////////////////////////



wire i_add=r_type & func[5] & ~func[4] & ~func[3] &~ func[2] & ~func[1] &
~func[0];
wire i_addu=r_type & func[5] & ~func[4] & ~func[3] &~ func[2] & ~func[1] &
func[0];
wire i_sub=r_type & func[5] & ~func[4] &~ func[3] & ~func[2] & func[1] &
~func[0];
wire i_subu=r_type & func[5] & ~func[4] & ~func[3] &~ func[2] & func[1] &
func[0];
wire i_and=r_type & func[5] & ~func[4] & ~func[3] & func[2] & ~func[1] &
~func[0];
wire i_or= r_type & func[5] & ~func[4] & ~func[3] & func[2] & ~func[1] &
func[0];
wire i_xor=r_type & func[5] & ~func[4] & ~func[3] & func[2] & func[1] &
~func[0];
wire i_nor= r_type & func[5] & ~func[4] & ~func[3] & func[2] & func[1] &
func[0];
wire i_slt= r_type & func[5] & ~func[4] & func[3] & ~func[2] & func[1] &
~func[0];
wire i_sltu= r_type & func[5] & ~func[4] & func[3] & ~func[2] & func[1] &
```

```
func[0];
wire i_sll= r_type & ~func[5] & ~func[4] & ~func[3] & ~func[2] & ~func[1] &
~func[0];
wire i_srl= r_type & ~func[5] & ~func[4] & ~func[3] & ~func[2] & func[1] &
~func[0];
wire i_sra= r_type & ~func[5] & ~func[4] & ~func[3] & ~func[2] & func[1] &
func[0];
wire i_sllv= r_type & ~func[5] & ~func[4] & ~func[3] & func[2] & ~func[1] &
~func[0];
wire i_srlv= r_type & ~func[5] & ~func[4] & ~func[3] & func[2] & func[1] &
~func[0];
wire i_srav= r_type & ~func[5] & ~func[4] & ~func[3] & func[2] & func[1] &
func[0];
wire i_jr= r_type & ~func[5] & ~func[4] & func[3] & ~func[2] & ~func[1] &
~func[0];

wire i_addi= ~op[5] & ~op[4] & op[3] & ~op[2] & ~op[1] & ~op[0];
wire i_addiu= ~op[5] & ~op[4] & op[3] & ~op[2] & ~op[1] & op[0];
wire i_andi= ~op[5] & ~op[4] & op[3] & op[2] & ~op[1] & ~op[0];
wire i_ori= ~op[5] & ~op[4] & op[3] & op[2] & ~op[1] & op[0];
wire i_xori= ~op[5] & ~op[4] & op[3] & op[2] & op[1] & ~op[0];
wire i_lw= op[5] & ~op[4] & ~op[3] & ~op[2] & op[1] & op[0];
wire i_sw= op[5] & ~op[4] & op[3] & ~op[2] & op[1] & op[0];
wire i_beq= ~op[5] & ~op[4] & ~op[3] & op[2] & ~op[1] & ~op[0];
wire i_bne= ~op[5] & ~op[4] & ~op[3] & op[2] & ~op[1] & op[0];
wire i_slti= ~op[5] & ~op[4] & op[3] & ~op[2] & op[1] & ~op[0];
wire i_sltiu= ~op[5] & ~op[4] & op[3] & ~op[2] & op[1] & op[0];
wire i_lui= ~op[5] & ~op[4] & op[3] & op[2] & op[1] & op[0];
wire i_j= ~op[5] & ~op[4] & ~op[3] & ~op[2] & op[1] & ~op[0];
wire i_jal= ~op[5] & ~op[4] & ~op[3] & ~op[2] & op[1] & op[0];

assign wreg            = i_add | i_sub | i_and | i_xor | i_sll | i_srl | i_sra | i_or |
                         i_addi | i_andi | i_ori | i_xori | i_lw | i_lui | i_jal |
                         i_addu | i_subu | i_nor | i_slt | i_sltu | i_sllv | i_srlv |
                         i_srav | i_addiu | i_slti | i_sltiu |
                         i_mfc0 | i_mul | i_clz | i_jalr | i_lb | i_lbu | i_lh | i_lhu |
i_mfhi | i_mflo;
assign  regrt          = i_addi | i_andi | i_ori | i_xori | i_lw | i_lui | //i_jal |
è¿™ä¸ªéœ?è¦åŠ ä¸Šä¹ˆï¼Ÿï¼Ÿï¼?
                         i_addiu | i_slti | i_sltiu |
                         i_mfc0 | i_lb | i_lbu | i_lh | i_lhu ;
assign jal             = i_jal;
assign m2reg           = i_lw |
                         i_lb | i_lbu | i_lh | i_lhu ;
```

