

实验 3-3 报告

第 28 小组
付琳晴、李舒博

一、实验任务（10%）

增加指令：DIV、DIVU、MULT、MULTU、MFHI、MFLO、MTHI、MTLO、BGEZ、BGTZ、BLEZ、BLTZ、BLTZAL、BGEZAL、JALR。

数据相关前递处理。

乘法：booth 算法+华莱士。

除法：迭代算法。

二、实验设计（30%）

（一）乘法：booth 算法+华莱士

利用 booth 编码（减少加法次数）和华莱士树（利用保留进位加法将累加改为不需要等待进位信号的形式）设计乘法器。

需要进行有符号和无符号运算，所以需要从仅支持 32 位有符号的 32 位部件扩展为 33 位部件，利用有符号数增添一位符号位，无符号数扩张一位 0，进行运算。

（二）除法：迭代算法

乘法器中，因各部分积之间没有直接的递推关系，所以可以同时形成各次部分积，实现快速运算。但在除法运算中，某一步的商值是由该步的余数和除数的比较结果决定的，而该商值又决定下一步的操作，即除法操作需分级递推地进行。

迭代除法（收敛除法）的思想是将除法转换为乘法，以便利用快速乘法器来实现快速除法。分析除法运算可知，两数相除，可以将被除数和除数分别看成是一个分子的分子和分母，若让它们分别乘以某一迭代系数 c_i 序列 ($i = 0, 1, \dots, m$)，使分母迅速收敛于 1，则分子便相应地收敛为商值，这就使除法转换为乘法。

$$\frac{x}{y} = \frac{x \times c_0 \times c_1 \times \dots \times c_m}{y \times c_0 \times c_1 \times \dots \times c_m}$$

对于有符号数，提前根据被除数和除数的符号判断商和余数的符号，然后利用迭代法，获得商和余数的绝对值，再通过先前判断得到的商和余数的符号获得最终的商和余数。

对于无符号数，省略判断符号和结果调整的步骤，直接得到无符号运算的结果。

用 `div_op` 作为除法控制信号，`~div_op | complete` 作为新的就绪信号。

问题 1：可以对 2 位 booth 编码器进行更进一步的优化吗？

可以。减少编码器，努力实现最大化重复利用。

问题 2：为什么在迭代除法器中，用被除数的绝对值除以出书的绝对值就可以得到商的绝对值和余数的绝对值？这和对余数符号的规定有什么关系？

在正常的除法运算中，将相关数字非正值的部分同乘-1 就可以调整成所需绝对值除以绝对值得绝对值形式的结果。余数符号需要和被除数符号相同，因为要相对于正除正得正余正中的被除数和余数同时做出调整。

问题 3：迭代除法可以提前结束吗？

case1：如果余数（被除数）整体小于除数，由于为定点运算部件，不考虑小数点之后的结果，那么就可以完全上商 0，提前结束除法。

case2：参照 2006 年中国科学院计算技术研究所高茁 paper “高性能除法电路仿真与实现”（文章编号 1002-1841（2006）06-0038-02），引入冗余程度 ρ 作为递推收敛约束条件，使 SRT 算法与递推形式的其他线性收敛的除法算法相比，性能大幅度提高。

（三）数据前递处理

数据前递需要解决的是有写回寄存器的指令还未将结果写回寄存器，下一条指令就要用。采用数据前递，即直接将 exe 级、mem 级、wb 级还未来得及写的结果直接前递到 decode 级里用。

将写回寄存器的指令分为 lw 和其余指令两类，遇到前面有指令是 lw 则必须阻塞，因为 lw 要从 data_ram 取数据回来，数据在第四个周期才能取回来，并且只有这一个指令需要写回的是存储器中的数。其余指令需要写回的都是 alu 计算出的结果，在 exe 级已经得到，因此无论是进行到了 exe、mem 还是 wb 级都可以直接把目的数据前递。

