五级流水线CPU设计文档

(一)CPU设计要求

32位五级流水线CPU

支持指令{add, sub, ori, lw, sw, beq, lui, jal, jr, nop }

(二)关键模块设计

IFU

模块定义

信号名	方向	描述
clk	I	时钟信号
Reset	I	异步复位信号
NPCop[1:0]	I	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31...28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
instr_index[26:0]	I	26位输入信号，用于PC的计算
offset[16:0]	I	16位输入信号，PC的偏移量
Reg[32:0]	I	32位输入信号，用于寄存器地址的跳转
Judge	I	一位输入，作为跳转的判断依据
PC_in[31:0]	I	32位输入，D级的PC地址
PCwrite	I	控制是否进行PC修改

信号名	方向	描述
Instr[31:0]	O	32为输出信号，输出当前要执行的指令
PC[31:0]	O	32位输出信号，当前PC的地址

NPC

模块定义

信号名	方向	描述
PC[31:0]	I	32位输入信号，当前指令的地址

信号名	方向	描述
NPCop[1:0]	I	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31…28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
instr_index[26:0]	I	26位输入信号，用于PC的计算
offset[16:0]	I	16位输入信号，PC的偏移量
Reg[32:0]	I	32位输入信号，用于寄存器地址的跳转
Judge	I	一位输入，作为跳转的判断依据
NPC[31:0]	O	32位输出，输出下一条指令的地址

功能定义

序号	功能名称	描述
1	计算PC的下一个值	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31…28}$||instr_index||0_2 10： 结合equal,n_equal,less,big,big or equal,less or equal判断是否需要跳转 PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
2	是否跳转判断	Judge=1，跳转 Judge=0，不跳转

IF_ID

模块定义

信号名	方向	描述
clk	I	时钟信号
reset	I	同步复位信号
IF_ID_en	I	寄存器写入控制信号
PC_F[31:0]	I	32位输入信号，F级的PC
Instr_F[31:0]	I	32位输入信号，F级的指令
PC_D	O	32位输出信号，D级的PC
Instr_D	O	32位输出，D级的指令

CTRL

模块定义

信号名	方向	描述
OP[5:0]	I	6位输入信号，指令操作码
Func[6:0]	I	6位输入信号，指令的func段
RegDst[1:0]	O	GRFA3输入端控制信号 00 : A3←$Instr_{20...16}$ 01 : A3←$Instr_{15...11}$ 11 : A3←0x1f
RegWrite	O	寄存器写入控制信号 0 : 不能向GRF写入 1 : 可以向GRF写入
EXTop	O	功能选择信号 0：Imm无符号拓展到32位 1：Imm符号位拓展到32位
ALUsrc[1:0]	O	ALUSrcB输入控制信号 00 : SrcB←RD2 01 : SrcB←EXTImm 10 : SrcB←sll指令的s SrcA←RD2
ALUctrl[2:0]	O	3位输出信号，选择ALU的功能 000 : SrcA + SrcB 001 : SrcA - SrcB 010 : A |B 011 : A & B 100 : A>>B 101 : $signed(A)>>>B 110 : A<B 置1 111 : A<<B
Menwrite	O	内存写入控制信号 0 : 不能向DM写入 1 : 可以向DM写入
MemtoReg[1:0]	O	控制向寄存器的写入数据 00 : WD←ALUResult 01 : WD←RD 10 : WD←$NPC_{31...0}$ 11 : WD←[$ALUResult_{15...0}$||0_{16}]

信号名	方向	描述

信号名	方向	描述
NPCop[1:0]	O	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31...28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
CMPop[2:0]	O	用于指示进行何种跳转判断 000 : 判断equal 001 : 判断n_equal 010 : 判断less 011 : 判断big 100 : 判断less or equal 101 : 判断big or equal
DMop[1:0]	O	用于lb,sb,lh,sh等操作的拓展 00 : 正常读取，以字为单位 01 : 用于lh或sh Menwrite = 1--->sh Menwrite = 0--->lh 11 : 用于lb或sb Menwrite = 1--->sb Menwrite = 0--->lb