```
assign shift          = i_sll | i_srl | i_sra ;
assign aluimm         = i_addi | i_andi | i_ori | i_xori | i_lw | i_lui | i_sw |
                        i_addiu | i_slti | i_sltiu |
                        i_lb | i_lbu | i_lh | i_lhu | i_sb | i_sh;
assign sext           = i_addi | i_lw | i_sw | i_beq | i_bne |
                        i_slti | i_addiu;
assign aluc[3]        = i_slt | i_sltu | i_sll | i_srl | i_sra | i_sllv | i_srlv | i_srav | i_slti |
i_sltiu | i_lui;
assign aluc[2]        = i_and | i_or | i_xor | i_nor | i_sll | i_srl | i_sra |i_sllv | i_srlv |
i_srav | i_andi | i_ori | i_xori ;
assign aluc[1]        = i_add | i_sub | i_xor | i_nor | i_slt | i_sltu | i_sll | i_sllv | i_addi
| i_xori | i_lw | i_sw | i_beq | i_bne | i_slti | i_sltiu ;
assign aluc[0]        = i_sub | i_subu | i_or | i_nor | i_slt | i_srl | i_srlv | i_ori | i_beq
| i_bne | i_slti;
assign wmem           = i_sw ;
assign pcsource[1]= i_jr | i_j | i_jal ;
assign pcsource[0]= i_beq&z | i_bne&~z | i_j | i_jal;
assign d_ram_wena = i_sw;
assign d_ram_rena = i_lw;
//////////////////////////////////////////////////////////////////////

endmodule


// wire i_unimplemented = ~(i_mfc0 | i_mtc0 | i_eret | i_syscall |
//                          i_add | i_sub | i_and | i_or | i_xor | i_sll | i_srl |
//                          i_sra | i_jr | i_addi | i_andi | i_ori | i_xori | i_lw |
//                          i_sw | i_beq | i_bne | i_lui | i_j | i_jal);



//?????????? 4? ???? ???? ????? ??

// wire overflow = ov & (i_add | i_sub | i_addi);
// wire int_int    = sta[0] & intr;
// wire exc_sys    = sta[1] & i_syscall;//??
// wire exc_uni    = sta[2] & unimplemented_inst;//??
// wire exc_ovr    = sta[3] & overflow;
// assign exc       = int_int | exc_ovr | exc_sys | exc_uni;
// assign inta      = int_int;
// // ExcCode 00 ???? 01 ????    10 ?????? 11 ??
// wire ExcCode0 = i_syscall | overflow;      //??
// wire ExcCode1 = unimplemented_inst | overflow ;//??
// assign cause    = { 28'h0 , ExcCode1 , ExcCode0 , 2'b00 };
```

```
// //??3??????????
// assign mtc0 = i_mtc0;//??
// assign wsta = exc | mtc0 & rd_is_status | i_eret;
// assign wcau = exc | mtc0 & rd_is_cause;
// assign wepc = exc | mtc0 & rd_is_epc;

// wire rd_is_status = {rd == 5'd12};//cp0 status register
// wire rd_is_cause  = {rd == 5'd13};//cp0 cause register
// wire rd_is_epc    = {rd == 5'd14};//cp0 epc register
// assign mfc0[0]    = i_mfc0 & rd_is_status | i_mfc0 & rd_is_epc;
// assign mfc0[1]    = i_mfc0 & rd_is_cause  | i_mfc0 & rd_is_epc;

// //00:??? 01:EPC 10:????????????
// assign selpc[0] = i_eret;
// assign selpc[1] = exc;
```

7. Mul_Div

以下是 Verilog HDL 代码:

```
`timescale 1ns / 1ps


module Mul_Div(
input [3:0] choice,//1000:div 0100:divu 0010:mul 0001:multu
input [31:0] a,
input [31:0] b,
input clk,
input rst,
output buzy,
output [31:0] q, //shang
output [31:0] r   //yushu
    );
parameter DIV   = 4'b1000,
          DIVU = 4'b0100,
          MUL   = 4'b0010,
          MULTU= 4'b0001;