修改 ready_go 信号，阻塞的情况只有两种：（1）当前指令的前几条指令中有 lw，且当前指令源操作寄存器号与 lw 目的寄存器号相同。（2）lw 指令前面有 sw 指令。不过第二种情况没有处理，因为跑测试的时候没有遇到，而且，这应该也不算是寄存器的数据相关吧，所以就没加这个判断条件。（如果第四次 task 栽在这里了再加上！）

鉴于从寄存器里直接读出的 rdata1 和 rdata2 已经不靠谱了，所以添加了两个 rdata1_true 和 rdata2_true 的 wire 类型信号，用来存前递过来的当前这个寄存器中应该的数据。以前其他需要用到 rdata1 和 rdata2 的代码部分，全部改成用 true 的值。判断用不用数据前递的条件是：前几条指令有写回指令（RegWrite=1）且 exe 级、mem 级、wb 级中任何一级的目的寄存器和 decode 级源寄存器号相等。当至少有两级的目的寄存器号都等于源

寄存器号，要前递的是最近的数据，因此要注意判断顺序。rdata1_true 的赋值如图 1。

```
assign rdata1_true = (pipe2_valid && (pipe2_data[16:12] == pipe1_data[25:21]) && pipe2_data[17] == 1'b1 && (pipe2_data[11:9] != 3'b1)) ? wdata :  
    (pipe3_valid && (pipe3_data[20:16] == pipe1_data[25:21]) && pipe3_data[21] == 1'b1 && (pipe3_data[15:13] != 3'b1)) ? pipe3_data[215:184] :  
    (pipe4_valid && (pipe4_data[87:83] == pipe1_data[25:21]) && pipe4_data[82] == 1'b1 && (pipe4_data[81:79] != 3'b1)) ? pipe4_data[119:88] :  
    rdata1;
```

图 1. rdata1_true 代码

(四) 指令添加

此次需要添加七条跳转指令，跳转的线路之前都有，只需要增加这几条指令的判断条件即可。相关判断条件如图 2。

```
280 assign ALUSrcB2 = (  
281     (BNE && !zero)  
282     || (BEQ && zero)  
283     || (BGEZ && ~rdata1_true[31])  
284     || (BGTZ && ~rdata1_true[31] && (rdata1_true | 32'b0))  
285     || (BLEZ && (rdata1_true[31] || rdata1_true == 32'b0))  
286     || (BLTZ && rdata1_true[31])  
287     || (BLTZAL && rdata1_true[31])  
288     || (BGEZAL && ~rdata1_true[31])  
289     ) ? 1:0;  
290
```

图 2. offset 计算信号代码

剩余八条与乘除法相关的指令，在乘除法模块成功加入流水线后非常容易实现，大部分都是与特殊寄存器 LO、HI 相关的数据操作，需要在代码中设相应的寄存器即可。

(五) 乘除法 IP

乘法 IP 设两个输入端口都是 33 位，处理周期设置为 2，两个源操作数根据指令是 MULT 还是 MULTU 进行位扩展，在 exe 级送入乘法 IP，理论上会在 wb 级得到结果送回来。但是实际实现中，流水线当遇到乘法指令会阻塞 decode 级，为结果写入 HI 和 LO 留出时间。并且这个阻塞会在四个周期后自动解除，因为这里阻塞用到了每一级流水线的 pipe_valid 信号，当阻塞之后 exe、mem、wb 级的 valid 依次变成 0 后会使 ready_go 重新变回 1，继续进行流水线。（其实这个信号我写进代码时并没有发现有什么奇妙之处，就顺手添加的，但是室友告诉我她们的乘法器阻塞之后会一直阻塞下去，于是我重新看了代码才发现是 valid 的奇妙作用...）

