流水线CPU设计

模块端口及其功能描述

·F级：

PC模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
PCSrc[2:0]	I	000:pc=pc+4,001:pc=pc+4+sign_extend(offset||0^2),010:pc=pc=pc31…28||instr||0^2,011:pc=ra
immediate_32[31:0]	I	32位立即数
immediate_26[25:0]	I	j指令的26位立即数
ra[31:0]	I	返回的地址
stall	I	暂停信号,1:pc保持不变，即暂停,0:pc正常计算
D_PC[31:0]	I	D级PC，用于计算beq型跳转地址
pc_out[31:0]	O	输出的pc

FD_reg模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
stall	I	1:冻结，机器码保持为上一个周期的值，0:机器码正常进行读取
F_PC[31:0]	I	F级的pc
F_Instr[31:0]	I	F级的指令
D_PC[31:0]	O	流水到D级的pc
D_Instr[31:0]	O	流水到D级的指令

·D级：

GRF模块

信号名	端口	描述
A1[4:0]	I	读到RD1的寄存器的地址
A2[4:0]	I	读到RD2的寄存器的地址

信号名	端口	描述
A3[4:0]	I	进行写操作的寄存器的地址
WD[31:0]	I	写入的数据
clk	I	时钟信号
reset	I	同步复位信号
W_PC[31:0]	I	流水到的W级的pc，因为我们设定是只有在W级才会进行写寄存器的操作，所以写的pc应该就是一直随着流水线传递的pc
RD1[31:0]	O	从A1的寄存器读出的数据
RD2[31:0]	O	从A2的寄存器读出的数据

EXT模块

信号名	端口	描述
immediate_16[15:0]	I	16位的立即数
ExtOp	I	1:符号扩展,0:0扩展
immediate_32[31:0]	O	扩展成32位的立即数

D_ctrl模块

信号名	端口	描述
D_Instr[31:0]	I	流水到D级的指令
D_forward_RD1[31:0]	I	考虑了有转发的D级真实的会流水下去的输出
D_forward_RD2[31:0]	I	考虑了有转发的D级真实的会流水下去的输出，它俩用来比较，来提前beq的判断，减少开销
D_A3[4:0]	O	判断要写的寄存器，**如果**该条指令**非写寄存器**操作，我们就不需要关心A3的值，为了操作的连贯性，置0
RegDst[2:0]	O	000:写入rt寄存器，001:写入rd寄存器，010:写入31号寄存器，(提供选择信号供D_A3选择要写入的寄存器)
ExtOp	O	1：对符号数进行符号扩展，0：0扩展
PCSrc[2:0]	O	决定PC的计算方式,000：正常计算，001：beq型跳转，010:jal型跳转，011：jr型跳转，具体跳转方式参考pc模块
D_Tuse_rs[1:0]	O	判断从现在开始经过几个周期会用到GPR[rs]的数据

信号名	端口	描述
D_Tuse_rt[1:0]	O	判断从现在开始经过几个周期会用到GPR[rt]的数据
HILO_operation	O	判断D型指令是否为乘除槽相关指令

DE_reg模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
stall	I	1：等价于插入了一个nop指令，0：正常进行
RD1_in[31:0]	I	考虑过转发后的rs寄存器真实数据
RD2_in[31:0]	I	考虑过转发后的rt寄存器真实数据
immediate_32_in[31:0]	I	32位立即数
D_PC[31:0]	I	D级PC
D_Instr[31:0]	I	
D_A3[4:0]	I	要进行写操作的寄存器，随流水线流水下去
RD1_out[31:0]	O	随流水线流下去的rs寄存器真实数据
RD2_out[31:0]	O	随流水线流下去的rt寄存器真实数据
immediate_32_out[31:0]	O	随流水线流水下去的立即数
E_PC[31:0]	O	将D_PC的值流水下去
E_Instr[31:0]	O	将D_Instr的值流水下去
E_A3[4:0]	O	将D_A3的值流水下去

