**实验六报告**

学号 2017K8009929044

2017K8009929025

姓名 李昊宸 李颖彦

箱子号 33

一、实验任务（10%）

(1) 深入理解译码和执行两个流水级的工作过程。

(2) 掌握在流水线 CPU中添加运算类指令的方法。

二、实验设计（40%）

（一）总体设计思路

在ID阶段完成新增指令的译码与数据通路的更新，在exe模块中新增hi、lo寄存器的使用，乘除法，其中乘法通过vivado自行调用ip，除法通过调用指定ip实现。

（二）重要模块1设计：译码模块

1. 工作原理

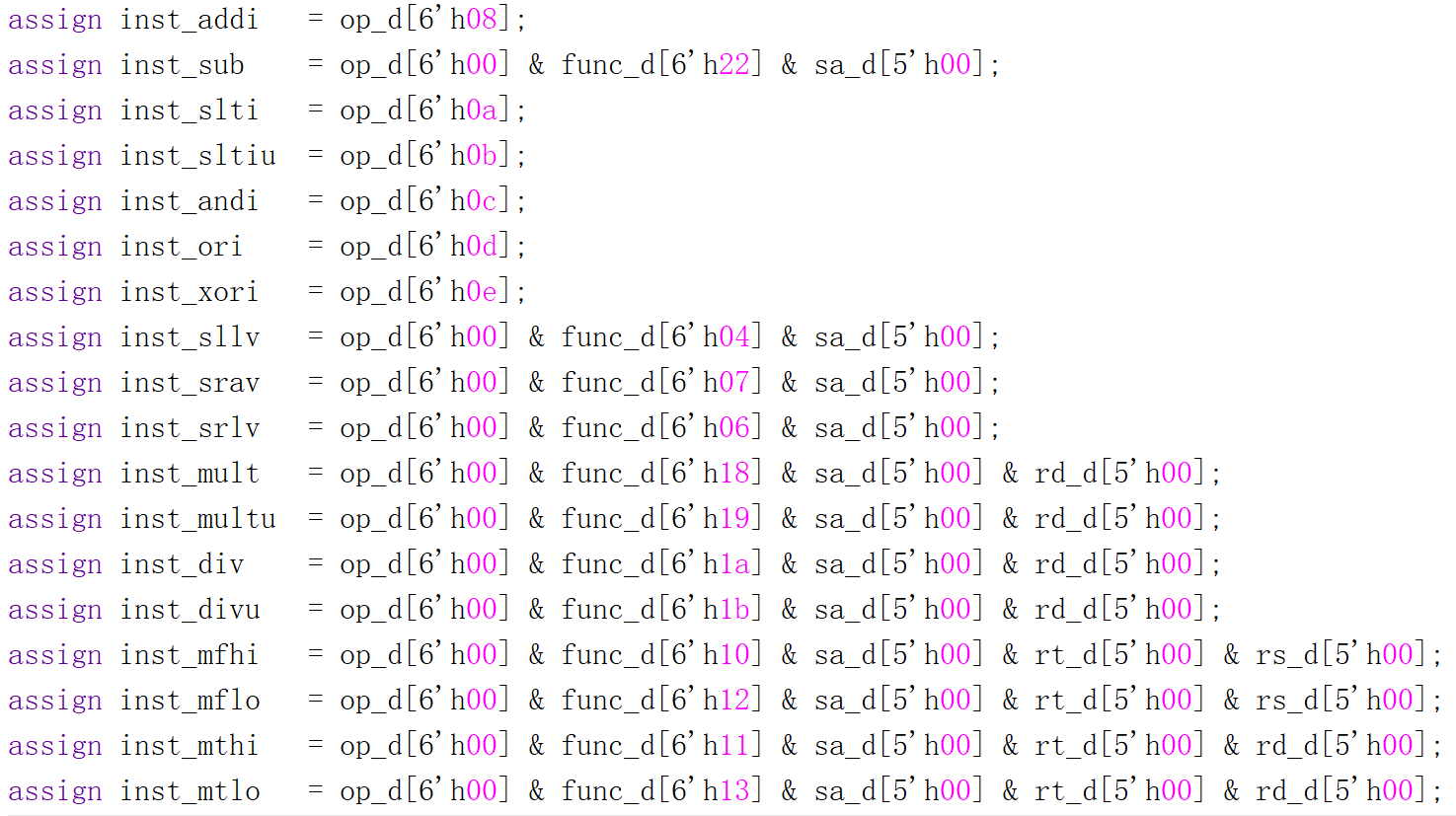
新增指令并译码，并相应的更新数据通路。

1. 功能描述

新增指令分为3类：算数逻辑指令：（本文将其分为3类）ADD,ADDI…..ANDI,ORI,…,SLLV…,乘除指令，乘除法对应的搬运指令

