**LAB5报告**

学号： 2021K8009929010

姓名： 贾城昊

学号： 2021K8009929016

姓名： 李金明

学号： 2021K8009929007

姓名： 牛浩宇

箱子号： 13

1. 实验任务

Exp10要求在exp9的基础上添加部分算术逻辑运算指令与乘除法运算指令。包括slti, sltui, andi, ori, xori, sll, srl, sra, pcaddu12i, mul.w, mulh.w, mulh.wu, div.w, mod.w, div.wu, mod.wu。其中乘除法运算指令通过额外例化新写的乘除法模块来实现。

Exp11要求在exp10的基础上添加转移指令blt，bge，bltu，bgeu和访存指令ld.b，ld.h，ld.bu，ld.hu，st.b，st.h。要求成功通过仿真与上板测试。

1. 实验设计
   1. 总体设计思路
      1. Exp10算术逻辑运算指令

9条算术逻辑运算类指令的实现主要复用已实现的部分指令的数据通路。slti、sltui、andi、ori、xori指令与之前实现的slt、sltu、and、or、xor指令的区别仅在于一个源操作数从寄存器中的数据变成了立即数，因此基本上只要修改与此相关的信号。sll、srl、sra指令类似，可以在slli、srli、srai指令的基础上修改，调整操作数来源与相关控制信号即可。pcaddu12i指令的通路主要复用add指令，两个操作数一个来自pc，另一个是需要零扩展12位的立即数，只需修改对应的控制信号。

* + 1. Exp10乘除指令

9条算术逻辑运算类指令的实现主要复用已实现的部分指令的数据通路。slti、sltui、andi、ori、xori指令与之前实现的slt、sltu、and、or、xor指令的区别仅在于一个源操作数从寄存器中的数据变成了立即数，因此基本上只要修改与此相关的信号。sll、srl、sra指令类似，可以在slli、srli、srai指令的基础上修改，调整操作数来源与相关控制信号即可。pcaddu12i指令的通路主要复用add指令，两个操作数一个来自pc，另一个是需要零扩展12位的立即数，只需修改对应的控制信号。

* + 1. Exp11跳转类指令

对已有的是否跳转的判断逻辑进行扩展，增加在有无符号的情况下rj<rd的判断。

* + 1. Exp11访存类指令

内存读写数据不再全部为32位，需要考虑alu计算所得地址后两位的偏移。

load指令：ld.b(u)、ld.h(u)为8位与16位，且地址后两位不一定对齐4字节为00。ld.b(u)有的偏移00、01、10、11四种情况，分别对应读数据为32位rdata中的第0、1、2、3字节；ld.h(u)的偏移有00、10两种情况，分别对应读数据为32位rdata中的第01、23字节。因此可以先进行移位操作，将要选取的部分数据调整到最低位。之后再进行扩展。

Store指令：st.b、st.h为8位与16位。st.b的四种偏移00、01、10、11分别对应mem\_we为0001、0010、0100、1000。st.h的两种偏移00、10分别对应mem\_we为0011、1100。如此对mem\_we进行赋值。32位的wdata将8位数据重复4次，16位数据重复2次填满即可。

