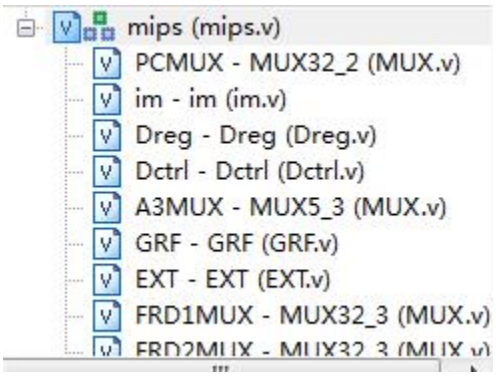
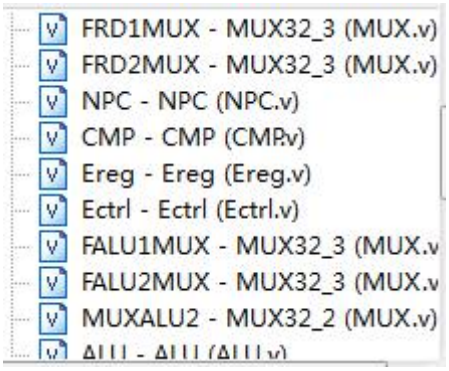
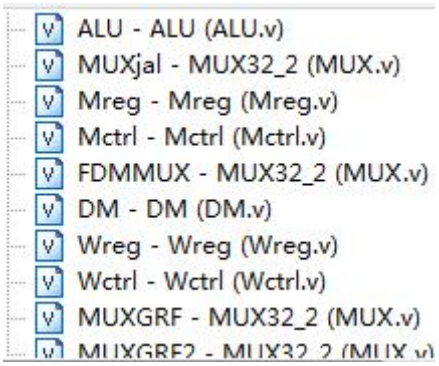
CPU 设计文档

一、总体设计







```
module mips(
    input clk,
    input reset
    );
wire [31:0]PC,PC_n,IR,PC4,PC8,IR_D,PC4_D,nPC,RD1,RD2,E32,CMP1,CMP2,AO_M,WD_W;
wire [31:0]IR_E,PC4_E,EXT_E,ALU1,ALU2,ALU3,rs_E,rt_E,AO,AOE,PC8_E;
wire [31:0]IR_M,PC4_M,rt_M,rt2,DMO,PC8_M;
wire [31:0]IR_W,PC4_W,AO_W,DR_W,WD1,PC8_W;
wire [4:0]A3,A3_E,A3_M,A3_W;
wire [1:0]Tnew,A3sel,FRD1D,FRD2D,Tnew_E,ALUop,FALUrsE,FALUrtE,Tnew_M;
wire stall,PCsrc,branch,jump,jrop,EXTop,RegWrite,W_W,equal,W_E,ALUsrc,W_M,FDMrtM,MemWrite,MemRead,MemToReg,jump1,jalE;

MUX32_2 PCMUX(.a1(PC4),.a2(nPC),.sel(PCsrc),.result(PC_n));
im im(.en(stall),.clk(clk),.reset(reset),.PC(PC),.PC_n(PC_n),.addr(IR));
assign PC4=PC+4;

Dreg Dreg(.clk(clk),.reset(reset),.en(stall),.IR(IR),.PC4(PC4),.IR_D(IR_D),.PC4_D(PC4_D));
Dctrl Dctrl(.op(IR_D[31:26]),.func(IR_D[5:0]),.branch(branch),.jump(jump),.jrop(jrop)
          ,.EXTop(EXTop),.Tnew(Tnew),.A3sel(A3sel),.RegWrite(RegWrite));
MUX5_3 A3MUX(.a1(IR_D[15:11]),.a2(IR_D[20:16]),.sel(A3sel),.result(A3));
GRF GRF(.WPC(PC4_W),.ReadReg1(IR_D[25:21]),.ReadReg2(IR_D[20:16]),.WriteReg(A3_W),.Writedata(WD_W),.clk(clk),.reset(res
       ,.we(W_W),.Readdata1(RD1),.Readdata2(RD2));
EXT EXT(.IMM(IR_D[15:0]),.EXTop(EXTop),.IMM_result(E32));
MUX32_3 FRD1MUX(.a1(RD1),.a2(WD_W),.a3(AO_M),.sel(FRD1D),.result(CMP1));
MUX32_3 FRD2MUX(.a1(RD2),.a2(WD_W),.a3(AO_M),.sel(FRD2D),.result(CMP2));
NPC NPC(.rs(CMP1),.PC4(PC4_D),.I26(IR_D[25:0]),.E32(E32),.sel({jrop,branch}),.nPC(nPC));
CMP CMP(.rs(CMP1),.rt(CMP2),.result(equal));
assign PCsrc=(branch&equal)|jrop|jump;
```

```
Ereg Ereg(.clk(clk),.reset(reset|stall),.IR(IR_D),.A3(A3),.PC4(PC4_D),.rs(CMP1),.rt(CMP2),.E32(E32),.Tnew(Tnew)
          ,.RegWrite(RegWrite),.IR_E(IR_E),.Tnew_E(Tnew_E),.A3_E(A3_E),.PC4_E(PC4_E),.rs_E(rs_E),.rt_E(rt_E)
          ,.EXT_E(EXT_E),.W_E(W_E),.PC8_E(PC8_E));
Ectrl Ectrl(.op(IR_E[31:26]),.func(IR_E[5:0]),.ALUop(ALUop),.ALUsrc(ALUsrc),.jalE(jalE));
MUX32_3 FALU1MUX(.a1(rs_E),.a2(WD_W),.a3(AO_M),.sel(FALUrsE),.result(ALU1));
MUX32_3 FALU2MUX(.a1(rt_E),.a2(WD_W),.a3(AO_M),.sel(FALUrtE),.result(ALU2));
MUX32_2 MUXALU2(.a1(ALU2),.a2(EXT_E),.sel(ALUsrc),.result(ALU3));
ALU ALU(.ALUData1(ALU1),.ALUData2(ALU3),.ALUOp(ALUop),.ALU_result(AOE));
MUX32_2 MUXjal(.a1(AOE),.a2(PC8_E),.sel(jalE),.result(AO));

Mreg Mreg(.clk(clk),.reset(reset),.IR(IR_E),.A3(A3_E),.PC4(PC4_E),.PC8(PC8_E),.AO(AO),.rt(ALU2),.Tnew(Tnew_E),.RegWrite
          ,.IR_M(IR_M),.Tnew_M(Tnew_M),.A3_M(A3_M),.PC4_M(PC4_M),.PC8_M(PC8_M),.AO_M(AO_M),.rt_M(rt_M),.W_M(W_M));
Mctrl Mctrl(.op(IR_M[31:26]),.func(IR_M[5:0]),.MemRead(MemRead),.MemWrite(MemWrite));
MUX32_2 FDMMUX(.a1(rt_M),.a2(WD_W),.sel(FDMrtM),.result(rt2));
DM DM(.MemAddr(AO_M[11:2]),.MemData(rt2),.PC(PC4_W),.MemWrite(MemWrite),.MemRead(MemRead)
      ,.clk(clk),.reset(reset),.MemDatao(DMO));

Wreg Wreg(.clk(clk),.reset(reset),.A3(A3_M),.PC4(PC4_M),.PC8(PC8_M),.AO(AO_M),.DR(DMO),.RegWrite(W_M),.IR(IR_M)
          ,.IR_W(IR_W),.A3_W(A3_W),.PC4_W(PC4_W),.PC8_W(PC8_W),.AO_W(AO_W),.DR_W(DR_W),.W_W(W_W));
Wctrl Wctrl(.op(IR_W[31:26]),.func(IR_W[5:0]),.MemToReg(MemToReg),.jump1(jump1));
MUX32_2 MUXGRF(.a1(AO_W),.a2(DR_W),.sel(MemToReg),.result(WD1));
MUX32_2 MUXGRF2(.a1(WD1),.a2(PC8_W),.sel(jump1),.result(WD_W));

HAZARD HAZARD(.IR_D(IR_D),.IR_E(IR_E),.IR_M(IR_M),.A3_E(A3_E),.A3_M(A3_M),.A3_W(A3_W),.Tnew_E(Tnew_E)
              ,.Tnew_M(Tnew_M),.W_E(W_E),.W_M(W_M),.W_W(W_W),.stall(stall),.FALUrsE(FALUrsE),.FALUrtE(FALUrtE)
              ,.FRD1D(FRD1D),.FRD2D(FRD2D),.FDMrtM(FDMrtM));
endmodule
```

