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

加载：

```
add $1, $0, 2560071317
sw $1, 0($0)
lbu $1, 0($0)
lbu $1, 1($0)
lbu $1, 2($0)
lbu $1, 3($0)
lb $1, 0($0)
lb $1, 1($0)
```

lb $1, 2($0)

lb $1, 3($0)

lhu $1, 0($0)

lhu $1, 2($0)

lh $1, 0($0)

lh $1, 2($0)

lw $1, 0($0)

储存：

add $1, $0, 2560071317

sw $1, 0($0)

sh $1, 4($0)

sh $1, 6($0)

sb $1, 8($0)

sb $1, 9($0)

sb $1, 10($0)

sb $1, 11($0)

运算：

lui $1, 60875

add $2, $0, 43399

add $3, $1, $2

addu $3, $1, $2

sub $3, $1, $2

subu $3, $1, $2

mult $1, $2

mfhi $3

mflo $4

multu $1, $2

mfhi $3

mflo $4

div $1, $2

mfhi $3

mflo $4

divu $1, $2

mfhi $3

mflo $4

slt $3, $1, $2

sltu $3, $1, $2

sll $3, $1, 3

srl $3, $1, 3

sra $3, $1, 3

ori $10, $0, 3

sllv $3, $1, $10

srlv $3, $1, $10

srav $3, $1, $10

and $3, $1, $2

or $3, $1, $2

xor $3, $1, $2

nor $3, $1, $2

addi $3, $1, 2557891634

addiu $3, $1, 2557891634

andi $3, $1, 2557891634

ori $3, $1, 2557891634

xori $3, $1, 2557891634

lui $3, 61441

slti $3, $1, 32749

sltiu $3, $1, 32749

跳跳跳：

ori $1, 1

ori $2, 1

lui $3, 61441

```
beq $1, $2, to1
ori $4, $0, 0
ori $30, $0, 4095
to1:
bne $1, $3, to2
ori $4, $0, 1
ori $30, $0, 4095
to2:
blez $0, to3
ori $4, $0, 2
ori $30, $0, 4095
to3:
bgtz $2, to4
ori $4, $0, 3
ori $30, $0, 4095
to4:
bltz $3, to5
ori $4, $0, 4
ori $30, $0, 4095
to5:
bgez $0, to6
ori $4, $0, 5
ori $30, $0, 4095
to6:
j to7
ori $4, $0, 6
ori $30, $0, 4095
to7:
jal to8
ori $4, $0, 7
```

```
j end
ori $4, $0, 2182
ori $30, $0, 4095
to8:
ori $5, 12420
jalr $3, $5 # to9
ori $4, $0, 8
ori $30, $0, 4095
to9:
jr $31
ori $4, $0, 9
ori $30, $0, 4095
end:
```

HiLo 寄存器：

```
ori $1, $0, 15
mthi $1
mfhi $2
mtlo $1
mflo $2
```

三、思考题

（一）为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

因为乘除法用的时间比较长，不能在一个周期以内出结果。

（二）参照你对延迟槽的理解，试解释“乘除槽”。

乘除槽就类似于延迟槽，如果是冲突指令就阻塞（8 个乘除相关的指令），如果不冲突就继续执行，比如其他的运算指令。

（三）举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。

有可能只需要一个字节，但是如果按字访问，就要再加指令把这个字节提出来。比如取最低的字节，如果只有按字访问，就要先取出整个字，然后和32’h000000ff 做与运算，非常麻烦。

（四）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？

我遇到了最高一位为 0 的情况，结合数字发现之后很多位为 1，于是查到 dm 中关于最高位的处理有问题。

还有评测提示 alu 的问题，只能查运算类指令，发现算数右移操作有误。

很多错误都是 Controller 的问题。建议在写控制器的时候高度集中，能够复制的尽量复制，有必要的话之后重新查一遍。

对于覆盖性，就按照运算类，分支，跳转，读写类查错。关于符号数和无符号数都要查。

（五）为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

在 Controller 中把指令分为 ifRR，ifRI，ifLoad，ifSave，ifBranch，ifJump，ifTrans 这几类。虽然有更方便的处理方法，就是把乘除法单独拿出来，然后在 RR 运算类排除乘除法，可以减少书写量。但是这样逻辑不清楚。我认为还是以意义清楚的分类方法为好。