一、模块

取址阶段(F级)

1. PC

Port name	Direction	Type	Description
clk	input		
reset	input		
In	input	[31:0]	下一条指令地址
stall	input		暂停信号
Out	output	[31:0]	当前指令地址

2. IM

Port name	Direction	Type	Description
Addr	input	[11:0]	指令地址
Ins	output	[31:0]	指令

3. MUX_PC

Port name	Direction	Type	Description
PCPlus4	input	[31:0]	PC+4
PCfromNPC	input	[31:0]	NPC的地址
branch	input	[2:0]	0：不跳转，PC+4 其他：NPC
PCAddr	output	[31:0]	下一指令地址

流水线寄存器F

Port name	Direction	Type	Description
clk	input		
reset	input		
stall	input		暂停型号
F_Ins	input	[31:0]	
F_PCPlus4	input	[31:0]	PC地址加4

F_PCAddr	input	[31:0]	PC地址
D_Ins	output	[31:0]	
D_PCPlus4	output	[31:0]	PC地址加4
D_PCAddr	output	[31:0]	PC地址

译码阶段(D级)

1. D_Control

Port name	Direction	Type	Description
Ins	input	[31:0]	
Sign	output	[2:0]	0：0扩展 1：符号扩展 2：加载到高位
Branch	output	[2:0]	0：不跳转 1：beq 2：jal 3：jr
Rs_T_use	output	[2:0]	从D级开始，多少周期后使用Rs(若不用则为3'b111)
Rt_T_use	output	[2:0]	从D级开始，多少周期后使用Rt(若不用则为3'b111)
T_new	output	[2:0]	从D级开始多少周期后更新寄存器(更新的值进入流水线寄存器)，若不更新，则为0

2. GRF

Port name	Direction	Type	Description
clk	input		
reset	input		
D_Rs	input	[4:0]	
D_Rt	input	[4:0]	
W_TargetReg	input	[4:0]	写回阶段写入哪个寄存器
W_Data	input	[31:0]	写回阶段写入寄存器的数
W_RegWrite	input		写回阶段是否写寄存器
WPC	input	[31:0]	当前（处于W级）的指令的地址
D_RsD	output	[31:0]	
D_RtD	output	[31:0]	

3. EXT

Port name	Direction	Type	Description

Port name	Direction	Type	Description
D_Imme16	input	[15:0]	
D_Sign	input	[2:0]	
D_Imme32	output	[31:0]	

4. MUX_DA

   branch时Rs的冲突

Port name	Direction	Type	Description
CDRs	input	[2:0]	1：与当前处于M阶段的指令写回的寄存器冲突 2：与当前处于W阶段的指令写回的寄存器冲突 0:不需转发
D_RsD	input	[31:0]	从寄存器读出的数
M_RegDst	input	[2:0]	M级写哪个寄存器 2：写31号（用于判断转发值）
M_PCPlus8	input	[31:0]	M阶段的PC地址+8
M_ALUAns	input	[31:0]	M级ALU的结果
W_RegDst	input	[2:0]	W级写哪个寄存器 2：写31号（用于判断转发值）
W_PCPlus8	input	[31:0]	W阶段的PC地址+8
W_MemData	input	[31:0]	W级的准备写回的数
B_Rs	output	[31:0]	用于分支判断的数

5. MUX_DB

   branch时Rt的冲突

Port name	Direction	Type	Description

Port name	Direction	Type	Description
CDRt	input	[2:0]	1：与当前处于M阶段的指令写回的寄存器冲突 2：与当前处于W阶段的指令写回的寄存器冲突 0:不需转发
D_RtD	input	[31:0]	从寄存器读出的数
M_RegDst	input	[2:0]	M级写哪个寄存器 2：写31号（用于判断转发值）
M_PCPlus8	input	[31:0]	M阶段的PC地址+8
M_ALUAns	input	[31:0]	M级ALU的结果
W_RegDst	input	[2:0]	W级写哪个寄存器 2：写31号（用于判断转发值）
W_PCPlus8	input	[31:0]	W阶段的PC地址+8
W_MemData	input	[31:0]	W级的准备写回的数
B_Rt	output	[31:0]	用于分支判断的数

6. NPC

Port name	Direction	Type	Description
Addr	input	[31:0]	当前地址指令
Branch	input	[2:0]	0：不跳转 1：beq 2：jal 3：jr
D_Imme	input	[31:0]	
equal	input		
D_Ins	input	[31:0]	
JumpAddr	input	[25:0]	
D_RaData	input	[31:0]	31号寄存器的值
NAddr	output	[31:0]	下一指令地址

7. CMP

Port name	Direction	Type	Description
B_Rs	input	[31:0]	
B_Rt	input	[31:0]	
equal	output		Rs和Rt是否相等