二、数据通路设计

1、im

```
module im(
    input [31:0] PC_n,
    input clk,
    input en,
    input reset,
    output [31:0] PC,
    output [31:0] addr
    );
reg [31:0] im_reg[0:1023];
reg [31:0] PC1;
wire [9:0] b;
wire [31:0] a;
initial begin
   PC1=32'h00003000;
   $readmemh("code.txt",im_reg,0,1023);

end
assign a=PC1-32'h00003000;
assign b=a[11:2];
assign PC=PC1;
assign addr=im_reg[b];
always @(posedge clk) begin
   if(reset)
      PC1<=32'h00003000;
   if(reset==0&&en==0)
      PC1<=PC_n;
end
endmodule
```

模块接口

信号名	方向	功能描述
reset	I	复位信号,当复位信号为 1 时，PC 被设置为 0x00003000
[31:0]PC_n	I	读入下一周期 PC 的值
clk	I	时钟信号
en	I	PC 使能端
[31:0]PC	O	当前 PC 的值
[31:0]addr	O	当前 32 位指令值

2、GRF

```
module GRF(
    input [4:0] ReadReg1,
    input [4:0] ReadReg2,
    input [4:0] WriteReg,
    input [31:0] Writedata,
    input [31:0]WPC,
    input clk,
    input reset,
    input we,
    output [31:0] Readdata1,
    output [31:0] Readdata2
    );
reg [31:0] r[0:31];
assign Readdata1=r[ReadReg1];
assign Readdata2=r[ReadReg2];
integer i;
initial begin
   for(i=0;i<=31;i=i+1)
         r[i]=0;
end
always @(posedge clk) begin
   if(reset)
      for(i=0;i<=31;i=i+1)
         r[i]<=0;
   if(reset==0&&we==1) begin
      if(WriteReg!=0)
         r[WriteReg]<=Writedata;
      $display("%d@%h: $%d <= %h",$time, WPC-4, WriteReg,Writedata);
   end

end
endmodule
```

模块接口

信号名	方向	功能描述
reset	I	复位信号，当复位信号为 1 时，所有寄存器被设置为 0x00000000
we	I	寄存器使能信号，当使能信号为 1 时，寄存器可以被写入数据
clk	I	时钟信号
[4:0]ReadReg1	I	寄存器 1 的地址
[4:0]ReadReg2	I	寄存器 2 的地址
[4:0]WriteReg	I	需要写入数据的寄存器的地址
[31:0]Wirtedata	I	写入的数据
[31:0]WPC	I	当前 PC 的值
[31:0]Readdata1	O	寄存器 1 中的数据
[31:0]Readdata2	O	寄存器 2 中的数据

3、ALU

```
module ALU(
    input [31:0] ALUData1,
    input [31:0] ALUData2,
    input [1:0] ALUOp,
    output [31:0] ALU_result
    );
assign ALU_result=(ALUOp==0)?ALUData1+ALUData2:
                  (ALUOp==1)?ALUData1-ALUData2:
                  (ALUOp==2)?ALUData1|ALUData2:ALUData2<<16;
endmodule
```