除法器 IP 端口比较多，如图 3. 其中 tready 和 tvalid 信号用来做握手信号，保证除法数据只送入一次。后来观察波形发现 tready 信号会出现周期性上升。代码中将 tvalid 信号设为 reg 类型，并受 tready 信号控制，当 tready 信号为 1 时，tvalid 将在下一个周期下降，保证了数据只输入一次。在设置 IP 时候，将输入数据位数设为 33 位，发现端口自动把位数扩到了 40 位，于是结果算出来一共 80 位，我们用到的商和余数都要取低 32 位。

```

732 mydiv div1(
733     .s_axis_divisor_tdata    (s_axis_divisor_tdata ),
734     .s_axis_divisor_tready   (s_axis_divisor_tready ),
735     .s_axis_divisor_tvalid   (s_axis_divisor_tvalid ),
736     .s_axis_dividend_tdata   (s_axis_dividend_tdata ),
737     .s_axis_dividend_tready   (s_axis_dividend_tready),
738     .s_axis_dividend_tvalid   (s_axis_dividend_tvalid),
739     .aclk                     (clk ),
740     .m_axis_dout_tdata        (m_axis_dout_tdata ),
741     .m_axis_dout_tvalid       (m_axis_dout_tvalid ),
742 );
743

```

图 3. 除法器 IP 调用

(六) HI、LO 寄存器

需要新设置两个寄存器 HI 和 LO 用来满足乘除法的特殊设置。代码中 HI 和 LO 中数据的来源有三种情况：

(1) MTHI、MTLO 指令：将寄存器数存入。(2) MULT 和 MULTU 指令。(3) DIV 和 DIVU 指令。三种情况的数据写入信号也不同。需要注意的是，以下情况会造成数据相关。mflo 紧接着 mtlo，使 LO 还没有写入数据就要被取出，导致程序出错。因此要考虑 HI 和 LO 的数据前递。

```

105436 bfc66a10 <n49_mflo_test>:
105437 n49_mflo_test():
105438 bfc66a10: 26100001 addiu s0,s0,1
105439 bfc66a14: 24120000 li s2,0
105440 bfc66a18: 3c0808fc lui t0,0x8fc
105441 bfc66a1c: 01000013 mtlo t0
105442 bfc66a20: 00001012 mflo v0
105443 bfc66a24: 3c1508fc lui s5,0x8fc

```

图 4. 数据相关代码

在代码中增加了 HI_true 和 LO_true 信号，类似于 rdata1_true 和 rdata2_true，都是用来数据前递的。判断条件是前面指令要在 HI 和 LO 中写数，后面指令紧接着要用到 HI 和 LO，此时直接将 exe 级的结果送到源操作数处。

