流水线 32 位简单 CPU——Verilog 实现

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 实现的流水线 MIPS - CPU，支持的指令集包含 MIPS-C3 指令。为了实现这些功能，CPU 主要包含了 If，Id，Ex，Mem，Wb，ForwardSel，StallSel 和各流水线寄存器等模块。

（二）关键模块定义

1. If

信号名	方向	描述
clk	I	时钟信号
reset	I	重置
ifStall	I	是否阻塞
ifBranchOrJump_Id	I	是否要用跳转的 pc
pc_Id	I	跳转的 pc
reset_IfId	O	重置流出
pc_IfId	O	pc 流出
instr_IfId	O	指令流出

内置 pcReg，如果跳转，就用 pc_Id 赋值，否则自增 4。

如果阻塞，则保持所有寄存器的值不变（包括内部的 pcReg）。

2. Id

信号名	方向	描述
clk	I	时钟信号
reset_IfId	I	重置信号
pc_IfId	I	pc 流入

instr_IfId	I	指令流入
pc_Wb	I	从 Wb 流入的 pc
grfWa_Wb	I	从 Wb 流入的写寄存器地址
grfWd_Wb	I	从 Wb 流入的写寄存器值
ifWrGrf_Wb	I	从 Wb 流入，是否要写寄存器
grfCmp1_Fs	I	从 ForwardSel 流入转发后的比较器 1 的值
grfCmp2_Fs	I	从 ForwardSel 流入转发后的比较器 2 的值
reset_IdEx	O	reset 流出
pc_IdEx	O	pc 流出
instr_IdEx	O	指令流出
grfRd1_IdEx	O	寄存器 1 的值流出
grfRd2_IdEx	O	寄存器 2 的值流出
immExt_IdEx	O	扩展立即数流出
pcNext_If	O	下一个 pc，给 If 使用
ifBranchOrJump_If	O	是否要用 pcNext，给 If 使用
grfDirRd1_Fs	O	寄存器 1 直接流出（未经转发）的值，给 Fs 统一处理
grfDirRd2_Fs	O	寄存器 2 直接流出（未经转发）的值，给 Fs 统一处理

此外，Id 级生成转发和阻塞使用用的信号，并在之后各级之间传递。

信号名	**描述**
ifReGrf1	是否要读 rs
ifReGrf2	是否要读 rt
ifWrGrf	是否要写 grf
grfRa1	rs
grfRa2	rt
grfWa	写 grf 的地址
grfWd	写 grf 的值
tUseRs	rs 的 tUse
tUseRt	rt 的 tUse

tNew	该指令的 tNew

注意有延迟槽，如果是 jal 指令，写入寄存器的值为 pc+8

注意修改 grf 的信号都要来自 Wb 级，不能用 Id 级的，而读寄存器要用 Id 级的信号

对于 B 或者 J 指令，这一级可以得到全部信息，其中比较器使用转发来的 cmp1、2，这两个信号也流出到下一级作为寄存器 1,2 的值。

3. Controller

控制器接收一个指令，并且译码成想要的控制信号。

aluCtrl：1.无符号加，2.符号加，3.无符号减，4.符号减，5.无符号小于置 1，6.小于置 1，7.逻辑左移，8.逻辑可变左移，9.逻辑右移，10.逻辑可变右移，11.算数右移，12.算数可变右移，13.与，14.或，15.异或，16.或非，17.加载到高位

hiloCtrl：1.无符号乘，2.乘，3.无符号除，4.除，5.读 Hi，6.读 Lo，7.写 Hi，8.写 Lo

loadCtrl：1.lb，2.lbu，3.lh，4.lhu，5.lw

saveCtrl：1.sb，2.sh，3.sw

注意 tUse 和 tNew：

都先把乘除法看成是 alu 简单的组合逻辑，默认很快能出结果，运算时间和阻塞由专门的阻塞单元控制。

注意 nop 指令要特判，对每个控制输出，先判断是不是 nop，如果是就赋值 0，不是就再判断其他的。

4. Ex

信号名	**方向**	**描述**
pc_IdEx	I	pc 流入
instr_IdEx	I	指令流入

reset_IdEx	I	重置流入
immExt_IdEx	I	扩展立即数流入
grfRd1_IdEx	I	grfRd1 流入
grfRd2_IdEx	I	grfRd2 流入
calA_Fs	I	转发来的运算器 A 端输入
calB_Fs	I	转发来的运算器 B 端输入
reset_ExMem	O	重置流出
pc_ExMem	O	pc 流出
instr_ExMem	O	指令流出
aluAns_ExMem	O	alu 运算结果流出
ifBusy_Ss	O	给 StallSel 使用，是否 busy
ifStart_Ss	O	给 StallSel 使用，是否 start

```
assign inB = (ifImmZeroExt | ifImmSignExt)?
                (immExt_IdEx):
                (calB_Fs);
```