1. 具体代码实现

先来看看译码，依照mips规定即可。

重点是数据通路的更新：

A）算数逻辑指令

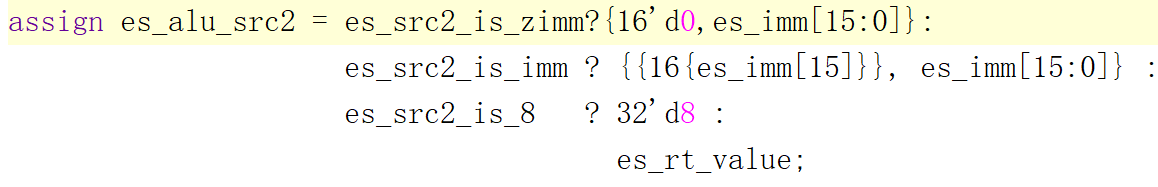
1.对于ADD,ADDI,SUB,SLTI,SLTIU,完全可以复用当前数据通路，不再概述。

2.对于ANDI,ORI,XORI，可以复用i型指令的数据通路，但要注意的是，此时立即数进行的是零拓展，这意味着需要新增数据通路。

在id模块中:



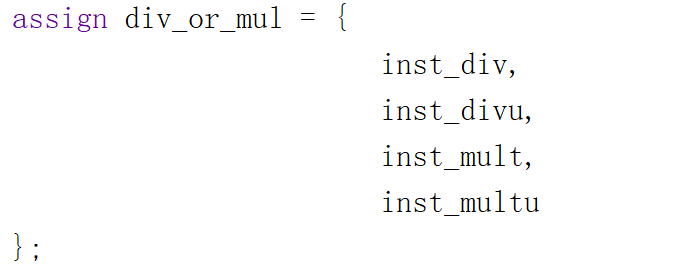
之后，在exe模块中：



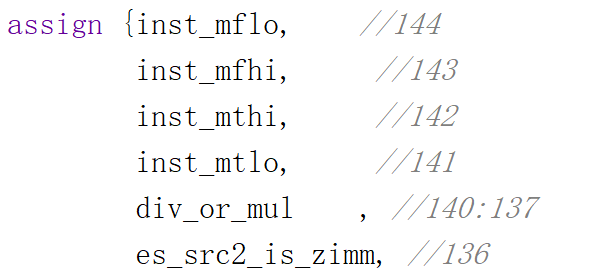
3.对于SLLV,SRAV,SRLV，同样复用数据通路，但要注意，不能用SLL等的直接数据通路，因为此时偏移量在寄存器而不是sa中。

