设计标准：
5 段流水线 可解决冲突、异常等情形 不支持中断

实现指令集：
or, and, xor, nor, sllv, srlv, srav, mfhi, mflo, mthi, mtlo, movn, movz, slt, sltu, add, addu, sub, subu, mult, multu, jr, jalr, ori, andi, xori, lui, slti, sltiu, addi, addiu, j, jal, beq, bgtz, blez, bne, lb, lbu, lh, lhu, lw, sb, sw, sh, bgez, bgezal, bltz, bltzal, clz, clo, mul, madd, msub, maddu, msubu

模块结构：
cpu 上层模块，调用下述所有模块
pc_reg 存储、更新 pc
inst_rom 存指令
if_id 段寄存器，传递控制信号
id 译码，生成控制信号
id_ex 段寄存器，传递控制信号
ex 执行指令
ex_mem 段寄存器，传递控制信号
mem 访存，与数据存储器交互
data_ram 存数据
mem_wb 段寄存器，传递控制信号
hilo_reg 保存 hi，lo 两个寄存器的值
ctrl 生成 stall 信号，处理数据冲突
另，为统一格式，编写头文件 header.v 并在各个模块中引用

具体实现：
pc_reg
使用 chipenable/disable 信号，enable 时 pc 先根据有无 stall 信号决定是否暂停流水线，再根据有无 branch 控制信号决定是跳转还是正常加 4

```
always@(posedge clk)
    begin
        if(ce==`ChipDisable)
        begin
            pc<=32'h00000000;
        end
        else if(stall[0]==`NoStop)
        begin
            if(branch_flag_i==`Branch)
            begin
                pc<=branch_target_address_i;
            end
            else
            begin
                pc<=pc+4'h4;
                end
            end
        end
```

inst_rom
内置寄存器组，由 inst_rom.data 初始化，取指时由 chipenable 信号控制，若 enable 则正常取指，否则输出 0

```
reg[`InstBus] inst_mem [0:`InstMemNum-1];
        initial $readmemh ("inst_rom.data",inst_mem);
        always@(*)
        begin
            if(ce==`ChipDisable)
            begin
                inst<=`ZeroWord;
           end
            else
            begin
                inst<=inst_mem[addr[`InstMemNumLog2+1:2]];
            end
        end
```

regfile
寄存器组，由写入，读一，读二三部分构成；为解决部分数据冲突，判断 ex/mem 阶段指令写入 reg 地址与后来指令读 reg 地址是否相同，若相同则可直接写出；0 号寄存器初始化为 0，且拒绝向 0 号寄存器的写操作

```
initial
        begin
            regs[0]<=`ZeroWord;
        end

        always@(posedge clk)
        begin
            if(rst==`RstDisable)
            begin
                if((we==`WriteEnable) && (waddr!=`RegNumLog2'h0))
                begin
                    regs[waddr]<=wdata;
                end
            end
        end

        always@(*)
```

```
begin
        if(rst==`RstEnable)
        begin
                rdata1<=`ZeroWord;
      end
      else if(raddr1==`RegNumLog2'h0)
      begin
                rdata1<=`ZeroWord;
      end
      else if((raddr1==waddr) && (we==`WriteEnable) && (re1==`ReadEnable))
        begin
                rdata1<=wdata;
      end
        else if(re1==`ReadEnable)
        begin
            rdata1<=regs[raddr1];
      end
        else
        begin
            rdata1<=`ZeroWord;
      end
end

always@(*)
begin
        if(rst==`RstEnable)
        begin
                    rdata2<=`ZeroWord;
        end
        else if(raddr2==`RegNumLog2'h0)
        begin
                rdata2<=`ZeroWord;
        end
        else if((raddr2==waddr) && (we==`WriteEnable) && (re2==`ReadEnable))
        begin
            rdata2<=wdata;
        end
        else if(re2 == `ReadEnable)
        begin
            rdata2 <= regs[raddr2];
        end
        else
        begin
            rdata2 <= `ZeroWord;
```

```
                end
        end
```

if_id

主要负责信号传递，但受 ctrl 模块产生的 stall 信号控制，若有 stall 信号则暂停流水线

```
always@(posedge clk)
        begin
                if(rst==`RstEnable)
                begin
                        id_pc<=`ZeroWord;
                        id_inst<=`ZeroWord;
                end
                else if(stall[1] == `Stop && stall[2] == `NoStop)
                begin
                        id_pc<=`ZeroWord;
                        id_inst<=`ZeroWord;
              end
                else if(stall[1] == `NoStop)
                begin
                    id_pc <= if_pc;
                    id_inst <= if_inst;
                end
        end
```

id

基本控制信号

alu 操作类型 alu_op/sel, alu 操作数来源 reg1/2_read_o, 读寄存器组地址 reg1/2_read_addr, 立即数 imm, 写回信号 wreg_o, 跳转控制 branch_flag_o, 跳转地址 branch_target_address_o, 针对 ld/st 指令数据冲突必须延迟流水线的控制信号 next_inst_in_delayslot_o

