实验 6 报告

学号：2017K8009922026，2017K8009922032
姓名：康齐瀚,赵鑫浩
箱子号：63

一、实验任务（10%）

向已有的 CPU 中添加 ADD, ADDI 等算术逻辑运算指令以及 MULT，MULTU 等乘除运算指令以及与之配套的数据搬运指令 MFHI,MFLO 等。最终能成功通过 42 个测试样例和上板

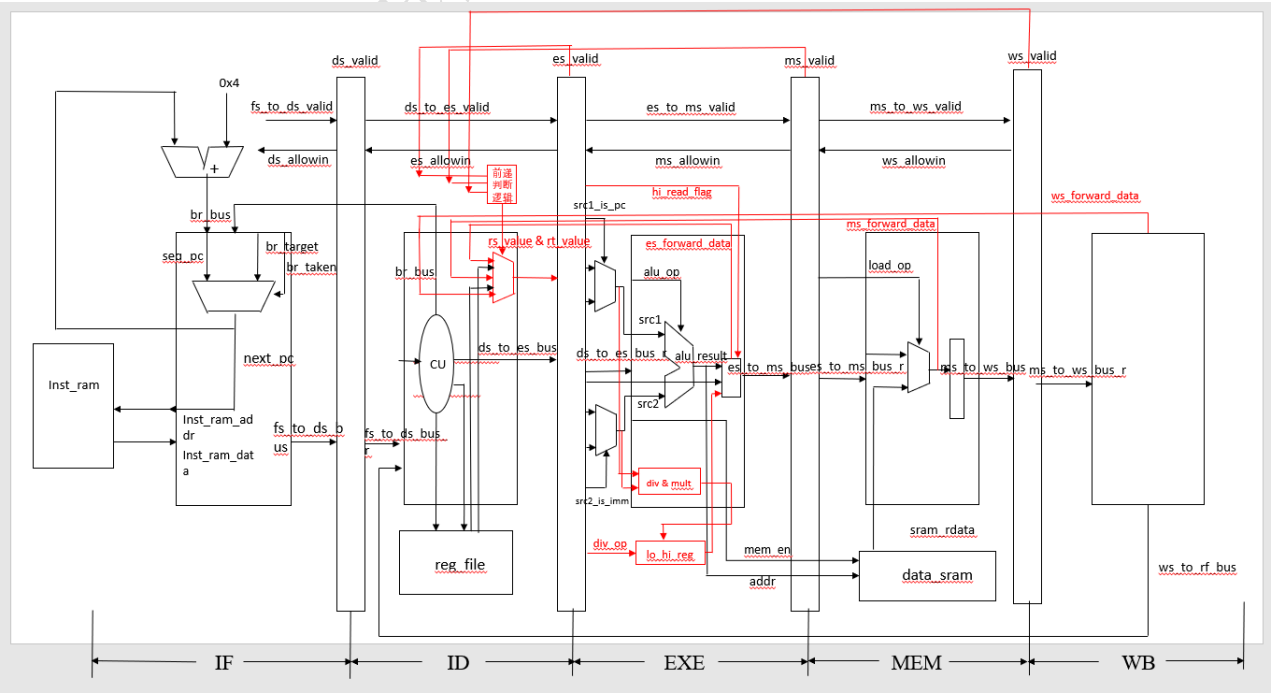
二、实验设计（40%）

（一）总体设计思路

总体设计上，仍然遵循了老师提供的代码框架的风格。使用 inst_具体指令标识的信号表示当前正在译码的指令属于何种类型。用这些信号之间的与，或运算来形成该指令所需的 ALU 运算类型和源操作数的相关控制信号。

在乘除法指令的实现上，在 EXE 级增加了 HI 和 LO 两个寄存器，在 EXE 级算出结果后直接写入。

在与乘除法配套的指令实现上，在 ID 级增加了与读写 HI，LO 相关的控制信号并加入 is_to_es_bus，在 EXE 级进行读出或写入。



（二）重要模块 1 设计：乘除法模块

1、工作原理

乘法直接使用了*运算符，除法调用 Vivado 自己实现的除法器 IP

2、接口定义

以下以有符号除法为例。

名称	方向	位宽	功能描述
clk			时钟信号
signed_divisor_tvalid	IN	1	除数 Valid 握手信号
signed_divisor_tready	OUT	1	除数 ready 握手信号
signed_divisor_tdata	IN	32	除数
signed_dividend_tvalid	IN	1	被除数 Valid 握手信号
signed_dividend_tready	OUT	1	被除数 ready 握手信号
signed_dividend_tdata	IN	32	被除数
signed_dout_tvalid	OUT	1	除法结束信号
signed_div_result	OUT	64	除法结构（商和余数）