·E级：

ALU模块

信号名	端口	描述
SrcA[31:0]	I	ALU模块的第一个操作数
SrcB[31:0]	I	ALU模块的第二个操作数
ALUOp[2:0]	I	

信号名	端口	描述
ALUResult	O	计算结果

E_ctrl模块

信号名	端口	描述
E_Instr[31:0]	I	
E_PC[31:0]	I	
ALUOp[2:0]	O	决定运算操作
ALUSrc	O	1：选取流水到E级的32位立即数，0：选取考虑了转发后的rt寄存器数据E_forward_RD2
E_Tnew[1:0]	O	根据指令具体类型判断该指令产生的最终结果写入寄存器需要几个周期，00：与PC有关的指令操作，在D级就已经完成结果写入寄存器且随流水线传递了,01:ALU运算指令，在E级经过ALU产生E_ALUResult，再等一个周期才能经EM_reg寄存器流水到M级,10:DM型指令
E_WD[31:0]	O	根据E_Instr判断该条指令是什么指令，产生的结果是什么，判断是否能够产生所需需要的数据进行转发，因为E级只有pc是已经产生的，alu的计算结果和dm的访存结果都没产生，所以只能转发pc回去,通过该输出，我们就封装了让模块自己决定转发什么数据回去供D级使用
start	O	乘除槽启动的信号
Op[3:0]	O	乘除槽具体做的运算类型 0000:mult,0001:multu,0010:div,0011:divu,0100:mfhi,0101:mflo,0110:mthi,0111:mtlo,1111:不做乘除运算

MDU模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
Op[3:0]	I	乘除槽运算类型
start	I	乘除槽启动信号
E_forward_RD1[31:0]	I	
E_forward_RD2[31:0]	I	
Busy	O	
HI[31:0]	O	
LO[31:0]	O	

EM_reg模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
E_PC[31:0]	I	
E_Instr[31:0]	I	
ALUResult_in[31:0]	I	
E_RD2[31:0]	I	
E_A3[4:0]	I	
E_Tnew[1:0]	I	
HI[31:0]	I	
LO[31:0]	I	
ALUResult_out[31:0]	O	
M_PC[31:0]	O	
M_Instr[31:0]	O	
M_RD2[31:0]	O	
M_A3[4:0]	O	
M_Tnew[1:0]	O	采用递减思路，免去了ctrl模块重复计算,M_Tnew={E_Tnew-1,0}
M_HI[31:0]	O	用于mfhi，mflo的转发
M_LO[31:0]	O	

·M级：

M_ctrl模块

信号名	端口	描述
M_Instr[31:0]	I	
M_PC[31:0]	I	
M_ALUResult[31:0]	I	
MemWrite	O	1:可以往内存写入数据，0：无效写入操作

信号名	端口	描述
DMOp[2:0]	O	决定DM的读写方式，是以字还是半字还是字节为单位进行读写。000：字方式，lw,sw指令，001:半字方式，lh,sh指令，010:字节方式，lb,sb指令
M_WD[31:0]	O	决定转发回去的数据是什么,因为M级ALUResult和PC结果已经产生，所以根据指令类型判断当前面有转发需求时转发什么数据回去，相当于临时结果变量的存储
M_HI[31:0]	O	
M_LO[31:0]	O	
m_data_byteen[3:0]	O	字节使能信号
BEOp[2:0]	O	load类型指令取得的数据的扩展

BE模块：

信号名	端口	描述
address[1:0]	I	M_ALUResult的最后两位，byte位
m_data_rdata[31:0]	I	从tb的DM中读出来的数据，需要根据指令类型进行扩展
BEOp[2:0]	I	000:lw,001:lb,011:lh
M_DMRD[31:0]	O	经过扩展的数据，可认为是从DM最终读出的正确数据

MW_reg模块

信号名	端口	描述
clk	I	
reset	I	
M_ALUResult[31:0]	I	
M_DMRD[31:0]	I	从内存中读取出来的数据
M_PC[31:0]	I	
M_Instr[31:0]	I	
M_A3[4:0]	I	
M_Tnew[1:0]	I	
M_HI[31:0]	I	

信号名	端口	描述
M_LO[31:0]	I	
W_ALUResult[31:0]	O	
W_DMRD[31:0]	O	
W_A3[4:0]	O	
W_PC[31:0]	O	
W_Instr[31:0]	O	
W_Tnew[1:0]	O	其余均为流水信号，W_Tnew恒等于0
W_HI[31:0]	O	
W_LO[31:0]	O	

·W级：

W_ctrl模块

信号名	端口	描述
W_Instr[31:0]	I	
W_ALUResult[31:0]	I	
W_DMRD[31:0]	I	
W_PC[31:0]	I	
W_HI[31:0]	I	
W_LO[31:0]		
W_WD[31:0]	O	此时alu的计算结果，pc，dm的访存结果均已经产生，根据需要挑选数据转发回去