各个指令的控制信号生成（由指令码先将各指令分类，首先 31：26 为 000000 则为 special 类，此类型中有 or, and, xor, nor, sllv, srlv, srav, mfhi, mflo, mthi, mtlo, movn, movz, slt, sltu, add, addu, sub, subu, mult, multu, jr, jalr；为 000001 中有 bgez,bgezal,bltz,bltzal；为 011100 则为 special2 类，有 clz,clo,mul,madd,maddu,msub,msubu ;其余指令各自判断，主要是 ld/st 类，跳转类和立即数运算操作指令

各指令分别写出所有控制信号，篇幅过长，在此不一一说明，详情请见工程目录下 “id.v”

id_ex

受 stall 信号控制，若无暂停流水线需求则正常传递控制信号，否则输出 0

```
always@(posedge clk)
        begin
                if(rst==`RstEnable)
```

```
            begin
                ex_aluop<=`EXE_NOP_OP;
                ex_alusel<=`EXE_RES_NOP;
                ex_reg1<=`ZeroWord;
                ex_reg2<=`ZeroWord;
                ex_wd<=`NOPRegAddr;
                ex_wreg<=`WriteDisable;
                ex_link_address<=`ZeroWord;
                ex_is_in_delayslot<=`NotInDelaySlot;
                is_in_delayslot_o<=`NotInDelaySlot;
                ex_inst <= `ZeroWord;
            end
            else if(stall[2]==`Stop && stall[3]==`NoStop)
            begin
                ex_aluop<=`EXE_NOP_OP;
                ex_alusel<=`EXE_RES_NOP;
                ex_reg1<=`ZeroWord;
                ex_reg2<=`ZeroWord;
                ex_wd<=`NOPRegAddr;
                ex_wreg<=`WriteDisable;
                ex_link_address<=`ZeroWord;
                ex_is_in_delayslot<=`NotInDelaySlot;
                ex_inst <= `ZeroWord;
            end
            else if(stall[2]==`NoStop)
            begin
                ex_aluop<=id_aluop;
                ex_alusel<=id_alusel;
                ex_reg1<=id_reg1;
                ex_reg2<=id_reg2;
                ex_wd<=id_wd;
                ex_wreg<=id_wreg;
                ex_link_address<=id_link_address;
                ex_is_in_delayslot<=id_is_in_delayslot;
            is_in_delayslot_o<=next_inst_in_delayslot_i;
            ex_inst<=id_inst;
            end
        end
```

ex

与 id 模块对应，先将各指令输出分成几种类型：逻辑指令 logicout，移位指令 shiftres，算数操作 arithmeticres，乘法指令 mulres，跳转地址 link_address_i；其中，逻辑、移位和大部分算数操作指令直接由指令传来的数据即可完成运算，而 clo，clz 各用一条特殊赋值指令解决；乘法和 madd 等指令需要与 hi，lo 寄存器的内容结合处理，而且 madd，msub 还需要

申请 stall 信号；mf/mthi/lo 指令容易实现，只需存取 hi，lo 寄存器中的值

各指令操作如上所述，具体篇幅过长，在此不一一详述，请见工程目录下”ex.v”

ex_mem
与前两个段模块类似，受 stall 信号控制，若无 stall 信号正常传递运算结果和控制信号，否则输出 0

```
always@(posedge clk)
begin
    if(rst==`RstEnable)
    begin
        mem_wd<=`NOPRegAddr;
        mem_wreg<=`WriteDisable;
        mem_wdata<=`ZeroWord;
        mem_hi<=`ZeroWord;
        mem_lo<=`ZeroWord;
        mem_whilo<=`WriteDisable;
        hilo_o<={`ZeroWord, `ZeroWord};
        cnt_o<=2'b00;
        mem_aluop<=`EXE_NOP_OP;
        mem_mem_addr<=`ZeroWord;
        mem_reg2<=`ZeroWord;
    end
    else if(stall[3]==`Stop && stall[4]==`NoStop)
    begin
        mem_wd<=`NOPRegAddr;
        mem_wreg<=`WriteDisable;
        mem_wdata<=`ZeroWord;
        mem_hi<=`ZeroWord;
        mem_lo<=`ZeroWord;
        mem_whilo<=`WriteDisable;
        hilo_o<=hilo_i;
        cnt_o<=cnt_i;
        mem_aluop<=`EXE_NOP_OP;
        mem_mem_addr<=`ZeroWord;
        mem_reg2<=`ZeroWord;
    end
    else if(stall[3]==`NoStop)
    begin
        mem_wd<=ex_wd;
        mem_wreg<=ex_wreg;
        mem_wdata<=ex_wdata;
        mem_hi<=ex_hi;
```

```
            mem_lo<=ex_lo;
            mem_whilo<=ex_whilo;
            hilo_o<={`ZeroWord, `ZeroWord};
            cnt_o<=2'b00;
            mem_aluop<=ex_aluop;
            mem_mem_addr<=ex_mem_addr;
            mem_reg2<=ex_reg2;
        end
        else
        begin
            hilo_o<=hilo_i;
            cnt_o<=cnt_i;
        end
    end
```

mem
主要处理访存指令 lb, lbu, lh, lhu, lw,lwl,lwr, sb, sw, sh, swl, swr 和传递控制信号，产生与 data_ram 交互的地址与使能信号，各个访存指令的控制信号分别讨论得出