B)乘除指令

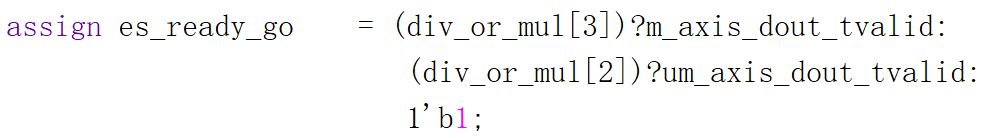
在id中：



之后，新增id与exe之间的总线接口：

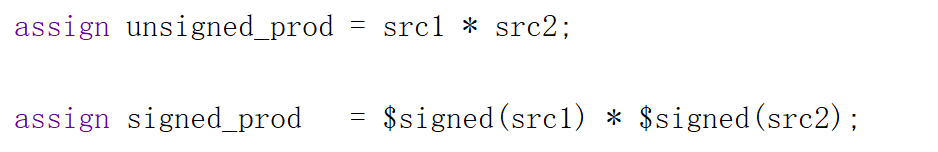


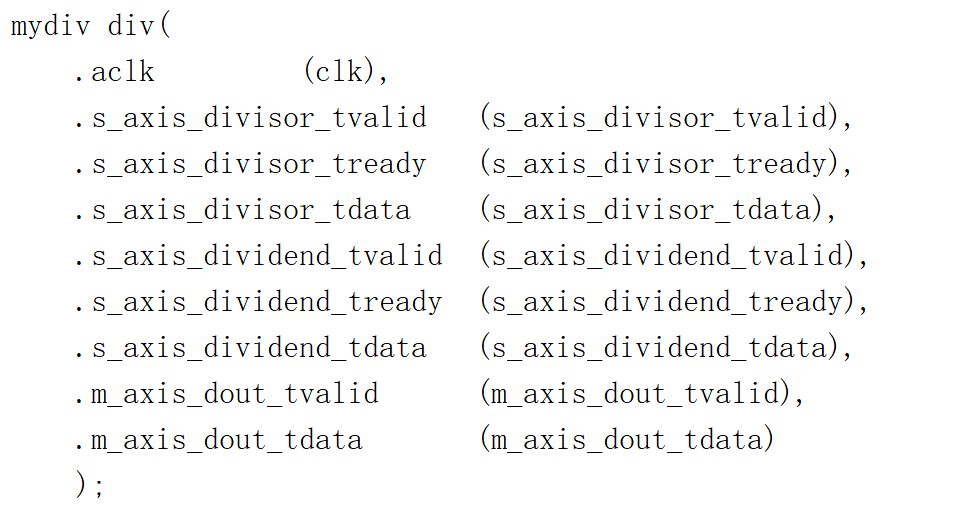
因为调用除法ip需要多排才能出结果，所以流水线需要堵塞：（更多细节见下面的除法模块）



