设计草稿

控制器设计

由于我使用的是控制信号驱动型的译码器，增加指令可以类比原有指令进行

新增的and、or、slt,、sltu指令都是R类型计算指令，类比add指令，只需增加ALU功能即可；

addi、andi是I型计算类指令，类比ori即可

lb、lh、sb、sh是I型存取指令，类比lw和sw，在控制器中增加Byte和Half控制信号

bne条件跳转指令，类比beq即可

对于乘除类指令，需要新增的控制端口为Start、LOWrite、HIWrite、LORead和HIRead

乘除模块设计

```
module multDiv(
    input clk,
    input reset,
    input Start,//启动乘除法信号
    input LOWrite,//写LO寄存器
    input HIWrite,//写HI寄存器
    input [3:0] ALUOp,//模式选择
    input [31:0] D1,//第一个数据
    input [31:0] D2,//第二个数据
    output reg Busy,//表示正在进行乘除运算
    output reg [31:0] LO,//LO寄存器输出
    output reg [31:0] HI//HI寄存器输出
);
```

冲突处理

Tuse和Tnew对应表格

指令	rsTuse	rtTuse	Tnew	指令	rsTuse	rtTuse	Tnew
add	1	1	1	mult	1	1	0

指令	rsTuse	rtTuse	Tnew	指令	rsTuse	rtTuse	Tnew
sub	1	1	1	multu	1	1	0
and	1	1	1	div	1	1	0
or	1	1	1	divu	1	1	0
slt	1	1	1	mfhi	x	x	1
sltu	1	1	1	mflo	x	x	1
ori	1	x	1	mthi	1	x	0
lui	1	x	1	mtlo	1	x	0
andi	1	x	1				
lw	1	x	2				
lb	1	x	2				
lh	1	x	2				
sw	1	2	0				
sb	1	2	0				
sh	1	2	0				
beq	0	0	0				
bne	0	0	0				
jal	x	x	0				
jr	0	x	0				

增加乘除阻塞

当乘除模块正在进行乘除运算（即Start或Busy信号有效）时，D级出现新的乘除相关指令（mult、multu、div、divu、mfhi、mflo、mtlo、mthi）时，需要阻塞

思考题

1、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？

> 因为乘除法比其他的ALU运算慢得多，为了能在进行乘除法时进行其他指令的运算，加快CPU的运行速率，所以将ALU与乘除模块分开。实际上乘除不是一个周期内能完成的，会产生一些中间值，所以需要HI和LO来保存计算的中间值和最后的结果。

2、真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。

> 乘法其实就是进行多个加法运算，首先CPU会初始化三个通用寄存器用来存放被乘数，乘数，部分积的二进制数，部分积寄存器初始化为0；然后再判断乘数寄存器的低位是0还是1，如果为0则将乘数寄存器右移一位，同时将部分积寄存器也右移一位，部分积寄存器低位溢出的一位填充到乘数寄存器的高位，同时部分积寄存器高位补0；如果为1则将部分积寄存器加上被乘数寄存器，再进行移位操作。当所有乘数位处理完成后部分积寄存器做高位乘数寄存器做低位就是最终乘法结果。
>
> 实现除法首先CPU会初始化三个寄存器,用来存放被除数，除数和部分商。余数(被除数与除数比较的结果)放到被除数的有效高位上，CPU做除法时和做乘法时是相反的，乘法是右移，除法是左移，乘法做的是加法，除法做的是减法。首先CPU会把被除数bit位与除数bit位对齐，然后在让对齐的被除数与除数比较。如果得数大于或等于则将比较的结果放到被除数的有效高位上，然后在商寄存器上商1并向后多看一位(上商就是将商的最低位左移1位腾出商寄存器最低位上新的商)如果得数小于则上商0并向后多看一位，然后循环做以上操作当所有的被除数都处理完后，商做结果被除数里面的值就是余数。

3、请结合自己的实现分析，你是如何处理 Busy 信号带来的周期阻塞的？

> 当Busy信号有效时，在D级如果译码出了乘除类指令，就将其阻塞在D级，直到E级乘除的结果获得再停止阻塞。

4、请问采用字节使能信号的方式处理写指令有什么好处？（提示：从清晰性、统一性等角度考虑）

> 能够清晰的看到哪个字节被写入或读出，并且统一了字、半字、字节读写的控制信号，使其只需一个信号就能完成区分。

5、请思考，我们在按字节读和按字节写时，实际从 DM 获得的数据和向 DM 写入的数据是否是一字节？在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢？