* + 1. 五级流水CPU总体设计图

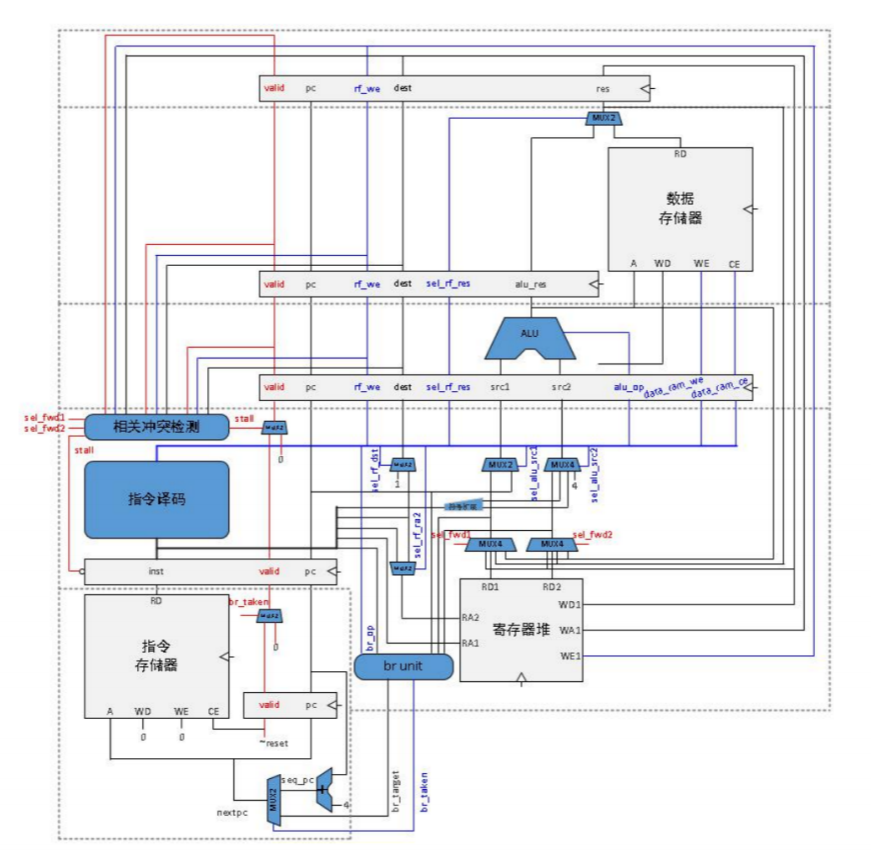
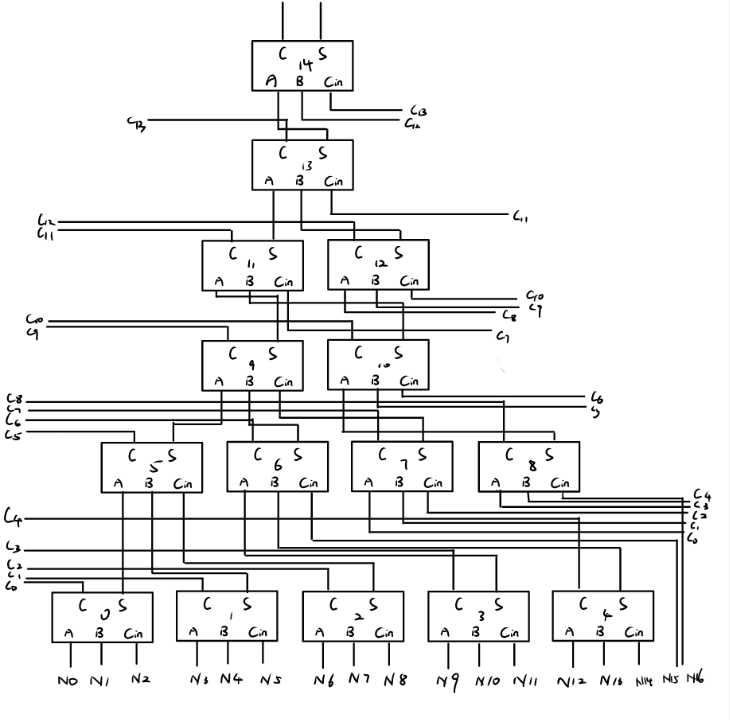


图6 五级流水CPU总体设计图

* 1. 重要模块1设计：乘法器
     1. 工作原理

乘法器中使用2位booth编码进行运算，使用2位booth编码可以将32位乘法转化为16个部分积相加，从而只使用15次加法运算，提高了效率。另外在此实验中，由于涉及到无符号乘法，为了便于booth算法的移位操作，需要将操作数扩展为34位进行运算，故实际上有17个部分积，需要进行16次加法运算。

直接进行17个68位数的相加会导致很大的延迟，使用华莱士树可以将其转化成2个68位数相加，图x是针对某一位构建的华莱士树。特别地，虽然booth算法输出的c有17位，但是由于最高的两位是扩展得到的，实际上c17一定为0，c16和c15在连接64位加法器时另外考虑，故华莱士树只需要考虑14个Cin即可。



图x 乘法器华莱士树结构

* + 1. 接口定义

表1 乘法器接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| mul\_clk | IN | 1 | 乘法器模块时钟信号 |
| resetn | IN | 1 | 复位信号 |
| mul\_signed | IN | 1 | 控制有符号乘法和无符号乘法 |
| x | IN | 32 | 被乘数 |
| y | IN | 32 | 乘数 |
| result | OUT | 64 | 乘法结果 |