CMP

信号名	方向	描述
RD1[31:0]	I	32位输入，作为要比较的值
RD2[31:0]	I	32位输入，作为要比较的值
CMPop[2:0]	I	用于指示进行何种跳转判断 000 : 判断equal 001 : 判断n_equal 010 : 判断less 011 : 判断big 100 : 判断less or equal 101 : 判断big or equal
Judge	O	一位输出，作为跳转的判断依据

GRF

模块定义

信号名	方向	描述
clk	I	时钟信号

信号名	方向	描述
reset	I	复位信号，将32个寄存器中的值全部清零 1：复位 0：无效
WE	I	写使能信号 1：可向GRF中写入数据 0：不能向GRF中写入数据
A1[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的值读出到RD1
A2[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的值读出到RD2
A3[4:0]	I	5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器
WD[31:0]	I	32位数据输入信号
RD1[31:0]	O	输出A1指定的寄存器的32位数据
RD2[31:0]	O	输出A2指定寄存器中的32位数据

功能定义

序号	功能名称	描述
1	复位	reset信号有效是，所有寄存器存储的数值清零
2	读数据	读出A1，A2地址对应寄存器中所存储的数据到RD1，RD2
3	写数据	当WE有效且时钟上升沿来临时，将WD写入A3对应的寄存器中

EXT

信号名	方向	描述
Imm[15:0]	I	15位输入立即数
EXTop	I	功能选择信号 0：Imm无符号拓展到32位 1：Imm符号位拓展到32位
EXTImm[31:0]	O	32位输出信号，输出Imm拓展之后的数

ID_EX

信号名	方向	描述
clk	I	时钟信号
reset	I	同步复位信号
ID_EX_clr	I	阻塞清零信号
PC_D[31:0]	I	32位输入，D级PC

信号名	方向	描述
A3_D[4:0]	I	5位输入，待写入寄存器编号
RD1_D[31:0]	I	32位输入，从GRF[A1]读出
RD2_D[31:0]	I	32位输入，从GRF[A2]读出
RD1_Sel_D[1:0]	I	RD1的转发控制信号
RD2_Sel_D[1:0]	I	RD2的转发控制信号
EXTImm_D[31:0]	I	拓展后的三十二位立即数
Instr_D[31:0]	I	32为输入，D级的指令
A2_D[4:0]	I	D级使用的寄存器编号
A1_D[4:0]	I	E级使用的寄存器编号
A1_E[4:0]	O	E级使用的寄存器编号
A2_E[4:0]	O	D级使用的寄存器编号
Instr_E[31:0]	O	32为输出，E级的指令
PC_E[31:0]	O	32位输出，E级PC
A3_E[4:0]	O	5位输出，待写入寄存器编号
RD1_E[31:0]	O	32位输出，从GRF[A2]读出
RD2_E[31:0]	O	32位输出，从GRF[A2]读出
EXTImm_E[31:0]	O	拓展后的三十二位立即数
RD1_Sel_D_reg[1:0]	O	RD1的转发控制信号
RD2_Sel_D_reg[1:0]	O	RD2的转发控制信号

ALU

信号名	方向	描述
SrcA[31:0]	I	32位输入信号，第一个操作数A
SrcB[31:0]	I	32位输入信号，第二个操作数B
s[4:0]	I	sll可能会用到的信号

信号名	方向	描述
ALUControl[2:0]	I	3位输入信号，选择ALU的功能 000 : SrcA + SrcB 001 : SrcA - SrcB 010 : A | B 011 : A & B 100 : 101 : 110 :
ALUResult[31:0]	O	32位输出信号，输出运算结果

EX_DM

信号名	方向	描述
clk	I	时钟信号
reset	I	同步复位信号
PC_E[31:0]	I	32位输入，E级PC
A3_E[4:0]	I	5位输入，待写入寄存器编号
Instr_E[31:0]	I	32位输入，E级的指令
RD2_E[31:0]	I	32位输入，从GRF[A2]读出
ALUresult_E[31:0]	I	32位输入，从ALU读出
A2_E[4:0]	I	5位输入，待写入WD的寄存器编号
A2_M[4:0]	O	5位输出，待写入WD的寄存器编号
PC_M[31:0]	O	32位输出，M级PC
A3_M[4:0]	O	5位输出，待写入寄存器编号
Instr_M[31:0]	O	32位输出，M级指令
ALUresult_M[31:0]	O	32位输出
RD2_M[31:0]	O	32位输出，待写入DM的WD端