3. 功能描述

（1）除法

在除法模块中引入一个 count 信号，来判断除法是否为第一次握手（因为每 8 拍可以重新进行一条除法指令，而我们只需要第一次的除法结果），当 ready 信号为 1 时，让 count 信号加 1，用 count 信号判断，当且仅当第一次 ready 信号拉高时，valid 信号和其成功握手。

```
always@ (posedge clk)
    begin
        if(reset) begin
            count<=0;
        end
        else if(signed_dout_tvalid | unsigned_dout_tvalid)
        begin
             count<=0;
        end
        else if( (es_div & signed_divisor_tready) | (es_divu & unsigned_divisor_tready) )
        begin
            count <= count + 1;
        end
        else
        begin
            count <= count;
        end
    end
```

接下来我们给 valid 信号赋值实现握手。

```
always @(posedge clk)
    begin
        if(reset) begin
            signed_divisor_tvalid = 1'b0;
            //divisor_tready = 1'b0;
            signed_dividend_tvalid = 1'b0;
            //dividend_tready = 1'b0;
        end
        else
        begin
            signed_divisor_tvalid = es_div & ~signed_divisor_tready & ~signed_dividend_tready & (count == 0);
            signed_dividend_tvalid = es_div & ~signed_divisor_tready & ~signed_dividend_tready & (count == 0);
        end
    end
```

最后给被除数和除数信号赋值并调用除法模块。

```
mydiv_signed u_mydiv_signed(
    .aclk                   (clk),
    .s_axis_divisor_tvalid  (signed_divisor_tvalid),
    .s_axis_divisor_tready  (signed_divisor_tready),
    .s_axis_divisor_tdata   (signed_divisor_tdata),
    .s_axis_dividend_tvalid (signed_dividend_tvalid),
    .s_axis_dividend_tready (signed_dividend_tready),
    .s_axis_dividend_tdata  (signed_dividend_tdata),
    .m_axis_dout_tvalid     (signed_dout_tvalid),
    .m_axis_dout_tdata      (signed_div_result)

    );
```

（2）乘法

至于乘法，我们简单地用乘号实现有符号和无符号数的乘法（vivado 会自动调用乘法 ip 实现乘法）。

（三）重要模块 2 设计：HI 和 LO 寄存器模块

1、工作原理

在 EXE 模块中新增 reg 型信号 HI 和 LO。在乘除法指令运算完毕后就写入结果。在与乘除法配套的数据搬运指令中，则是解析来自 ID 级的 id_to_es_bus 指令中的读写 HI 和 LO 寄存器的信号，做相应的赋值或读数据操作。

2、接口定义

该部分并未以模块封装的形式给出，而是直接“嵌入”在 EXE 模块中。

名称	方向	位宽	功能描述
lo_wen	/	1	LO 寄存器写信号
hi_wen	/	1	HI 寄存器写信号
lo_read	/	1	LO 寄存器数据读信号
hi_read	/	1	HI 寄存器数据读信号
lo_reg	/	32	LO 寄存器
hi_reg	/	32	HI 寄存器

3、功能描述

如果检测当当前指令是乘除法指令，就直接在时钟上升沿将乘除法指令的结果写入 LO 和 HI 寄存器。这样实际上在除法时并不会导致问题，因为流水线被阻塞的缘故，HI 和 LO 会一直被写入直到最后运算完成，指令不再是除法。如果是 MTHI，MTLO 的指令，就由写使能信号 hi_wen 和 lo_wen 控制写入。代码如下：

```
always@(posedge clk) begin
    if(reset) begin
        hi_reg <= 32'b0;
    end
    else if(hi_wen) begin
        hi_reg <= hi_wdata;
    end
    else if(op_div | op_divu) begin
        hi_reg <= mult_div_result[31:0];
    end
    else if(op_mult | op_multu) begin
        hi_reg <= mult_div_result[63:32];
    end
end
```

```
always@(posedge clk) begin
    if(reset) begin
        lo_reg <= 32'b0;
    end
    else if(lo_wen) begin
        lo_reg <= lo_wdata;
    end
    else if(op_div | op_divu) begin
        lo_reg <= mult_div_result[63:32];
    end
    else if(op_mult | op_multu) begin
        lo_reg <= mult_div_result[31: 0];
    end
end
```

注意除法指令和乘法指令在 HI 和 LO 中存入数据需要的是高 32 位还是低 32 位是相反的

在从 HI 和 LO 寄存器中读出数据，是体现在对 EXE 模块的 es_final_result 的赋值上：

```
assign es_to_ms_result = (hi_read) ? hi_read_data :
                         (lo_read) ? lo_read_data : es_alu_result;
```