三、实验过程（60%）

(一) 实验流水账

付琳晴：

10.21：改数据前递。

10.22：添加 7 条跳转指令。

10.23：用乘除法器 IP，加剩下 8 条乘除法相关的指令，debug。

李舒博：

10.21 晚：看 booth 算法、华莱士树相关介绍

10.22 早：看迭代除法相关介绍

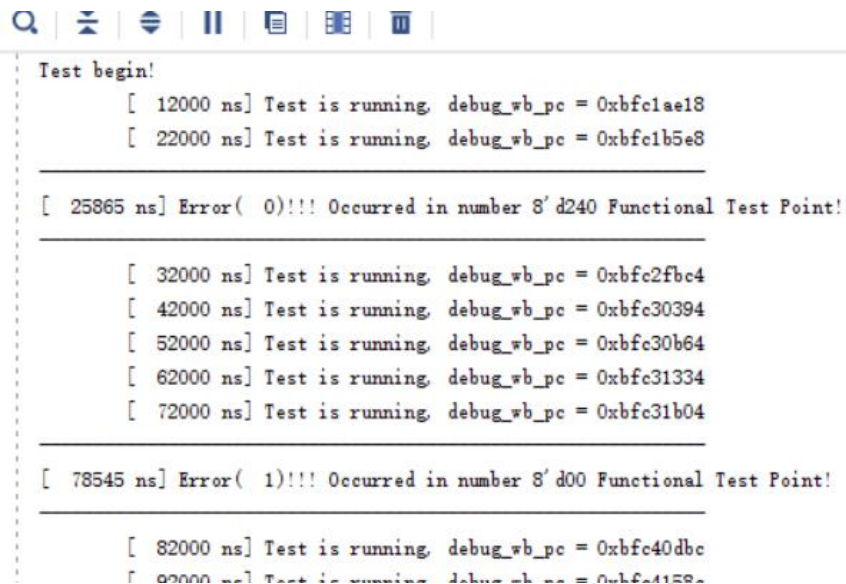
10.22：写乘除法代码、撸实验报告

（二）错误记录

1、错误 1

（1）错误现象

打印信息出现 error 但没有影响指令继续执行，如图 5。



```
Test begin!
[ 12000 ns] Test is running, debug_wb_pc = 0xbfc1ae18
[ 22000 ns] Test is running, debug_wb_pc = 0xbfc1b5e8

[ 25865 ns] Error( 0)!!! Occurred in number 8'd240 Functional Test Point!

[ 32000 ns] Test is running, debug_wb_pc = 0xbfc2fbc4
[ 42000 ns] Test is running, debug_wb_pc = 0xbfc30394
[ 52000 ns] Test is running, debug_wb_pc = 0xbfc30b64
[ 62000 ns] Test is running, debug_wb_pc = 0xbfc31334
[ 72000 ns] Test is running, debug_wb_pc = 0xbfc31b04

[ 78545 ns] Error( 1)!!! Occurred in number 8'd00 Functional Test Point!

[ 82000 ns] Test is running, debug_wb_pc = 0xbfc40dbc
[ 88000 ns] Test is running, debug_wb_pc = 0xbfc4158a
```

图 5. Error 信息

（2）分析定位过程

求助室友，室友表示是 sw 指令没有执行对，让我找到 num_data 信号波形，若出现高两位和低两位不一致的情况就是出错的地方。由此果然定位到了 sw 指令前面的 or 指令出错。

（3）错误原因

给 exe 流水级输的 rdata2 是寄存器被写之前的值，但其实需要的是被上条指令写完后的值。

（4）修正效果

事实上已经另设了 rdata2_true 信号，是寄存器中真实值（因为进行了数据前递），只不过代码中原来 rdata2 信号没有被替换成 rdata2_true，动手把所有以前用到 rdata2 信号全改换成 true 信号，便对了。

2、错误 2

（1）错误现象

波形初始化后全高阻态，没法取下一条指令。

（2）分析定位过程

没有取下一条指令说明被阻塞了，检查 ready_go 信号，发现复位信号结束后 decode 级的 ready_go 是高阻态。

（3）错误原因

decode 级的 ready_go 信号依赖了后面几级流水的 data 信号，但刚开始数据信号还没传到后几级流水，因此后

面几级流水里的信号都是高阻态，导致 ready_go 信号也是高阻态。

(4) 修正效果

更改了 ready_go 信号。加入了后面几级流水的 valid 信号，修改过后 ready_go 信号代码如图 6。由于 valid 信号在复位时均有信号，为 0，因此 ready_go 也有信号。

```
assign pipe1_ready_go = !(( pipe2_valid && pipe2_data[11:9] == 3'b1 && (pipe2_data[16:12] == rs || pipe2_data[16:12] == rt))
|| ( pipe3_valid && pipe3_data[15:13] == 3'b1 && (pipe3_data[20:16] == rs || pipe3_data[20:16] == rt))
|| ( pipe4_valid && pipe4_data[81:79] == 3'b1 && (pipe4_data[87:83] == rs || pipe4_data[87:83] == rt)) );
```

图 6. decode 级 ready_go 信号

3、错误 3

(1) 错误现象

用除法器 IP 算除法后结果无法及时送到 HI 和 LO 寄存器供下条指令用。

(2) 分析定位过程

MFHI 写数据出错，说明是之前的除法运算结果没有正确写入 HI，HI 又没有正确作为源操作数之一。查看 HI 波形，发现 HI 真正写入运算结果的时候已经来不及传入前面作为源操作数了，后面指令取到的 HI 值是错的。

(3) 错误原因

当除法运算得到后，流水线不阻塞，开始继续进行，但由于 HI 和 LO 两个寄存器写入值需要等到下个时钟沿，导致 MFHI 和 MFLO 指令用到的数据错误。

(4) 修正效果

采用了数据前递，即直接将除法器算出的结果作为下条指令需要的目的操作数送到 deccode 级。并将除法器输出结果信号 m_axis_dout_tvalid 作为传递数据的判断信号。并且只有这一种用到 HI 或 LO 的情况需要数据前递，若指令前面并不是紧跟除法计算，m_axis_dout_tvalid 此时是 0，则用 HI 和 LO 寄存器的数据。