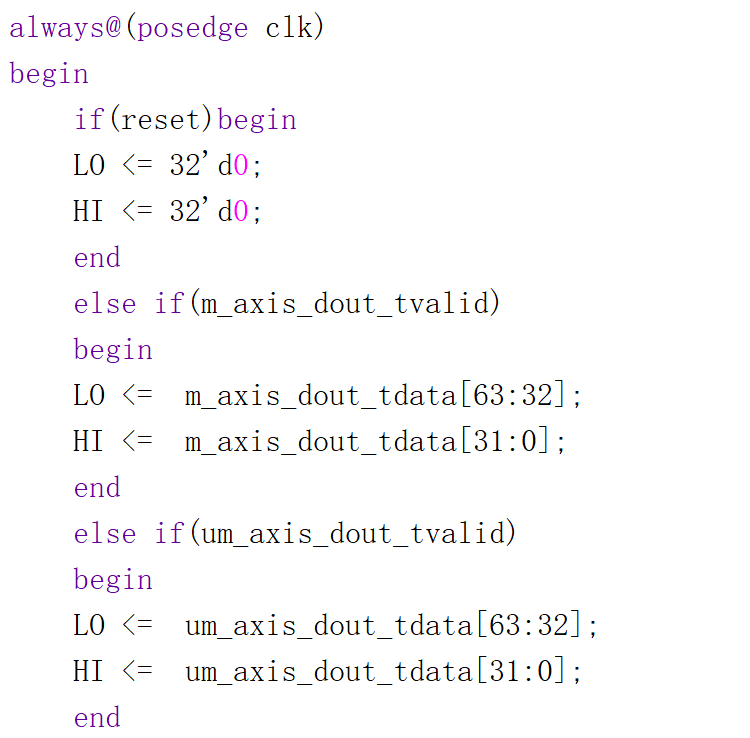
第一行是有符号除法，第二行是无符号除法。

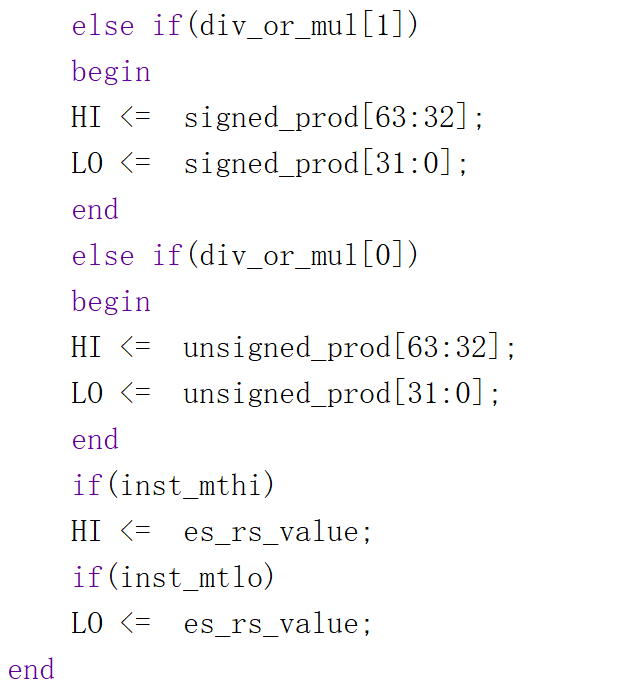
之后运算：





输出结果后：

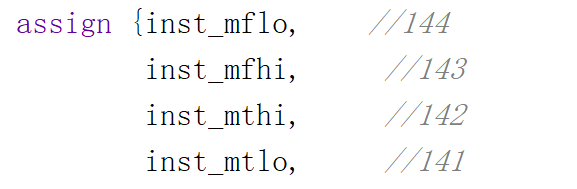




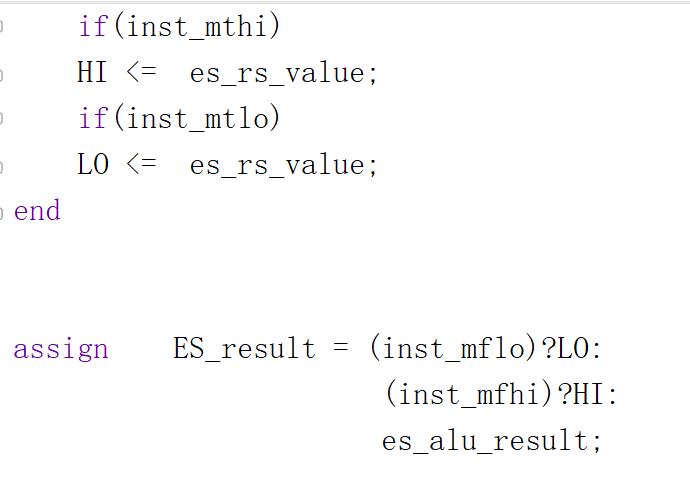
写入hi，lo寄存器即可。

C）乘除法数据搬运指令

首先，将指令从id传至exe



之后，进行读或写操作：



（二）重要模块2设计：除法模块

1. 工作原理

调用除法器IP来实现除法逻辑的运算。在IP Catalog中引入Divider Generator，设置算法种类为Radix2，被除数宽度和除数宽度均设置为32位，余数类型设置为整数，设置8个除法器拍数处理一次除法，在数据出口选择Non Blocking模式，这样在结果生成之后，后续数据通路可以直接取走除法结果，不必被握手信号阻塞在这里。考虑到除法指令有无符号除法和有符号除法，并且计算的结果不同，需要设置两个除法器IP，一个设置为signed，另一个设置为unsigned。

1. 接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| aclk | IN | 1 | 时钟 |
| s\_axis\_divisor\_tvalid | IN | 1 | 传入除数信号有效 |
| s\_axis\_divisor\_tready | OUT | 1 | 除数通道可以接受输入 |
| s\_axis\_divisor\_tdata | IN | 32 | 输入除数数据 |
| s\_axis\_dividend\_tvalid | IN | 1 | 传入被除数信号有效 |
| s\_axis\_dividend\_tready | OUT | 1 | 被除数通道可以接受输入 |
| s\_axis\_dividend\_tdata | IN | 32 | 输入被除数数据 |
| m\_axis\_dout\_tvalid | OUT | 1 | 商和余数输出有效 |
| m\_axis\_dout\_tdata | OUT | 32 | 商和余数数据输出有效 |

表一 除法器IP AXI接口定义

1. 功能描述

我们把除法器IP放在执行级。对于该IP的功能，简单来说可以分为以下几步：

1. 被除数与除数准备好之后，两个s\_axis\_tvaild信号拉高，准备等待除法器响应；同时将exe级阻塞在这里。
2. 除法器输出两个s\_axis\_tready信号拉高，在时钟上升沿时将被除数与除数输入读入，开始除法运算；同时应该将两个tvalid信号手动拉低，避免下一拍时钟到来时有另外的信号读入。
3. 等待其执行完毕（实测32拍），除法器会把m\_axis\_dout\_tready拉高，此时将输出数据存入HI和LO寄存器。随后把exe级的阻塞释放。
4. 具体代码实现
5. 变量声明