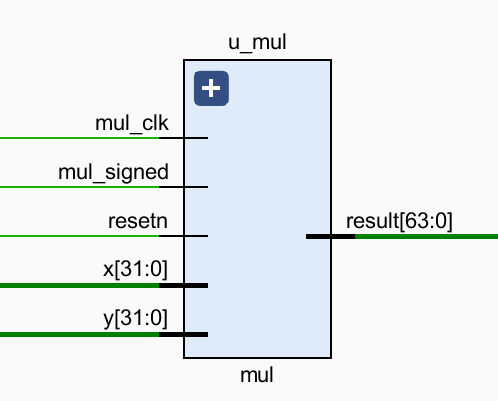


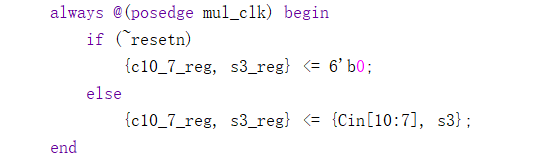
图8 乘法器接口

* + 1. 功能描述

在每个时钟周期，乘法器将输入的32位数根据译码结果有符号或无符号扩展位34位（图x），通过booth编码和华莱士树将其转化位2个64位数相加。特别地，为了优化时序结果，需要在华莱士树中间加上一级切分（图x），以达到优化时序结果的目的。最后乘法器将在输入操作数的下一个时钟周期输出64位的计算结果。



图9 乘法器操作数的扩展



图x 乘法器二级流水

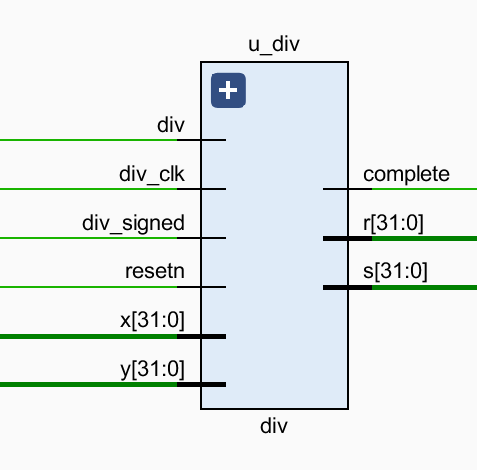
* 1. 重要模块2设计：除法器
     1. 工作原理

本实验中使用的是1位恢复余数绝对值迭代触发器。除法器工作时，首先根据被除数和除数确定商和余数的符号，并计算被除数和除数的绝对值，然后对绝对值进行除法运算，迭代运算得到商和余数的绝对值，最后根据被除数和除数原先的符号确定商和余数最终的符号和值。在这个除法器中，若想得到无符号除法的结果，只需要在计算被除数和除数的绝对值时保持值不变即可。

* + 1. 接口定义

表2 除法器接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| div\_clk | IN | 1 | 除法器模块时钟信号 |
| resetn | IN | 1 | 复位信号 |
| div | IN | 1 | 除法运算的命令 |
| div\_signed | IN | 1 | 控制有符号除法和无符号除法 |
| x | IN | 32 | 被除数 |
| y | IN | 32 | 除数 |
| s | OUT | 32 | 除法结果，商 |
| r | OUT | 32 | 除法结果，余数 |
| complete | OUT | 1 | 除法完成信号 |



图x 除法器接口

* + 1. 功能描述

当div信号拉高时，根据EXE模块传递的操作数进行除法运算。当运算结束后，拉高complete信号，此时r和s的值分别是除法运算得到的余数和商。

* 1. 重要模块3设计：ID流水级
     1. 工作原理

为新增加的指令添加译码信号，将它们传给EXE阶段。修改与alu、立即数有关的信号，实现算术逻辑运算指令。对于转移指令则需要判断是否进行跳转，并计算跳转地址。

本次实验中采用前递的方法，故当遇到访存指令和乘法指令且与紧跟的吓一跳指令有数据相关时，也需要阻塞一周期。阻塞是通过控制id\_ready\_go 的信号实现的，其拉低即可确保流水线被阻塞。

* + 1. 接口定义

表2 ID流水级接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号 |
| ex\_allowin | IN | 1 | EXE流水级允许ID流水级传入数据 |
| id\_allowin | OUT | 1 | ID流水级允许IF流水级传入数据 |
| if\_to\_id\_valid | IN | 1 | 标记IF流水级传入ID流水级的数据是否有效 |
| if\_to\_id\_wire | IN | 64 | IF流水级传给ID流水级的数据 |
| id\_to\_ex\_valid | OUT | 1 | 标记ID流水级传入EXE流水级的数据是否有效 |
| id\_to\_ex\_wire | OUT | 160 | ID流水级传入EXE流水级的数据 |
| br\_taken | OUT | 32 | 传给IF流水级的跳转信号 |
| br\_target | OUT | 1 | 传给IF流水级的跳转地址 |
| wb\_rf\_zip | IN | 38 | WB流水级向ID流水级传递的regfile的写回信息和前递数据 |
| ex\_rf\_zip | IN | 39 | EXE流水级前递到ID流水级的数据 |
| mem\_rf\_zip | IN | 38 | MEM流水级前递到ID流水级的数据 |

