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
pc_out[31:0]	O	输出的pc

IM模块

信号名	端口	描述
PC[31:0]	I	程序计数器
Instr[31:0]	O	从ROM取得的指令

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
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

信号名	端口	描述
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
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

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	
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

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	

信号名	端口	描述
E_RD2[31:0]	I	
E_A3[4:0]	I	
E_Tnew[1:0]	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级：

DM模块

信号名	端口	描述
address[31:0]	I	从RAM读出的地址
writedata[31:0]	I	写入的数据
clk	I	时钟信号
WE	I	1：能写入，0：不能写入
reset	I	1：RAM清0，0：无效
DMOp[2:0]	I	000:无所谓，因为WE会恒为0，001:lw,sw指令,010:lh,sh指令,011:lb,sb指令
PC	I	
readdata	O	从RAM读出的数据

M_ctrl模块

信号名	端口	描述
M_Instr[31:0]	I	
M_PC[31:0]	I	

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

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	
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级：

W_ctrl模块

信号名	端口	描述
W_Instr[31:0]	I	
W_ALUResult[31:0]	I	
W_DMRD	I	
W_PC	I	
W_WD	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	
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	
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beq型指令的T_use=0，如果beq有用到正在数据计算的寄存器很有可能会阻塞。

```
ori $1,$1,1
ori $2,$2,2
beq $1,$2,loop
loop:
  ori $3,$3,3
```

如图，因为beq用到了$1,$2，而2的结果还没流水经过寄存器，即E_Tnew=1,所以T_use_rt<E_Tnew，即此刻需要暂停，等待$2的ALUResult流水到M级转发回D级进行beq的比较，才能实现正确的行为，可以发现，提前分支判断但却需要阻塞。

2.因为延迟槽之后的指令是一定会执行的，即PC+4的指令已经执行过了，如果将PC+4写入寄存器，在`jr $ra`的时候就会重新跳转回该指令又重新执行一遍，是不符合我们预想的结果的，所以将PC+8写入寄存器，就可以避免重复执行的结果。

3.功能部件输出是有延迟的，如果转发数据采用功能部件，得到的结果不稳定，会产生波动，而经过流水寄存器后，该值是稳定的，不会产生波动，所以我们所有的转发数据都应该来源于流水寄存器，即产生数据的下一个周期转发。

4.GPR内部转发其实就是W级的数据回写，它处理了数据冒险，使得从寄存器中读出的数值是还没有完成回写操作但已经经过运算了的，不会是原先的值。

```
assign RD1 = (A1==A3)&&(A3!=0)? W_WD:registers[A1];
assign RD2 = (A2==A3)&&(A3!=0)? W_WD:registers[A2];
```

5.(**需要即转发,我们根本不需要关心值是否准备好，因为没准备好，程序会自动帮我们阻塞直到能够转发为止，并且我们不需要知道具体转移的值是多少，我只需要这个时候值被更新，我需要用新值**)(想想T_use，就知道需求者和转发者了)

需求者：GRF端口的输出(GPR[rs],GPR[rt]),SrcA,SrcB,MemoryWriteData

供给者：E级的PC回写，M级的ALUResult或PC回写，W级的ALUResult或PC或DMRD回写

转发通路的代码如下:

```
assign D_forward_RD1 = (D_A1==0)? 0:
    (D_A1==E_A3)? E_WD:
    (D_A1==M_A3)? M_WD:
    D_RD1;//W级的转发由grf模块内部转发实现
assign D_forward_RD2 = (D_A2==0)? 0:
    (D_A2==E_A3)? E_WD:
    (D_A2==M_A3)? M_WD:
    D_RD2;
assign E_forward_RD1 = (E_A1==0)? 0:
    (E_A1==M_A3)? M_WD:
    (E_A1==W_A3)? W_WD:
    E_RD1;
assign E_forward_RD2 = (E_A2==0)? 0:
    (E_A2==M_A3)? M_WD:
    (E_A2==W_A3)? W_WD:
    E_RD2;
assign M_forward_RD2 = (M_A2==0)? 0:
    (M_A2==W_A3)? W_WD;
```

6.在我的主线路中mips模块的转发线路可以不变，因为我的转发数据的选取已经封装，所以说如果有新增的数据类型需要转发的，只需要在ctrl模块改变WD即要写的数据即可。

T_use=0：PC型指令，主线路无需修改，只需要修改E级，M级，W级的ctrl模块，对于E_WD,M_WD,W_WD的数据选取即可。

T_use=1：ALU型指令，修改ALU,E级ctrl模块，ALU模块主要改变计算方式和ALUOp，ALUOp来自E_ctrl模块产生，修改即可

T_use=2：DM型指令，修改M级ctrl模块，因为可能读取内存的方式发生变化，需要改变DMOp

7.我的译码器架构是分布式译码，是将指令流水下去，优势是降低了流水级间传递的信号，但是需要实例化多个控制器，增加了后续流水级的逻辑复杂度。