有符号除法部分：

*reg s\_axis\_divisor\_tvalid;*

*wire s\_axis\_divisor\_tready;*

*wire [31:0] s\_axis\_divisor\_tdata;*

*reg s\_axis\_dividend\_tvalid;*

*wire s\_axis\_dividend\_tready;*

*wire [31:0] s\_axis\_dividend\_tdata;*

*wire m\_axis\_dout\_tvalid;*

*wire [63:0] m\_axis\_dout\_tdata;*

无符号除法部分：

*reg us\_axis\_divisor\_tvalid;*

*wire us\_axis\_divisor\_tready;*

*wire [31:0] us\_axis\_divisor\_tdata;*

*reg us\_axis\_dividend\_tvalid;*

*wire us\_axis\_dividend\_tready;*

*wire [31:0] us\_axis\_dividend\_tdata;*

*wire um\_axis\_dout\_tvalid;*

*wire [63:0] um\_axis\_dout\_tdata;*

1. 逻辑通路

*always @(posedge clk) begin*

*if (reset) begin*

*s\_axis\_divisor\_tvalid <= 1'b0;*

*s\_axis\_dividend\_tvalid <= 1'b0;*

*us\_axis\_divisor\_tvalid <= 1'b0;*

*us\_axis\_dividend\_tvalid <= 1'b0;*

*last <= 1'b0;*

*end* //last信号用来表示除法是否在执行，拉高为正在执行，所以阻塞

*else if(s\_axis\_divisor\_tready & s\_axis\_dividend\_tready )begin*

*s\_axis\_divisor\_tvalid <= 1'b0;*

*s\_axis\_dividend\_tvalid <= 1'b0;*

*end*

*else if(us\_axis\_divisor\_tready & us\_axis\_dividend\_tready )begin*

*us\_axis\_divisor\_tvalid <= 1'b0;*

*us\_axis\_dividend\_tvalid <= 1'b0;*

*end* //当tready信号被拉高后，下一拍时钟上升沿后把tvalid信号拉低

*else if (div\_or\_mul[3] & ~last) begin*