三、实验过程（50%）

（一）实验流水账

2019/10/10　14：00～18：00　完成简单算术逻辑指令的设计并仿真通过

2019/10/11　8：00～12：00　完成乘除法指令设计并仿真通过

（二）错误记录

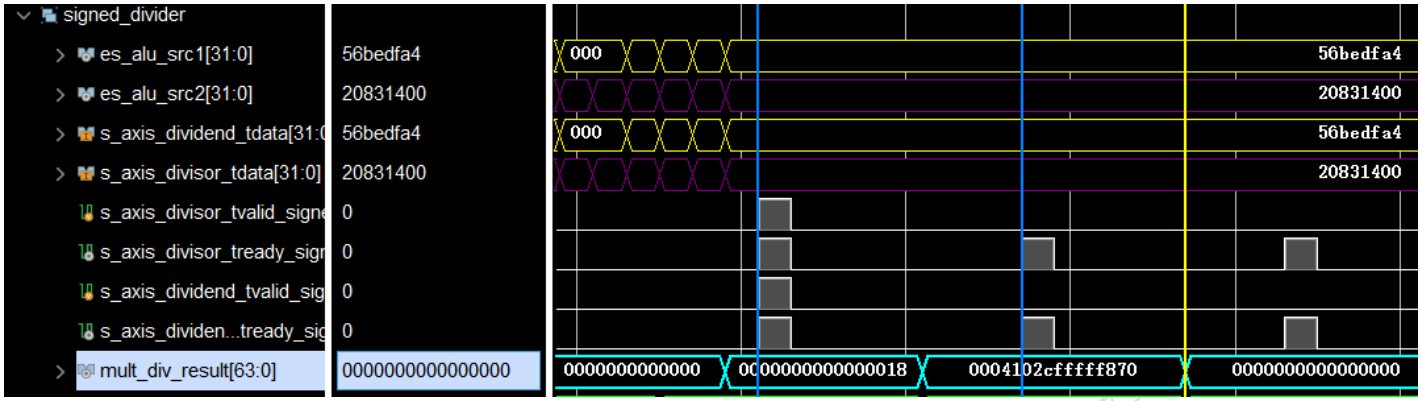
1、错误 1：寄存器写回错误

（1）错误现象

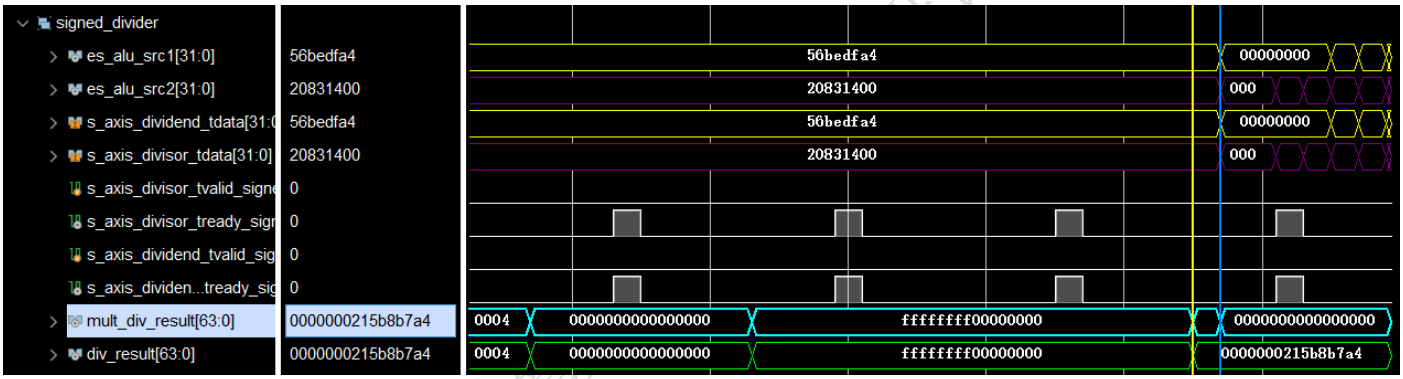
仿真时，控制台对比报错，PC=0xbfc3095c 时写回寄存器的数据有误。

（2）分析定位过程

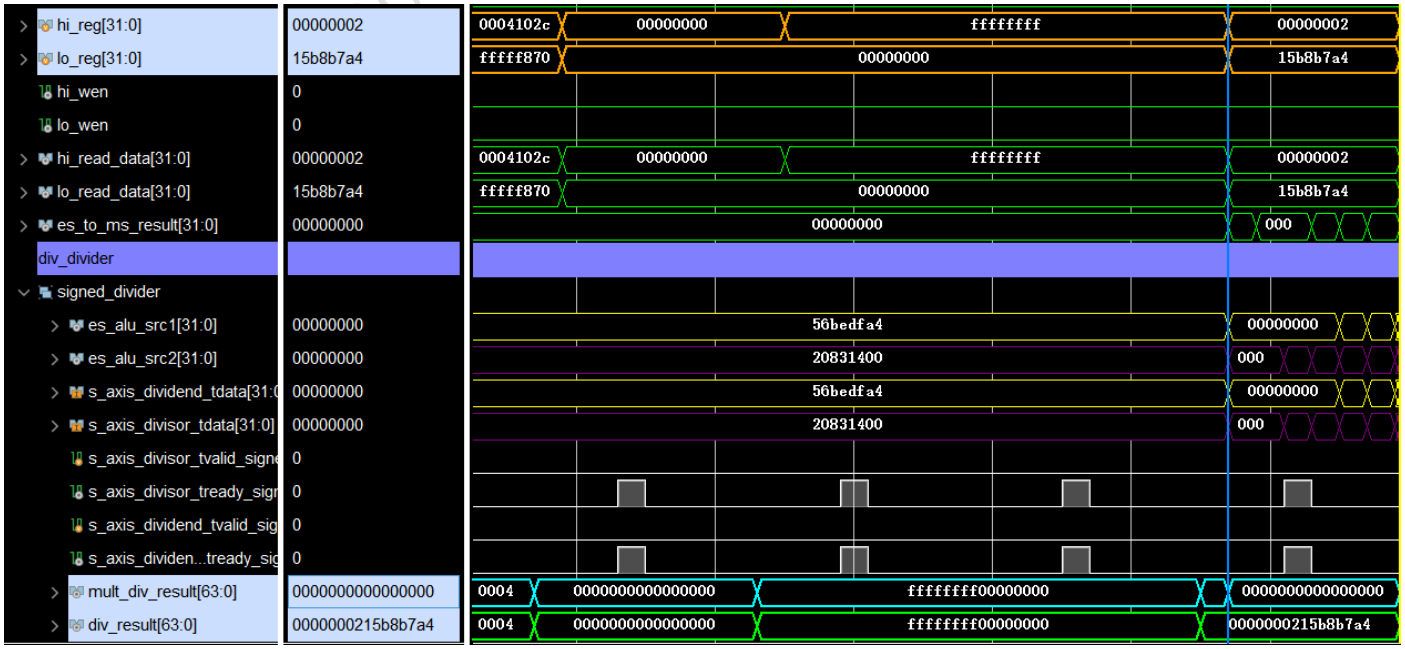
查看 test.s 文件发现 PC=0xbfc3095c 是一条 mflo 指令。前面正好是一条 div 指令，因此可能是 div 指令的计算出错，也可能是 mflo 的逻辑有误。



观察 div 指令的执行情况，被除数和除数分别为 0x56bedfa4 和 0x20831400。在两个 tready 信号拉高的时候，tvalid 输入信号也拉高，因此在此时两个输入数据是写进了除法器 IP 中的。



在除法器运算完毕时可以看到输出结果是 0x0000000215b8b7a4, 高位为商，低位为余数。用计算器计算发现结果是正确的。考虑是 hi 和 lo 寄存器写入的问题：



从这张图的最顶端可以看到 hi 和 lo 寄存器都被赋值了，但是在 mflo 指令执行时却写回了 0x15b8b7a4 的数据，正确数据应该是 0x00000002，因此错误是 lo 和 div 寄存器存放的数据放反了。lo 应该存放商，hi 存放余数。