流水线寄存器D

Port name	Direction	Type	Description
clk	input		
reset	input		
stall	input		暂停
D_Ins	input	[31:0]	D级指令
D_Imme	input	[31:0]	立即数，E阶段使用（以及扩展）
D_T_new	input	[2:0]	从D级开始多少周期更新寄存器
D_A	input	[31:0]	寄存器第一个数
D_B	input	[31:0]	寄存器第二个数
D_PCAddr	input	[31:0]	D级指令地址
D_Rs	input	[4:0]	寄存器编号
D_Rt	input	[4:0]	寄存器编号
D_Rd	input	[4:0]	目标寄存器编号
E_Ins	output	[31:0]	E级指令
E_Imme	output	[31:0]	E级立即数
E_T_new	output	[2:0]	max{D_T_new-1,0}
E_A	output	[31:0]	
E_B	output	[31:0]	
E_PCAddr	output	[31:0]	E级指令地址
E_Rs	output	[4:0]	寄存器编号
E_Rt	output	[4:0]	寄存器编号
E_Rd	output	[4:0]	目标寄存器编号

执行阶段(E级)

1. E_Control

Port name	Direction	Type	Description

Port name	Direction	Type	Description
Ins	input	[31:0]	当前指令
RegWrite	output		
ALUOp	output	[3:0]	
ALUSrc	output		进入ALU的数来自寄存器还是立即数
RegDst	output	[2:0]	写入的寄存器 0：rt 1：rd 2：31号

2. ALU

Port name	Direction	Type	Description
E_ALUA	input	[31:0]	
E_ALUB	input	[31:0]	
E_ALUOp	input	[3:0]	执行阶段使用，判断ALU运算种类
E_ALUAns	output	[31:0]	

3. MUX_EA

Port name	Direction	Type	Description
CEA	input	[2:0]	1：与当前处于M阶段的指令写回的寄存器冲突 2：与当前处于W阶段的指令写回的寄存器冲突 0:不需转发
E_A	input	[31:0]	寄存器Rs的值
M_RegDst	input	[2:0]	M级的目标寄存器
M_PCPlus8	input	[31:0]	M级的指令地址+8
M_ALUAns	input	[31:0]	M级的ALU的结果
W_RegDst	input	[2:0]	W级的目标寄存器
W_PCPlus8	input	[31:0]	W级的指令地址+8
W_MemData	input	[31:0]	W级的准备写回的数
E_ALUA	output	[31:0]	进入ALU的数

4. MUX_EB

Port name	Direction	Type	Description

Port name	Direction	Type	Description
CEB	input	[2:0]	1：与当前处于M阶段的指令写回的寄存器冲突 2：与当前处于W阶段的指令写回的寄存器冲突 0:不需转发
E_B	input	[31:0]	寄存器Rs的值
M_RegDst	input	[2:0]	M级的目标寄存器
M_PCPlus8	input	[31:0]	M级的指令地址+8
M_ALUAns	input	[31:0]	M级的ALU的结果
W_RegDst	input	[2:0]	W级的目标寄存器
W_PCPlus8	input	[31:0]	W级的指令地址+8
W_MemData	input	[31:0]	W级的准备写回的数
E_NextB	output	[31:0]	进入下一个多路选择器，选择寄存器值还是立即数的选择器的数，或者写入内存

5. MUX_EImmeReg

Port name	Direction	Type	Description
E_ALUSrc	input		0：来自上一个多路选择器，MUX_EB 1：立即数
E_InB	input	[31:0]	来自上一个多路选择器，MUX_EB
E_Imme	input	[31:0]	立即数
E_ALUB	output	[31:0]	进入ALU的数

6. MUX_Target

Port name	Direction	Type	Description
E_RegDst	input	[2:0]	0：写Rt 1：写Rd 2：写31号
E_Rt	input	[31:0]	
E_Rd	input	[31:0]	
E_TargetReg	output	[31:0]	目标寄存器

流水线寄存器E

Port name	Direction	Type	Description
clk	input		
reset	input		
clr	input		
E_Ins	input	[31:0]	指令
E_TargetReg	input	[4:0]	目标寄存器
E_T_new	input	[2:0]	
E_ALUAns	input	[31:0]	ALU的运算结果
E_NextB	input	[31:0]	转发判断后的结果，存入内存的数值
E_PCAddr	input	[31:0]	指令地址
M_Ins	output	[31:0]	
M_TargetReg	output	[4:0]	目标寄存器
M_T_new	output	[2:0]	
M_ALUAns	output	[31:0]	ALU运算结果
M_WriteData	output	[31:0]	对应E_NextB
M_PCAddr	output	[31:0]	指令地址

存储阶段(M级)