DM

模块定义

信号名	方向	描述
clk	I	时钟信号

信号名	方向	描述
Reset	I	复位信号，将RAM中的值全部清零 1：复位 0：无效
WE	I	写使能信号 1：可向GRF中写入数据 0：不能向GRF中写入数据
WD[31:0]	I	32位数据输入信号，要写入的数据
A[31:0]	I	32位输入信号，指定RAM中的的一个地址
DMop[1:0]	I	2位输入信号，用于lb，lh等特殊指令 00 ： 正常读写，lw，sw 01 ： 用于lh和sh，根据WE选择进行哪一条指令 10 ： 用于lb和sb，根据WE选择进行哪一条指令
RD[31:0]	O	32位输出信号，读出A指定的地址中的数据

DM_WB

模块定义

信号名	方向	描述
clk	I	时钟信号
reset	I	同步复位信号
PC_M[31:0]	I	32位输入，M级PC
A3_M[4:0]	I	5位输入，待写入寄存器编号
Instr_M[31:0]	I	32位输入，M级的指令
RD_M[31:0]	I	32位输入，DM的输出端
PC_WB[31:0]	O	32位输出，WB级PC
ALUresult_M[31:0]	I	32位输入，从ALU读出
A3_WB[4:0]	O	5位输出，待写入寄存器编号
Instr_WB[31:0]	O	32位输出，WB级指令
RD_WB[31:0]	O	32位输出，DM的输出端的值
ALUresult_WB[31:0]	O	32位输出

HAZARD

模块定义

信号名	方向	描述
clk	I	时钟信号
Instr_D[31:0]	I	D级指令
Instr_E[31:0]	I	E级指令
Regwrite_E	I	E级寄存器写使能
Regwrite_M	I	M级寄存器写使能
PCwrite	O	PC写使能
IF_ID_en	O	IF_ID寄存器写使能
ID_EX_clr	O	ID_EX寄存器清零
Num_use_rs_D[4:0]	O	D指令待使用的rs
Num_use_rt_D[4:0]	O	D指令待使用的rt
Tnew_E_[1:0]	O	D指令待使用rs时间
Tnew_M_[1:0]	O	D指令待使用rt时间

HAZARD_E

模块定义

信号名	方向	描述
Instr_E[31:0]	I	E级指令
Tnew_E[1:0]	O	E级指令的Tnew
Num_new_E[4:0]	O	E级指令待修改寄存器编号

HAZARD_M

模块定义

信号名	方向	描述
Instr_E[31:0]	I	E级指令
Tnew_E[1:0]	I	E级指令的Tnew
Num_new_E[4:0]	I	E级指令待修改寄存器编号
Tnew_M	O	M级指令的Tnew

信号名	方向	描述
Num_new_M	O	M级指令待修改寄存器编号

(三)思考题

1、我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。

提早了判断，导致分支指令的Tuse_rs,Tuse_rt在D级就是0,很容易在待使用的寄存器Tnew还未到0时导致stall，消耗了周期。

```
lw $a0,0($0)
nop
beq $a0,$a1,branch
……
branch xxx
```

这个例子需要beq在D级stall一个周期等待$a0写回，但是如果分支判断在E级,就可以直接通过WB级到E级的转发得到跳转判断信号。

2、因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回 PC + 8，请思考为什么这样设计?

因为J型指令的跳转计算，即NextPc的地址在D级才能算出来，此时F级中已经取出了jal等指令的下一条指令，此条指令的PC值已经是jal的PC值加4了，且这条指令不论是否跳转都会执行的。这条指令不能被重复操作，所以链接地址就需要链接到此时F级指令的下一条，即PC的值再多加4，所以对于jal指令的写回是PC+8。

3、我们要求大家所有转发数据都来源于流水寄存器而不能是功能部件（如 DM 、ALU ），请思考为什么?

首先，在寄存器端的输出数据是稳定的，可以保持一个稳定的信号，其次，这样对于组合逻辑的延迟就会小很多，否则功能部件的组合逻辑时长会加在转发的时长中，导致组合逻辑时间太长，时钟周期就会增加，总效率反而降低。

4、我们为什么要使用 GPR 内部转发？该如何实现?