信号传递

```
always@(*)
    begin
        if(rst==`RstEnable)
        begin
            wd_o<=`NOPRegAddr;
            wreg_o<=`WriteDisable;
            wdata_o<=`ZeroWord;
            hi_o<=`ZeroWord;
            lo_o<=`ZeroWord;
            whilo_o<=`WriteDisable;
            mem_addr_o<=`ZeroWord;
            mem_we<=`WriteDisable;
            mem_sel_o<=4'b0000;
            mem_data_o<=`ZeroWord;
            mem_ce_o<=`ChipDisable;
        end
        else
        begin
            wd_o<=wd_i;
            wreg_o<=wreg_i;
            wdata_o<=wdata_i;
            hi_o<=hi_i;
            lo_o<=lo_i;
            whilo_o<=whilo_i;
            mem_we<=`WriteDisable;
            mem_addr_o<=`ZeroWord;
            mem_sel_o<=4'b1111;
            mem_ce_o<=`ChipDisable;
…
```
指令操作在此不一一详述，具体请见工程目录下 “mem.v”

data_ram
内置寄存器组，分读写两部分

```
always@(posedge clk)
    begin
        if(ce==`ChipDisable)
        begin
            data_o<=`ZeroWord;
        end
        else if(we==`WriteEnable)
        begin
            if(sel[3]==1'b1)
            begin
                data_mem[addr[`DataMemNumLog2+1:2]][31:24]<=data_i[31:24];
           end
            if(sel[2]==1'b1)
            begin
                data_mem[addr[`DataMemNumLog2+1:2]][23:16]<=data_i[23:16];
           end
           if(sel[1]==1'b1)
            begin
                data_mem[addr[`DataMemNumLog2+1:2]][15:8]<=data_i[15:8];
           end
            if(sel[0]==1'b1)
            begin
                data_mem[addr[`DataMemNumLog2+1:2]][7:0]<=data_i[7:0];
           end
        end
    end

    always@(*)
    begin
        if(ce==`ChipDisable)
        begin
            data_o<=`ZeroWord;
        end
        else if(we==`WriteDisable)
```

```
            begin
                data_o<=data_mem[addr[`DataMemNumLog2+1:2]];
            end
            else
            begin
                data_o<=`ZeroWord;
            end
        end
```

mem_wb
与前面的段寄存器类似，受 stall 信号控制，负责结果传递

```
always@(posedge clk)
        begin
            if(rst==`RstEnable)
            begin
                wb_wd<=`NOPRegAddr;
                wb_wreg<=`WriteDisable;
              wb_wdata<=`ZeroWord;
              wb_hi<=`ZeroWord;
              wb_lo<=`ZeroWord;
              wb_whilo<=`WriteDisable;
            end else if(stall[4]==`Stop && stall[5]==`NoStop)
            begin
                wb_wd<=`NOPRegAddr;
                wb_wreg<=`WriteDisable;
              wb_wdata<=`ZeroWord;
              wb_hi<=`ZeroWord;
              wb_lo<=`ZeroWord;
              wb_whilo<=`WriteDisable;
            end
            else if(stall[4]==`NoStop)
            begin
                wb_wd<=mem_wd;
                wb_wreg<=mem_wreg;
                wb_wdata<=mem_wdata;
                wb_hi<=mem_hi;
                wb_lo<=mem_lo;
                wb_whilo<=mem_whilo;
            end      //if
        end
```

hilo_reg
实现 hi 和 lo 寄存器存取

```
always@(posedge clk)
        begin
                if(rst==`RstEnable)
                begin
                        hi_o<=`ZeroWord;
                        lo_o<=`ZeroWord;
                end
                else if((we==`WriteEnable))
                begin
                        hi_o<=hi_i;
                        lo_o<=lo_i;
                end
        end
```

ctrl
产生 stall 信号

```
always@(*)
begin
                if(rst==`RstEnable)
                begin
                        stall<= 6'b000000;
                end
                else if(stallreq_from_ex==`Stop)
                begin
                        stall<=6'b001111;
                end
                else if(stallreq_from_id==`Stop)
                begin
                        stall<=6'b000111;
                end
                else
                begin
                        stall<=6'b000000;
                end
        end
```