模块接口

信号名	方向	功能描述
[31:0]ALUData1	I	ALU 数据 1
[31:0]ALUData2	I	ALU 数据 2
[1:0]ALUOp	I	ALU 控制信号 00:加法运算　　01:减法运算（Data1-Data2） 10:或运算　　11:左移 16 位
[31:0]ALU_result	O	输出 ALU 计算结果

4、DM

```
module DM(
    input [9:0] MemAddr,
    input [31:0] MemData,
    input [31:0] PC,
    input MemWrite,
    input clk,
    input MemRead,
    input reset,
    output [31:0] MemDatao
    );
integer i;
wire [31:0] a;
reg [31:0] dm_reg[0:1023];
assign MemDatao=(MemRead==1)?dm_reg[MemAddr]:0;
assign a={{21'b0},MemAddr};
initial begin
   for(i=0;i<1024;i=i+1)
      dm_reg[i]<=0;
end
always @(posedge clk) begin
   if(reset)
      for(i=0;i<1024;i=i+1)
         dm_reg[i]<=0;
   if(reset==0&&MemWrite==1) begin
         dm_reg[MemAddr]<=MemData;
         $display("%d@%h: *%h <= %h",$time,PC, a<<2,MemData);
   end
end
endmodule
```

模块接口

信号名	方向	功能描述
reset	I	复位信号,当复位信号为 1 时，所有寄存器被设置为 0x00000000
clk	I	时钟信号

MemWrite	I	内存寄存器写使能信号，当该信号为 1 时，可将数据写入内存寄存器中
MemRead	I	内存寄存器读使能信号，当该信号为 1 时，可输出内存寄存器中的数据
[9:0]MemAddr	I	内存寄存器的地址值
[31:0]MemData	I	需要写入内存寄存器的数据
[31:0]PC	I	当前 PC 值
[31:0]MemData_o	O	输出内存寄存器中的数据

5、EXT

```
module EXT(
    input [15:0] IMM,
    input  EXTop,
    output [31:0] IMM_result
    );
assign IMM_result=(EXTop==1)?{{16{IMM[15]}},IMM}:{{16{1'b0}},IMM};
endmodule
```

模块接口

信号名	方向	功能描述
[15:0]IMM	I	16 为待扩展数据
EXTop	I	扩展方式控制信号 0:无符号扩展　　1:有符号扩展
[31:0]IMM_result	O	扩展后数据

6、CMP

```
module CMP(
    input [31:0] rs,
    input [31:0] rt,
    output result
    );
assign result=(rs==rt)?1:0;
endmodule
```

模块接口

信号名	方向	功能描述
[31:0]rs	I	寄存器 rs 的值
[31:0]rt	I	寄存器 rt 的值
result	O	若 rs==rt，则输出 1

7、NPC

```
module NPC(
    input [31:0] PC4,
    input [25:0] I26,
    input [31:0] E32,
    input [31:0] rs,
    input [1:0]sel,
    output [31:0] nPC
    );
wire [31:0]b,j;
assign b=PC4+(E32<<2);
assign j={PC4[31:28],I26,{2{1'b0}}};
assign nPC=(sel==2)?rs:
           (sel==1)?b:j;
endmodule
```

模块接口

信号名	方向	功能描述
[31:0]PC4	I	PC+4 的值
[25:0]I26	I	J 型指令后 26 位
[31:0]E32	I	B 型指令扩展后的 32 位数
[31:0]rs	I	rs 寄存器中的值
[1:0]sel	I	选择信号 2：选择 rs 寄存器中的值　1：b 型指令　0：j 型指令
[31:0]nPC	O	下一周期 PC 的值

8、MUX

```
module MUX32_3(
   input [31:0]a1,
   input [31:0]a2,
   input [31:0]a3,
   input [1:0]sel,
   output [31:0]result
    );
assign result=(sel==2)?a3:
              (sel==1)?a2:a1;
endmodule

module MUX32_2(
   input [31:0]a1,
   input [31:0]a2,
   input sel,
   output [31:0]result
    );
assign result=(sel==1)?a2:a1;

endmodule
module MUX5_3(
   input [4:0]a1,
   input [4:0]a2,
   input [1:0]sel,
   output [4:0]result
    );
assign result=(sel==2)?31:
              (sel==1)?a2:a1;
endmodule
```

模块接口

信号名	方向	功能描述
[31:0]a1	I	多选器 0 接口

[31:0]a2	I	多选器 1 接口
[31:0]a3	I	多选器 2 接口
[1:0]sel	I	选择信号
[31:0]result	O	选择结果
[31:0]a1	I	多选器 0 接口
[31:0]a2	I	多选器 1 接口
sel	I	选择信号
[31:0]result	O	选择结果
[4:0]a1	I	多选器 0 接口
[4:0]a2	I	多选器 1 接口
[1:0]sel	I	选择信号（多选器 2 接口为$31）
[4:0]result	O	选择结果

9、Dreg

```
module Dreg(
    input clk,
    input reset,
    input en,
    input [31:0] IR,
    input [31:0] PC4,
    output reg [31:0] IR_D,
    output reg [31:0] PC4_D
    );
initial begin
   IR_D<=0;
   PC4_D<=0;
end

always @(posedge clk) begin
   if(reset) begin
      IR_D<=0;
      PC4_D<=0;
   end
   if(reset==0&&en==0) begin
      IR_D<=IR;
      PC4_D<=PC4;
   end
end
endmodule
```