Stall模块

信号名	端口	描述
D_Tuse_rs[1:0]	I	
D_Tuse_rt[1:0]	I	
E_Tnew[1:0]	I	

信号名	端口	描述
M_Tnew[1:0]	I	
D_A1[4:0]	I	
D_A2[4:0]	I	
E_A3[4:0]	I	
M_A3[4:0]	I	
HILO_operation	I	
start	I	
Busy	I	
stall	O	1:发送阻塞信号，进行pc，FD_reg，DE_reg寄存器的相应操作,0:无事发生

思考题

1.因为乘除法需要的时钟周期很长，乘法需要5个时钟周期，除法需要10个时钟周期，并且乘除法的结果是存入HI，LO寄存器的，与grf，dm无关，若整合进ALU里，遇上其他指令的时候会阻塞其他指令的ALU计算，所以需要有单独的乘除法部件进行乘除法运算，减少阻塞的可能。

模拟Mars的情况，可以进行稳定的转发。

2.乘法由若干个乘法单元组成，每个周期计算几位，最后依次累加起来，所以会在几个周期后得到结果。除法采用试商法，一个周期内计算4位左右的商，经过8个周期计算结束。

3.Busy信号当且仅当乘除槽内进行的是mult,multu,div,divu时才会生成并持续，我在stall模块中新增了HILO_operation,start,Busy端口，当(start|Busy)&&(HILO_operation),也就是说乘除槽内正在进行乘除运算并且D级指令为乘除槽指令的时候，实现阻塞。

4.控制器负责提供写使能信号，实现了控制模块提供控制信号的统一性，同时，无需关心字节使能为0的位上数据的正确性，只需要考虑字节使能位为1上的数据是否正确，以及不正确如何调整即可，实现了清晰性。

5.不是，实际从DM获得数据是load型指令的结果，向DM写入数据是save型指令的要求，在我的实现中写入的数据是M_forward_RD2，根据字节使能信号写入相应的位即可。任何情况下都高于，因为按字读写实际上是字节使能信号为全1的特殊情况，只要要求有按字节读写的功能都一定能实现按字读写，若只考虑按字读写，实现按字节读写时候就需要扩展额外的数据通路，降低了流水线的总效率。

6.专门生成一个constant.v文件用来存放宏定义，在每个ctrl模块include，而不是在每个ctrl模块中单独宏定义，保证了规范性，需要常量的时候用宏定义用名字去替换掉常量，保证了可读性。添加指令的时候根据指令类型添加，利用指令的类型的相似性保证了规范性，减少了出错的可能。

7.乘除槽指令的冲突。当(start|Busy)&&(HILO_operation),也就是说乘除槽内正在进行乘除运算并且D级指令为乘除槽指令的时候，实现阻塞。测试样例如下

```
ori $1,$1,1
ori $2,$2,2
mult $1,$2
mflo $3
div $1,$2
mfhi $4
multu $2,$3
mthi $3
divu $2,$3
mtlo $4
```

8.本次添加指令除了乘除槽外其实与P5的冲突差不多，我在本次P6课下遇到的最大困难是outputs are too much，一开始我认为出现如此情况有两种可能，一是在不该写寄存器的时候进行了写操作，于是我分别测了每条指令，发现行为均正确，便排除了误写的可能性。二是延迟槽的处理有误，但是本次新增的延迟槽只有bne，处理同beq很相似，便也排除了延迟槽的问题。最后我推测expected nothing是因为评测机的输出已经到末尾了，而我还有输出。经同学提醒，我构造了如下样例，最终发现是死循环的时候我还会一直输出的问题

```
ori $1,$1,1
sw $1,0($0)
nop
nop
lb $2,0($0)
nop
label1:beq $0,$0,label1
```

因为我pc的实现方式与推荐方式不一样，经过仿真波形图的对比，我发现是我误认为F_PC一定比D_PC大4，这种情况在一般情况下确实成立，但是在上述代码提到的死循环的时候会在某一个周期内，出现F_PC=D_PC的情况，而我计算pc始终用的是F_PC计算的，这也就导致了输出的F_PC的错误，也就导致了我的pc在死循环的时候没有冻结，而是减减加加，所以会执行很多遍lb,也就导致了在评测机expected nothing的时候，我得到的结果是outputs are too much。

除上文提到的问题外，我还遇到了对符号数操作不对的问题，我误认为verilog的最高位是符号位，忘记了$signed才是取符号，经过构造负数的slt，检查出了我的问题。

还有评测机报错写入DM的时候的错误，发现写入DM的数据不为0而我为0，于是构造了sb,sh的数据，也成功找出了错误。

综上，我的构造策略是根据新增指令类型的特点构造数据，根据评测机报错信息推测可能是哪条指令的错误，再对应构造测试样例。