//div
reg r_sign = 0;
reg a_sign = 0;
reg b_sign = 0;
```

```
reg [5:0]    count = 0;
reg [63:0] a_div , b_div;
wire [31:0] q_div;
wire [31:0] r_div;
reg bz_div = 0;
wire start_div = ~bz_div && (choice == DIV);

assign r_div= a_sign ?
                (b_sign ? ~a_div[63:32]+1 : ~a_div[63:32]+1) :
                (b_sign ? a_div[63:32] : a_div[63:32]) ;

assign q_div= a_sign ?
                (b_sign ? a_div[31:0] : ~a_div[31:0]+1) :
                (b_sign ? ~a_div[31:0]+1 : a_div[31:0]) ;
//divu

reg r_sign_u = 0;
reg [31:0] reg_q;
reg [31:0] reg_r;
reg [31:0] reg_b;
wire [32:0] sub_add;
assign sub_add = r_sign_u?({reg_r,q[31]} + reg_b):({reg_r,q[31]} - reg_b);

reg bz_divu = 0;
wire start_divu = ~bz_divu & (choice == DIVU);
wire [31:0] r_divu;
wire [31:0] q_divu;
assign r_divu = r_sign_u?( reg_r + reg_b ): reg_r;
assign q_divu = reg_q;

//mul
reg [63:0] result = 0;
reg [31:0] x = 0;
reg [31:0] y = 0;
reg flag = 0;
reg [31:0] high = 0;
wire [31:0] r_mul;
wire [31:0] q_mul;
assign r_mul = result[31:0];//mul ouput low 32 bit
assign q_mul = result[63:32];

//multu
wire [31:0] r_multu;
wire [31:0] q_multu;
```

```
assign r_multu = result[31:0];
assign q_multu = result[63:32];


//////////////////////////////////////



always @(*)begin
case (choice)
DIV:begin
      if(rst)begin
         count = 0;
         bz_div = 0;
      end
      else begin
         if( start_div )begin
            r_sign = 0;
            a_sign = a[31];
            b_sign = b[31];
            a_div = a_sign ? {32'b0,~a+1} : {32'b0,a}; //è¢«é™¤æ•°å–
ç»å¯¹å€¼æ‰©å±?
            b_div = b_sign ? {~b+1,32'b0} : {b,32'b0}; //é™¤æ•°å–
ç»å¯¹å?¼æ‰©å±?
            count = 6'b0;                          //è®¡æ•°å™¨ç½®é›?
            bz_div = 1;                            //å¼?å§‹æ‰§è¡Œé™¤æ³•æŒ‡ä»?
         end
         else if ( bz_div )begin
            count = count + 1;                  //è®¡æ•°å™¨åŠ ä¸?
            a_div = {a_div[62:0],1'b0};      //å·¦ç§»
            if( a_div >= b_div )
                  a_div = a_div - b_div + 1; //å¤Ÿå‡åˆ™æ‰§è¡Œæ³?,å•†ä¸Š1
            else
                  a_div = a_div;                //ä¸å¤Ÿå‡åˆ™ä¸å˜,å•†ä¸Š0
            if( count == 6'h20 )
                  bz_div = 0;                         //ç»“æŸæ‰§è¡Œé™¤æ³•æŒ‡ä»¤
         end
      end
end

DIVU:begin
      if(rst)begin
                  count<=0;
                  reg_q<=0;
```

```
                reg_r<=0;
                reg_b<=0;
                bz_divu<=0;
            end
        else begin
            if(start_divu)begin
                reg_q <= a; // dividend;
                reg_b <= b; //divisor;
                reg_r <= 0;
                count <= 0;
                bz_divu <= 1;
                r_sign_u<=0;
            end
            else if(bz_divu)begin
                reg_r <= sub_add[31:0];
                r_sign_u <= sub_add[32];
                reg_q <= {reg_q[30:0],~sub_add[32]};
                count <= count +1;
                if (count == 31)
                    bz_divu <= 0;
            end
        end
end

MUL:begin
        if (rst)begin
            result=0;
            x = 0;
            y = 0;
            high = 0;
            flag = 0;
        end
        else begin
            x = a;
            y = b;
            high = 0;
            result = 0;
            flag = (~x[31] & y[31]) | (x[31] & ~y[31]);
            if (x[31] == 1)begin
                x = x - 1;
                x = ~x;
            end
            if (y[31] == 1)begin
                y = y - 1;
```

```
                y = ~y;
            end
            while(y != 0)begin
                if(y[0]==1)
                    result = result + {high,x};
                high = high << 1;
                high[0] = x[31];
                x = x << 1;
                y = y >> 1;
            end
            if (flag == 1)begin
                result = ~result;
                result = result + 1;
                flag = 0;
             end
        end
end