模块接口

信号名	方向	功能描述
clk	I	时钟信号
en	I	D 级流水线寄存器使能信号
reset	I	D 级流水线寄存器复位信号
[31:0]IR	I	传入的 32 位指令
[31:0]PC4	I	传入的 PC+4 的值
[31:0]IR_D	O	输出 D 级寄存器的指令
[31:0]PC4_D	O	输出 D 级寄存器的 PC+4

10、Dctrl

```
module Dctrl(
    input [5:0] op,
    input [5:0] func,
    output branch,
    output jump,
    output jrop,
    output EXTop,
    output [1:0]Tnew,
    output [1:0] A3sel,
    output RegWrite
     );
wire addu,subu,lw,sw,beq,ori,lui,jal,jr,j;
assign addu=(op==6'b000000&&func==6'b100001)?1:0;
assign subu=(op==6'b000000&&func==6'b100011)?1:0;
assign lw=(op==6'b100011)?1:0;
assign sw=(op==6'b101011)?1:0;
assign beq=(op==6'b000100)?1:0;
assign ori=(op==6'b001101)?1:0;
assign lui=(op==6'b001111)?1:0;
assign jal=(op==6'b000011)?1:0;
assign jr=(op==6'b000000&&func==6'b001000)?1:0;
assign j=(op==6'b000010)?1:0;

assign Tnew=(lw==1)?2:
            (addu|subu|ori|lui==1)?1:0;

assign branch=beq;
assign jump=jal|j;
assign jrop=jr;
```

```
    ,,
wire addu,subu,lw,sw,beq,ori,lui,jal,jr,j;
assign addu=(op==6'b000000&&func==6'b100001)?1:0;
assign subu=(op==6'b000000&&func==6'b100011)?1:0;
assign lw=(op==6'b100011)?1:0;
assign sw=(op==6'b101011)?1:0;
assign beq=(op==6'b000100)?1:0;
assign ori=(op==6'b001101)?1:0;
assign lui=(op==6'b001111)?1:0;
assign jal=(op==6'b000011)?1:0;
assign jr=(op==6'b000000&&func==6'b001000)?1:0;
assign j=(op==6'b000010)?1:0;

assign Tnew=(lw==1)?2:
            (addu|subu|ori|lui==1)?1:0;

assign branch=beq;
assign jump=jal|j;
assign jrop=jr;
assign EXTop=lw|sw|beq;
assign RegWrite=addu|subu|lui|ori|lw|jal;

assign A3sel[0]=lw|ori|lui;
assign A3sel[1]=jal;
endmodule
```

模块接口

信号名	方向	功能描述
[5:0]op	I	指令的[31:26]位
[5:0]func	I	指令的[5:0]位
branch	O	B 型指令控制信号
jump	O	J 型指令控制信号
Jrop	O	jr 指令控制信号
EXTop	O	扩展控制信号
[1:0]Tnew	O	当前指令产生结果所需要的周期数
[1:0]A3sel	O	写回寄存器的选择信号
RegWrite	O	写寄存器信号

11、Ereg

```
module Ereg(
    input clk,
    input reset,
    input [31:0] IR,
    input [4:0] A3,
    input [31:0] PC4,
    input [31:0] rs,
    input [31:0] rt,
    input [31:0] E32,
    input [1:0] Tnew,
    input RegWrite,
    output reg [31:0] IR_E,
    output reg [1:0]Tnew_E,
    output reg [4:0] A3_E,
    output reg [31:0] PC4_E,
    output reg [31:0] rs_E,
    output reg [31:0] rt_E,
    output reg [31:0] EXT_E,
    output reg W_E,
    output reg [31:0] PC8_E
    );
initial begin
    IR_E<=0;
    Tnew_E<=0;
    A3_E<=0;
    PC4_E<=0;
    rs_E<=0;
    rt_E<=0;
    EXT_E<=0;
```

```
    rt_E<=0;
    EXT_E<=0;
    W_E<=0;
    PC8_E<=0;
end
always @(posedge clk) begin
    if(reset) begin
        IR_E<=0;
        Tnew_E<=0;
        A3_E<=0;
        PC4_E<=0;
        rs_E<=0;
        rt_E<=0;
        EXT_E<=0;
        W_E<=0;
        PC8_E<=0;
    end
    else begin
        IR_E<=IR;
        Tnew_E<=Tnew;
        A3_E<=A3;
        PC4_E<=PC4;
        rs_E<=rs;
        rt_E<=rt;
        EXT_E<=E32;
        W_E<=RegWrite;
        PC8_E<=PC4+4;
    end
end
```

模块接口

信号名	方向	功能描述
clk	I	时钟信号
reset	I	E 级寄存器复位信号
[31:0]IR	I	传入的指令
[4:0]A3	I	要写入的寄存器的地址值
[31:0]PC4	I	传入的 PC+4 的值
[31:0]rs	I	传入的 rs 寄存器的值
[31:0]rt	I	传入的 rt 寄存器的值
[31:0]E32	I	传入的扩展后的立即数值
[1:0]Tnew	I	传入的 Tnew 值
RegWrite	I	写寄存器信号
[31:0]IR_E	O	输出指令值
[1:0]Tnew_E	O	输出 Tnew 值
[4:0]A3_E	O	输出写回寄存器地址值
[31:0]PC4_E	O	输出 PC+4 的值
[31:0]rs_E	O	输出 rs 寄存器的值
[31:0]rt_E	O	输出 rt 寄存器的值
[31:0]EXT_E	O	输出扩展后立即数的值
W_E	O	输出寄存器写使能信号
[31:0]PC8_E	O	输出 PC+8 的值