1. M_Control

Port name	Direction	Type	Description

Port name	Direction	Type	Description
Ins	input	[31:0]	
RegWrite	output		是否写寄存器
MemRead	output		是否读寄存器
MemWrite	output		是否写内存
MemWriteOp	output	[2:0]	0：sw
MemReadOp	output	[2:0]	0：lw

2. MUX_DMIN

Port name	Direction	Type	Description
CMI	input	[2:0]	0：不需要转发 1：转发结果
W_RegDst	input	[2:0]	
M_Write	input	[31:0]	不转发写入内存的数
W_PCPlus8	input	[31:0]	指令地址+8
W_Write	input	[31:0]	W级写回的数
M_WriteData	output	[31:0]	真正写入内存的数

3. DM

Port name	Direction	Type	Description
M_ALUAns	input	[31:0]	ALU运算结果
M_WriteData	input	[31:0]	写入内存的数
clk	input		
reset	input		
M_MemWrite	input		是否写内存
M_MemRead	input		是否读内存
M_MemWriteOp	input	[2:0]	0：sw
M_MemReadOp	input	[2:0]	0：lw
PC	input	[31:0]	当前指令地址
M_MemReadData	output	[31:0]	从内存读出的数

流水线寄存器M

Port name	Direction	Type	Description
clk	input		
reset	input		
M_TargetReg	input	[4:0]	写回的寄存器地址
M_T_new	input	[2:0]	
M_ReadData	input	[31:0]	内存读出的数
M_WriteData	input	[31:0]	ALU的结果
M_Ins	input	[31:0]	指令
M_PCAddr	input	[31:0]	指令地址
W_ReadData	output	[31:0]	内存读出的数字
W_ALUData	output	[31:0]	ALU算出的数字
W_TargetReg	output	[4:0]	目标寄存器
W_T_new	output	[2:0]	
W_Ins	output	[31:0]	指令
W_PCAddr	output	[31:0]	指令地址

写回阶段(W级)

1. W_Control

Port name	Direction	Type	Description
Ins	input	[31:0]	指令
RegWrite	output		是否写寄存器
MemToReg	output	[2:0]	写回的数来自内存还是ALU
RegDst	output	[2:0]	目标寄存器

2. MUX_BackDate

Port name	Direction	Type	Description

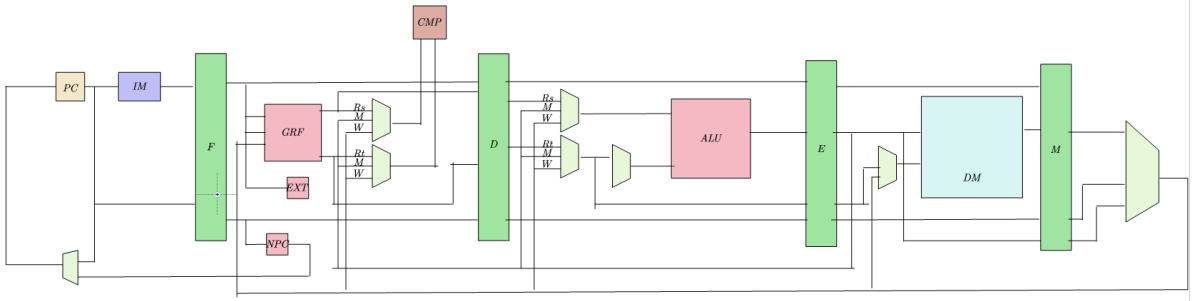
Port name	Direction	Type	Description
W_RegToWrite	input	[2:0]	0：ALU 1：内存 2：PC+8
W_ReadData	input	[31:0]	内存读出的数
W_ALUData	input	[31:0]	ALU的结果
W_PCPlus8	input	[31:0]	地址+8
W_BackData	output	[31:0]	写回的数

冲突处理

Port name	Direction	Type	Description
D_Rs_T_use	input	[2:0]	D级指令过多少个周期使用Rs
D_Rt_T_use	input	[2:0]	D级指令过多少个周期使用Rt
D_Rs	input	[4:0]	D级Rs
D_Rt	input	[4:0]	D级Rt
E_Rs	input	[4:0]	E级Rs
E_Rt	input	[4:0]	E级Rt

Port name	Direction	Type	Description
M_Rt	input	[4:0]	M级Rt
E_TargetReg	input	[4:0]	目标寄存器
M_TargetReg	input	[4:0]	目标寄存器
W_TargetReg	input	[4:0]	目标寄存器
E_T_new	input	[2:0]	E级指令过多少周期更新寄存器
M_T_new	input	[2:0]	M级指令过多少周期更新寄存器
W_T_new	input	[2:0]	W级指令过多少周期更新寄存器
E_RegWrite	input		是否写寄存器
M_RegWrite	input		是否写寄存器
W_RegWrite	input		是否写寄存器
CDRs	output	[2:0]	D级时分支转发判断
CDRt	output	[2:0]	D级时分支转发判断
CEA	output	[2:0]	E级时分支转发判断
CEB	output	[2:0]	E级时分支转发判断
CMI	output	[2:0]	M级时分支判断
Stall	output		暂停信号

二、电路图



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。

lw $t0,0($0)

beq $t0,$0,jump

add

add

...

jump:

如果lw得到的$t0的值不是0，则会比不提前判断读花费时间，因为需要在beq进入D级后暂停F、D，等待lw计入W级后才能从M级寄存器转发回结果

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?