注意 aluCtrl 做三目运算时对有符号数的处理方法。

5.Hilo

信号名	**方向**	**描述**
clk	I	pc 流入
reset	I	指令流入
start	I	重置流入
ctrl	I	运算控制
A	I	运算数 1
B	I	运算数 2
rdHi	O	hi 寄存器
rdLo	O	lo 寄存器
busy	O	busy 信号

注意如果是运算操作内部要用寄存器保存 A，B 的值，如果是读写操作就

直接存，不用寄存器。因为运算时会阻塞包括读写的信号，所以就不用考虑运算后读写造成 A，B 错误的情况。

6.Dm

信号名	方向	描述
pc	I	当前 pc
clk	I	时钟
reset	I	重置
rEn	I	读使能
wEn	I	写使能
addr	I	写入地址
dIn	I	写入值
loadCtrl	I	读数选择控制
saveCtrl	I	写数选择控制
dOut	O	读出值

为了尽量使用非阻塞的赋值，只能对写入的情况进行逐一讨论，然后用位拼接输出。

7.各级流水线寄存器

承上启下。如果遇到阻塞信号，IfId 级冻结，IdEx 级清空

7.ForwardSel

转发选择，就用转发点这级之前的流水线寄存器中的 grfRa 与之后各级流水线寄存器中的 grfWa 比较，如果相等且不为零，就选择这一级的 grfWd 作为转发值。都不匹配就选择默认值。

转发顺序：优先先比较靠前的 grfWa。

转发点：Id 中的 cmp1,2，Ex 中的运算数 1,2，Mem 中的 dm 写入值。

8.StallSel

1.grf 的阻塞。首先判断 Id 的指令是否要读寄存器，不读就不用阻塞。如果要写，之后 Ex，Mem 级有一级的判断成立就要阻塞：（同时满足 1.要写寄存器，2.写寄存器地址和 Id 级读寄存器地址相等且不为零，3.Id 级 tUse 小于这一级的 tNew）

2.hiLo 的阻塞。Id 级指令是 8 个乘除相关的指令，且 busy 或 start 有效，就阻塞。

1,2 满足一条整体就阻塞。

（三）重要机制实现方法

1.转发

转发统一由 ForwardSel 处理，利用的是 IdEx，ExMem，MemWb 和默认的数据。注意判断转发要从前往后。

2.阻塞

阻塞信号来源于 StallSel 模块，作用于 If，IfId，IdEx 级。

二、测试方案

（一）典型测试样例

li $4,0x98765432，li $5,0x23456789，li $6,0xfedcba98，li $10,1，li $9,0，add $3,$4,$5，addu $2,$4,$6，and $11,$4,$5，or $12,$5,$4，nor $13,$4,$5，xor $14,$4,$5，move $31,$0，mult $4,$6，mfhi $15，mflo $16，multu $4,$6，mfhi $17，mflo $18，div $4,$6，mfhi $19，mflo $20，divu $4,$6，mfhi $21，mflo $22，nop，nop，nop，nop，nop，#start，，#add/sub,beq/bne，add $7,$5,$4，beq $7,$3,label1，nop，ori $1,$0,1，label1:，sub $7,$7,$5，beq $7,$4,label2，nop，ori $1,$1,2，label2:，，#addu/subu,beq/bne，addu $7,$6,$4，beq $7,$2,label3，nop，ori $1,$0,4，label3:，subu $7,$7,$6，bne $7,$4,label4，nop，ori $1,$1,8，label4:，，#slt/sltu,beq/bne，slt $7,$6,$5#1，bne $7,$10,label5，nop，ori $1,$0,4，