这个相当于从WB级向E级的转发，需要得到WB级的Regwrite信号和待写入的寄存器编号，如果寄存器编号相同(不为0)且待写入，则直接输出写入的内容，不从RAM中读取。

```
assign RD1 = (A3 == A1 && Regwrite == 1 && A3!=5'b00000) ? WD : regester[A1];
//内部转发,注意0号寄存器
assign RD2 = (A3 == A2 && Regwrite == 1 && A3!=5'b00000) ? WD : regester[A2];
```

5、我们转发时数据的需求者和供给者可能来源于哪些位置？共有哪些转发数据通路?

需求者可能会来自D级的RD1，RD2；E级的ALUsrcA和ALUsrcB；M级的WD；

供给者可能来自E级的PC；M级的ALUresult_M；WB级的WD_WB;

供给者	需求者	供给者	需求者
PC_E	RD1	WD_WB	RD1
PC_E	RD2	WD_WB	RD2
ALUresult_M	RD1	WD_WB	ALUsrcA
ALUresult_M	RD2	WD_WB	ALUsrcB
ALUresult_M	ALUsrcA	WD_WB	WD_M
ALUresult_M	ALUsrcB		

6、在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改？提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。

R型指令

需要拓展ALU的功能，修改CTRL的控制信号，对转发阻塞单元设置Tuse_rt,Tuse_rt,Tnew和相对应的寄存器。

I型指令

如果是立即数操作，可能需要拓展ALU功能，对转发阻塞单元设置Tuse_rt,Tuse_rt,Tnew和相对应的寄存器。

如果是访存操作，可能需要修改DM，并且在外部可能需要流水M级的WD到WB级，或者需要一些特定的判断单元又来判断是否需要写回寄存器堆，最后写入寄存器堆。

J型指令

可能会有需要有条件链接的，需要将Jump信号跟随流水线寄存器流水，辅助判断是否Regwrite，对转发阻塞单元设置Tuse_rt,Tuse_rt,Tnew和相对应的寄存器。**如果是有条件写入，要对Tnew进行特判**

可能会有类似beq的指令有条件的判断跳转，需要拓展CMP模块的功能。对转发阻塞单元设置Tuse_rt,Tuse_rt,Tnew和相对应的寄存器。

7、简要描述你的译码器架构，并思考该架构的优势以及不足。

我使用了分布式译码的方式，优点在于减少了流水线寄存器的流水数据数量，但是缺点是增加了元器件的使用，提高了成本。

在控制单元内部，我使用了类似于logisim的按位译码方式

控制信号每种取值所对应的指令

```
wire add=(Instr[31:26]==6'b000000 && Instr[5:0] == 6'b100000);
wire sub=(Instr[31:26]==6'b000000 && Instr[5:0] == 6'b100010);
……
```

```
    assign RegDst[1]=1'b0|jal;
    assign RegDst[0]=1'b0|add|sub;
    assign Regwrite=1'b0|add|sub|ori|lw|lui|jal;
    assign EXTop=1'b0|lw|sw;
    assign ALUsrc[1]=1'b0;
    assign ALUsrc[0]=1'b0|ori|lw|sw|lui;
    assign ALUctrl[2]=1'b0;
    assign ALUctrl[1]=1'b0|ori;
    assign ALUctrl[0]=1'b0|sub;
    assign Memwrite=1'b0|sw;
    assign MemtoReg[1]=1'b0|lui|jal;
    assign MemtoReg[0]=1'b0|lw|lui;
    assign NPCop[1]=1'b0|beq|jr;
    assign NPCop[0]=1'b0|jal|jr|j;
    assign Branchop[2]=1'b0;
    assign Branchop[1]=1'b0;
    assign Branchop[0]=1'b0;
    assign DMop[1]=1'b0;
    assign DMop[0]=1'b0;
```

这一种译码方式，是对控制信号用了或指令的方式，如果满足这条指令，就会使的控制信号有效，这种方式的优点在于可以很容易的添加指令，对于指令只需要在相应控制信号之后或上一个即可，但是缺点是不够直观，可能会造成漏加信号的错误。