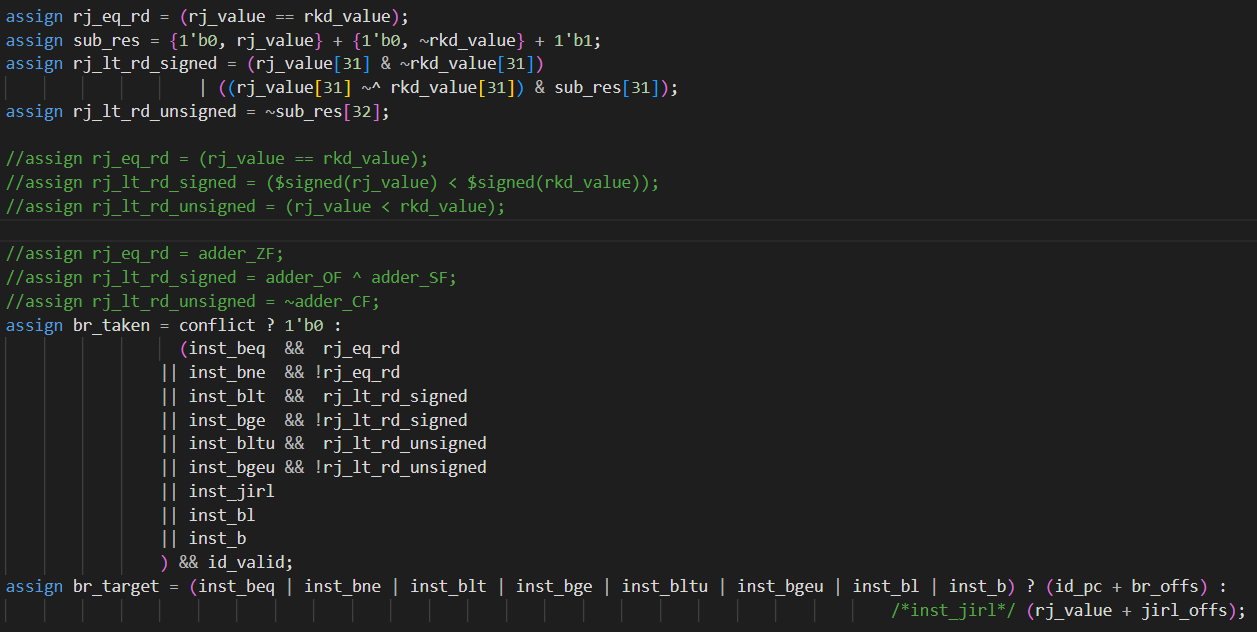


图11 ID流水级接口

* + 1. 功能描述

对于exp10增加的算术逻辑运算类指令，基本上都可以复用现有的数据通路，从寄存器中读出数据，根据指令类型对立即数进行扩展，计算出两个操作数后送入EXE阶段，使用alu算出结果。andi、ori、xori的立即数是由12位立即数无符号扩展得到的，之前没有这类立即数的数据通路，需要添加对应的信号和逻辑。

对于exp11增加的跳转指令（blt、bltu、bge、bgeu），可以复用之前的br\_taken与br\_target信号，只需在br\_taken的赋值逻辑中增加对应的条件判断的逻辑，即判断rj寄存器中的值在有无符号的两种情况下是否小于rd寄存器中的值。逻辑如下：



共尝试了三种判断逻辑，包括直接使用小于号、连接加法器模块使用Flag值、以及不连接单独的加法器模块，而是直接进行运算并判断。最终发现第三种赋值方法时序性能最好，故采用。

exp11中还增加了几种load和store指令，内存读写不再固定为32位。 改动为不再在ID流水级产生res\_from\_mem、mem\_we以及data\_sram\_wdata信号，而是将5种load指令信号与3种store指令信号向后传递到对应流水级再进行处理。