> 按字节写入时只替换了DM的一个字节，但按字节读时实际上读出的是一个字，再按照byteen选择对应字节。只需修改DM中一个字节时，使用sb只替换一个字节，若用sw必须先用lw读出修改后再写入。

6、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？

> 对于控制器使用控制信号驱动型的译码器，对指令进行分类，同类型的指令可以类比进行添加，通过这种方式，在译码时只需修改很小一部分代码，并且相比起指令驱动型的译码器减少了每次添加出错的可能性；处理数据冲突时，先利用AT法判断出每种指令Tuse和Tnew，此时同样可以进行类比，比如R型ALU指令的Tuse和Tnew是相同的，最后单独处理乘除类型指令，采用这种抽象的方式将同类型的指令写到一块，减少了出bug的可能性。

7、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？

> 对于新增的R类型ALU指令和I类型ALU指令，冲突的解决与P5相同，测试样例与P5相同，只需将新增指令加入就行。乘除类型指令冲突在于未得到结果时读LO和HI寄存器，所以只需要在Busy或Start有效时将乘除类指令阻塞在D级即可，对应测试样例如下：
>
> ```
> lui $1,$1,0x6789
> ori $1,$1,0xfced
> andi $2,$0,6
> mult $1,$2
> sw $1,0($0)
> lb $3,1($0)
> mflo $4
> mfhi $5
> mtlo $3
> mthi $1
> ```
>
> 然后替换mult为div等重复测试

8、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。

rs策略矩阵

T_{use} \ T_{new}	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F

rt策略矩阵

T_{use} \ T_{new}	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F

> 我是手动构造的样例，对于除乘除之外的指令，依照AT法表格中每种情况进行测试，因为ALU类型指令的相似性，替换不同的ALU类型指令即可测试所有ALU类型指令，同理可替换DM类型和跳转指令达到测试所有指令的冲突情况。对于乘除类型指令，单独构造测试样例，如上题所述。最后对于有符号区别的指令单独进行测试，再添加一些随机指令进行测试。

测试数据

```
a:
jal b
ori $s0,$s0,0x6666
lui $s0,0x5555
sub $t3,$t1,$s0
addi $s0,$0,9
addi $s1,$0,1
addi $t1,$0,0
addi $s2,$0,2
loop_1_begin:
    slt $t4,$t0,$s0
    beq $t4,$0,loop_1_end
    sll $t2,$t0,2
    lui $1,0x0000ffff
    ori $t3,$1,0x0000ffff
    sw $t3,0($t2)
    add $t0,$t0,$s1
    sllv $t2,$t0,$s2
    sub $t3,$t2,$s1
    sw $t3,0($t2)
    addi $t0,$t0,1
    j loop_1_begin
loop_1_end:
ori $2,$2,0x3080
jr $2
nop
b:
addi $t1,$0,100
addi $0,$0,100
sw $t1,0($0)
sb $t1,1($0)
sh $1,2($0)
sw $0,4($0)
add $t2,$t1,$0
ori $t2,$t2,0xabcd
lui $t2,0x8765
sw $t2,8($0)
lw $t3,8($0)
lh $4,6($0)
lb $5,7($0)
lb $5,9($0)
jr $ra
nop
ori $t0,$t0,0xfcfc
lui $t0,0x8f8f
beq $0,$0,c
ori $t1,$t1,100
add $t1,$t0,$t1
c:
sub $t1,$t0,$t1
```

```
sw $t1,0($0)
lw $t2,0($0)
jal d
nop
lui $t2,100
sw $t2,4($0)
and     $a0,$t2,$2
or $3,$a0,$t2
jal e
sltu $3,$2,$4
d:
ori $t2,$0,200
lh $0,0($0)
jr $ra
e:
lui $1,0x6789
ori $1,$1,0xfced
andi $2,$0,6
mult $1,$2
sw $1,0($0)
lb $3,1($0)
mflo $4
mfhi $5
mtlo $3
mthi $1
lui $3,0x8111
ori $3,$3,0x7777
div $1,$3
mflo $4
mfhi $5
divu $1,$2
multu $3,$4
mflo $7
divu $7,$4
mflo $7
mfhi $8
lui $t0,0xffff
ori $t0,$t0,0x1234
lui $s1,0x2333
ori $s1,0xffff
mult $t0,$s1
mfhi $9
mflo $9
multu $t0,$s1
mfhi $9
mflo $9
div $t0,$s1
mfhi $9
mflo $9
divu $t0,$s1
```

```
mfhi $9
mflo $9
```