*s\_axis\_divisor\_tvalid <= 1'b1;*

*s\_axis\_dividend\_tvalid <= 1'b1;*

*last <= 1'b1;*

*end*

*else if (div\_or\_mul[2] & ~last) begin*

*us\_axis\_divisor\_tvalid <= 1'b1;*

*us\_axis\_dividend\_tvalid <= 1'b1;*

*last <= 1'b1;*

*end* //div\_or\_mul信号用来记录当前在exe阶段执行的是否是除法指令

//如果是并且还没开始执行，就将两个tvalid信号拉高，并且将last拉高

*if(m\_axis\_dout\_tvalid)*

*last <= 1'b0;*

*else if(um\_axis\_dout\_tvalid)*

*last <= 1'b0;*

//重置

*end*

*assign es\_ready\_go = (div\_or\_mul[3])?m\_axis\_dout\_tvalid:*

*(div\_or\_mul[2])?um\_axis\_dout\_tvalid:*

*1'b1;*

*assign s\_axis\_divisor\_tdata = es\_rt\_value;*

*assign s\_axis\_dividend\_tdata = es\_rs\_value;*

*assign us\_axis\_divisor\_tdata = es\_rt\_value;*

*assign us\_axis\_dividend\_tdata = es\_rs\_value;*

1. 例化

*mydiv div(*

*.aclk (clk),*

*.s\_axis\_divisor\_tvalid (s\_axis\_divisor\_tvalid),*

*.s\_axis\_divisor\_tready (s\_axis\_divisor\_tready),*

*.s\_axis\_divisor\_tdata (s\_axis\_divisor\_tdata),*

*.s\_axis\_dividend\_tvalid (s\_axis\_dividend\_tvalid),*

*.s\_axis\_dividend\_tready (s\_axis\_dividend\_tready),*

*.s\_axis\_dividend\_tdata (s\_axis\_dividend\_tdata),*

*.m\_axis\_dout\_tvalid (m\_axis\_dout\_tvalid),*

*.m\_axis\_dout\_tdata (m\_axis\_dout\_tdata)*