因为这些跳转或分支指令需要到D级才能被识别出来，为了不暂停流水线，则会跟进一条指令进入F级，这条指令不应对程序产生影响，应是对程序无意义的，所以不需要在回到跳转指令时在去执行这条指令，而是应从延迟操槽中指令的下一条指令开始执行。

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?

从寄存器中读出的数是稳定的，而从DM、ALU读出的数据如果尚未稳定就直接转发可能会导致错误。

4、我们为什么要使用 GPR 内部转发？该如何实现?

不需要在冲突处理的单元额外判断从W到D的转发

在从寄存器中读数时加一个多路选择器，如果有冲突，则直接读写回的数，否则读寄存器中的数

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：D级：分支（如beq）的两个数

E级：进入ALU的两个数

M级：写入内存的数

供给者：M级：从E寄存器读出的ALU的结果或PC+8

W级：从M寄存器读出的内存读出的数或PC加8

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。

计算类型的指令：只需要需改ALU部分即可

分支或跳转：改变NPC，以及冲突处理单元以及D级的相关部件

条件读写内存：比较复杂，需要额外的信号流水来判断。

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。

分布式译码：在每一个流水线阶段只产生其需要的信号。

优点：流水的信号较少，代码量少，思路比较清晰，结构更为明朗。

缺点：需要额外流水一些控制信号，在条件读写时起到判断作用。

四、测试方案

通过自己的小量的构造数据，测试流水线对单条指令的执行是否正确，即在指令间插入nop，确保不会有冲突发生。

然后构造小规模数据，验证冲突解决是否正确。

最后通过借鉴其他同学的自动测试工具进行验证。

第一步：采用P4的测试数据，但在指令间加入4条nop

ori $0,$0,0x4454

测试是否会改变0号寄存器的值

ori $7,$0,0x1454

测试指令执行是否正确

ori $7,$0,0xf454

测试是否为0扩展

lui $0,0x4785

测试是否会改变0号寄存器的值

lui $7,0x1454

ori $8,$0,0x1454

lui $8,0x1454

测试指令执行是否正确

ori $7,$0,0x1454

ori $8,$0,0x1454

add $0,$7,$8

add $4,$7,$8

sub $0,$7,$8

sub $4,$7,$8

测试加减法

ori $7,$0,0x1454

sw $7,0,($0)

ori $8,$0,4

lui $11,0x478
add $7,$7,$11

sw $7,0,($8)

sw $7,8,($0)

lw $9,0($0)

lw $10,0($8)

非跳转指令测试完成

ori $t0,$0,0x4

ori $t1,$0,0x4

ori $t2,$0,0x7

beq $t0,$t1,jump

add $t0,$t1,$t0

add $t0,$t0,$t0

jump:

sw $t0,0($0)

测试跳转以及跳转位置是否正确

ori $t0,$0,0x4

ori $t1,$0,0x4

ori $t2,$0,0x7

beq $t0,$t2,jump

add $t0,$t1,$t0

add $t0,$t0,$t0

jump:

sw $t0,0($0)

不跳转的情况是否正确

ori $t0,$0,0x4

ori $t1,$0,0x4

jal jump

add $t0,$t0,$t1

add $t1,$t0,$t1

jal end

jump:

sw $t0,0($t2)
ori $t2,$0,4
jr $ra

end:

第二步：

add $t0,$t1,$t2

add $t4,$t1,$t0

add $t5,$t7,$t0

分别检验从M级以及从W级转发回E级的行为是否正确

lw $t0,0,($0)

sw $t1,4,($t0)

和

lw $t0,0,($0)

add $t2,$t1,$t0

判断暂停行为是否正确

jal func

add $t0,$t1,$t2

add $t0,$t1,$t2

func:

jr $31

nop

判断跳转时延迟槽执行是否正确，以及PC+8的转发是否正确

add $t0,$t1,$t2

beq $t0,$t1,jump

jump：

判断从M到D的转发是否正确

lw $t0,0,($t2)

beq $t0,$t1,jump

jump：

判断从W到D的转发是否正确

第三步：

自动化测试

五、冲突处理

转发采用无脑转发的方式。

暂停，T_new > T_use 时暂停F,D，清空E

如果不更新寄存器，则T_new = 0

若不用寄存器，则T_use = 3'b111