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