(5) 归纳总结

Debug 真痛苦，尤其是调用一个黑箱子。输入输出信号要考虑很多条件，修修补补才对。

4、错误 4

(1) 错误现象

DIV 指令未阻塞，仅仅执行了一拍就取了下条指令。

(2) 分析定位过程

取下一条指令说明阻塞条件有问题，检查 decode 级的 ready_go 信号，发现钻了老师测试代码的空子，阻塞条件是当前指令是 MFHI 或 MFLO 并且前面有除法指令。但其实这样是不对的，当除法指令后面跟着其他指令时候，除法指令不能被识别出来。

(3) 错误原因

div 指令阻塞的判断条件没考虑完全。（事实上所有指令前面有 div 都要阻塞）

(4) 修正效果

此处依然有 bug 没有解决。理论上只需要判断前面流水中有没有除法指令，有就阻塞 decode 级，但是实际实施起来，这样赋值会导致最初 ready_go 信号高阻态（判断信号在最初都没传到后面，是高阻态）。不知道怎么做才能让 ready_go 信号初始有信号。最终钻了一个测试用例的空子，测试用例中 div 后面的指令只有三种情况 MFHI、MFLO、ADDU，因此我只用了这三条指令结合前面流水中有没有除法指令作为判断条件对 div 进行阻塞。由于多问了助教一句，我觉得在 func4 里助教肯定会增加各种用例的...很后悔告诉了助教自己的钻空子历程...属于自己给自己挖坑。

5、错误 5

(1) 错误现象

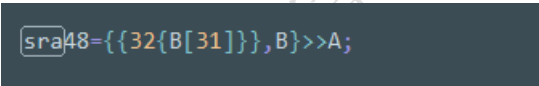
移位指令 SRAV 指令执行错误，移位结果出错。

(2) 分析定位过程

分析两个源操作数送入应该是正确的，一定是 ALU 计算出现问题，在 ALU 模块中发现是代码写的不准确导致的。

(3) 错误原因

移位指令的移位数来自 A 的低 5 位，但代码中用的 A 作为移位数，导致出错，错误如图 7。



```
sra48 = {{32{B[31]}}}, B >> A;
```

图 7. 错误代码

(4) 修正效果

将 >>A 改为 >>A[4:0]，波形正确。

(5) 归纳总结

怀疑 vivado 对这种代码的识别有不确定性，跑一次可能跑过，第二次就跑不过，不然为什么在第二阶段时候没有报错。看来代码还是应该写的规范一点。

四、实验总结

（一）组员：付琳晴

大体框架建好之后添跳转指令很顺，大部分线路都是原来的，只需要添加些控制信号情况即可。极感谢队友的代码框架，上星期写这个框架时候 debug 一定很费劲。队友写了乘除法模块，解决了最难的部分，极棒。

周一晚上开始添 IP，问助教一些问题，助教还问我进度怎么样了来着，我还一脸自信说就差把 IP 塞进去了... 结果完全没想到添两个 IP 这么难，哭了，各种控制信号各种连，直到最后即使 PASS 了线路其实还是不怎么好，有种投机取巧的画风。然后发现 IP 和队友写的乘除法模块信号好像还很多不一样，已经预料到以后改代码的痛苦...

（二）组员：李舒博

队友专治拖延！夸！及时拽出被 os 埋了半截身子的渣渣！

用从队友那里学来的 sublime 徒手写了乘除法模块！没往流水线里装！是我！

徒手写完模块心里感觉虚，手抖把实验报告模板写废了，上 sep 我发现这乘除法器有讲义 orz。面壁思过

五、参考文献

1. 刘慧英，戴春蕾，高苗：“高性能除法电路仿真与实现”

链接：<http://sourcedb.ict.cas.cn/cn/ictthesis/200907/P020090722612840282646.pdf>