*);*

*mydivu divu(*

*.aclk (clk),*

*.s\_axis\_divisor\_tvalid (us\_axis\_divisor\_tvalid),*

*.s\_axis\_divisor\_tready (us\_axis\_divisor\_tready),*

*.s\_axis\_divisor\_tdata (us\_axis\_divisor\_tdata),*

*.s\_axis\_dividend\_tvalid (us\_axis\_dividend\_tvalid),*

*.s\_axis\_dividend\_tready (us\_axis\_dividend\_tready),*

*.s\_axis\_dividend\_tdata (us\_axis\_dividend\_tdata),*

*.m\_axis\_dout\_tvalid (um\_axis\_dout\_tvalid),*

*.m\_axis\_dout\_tdata (um\_axis\_dout\_tdata)*

*);*

三、实验过程（50%）

（一）实验流水账

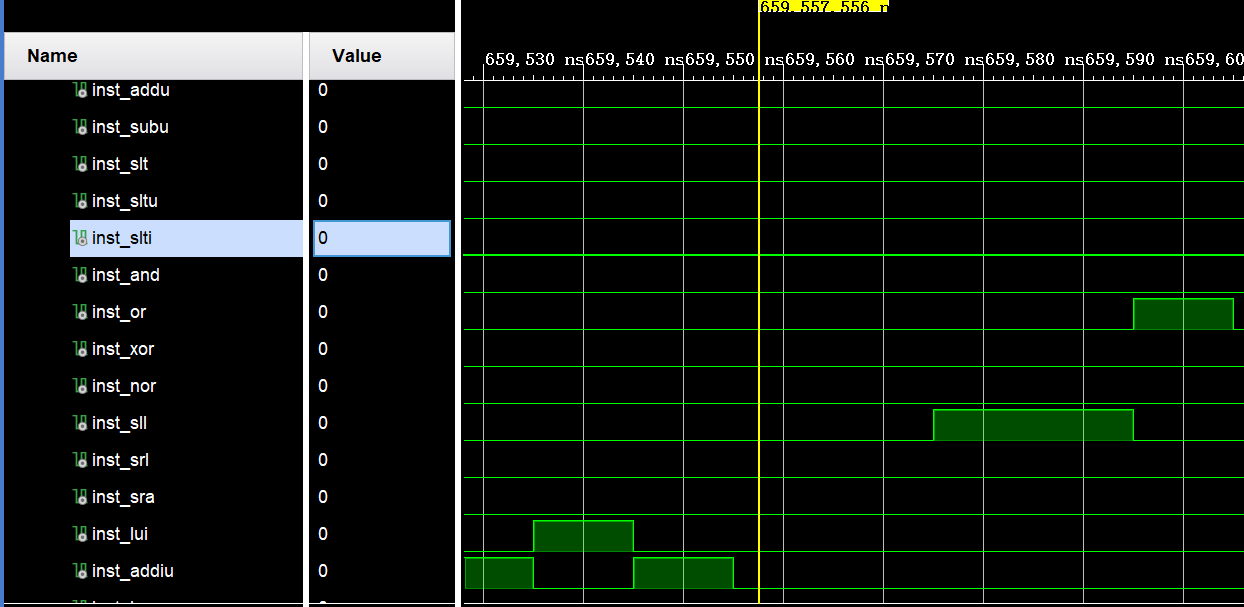
2019/9/30 14:00 – 18:00 加入运算类指令

2019/10/2 8:00 – 13:00 完善乘除法等

（二）错误记录

1、错误1：译码时10进制与16进制弄混

（1）错误现象



图一：没有指令

注意黄线处，发现此拍没有指令！

（2）分析定位过程

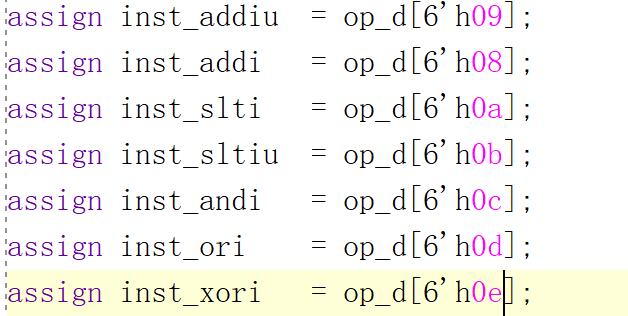
查看反汇编代码，此时应该时slti

（3）错误原因



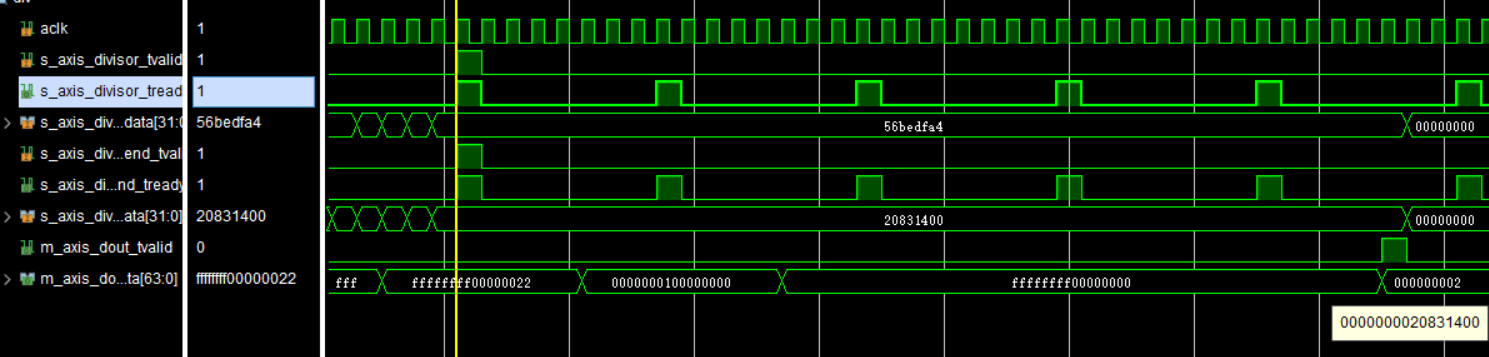
在十进制下为10，在十六进制下应该为0a

（4）修正效果



2、错误2：被除数与除数弄反

（1）错误现象

在不断测试之后，发现在执行除法指令时，余数出现了异样：大数据除以小数据之后，商为0，余数与小数据相同，不应该出现这种情况。

图二 被除数与除数弄反的波形

（2）分析定位过程

考虑到除法器有输出，问题应该不是调用的问题。调整了汇编测试程序中的测试顺序，发现在测试小数据除以大数据时商竟然不是0。问题变得有趣了起来。

（3）错误原因

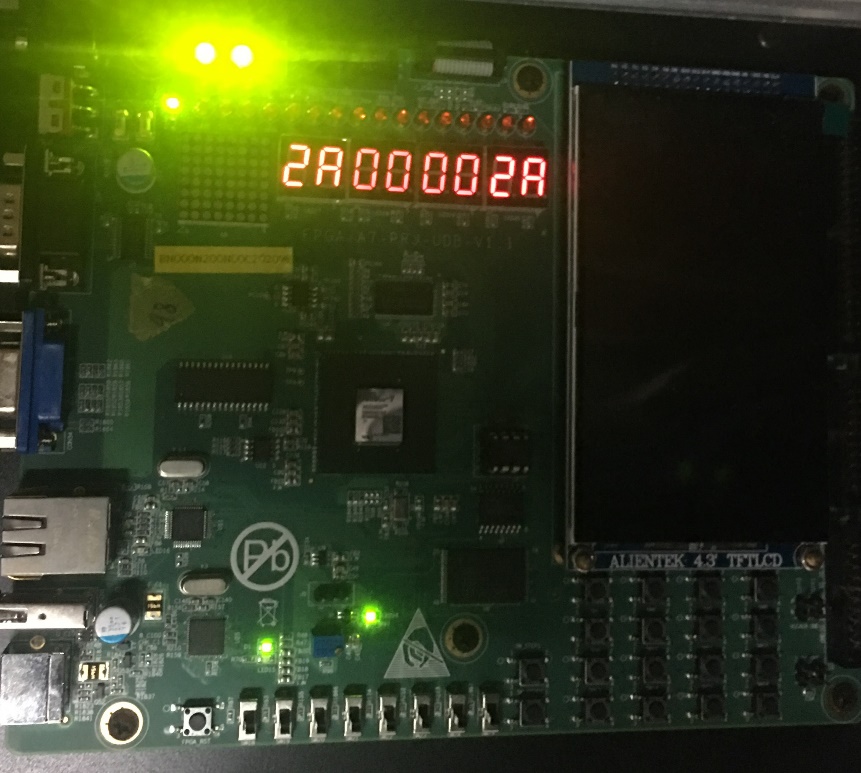
灵光一闪，我选择打开了网易词典，查一查divisor和dividend的意思。

交换了被除数和除数的赋值之后，问题得到解决。

（4）修正效果

结果输出正常，测试通过。

四、实验总结（可选）



图二 上板成功拍照留念

漏洞百出，果然debug能力最重要。