12、Ectrl

```
module Ectrl(
   input [5:0] op,
   input [5:0] func,
   output [1:0] ALUop,
   output ALUsrc,
   output jalE
    );
wire addu,subu,lw,sw,beq,ori,lui,jal,jr;
assign addu=(op==6'b000000&&func==6'b100001)?1:0;
assign subu=(op==6'b000000&&func==6'b100011)?1:0;
assign lw=(op==6'b100011)?1:0;
assign sw=(op==6'b101011)?1:0;
assign beq=(op==6'b000100)?1:0;
assign ori=(op==6'b001101)?1:0;
assign lui=(op==6'b001111)?1:0;
assign jal=(op==6'b000011)?1:0;
assign jr=(op==6'b000000&&func==6'b001000)?1:0;

assign ALUop[1]=ori|lui;
assign ALUop[0]=subu|lui;
assign ALUsrc=lw|sw|ori|lui;
assign jalE=jal;
endmodule
```

模块接口

信号名	方向	功能描述
[5:0]op	I	指令[31:26]位
[5:0]func	I	指令[5:0]位
[1:0]ALUop	O	ALU 控制信号

ALUsrc	O	ALU 接口 2 数据选择信号
jalE	O	jal 指令控制信号

13、Mreg

```
module Mreg(
    input clk,
    input reset,
    input [4:0] A3,
    input [31:0] PC4,
    input [31:0] AO,
    input [31:0] rt,
    input [1:0] Tnew,
    input RegWrite,
    input [31:0] IR,
    input [31:0] PC8,
    output reg [31:0] IR_M,
    output reg [1:0] Tnew_M,
    output reg [4:0]A3_M,
    output reg [31:0]PC4_M,
    output reg [31:0]AO_M,
    output reg [31:0]rt_M,
    output reg W_M,
    output reg [31:0] PC8_M
    );
initial begin
   A3_M<=0;
   PC4_M<=0;
   AO_M<=0;
   rt_M<=0;
   Tnew_M<=0;
   W_M<=0;
   IR_M<=0;
   PC8_M<=0;
   PC8_M<=0;
end
always @(posedge clk) begin
   if(reset) begin
      A3_M<=0;
      PC4_M<=0;
      AO_M<=0;
      rt_M<=0;
      Tnew_M<=0;
      W_M<=0;
      IR_M<=0;
      PC8_M<=0;
   end
   else begin
      IR_M<=IR;
      A3_M<=A3;
      PC4_M<=PC4;
      rt_M<=rt;
      AO_M<=AO;
      Tnew_M<=($signed(Tnew-1)>0)?Tnew-1:0;
      W_M<=RegWrite;
      PC8_M<=PC8;
   end
end

endmodule
```

模块接口

信号名	方向	功能描述
clk	I	时钟信号
reset	I	M 级寄存器复位信号
[31:0]IR	I	传入的指令
[4:0]A3	I	要写入的寄存器的地址值
[31:0]PC4	I	传入的 PC+4 的值
[31:0]AO	I	传入的 ALU 结果
[31:0]rt	I	传入的 rt 寄存器的值
[31:0]PC8	I	传入的 PC+8 的值
[1:0]Tnew	I	传入的 Tnew 值
RegWrite	I	写寄存器信号
[31:0]IR_M	O	输出指令值
[1:0]Tnew_M	O	输出 Tnew 值
[4:0]A3_M	O	输出写回寄存器地址值
[31:0]PC4_M	O	输出 PC+4 的值
[31:0]AO_M	O	输出 ALU 结果
[31:0]rt_M	O	输出 rt 寄存器的值
W_M	O	输出寄存器写使能信号
[31:0]PC8_M	O	输出 PC+8 的值

14、Mctrl

```
module Mctrl(
    input [5:0] op,
    input [5:0] func,
    output MemRead,
    output MemWrite
     );
wire addu,subu,lw,sw,beq,ori,lui,jal,jr;
assign addu=(op==6'b000000&&func==6'b100001)?1:0;
assign subu=(op==6'b000000&&func==6'b100011)?1:0;
assign lw=(op==6'b100011)?1:0;
assign sw=(op==6'b101011)?1:0;
assign beq=(op==6'b000100)?1:0;
assign ori=(op==6'b001101)?1:0;
assign lui=(op==6'b001111)?1:0;
assign jal=(op==6'b000011)?1:0;
assign jr=(op==6'b000000&&func==6'b001000)?1:0;

assign MemRead=lw;
assign MemWrite=sw;
endmodule
```

模块接口

信号名	方向	功能描述
[5:0]op	I	指令[31:26]位
[5:0]func	I	指令[5:0]位
MemRead	O	寄存器读使能信号
MemWrite	O	寄存器写使能信号

15、Wreg

```
module Wreg(
    input clk,
    input reset,
    input [4:0] A3,
    input [31:0] PC4,
    input [31:0] AO,
    input [31:0] DR,
    input RegWrite,
    input [31:0] IR,
    input [31:0] PC8,
    output reg [31:0] IR_W,
    output reg [4:0]A3_W,
    output reg [31:0]PC4_W,
    output reg [31:0]AO_W,
    output reg [31:0]DR_W,
    output reg W_W,
    output reg [31:0]PC8_W
    );
initial begin
   IR_W<=0;
   A3_W<=0;
   PC4_W<=0;
   AO_W<=0;
   DR_W<=0;
   W_W<=0;
   PC8_W<=0;
end
always @(posedge clk) begin
   if(reset) begin
   PC4_W<=0;
   AO_W<=0;
   DR_W<=0;
   W_W<=0;
   PC8_W<=0;
end
always @(posedge clk) begin
   if(reset) begin
      IR_W<=0;
      A3_W<=0;
      PC4_W<=0;
      AO_W<=0;
      DR_W<=0;
      W_W<=0;
      PC8_W<=0;
   end
   else begin
      IR_W<=IR;
      A3_W<=A3;
      PC4_W<=PC4;
      AO_W<=AO;
      DR_W<=DR;
      W_W<=RegWrite;
      PC8_W<=PC8;
   end
end

endmodule
```