MULTU:begin
      if (rst)begin
            result=0;
            x=0;
            y=0;
            high=0;
      end
      else begin
            x=a;
            y=b;
            high=0;
            result=0;
            while(y!=0) begin
                if(y[0]==1)
                result=result+{high,x};
                high=high<<1;
                high[0]=x[31];
                x=x<<1;
                y=y>>1;
            end
      end
end
   default:
      ;
endcase
end
```

```
assign q = choice[3] ? q_div    :
               ( choice[2] ? q_divu : (
                  choice[1] ? q_mul : (
                  choice[0] ? q_multu: 0
                  ))) ;

assign r = choice[3] ? r_div    :
               ( choice[2] ? r_divu : (
                  choice[1] ? result[31:0]    : (
                  choice[0] ? r_multu: 0
                  ))) ;

assign buzy = choice[3] ? bz_div    :
                   ( choice[2] ? bz_divu : 0) ;


endmodule
```

8. PC

以下是 Verilog HDL 代码:

```
                                `timescale 1ns / 1ps
module PC(
input clk,
input rst,
input jal,
input [31:0] jpc,
input [1:0] pcsource,
input [31:0] ra,
input [31:0] offset,
input [3:0] cause,
input buzy,
input i_jalr,
input i_bgez,
input i_eret,
input [31:0] inst,
input [31:0] exc_addr,
output [31:0] pc,
output [31:0] p4
      );
```

```
reg [31:0]pc_show_temp=0;
reg [31:0]p4_temp=0;
reg [31:0]adr_temp=0;
reg [31:0]pc_temp=0;
reg [31:0] jpc_temp=0;
parameter EXC_BASE = 32'h00000004; //base
parameter      SYSCALL = 4'b1000,
                  BREAK      = 4'b1001,
                  TEQ         = 4'b1101;

always @(posedge clk or posedge rst)
begin
   if (rst==1)begin
      pc_temp=0;
      pc_show_temp=pc_temp+32'h400000;
   end
   else
   begin
      jpc_temp = jpc-32'h400000;
      if ((cause == SYSCALL | cause == BREAK | cause == TEQ) == 1)
         p4_temp = EXC_BASE;
      else if (i_eret)
         p4_temp = exc_addr;
      else if ( buzy )
         p4_temp = p4_temp;
      else if (i_jalr)
         p4_temp = ra -32'h400000;
      else if (i_bgez & ra[31]==0)
         p4_temp = pc_temp + 4 + {2'b00 , inst[15:0] , 2'b00};
      else
         p4_temp = pc_temp + 4;
      adr_temp = p4_temp+offset;
      pc_show_temp=pc_temp+32'h400000;
      case (pcsource)
         2'b00: pc_temp=p4_temp;
         2'b01: pc_temp=adr_temp;
         2'b10: pc_temp=ra-32'h400000;
         2'b11: pc_temp=jpc_temp;
         default: pc_temp=0;
      endcase
   end
      if (pc_temp==32'h00000b54)
         pc_temp=pc_temp;
end
```

```
assign p4=jal?p4_temp+32'h400004:p4_temp;
// assign adr=adr_temp;
assign pc=pc_temp;
// assign pc_show=pc_temp+32'h400000;

endmodule
```

9. regfile

以下是 Verilog HDL 代码:

```
`timescale 1ns / 1ps

module regfile(rna,rnb,d,wn,we,clk,clrn,qa,qb);
    input clk;                  //时钟信号
    input clrn;                 //清零信号
    input we;                   //写使能信号
    input [4:0] rna, rnb; //读端口 a,b 的寄存器地址
    input [4:0] wn;             //写端口的寄存器地址
    input [31:0] d;             //写端口的 32 位数据
    output [31:0] qa,qb;    //读端口 a,b 的 32 为数据

    reg [31:0] array_reg [0:31]; //31 * 32-bit regs

    //读寄存器
    assign qa = (rna == 0) ? 0 : array_reg[rna];
    assign qb = (rnb == 0) ? 0 : array_reg[rnb];

    integer i;
    initial
    begin
        for(i = 0; i < 32; i = i + 1)
            array_reg[i] <= 0;
    end



    //写寄存器  //下降沿写入？？？
    always @(negedge clk or negedge clrn)
    begin
        if(clrn == 1)
        begin
```

```
                for(i = 0; i < 32; i = i + 1)
                    array_reg[i] <= 0;
            end
            else if((wn != 0) && we)
                array_reg[wn] = d;
        end
    endmodule
```

10. Hi_Lo

以下是 Verilog HDL 代码:

```
`timescale 1ns / 1ps


module Hi_Lo(
input clk,
input rst,
input [4:0] choice,
input [31:0] Hi_i,
input [31:0] Lo_i,
input [31:0] mul_h,
input [31:0] mul_l,
output [31:0] Hi_o,
output [31:0] Lo_o
    );
reg [31:0] H;
reg [31:0] L;


parameter MFHI = 5'b10000,
              MFLO = 5'b01000,
              MTHI = 5'b00100,
              MTLO = 5'b00010,
              D_M    = 5'b00001;

assign Hi_o =(choice == MFHI) ? H : 0;
assign Lo_o =(choice == MFLO) ? L : 0;


always @(posedge clk or posedge rst)begin
  if(rst)begin
```

```
        H = 0;
        L = 0;
    end
    else begin
        case (choice)
            MTHI:
                H = Hi_i;
            MTLO:
                L = Lo_i;
            D_M:begin
                H = mul_h;
                L = mul_l;
            end
            default: begin
                H = H;
                L = L;
            end
        endcase
    end
end
endmodule
```

11. CP0

以下是 Verilog HDL 代码:

```
                    `timescale 1ns / 1ps
module CP0(
input clk,
input rst,
// input [3:0] choice,
// input teq,                    // é©å“¥ç“‘å¯®æ›žå½,é‘·îˆæ«¡å¯®å,šçˆ¶ i_teq &&
zero
//  input  eret,                     //  æ¶“î…ŸæŸ?/å¯®å,šçˆ¶æ©æ–¿æ´–æ·‡â?³å½¿
ERET(Exception Return)
// input mtc0,                    // éæ©ŸP0æ·‡â?³å½¿
// input mfc0,                    // ç’‡ç±†P0æ·‡â?³å½¿
input [3:0] choice,
input [31:0] pc,
input [4:0] addr,           // cp0¼Ä´æÆ÷µØÖ·
input [3:0] cause,          // [6:2]=ExCode (syscall break teq)
input [31:0] wdata,         // write data
output [31:0] rdata,        // read data
```

```
output [31:0] exc_addr    // Êä³öPCµØÖ·
);
parameter      SYSCALL = 4'b1000,
                BREAK     = 4'b1001,
                TEQ        = 4'b1101,
                IE          = 0;
wire teq    = choice [3];
wire eret = choice [2];
wire mtc0 = choice [1];
wire mfc0 = choice [0];

reg [31:0] cop0 [0:31];
wire                                                                    [31:0]
status=cop0[12];//cop[12]é¨å‹«åžµæ¿®å¬¬å§‚éŽ¬ä½‚åŸŒæ´æ›Ÿæ§‚éŽ¬åºç®žç
¼æ¬åš-é‰ãƒ§æ®‘é”›ç‡‚ç´µé”Ÿï¿½?
integer i;
initial
 begin
   for (i=0;i<32;i=i+1)
         cop0[i] = 0;
   cop0[12]={28'b0,4'b1};
end

wire exception       =       status[0]&&      ((status[1]&&cause==SYSCALL)||
                                                    (status[2]&&cause==BREAK)
||

(status[3]&&cause==TEQ&&teq));

always@(posedge clk or posedge rst)
 begin
   if(rst)
     begin
       for (i=0;i<32;i=i+1)
          cop0[i] = 0;
       cop0[12]={28'b0,4'b1111};
     end
   else
     begin
       if (mtc0)
            cop0[addr] <= wdata;
       else if(exception)
            begin
               cop0[12] <= status<<5;
```

```
                cop0[13] <= {24'b0,cause,2'b0};
                cop0[14] <= pc + 32'h400000;
            end
        else if (eret)
            begin
                cop0[12] <= status>>5;
            end
    end
end

assign exc_addr = eret ? cop0[14] - 32'h400000 : 32'h4;
assign rdata = mfc0 ? cop0[addr] : 32'hx;
endmodule
```

12. Clz

以下是 Verilog HDL 代码:

```
module Clz(
input    clk,
input    [31:0] data,
output [31:0] num
       );
integer i;
reg [31:0] num_temp = 32;
reg flag = 1 ;
always @( *)begin
   flag = 1;
   i = 0;
   num_temp = 32;
   for (i=0 ; i<32 ; i=i+1)begin
      if(data[31-i] == 1 && flag == 1)begin
         num_temp = i;
         flag = 0;
      end
   end
   i = 0;
end
assign num = num_temp;
endmodule
```

四、实验结果