（3）错误原因

hi 寄存器和 lo 寄存器功能倒置了。

（4）修正效果

修改 EXE 级对 lo 和 hi 寄存器的赋值，使 lo 寄存器在除法指令中获得 div_result 的高 32 位数据，hi 寄存器在除法指令中获得 div_result 的低 32 位数据。

（5）归纳总结（可选）

在动手写代码之前没有仔细阅读指令和寄存器的规范。

2、错误 2：除法输入信号错误

（1）错误现象

仿真时，控制台对比报错，写回寄存器的数据有误。

（2）分析定位过程

查看 test.s 文件发现是一条 div 指令。且调用模块后除法运算结果不对。进一步手动计算后发现，除数和被除数的信号接反导致除法结果出错。

（3）错误原因

除数和被除数的信号接反。

（4）修正效果

修改 EXE 级对除数和被除数的信号调用后，波形正常。

3、错误 3：除法多次握手

（1）错误现象

仿真时，控制台对比报错，写回寄存器的数据有误，并且观察除法 valid 的信号后发现，其多次拉高和 ready 握手。

（2）分析定位过程

查看 test.s 文件发现是一条 div 指令。且调用模块后除法模块多次握手，即每 8 拍进行了一次除法指令，不符合要求，所以应该添加一个控制信号，判断是否为第一次握手，如果不是，则 valid 一直为 0。

（3）错误原因

没有考虑到 ready 信号每 8 拍拉高一次，表示可以运行一次除法。

（4）修正效果

加入判断 count 信号后，每一次除法指令仅有第一次握手成功并执行除法指令，符合预期。