模块接口

信号名	方向	功能描述
clk	I	时钟信号
reset	I	W 级寄存器复位信号

[31:0]IR	I	传入的指令
[4:0]A3	I	要写入的寄存器的地址值
[31:0]PC4	I	传入的 PC+4 的值
[31:0]AO	I	传入的 ALU 结果
[31:0]DR	I	传入的 DM 输出的值
[31:0]PC8	I	传入的 PC+8 的值
RegWrite	I	写寄存器信号
[31:0]IR_W	O	输出指令值
[4:0]A3_W	O	输出写回寄存器地址值
[31:0]PC4_W	O	输出 PC+4 的值
[31:0]AO_W	O	输出 ALU 结果
[31:0]DR_W	O	输出传入的 DM 的值
W_M	O	输出寄存器写使能信号
[31:0]PC8_M	O	输出 PC+8 的值

15、Wctrl

```
module Wctrl(
    input [5:0] op,
    input [5:0] func,
    output jump1,
    output MemToReg
     );
wire addu,subu,lw,sw,beq,ori,lui,jal,jr;
assign addu=(op==6'b000000&&func==6'b100001)?1:0;
assign subu=(op==6'b000000&&func==6'b100011)?1:0;
assign lw=(op==6'b100011)?1:0;
assign sw=(op==6'b101011)?1:0;
assign beq=(op==6'b000100)?1:0;
assign ori=(op==6'b001101)?1:0;
assign lui=(op==6'b001111)?1:0;
assign jal=(op==6'b000011)?1:0;
assign jr=(op==6'b000000&&func==6'b001000)?1:0;

assign jump1=jal;
assign MemToReg=lw;
endmodule
```

模块接口

信号名	方向	功能描述
[5:0]op	I	指令[31:26]位
[5:0]func	I	指令[5:0]位
jump1	O	Jal 跳转控制信号
MemToReg	O	寄存器写回数据控制信号

16、HAZARD

```
module HAZARD(
    input [31:0]IR_D,
    input [31:0]IR_E,
    input [31:0]IR_M,
    input [4:0]A3_E,
    input [4:0]A3_M,
    input [4:0]A3_W,
    input [1:0] Tnew_E,
    input [1:0] Tnew_M,
    input W_E,
    input W_M,
    input W_W,
    output stall,
    output [1:0]FALUrsE,
    output [1:0]FALUrtE,
    output [1:0]FRD1D,
    output [1:0]FRD2D,
    output FDMrtM
    );
wire addu,subu,lw,sw,beq,ori,lui,jal,jr;
wire Tuse_rs0,Tuse_rs1,Tuse_rt0,Tuse_rt1,Tuse_rt2;
wire stall_rs0_E1,stall_rs0_E2,stall_rs0_M1,stall_rs1_E2,stall_rs;
wire stall_rt0_E1,stall_rt0_E2,stall_rt0_M1,stall_rt1_E2,stall_rt;
wire [5:0]op,func;

assign op=IR_D[31:26];
assign func=IR_D[5:0];
assign addu=(op==6'b000000&&func==6'b100001)?1:0;
assign subu=(op==6'b000000&&func==6'b100011)?1:0;
```

```
assign lw=(op==6'b100011)?1:0;
assign sw=(op==6'b101011)?1:0;
assign beq=(op==6'b000100)?1:0;
assign ori=(op==6'b001101)?1:0;
assign lui=(op==6'b001111)?1:0;
assign jal=(op==6'b000011)?1:0;
assign jr=(op==6'b000000&&func==6'b001000)?1:0;

assign Tuse_rs0=beq|jr;
assign Tuse_rs1=addu|subu|lw|sw|ori|lui;
assign Tuse_rt0=beq;
assign Tuse_rt1=addu|subu;
assign Tuse_rt2=sw;

assign stall_rs0_E1=(Tuse_rs0)&(Tnew_E==1)&(IR_D[25:21]==A3_E)&W_E;
assign stall_rs0_E2=(Tuse_rs0)&(Tnew_E==2)&(IR_D[25:21]==A3_E)&W_E;
assign stall_rs0_M1=(Tuse_rs0)&(Tnew_M==1)&(IR_D[25:21]==A3_M)&W_M;
assign stall_rs1_E2=(Tuse_rs1)&(Tnew_E==2)&(IR_D[25:21]==A3_E)&W_E;
assign stall_rs=stall_rs0_E1|stall_rs0_E2|stall_rs0_M1|stall_rs1_E2;

assign stall_rt0_E1=(Tuse_rt0)&(Tnew_E==1)&(IR_D[20:16]==A3_E)&W_E;
assign stall_rt0_E2=(Tuse_rt0)&(Tnew_E==2)&(IR_D[20:16]==A3_E)&W_E;
assign stall_rt0_M1=(Tuse_rt0)&(Tnew_M==1)&(IR_D[20:16]==A3_M)&W_M;
assign stall_rt1_E2=(Tuse_rt1)&(Tnew_E==2)&(IR_D[20:16]==A3_E)&W_E;
assign stall_rt=stall_rt0_E1|stall_rt0_E2|stall_rt0_M1|stall_rt1_E2;

assign stall=stall_rs|stall_rt;

assign FRD1D=((IR_D[25:21]==A3_M)&(Tnew_M==0)&W_M&A3_M!=0)?2:
```

```
assign FRD1D=((IR_D[25:21]==A3_M)&(Tnew_M==0)&W_M&A3_M!=0)?2:
             ((IR_D[25:21]==A3_W&A3_W!=0)&W_W)?1:0;
assign FRD2D=((IR_D[20:16]==A3_M)&(Tnew_M==0)&W_M&A3_M!=0)?2:
             ((IR_D[20:16]==A3_W)&W_W&A3_W!=0)?1:0;
assign FALUrsE=((IR_E[25:21]==A3_M)&(Tnew_M==0)&W_M&A3_M!=0)?2:
               ((IR_E[25:21]==A3_W)&W_W&A3_W!=0)?1:0;
assign FALUrtE=((IR_E[20:16]==A3_M)&(Tnew_M==0)&W_M&A3_M!=0)?2:
               ((IR_E[20:16]==A3_W)&W_W&A3_W!=0)?1:0;
assign FDMrtM=((IR_M[20:16]==A3_W)&W_W&A3_W!=0)?1:0;
endmodule
```