label5:，slt $7,$5,$4#0，beq $7,$10,label6，nop，ori $1,$1,8，label6:，sltu $7,$5,$4#1，bne $7,$10,label7，nop，ori $1,$0,4，label7:，sltu $7,$6,$5#0，bne $7,$10,label8，nop，ori $1,$1,8，label8:，，#slt/sltu,blez/bltz/bgez/bgtz，slt $7,$6,$5#1，blez $7,label9，nop，ori $1,$0,4，label9:，slt $7,$5,$4#0，blez $7,label10，nop，ori $1,$1,8，label10:，sltu $7,$5,$4#1，bgtz $7,label11，nop，ori $1,$0,4，label11:，sltu $7,$6,$5#0，bgtz $7,label12，nop，ori $1,$1,8，label12:，，#sll,srl,sra,sllv,srlv,srav，sll $7,$4,3#<0，bgez $7,label13，nop，ori $1,$0,1，label13:，srl $7,$4,1，bgez $7,label14#>0，nop，ori $1,$1,2，label14:，sra $7,$6,14#<0，blez $7,label15，nop，ori $1,$0,1，label15:，sllv $7,$4,$4#>0，blez $7,label16，nop，ori $1,$0,1，label16:，srav $7,$6,$6#>0，bgtz $7,label17，nop，ori $1,$1,2，label17:，srlv $7,$6,$6，bgtz $7,label18#<0，nop，ori $1,$0,1，label18:，srlv $7,$5,$4，bltz $7,label19#>0，nop，ori $1,$1,2，label19:，srav $7,$4,$5，bltz $7,label20，nop，ori $1,$0,1，label20:，，#and,or,nor,xor，and $7,$4,$5，beq $7,$11,label21，nop，ori $1,$0,1，label21:，or $7,$4,$5，beq $7,$12,label22，nop，ori $1,$0,1，label22:，nor $7,$4,$5，beq $7,$13,label23，nop，ori $1,$0,1，label23:，xor $7,$4,$5，beq $7,$14,label24，nop，ori $1,$0,1，label24:，，#mflo,mfhi，mult $4,$6，，mfhi $7，beq $7,$15,mul1，nop，xori $1,$1,1，mul1:，mflo $7，beq $7,$16,mul2，nop，xori $1,$1,1，mul2:，，multu $4,$6，，mfhi $7，beq $7,$17,mul3，nop，xori $1,$1,1，mul3:，mflo $7，beq $7,$18,mul4，nop，xori $1,$1,1，mul4:，，，div $4,$6，，mfhi $7，beq $7,$19,div1，nop，xori $1,$1,1，div1:，mflo $7，beq $7,$20,div2，nop，xori $1,$1,1，div2:，，divu $4,$6，，mfhi $7，beq $7,$21,div3，nop，xori $1,$1,1，div3:，mflo $7，beq $7,$18,div4，nop，xori $1,$1,1，div4:，beq $0,$0,div4，nop，

三、思考题

（一）

在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数

据通路和控制信号两方面进行说明。

如果不跳转，则 pc 自增 4。如果要跳转，从 Id 级流入 If 一个跳转的信号，和 Id 算出来的 pcNext 信号。跳转信号有效时，就写入这个 pcNext

（二）

对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8？

因为考虑延迟槽，跳转的接下来的指令也会执行，所以以后 jr 返回之后的第 2 条指令，所以要 pc+8

（三）

为什么所有的供给者都是存储了上一级传来的各种数据的**流水级寄存器**，而不是由 ALU 或者 DM 等部件来提供数据？

因为流水级寄存器才表示这一条指令的结果，在 alu 或者 dm 中，结果正在计算，如果从这里提供数据判断自身的输入，可能会造成信号不稳定。

（四）

Thinking 1：如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。

有可能数据不正确。比如

```
ori $2, $0, 100
nop
.....
nop
addu $2, $4, $5
addu $1, $2, $3
```

aluA 端口默认输入转发后的寄存器 2 储存的值。如果用原来的数据，2 号寄存器就可能不是$4 + $5 的数，而是另外的，比如 ori 的结果

Thinking ***2***：我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？

减少和外界的接口。如果不用内部转发，还需要从 Wb 的值来作为 gpr 的写入地址和写入值。

Thinking 3：为什么 0 号寄存器需要特殊处理？

0 号寄存器保持为 0，即使修改也无效。

Thinking 4：什么是"最新产生的数据"？

越靠前的流水线寄存器的数据就是越新的数据。

（五）

在 AT 方法讨论转发条件的时候，只提到了"供给者需求者的 A 相同，且不为 0"，但在 CPU 写入 GRF 的时候，是有一个 we 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了**用且仅用** A 和 T 完成转发，在翻译出 A 的时候，要结合 we 做什么操作呢？

因为如果该判断成立，就说明要写的数据已经生成了，可以直接转发来用。如果不匹配，则写入寄存器为 0，也不会转发。

（六）

在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略，**比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。

我遇到了 alu 的转发处理冲突，解决方案是调换了 alu 输入判断的优先级，先判断靠前的流水线寄存器，再判断靠后的。

测试案例：

addu $1, $1, $2

addu $1, $1, $2

addu $1, $1, $2

手动构造时，先构造所有运算相关的指令，再构造 lw，sw 指令，再结合二者。然后构造跳转相关的指令，插入延迟槽，看是否执行了下面一条，下面第二条不执行，这样可以验证是否跳转，是否执行延迟槽。最后再随机组合。