模块接口

信号名	方向	功能描述
[31:0]IR_D	I	D 级寄存器的指令
[31:0]IR_E	I	E 级寄存器的指令
[31:0]IR_M	I	M 级寄存器的指令
[4:0]A3_E	I	E 级要写入的寄存器地址
[4:0]A3_M	I	M 级要写入的寄存器地址
[4:0]A3_W	I	W 级要写入的寄存器地址
[1:0]Tnew_E	I	E 级 Tnew 的值
[1:0]Tnew_M	I	M 级 Tnew 的值
W_E	I	E 级寄存器写使能信号
W_M	I	M 级寄存器写使能信号
W_W	I	W 级寄存器写使能信号
stall	O	暂停信号
[1:0]FALUrsE	O	ALUrs 寄存器转发控制信号
[1:0]FALUrtE	O	ALUrt 寄存器转发控制信号
[1:0]FRD1D	O	寄存器堆 rd1 转发控制信号
[1:0]FRD2D	O	寄存器堆 rd2 转发控制信号
FDMrtM	O	DM 写入数据转发控制信号

三、控制器及冲突设计

func[5:0]	100001	100011	N/A						001000	000000	N/A
op[5:0]	000000	000000	100011	101011	000100	001101	001111	000011	000000	000000	000010
	addu	subu	lw	sw	beq	ori	lui	jal	jr	nop	j
RegDst	1	1	0	X	X	0	0	X	X	X	X
RegWrite	1	1	1	0	0	1	1	0	0	X	0
ALUOp[1:0]	00	01	00	00	01	10	11	XX	XX	XX	XX
ALUsrc	0	0	1	1	0	1	1	X	X	X	X
MemToReg	0	0	1	X	X	0	0	X	X	X	X
MemRead	X	X	1	X	X	X	X	X	X	X	X
MemWrite	0	0	0	1	0	0	0	0	0	X	0
Branch	0	0	0	0	1	0	0	0	0	X	0
EXTop	X	X	1	1	1	0	0	X	X	X	X
jal	0	0	0	0	0	0	0	1	0	X	0
jrop	0	0	0	0	0	0	0	0	1	X	0
jump	0	0	0	0	0	0	0	1	0	X	1

	addu	subu	lw	sw	beq	ori	lui	jal	jr	nop	j
Tuse_rs0	0	0	0	0	1	0	0	X	1	X	X
Tuse_rs1	1	1	1	1	0	1	1	X	0	X	X
Tuse_rt0	0	0	X	0	1	X	X	X	X	X	X
Tuse_rt1	1	1	X	0	0	X	X	X	X	X	X
Tuse_rt2	0	0	X	1	0	X	X	X	X	X	X

[1:0]Tnew	01	01	10	00	00	01	01	00	00	00	00

四、测试程序

```
ori $0,$0,0
ori $1,$0,1
ori $2,$0,2
ori $3,$0,3
ori $4,$0,4
ori $5,$0,5
ori $6,$0,6
ori $7,$0,7
ori $8,$0,8
ori $9,$0,9
lui $10,10
ori $11,$0,11
ori $12,$0,12
ori $13,$0,13
ori $14,$0,14
ori $15,$0,15
ori $16,$0,16
ori $17,$0,17
ori $18,$0,18
ori $19,$0,19
ori $20,$0,20
ori $21,$0,21
ori $22,$0,22
ori $23,$0,23
ori $24,$0,24
ori $25,$0,25
ori $26,$0,26
ori $27,$0,27
lui $28,28
ori $29,$0,29
ori $30,$0,30
ori $31,$0,31
addu $t1,$t1,$t2
subu $t1,$t1,$t2
ori $t1,$0,8
sw $t1,0($0)
lw $t1,0($0)
ori $t1,$0,12
sw $t1,-4($t1)
sw $t1,4($t1)
ori $t1,$0,4
```

```
loop1:
lw $t1,4($t1)
lw $t1,0($t1)
beq $t1,$0,loop1
addu $t1,$t1,$8
beq $t1,$s0,loop2
nop
a3:
j a1
nop
nop
nop
nop
nop
a1:
jal a2
addu $t1,$ra,$t1
sw $ra,0($8)
addu $8,$8,$4
a2:
subu $t1,$ra,$t1
jr $ra
nop
loop2:
addu $t1,$t1,$t2
subu $t3,$t3,$t4
ori $t1,$t1,0x00001234
jal a3
addu $t1,$ra,$t1
```

期望输出：

```
45@00003000: $ 0 <= 00000000
55@00003004: $ 1 <= 00000001
65@00003008: $ 2 <= 00000002
75@0000300c: $ 3 <= 00000003
85@00003010: $ 4 <= 00000004
95@00003014: $ 5 <= 00000005
105@00003018: $ 6 <= 00000006
115@0000301c: $ 7 <= 00000007
125@00003020: $ 8 <= 00000008
135@00003024: $ 9 <= 00000009
145@00003028: $10 <= 000a0000
155@0000302c: $11 <= 0000000b
165@00003030: $12 <= 0000000c
175@00003034: $13 <= 0000000d
```

```
185@00003038: $14 <= 0000000e
195@0000303c: $15 <= 0000000f
205@00003040: $16 <= 00000010
215@00003044: $17 <= 00000011
225@00003048: $18 <= 00000012
235@0000304c: $19 <= 00000013
245@00003050: $20 <= 00000014
255@00003054: $21 <= 00000015
265@00003058: $22 <= 00000016
275@0000305c: $23 <= 00000017
285@00003060: $24 <= 00000018
295@00003064: $25 <= 00000019
305@00003068: $26 <= 0000001a
315@0000306c: $27 <= 0000001b
325@00003070: $28 <= 001c0000
335@00003074: $29 <= 0000001d
345@00003078: $30 <= 0000001e
355@0000307c: $31 <= 0000001f
365@00003080: $ 9 <= 000a0009
375@00003084: $ 9 <= 00000009
385@00003088: $ 9 <= 00000008
385@0000308c: *00000000 <= 00000008
405@00003090: $ 9 <= 00000008
415@00003094: $ 9 <= 0000000c
415@00003098: *00000008 <= 0000000c
425@0000309c: *00000010 <= 0000000c
445@000030a0: $ 9 <= 00000004
455@000030a4: $ 9 <= 0000000c
475@000030a8: $ 9 <= 00000000
515@000030b0: $ 9 <= 00000008
525@000030a4: $ 9 <= 00000000
545@000030a8: $ 9 <= 00000008
585@000030b0: $ 9 <= 00000010
625@000030f0: $ 9 <= 000a0010
635@000030f4: $11 <= ffffffff
645@000030f8: $ 9 <= 000a1234
655@000030fc: $31 <= 00003104
665@00003100: $ 9 <= 000a4338
695@000030d4: $31 <= 000030dc
705@000030d8: $ 9 <= 000a7414
715@000030e4: $ 9 <= fff5bcc8
735@000030f0: *00000008 <= 000030dc
755@000030e0: $ 8 <= 0000000c
765@000030e4: $ 9 <= 000a7414
```

```
785@000030f0: *0000000c <= 000030dc
805@000030e0: $ 8 <= 00000010
815@000030e4: $ 9 <= fff5bcc8
835@000030f0: *00000010 <= 000030dc
855@000030e0: $ 8 <= 00000014
865@000030e4: $ 9 <= 000a7414
885@000030f0: *00000014 <= 000030dc
905@000030e0: $ 8 <= 00000018
915@000030e4: $ 9 <= fff5bcc8
935@000030f0: *00000018 <= 000030dc
955@000030e0: $ 8 <= 0000001c
965@000030e4: $ 9 <= 000a7414
985@000030f0: *0000001c <= 000030dc
```

该程序最后为死循环，将所有 DM 中的值都赋为 0x000030dc

五、思考题

1. 在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。(**非常重要**)

Tuse_rs	Tuse_rt	Tnew_E	冲突寄存器	解决方案	测试程序
0		1	rs	暂停 1 个时钟周期后从 M 到 D 转发	ori $ra $0,0x0000300c jr $ra
0		2	rs	暂停 2 个时钟周期后从 W 到 D 转发	lw $ra,0($0) jr $ra
0	0	1	rs 或 rt	暂停 1 个时钟周期后从 M 到 D 转发	addu $t1,$t1,$t2 beq $t1,$t2,loop
0	0	2	rs 或 rt	暂停两个时钟周期后从 W 到 D 转发	lw $t1,0($0) beq $t1,$t2,loop
1	1	1	rs 或 rt	从 M 到 E 转发	addu $t1,$t1,$t2 subu $t1,$t1,$t2
1	1	2	rs 或 rt	暂停一个时钟周期后从 W 到 E 转发	lw $t1,0($0) addu $t1,$t1,$t2
1		1	rs	从 M 到 E 级转发	ori $t1,$0,1 lui $t1,1
1		2	rs	暂停一个时钟周期后从 W 到 E 转发	lw $t1,0($0) lui $t1,1
1	2	1	rs	从 M 到 E 级转发	lui $t1,1 sw $t1,0($0)
1	2	2	rs	暂停 1 个周	lw $t1,0($0)

				期后从 W 到 E 级转发	sw $t2,0($t1)
1	2	2	rt	从 W 到 M 级转发	lw $t1,0($0) sw $t1,0($0)

Tuse_rs	Tuse_rt	Tnew_M	冲突寄存器	解决方案	测试程序
0		1	rs	暂停一个周期后从 W 到 D 级转发	lw $ra,0($0) lui $t2,1 jr $ra
0	0	1	rs 或 rt	暂停一个周期后从 W 到 D 级转发	lw $t1,0($0) lui $t2,1 beq $t1,$t2,loop

1	1	1	rs 或 rt	从 W 到 E 级转发	lw $t1,0($0)
					lui $t2,1
					addu $t1,$t3,$t1

1		1	rs	从 W 到 E 级转发	lw $t1,0($0) lui $t2,1 ori $t1,$t1,1

1	2	1	rs	从 W 到 E 级转发	lw $t1,0($0)
					lui $t2,1
					sw $t1,0($0)

0		0	rs	从 M 级到 D 级转发	addu $ra,$t1,$t2 lui $t2,1 jr $ra
0	0	0	rs 或 rt	从 M 级到 D 级转发	addu $t1,$t1,$t2 lui $t3,1 beq $t1,$t2,loop

Tuse_rs	Tuse_rt	Tnew_W	冲突寄存器	解决方案	测试程序
0		0	rs	从 W 到 D 级转发	addu $ra,$t1,$t2 nop nop jr $ra
0	0	0	rs 或 rt	从 W 到 D 级转发	addu $t1,$t1,$t2 nop nop beq $t1,$t2,loop