* 1. 重要模块4设计：EXE流水级
     1. 工作原理

EXE阶段从ID阶段取得有效操作数和操作码后，进行算术逻辑运算，以及在EXE流水级向数据ram发送读写请求，读写地址以及写数据。在Exp10中，EXE流水级需要新添对乘法器和除法器的例化，对于乘法器，EXE流水级只需要把相关信号传递给乘法器，而对于除法器，若为除法指令，则需要等待其算出结果，然后选择结果传递给下一流水级。

* + 1. 接口定义

表3 EXE流水级接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号 |
| mem\_allowin | IN | 1 | MEM模块允许EXE模块传入数据 |
| ex\_allowin | OUT | 1 | EXE模块允许ID模块传入数据 |
| id\_to\_ex\_valid | IN | 1 | 标记ID模块传入EXE模块的数据是否有效 |
| id\_to\_ex\_wire | IN | 160 | ID模块传入EXE模块的数据 |
| ex\_to\_mem\_valid | OUT | 1 | 标记EXE模块传入MEM模块的数据是否有效 |
| ex\_to\_mem\_wire | OUT | 110 | EXE模块传入MEM模块的数据 |
| ex\_rf\_zip | OUT | 39 | EXE模块前递到ID模块的数据 |
| data\_sram\_en | OUT | 1 | 数据ram片选信号 |
| data\_sram\_we | OUT | 4 | 数据ram写使能 |
| data\_sram\_addr | OUT | 32 | 数据ram地址信号 |
| data\_sram\_wdata | OUT | 32 | 数据ram写数据 |

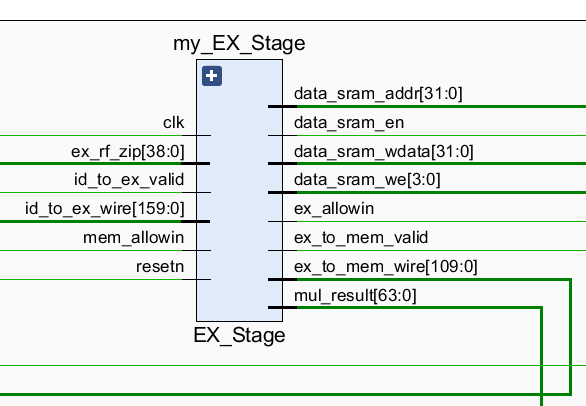
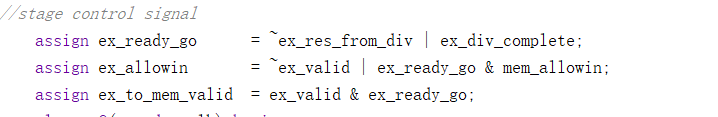


图14 EXE流水级接口

* + 1. 功能描述

结合从ID级传递来的控制信号和数据进行指令的执行。新增对于乘法指令和除法指令的处理：例化乘法器和除法器模块，并传递相应信号，乘法指令的阻塞处理在ID进行，除法指令的阻塞则需要在EXE进行，若为除法指令，需要等到除法器模块发送complete信号后，才能将ex\_ready\_go拉高，同时传递给MEM的数据也需要进行选择。具体如下所示：



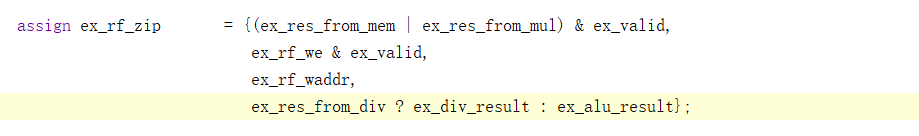
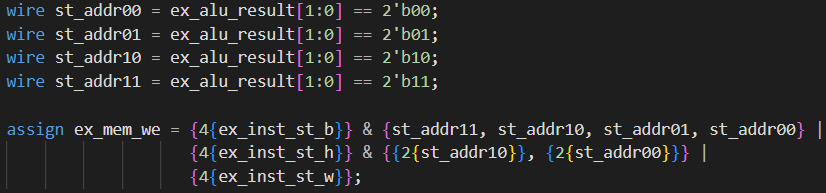
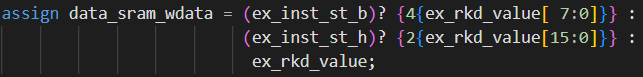


图 15 EXE流水级对除法指令的处理

对于访存：根据alu计算得到的地址第二位与内存写的位数来对mem\_we进行赋值：



32位的wdata可以将8位数据重复4次，16位数据重复两次，再由mem\_we进行选择。



* 1. 重要模块5设计：MEM流水级
     1. 工作原理

接收数据ram返回的读出数据，根据控制信号生成最终的写回数据。

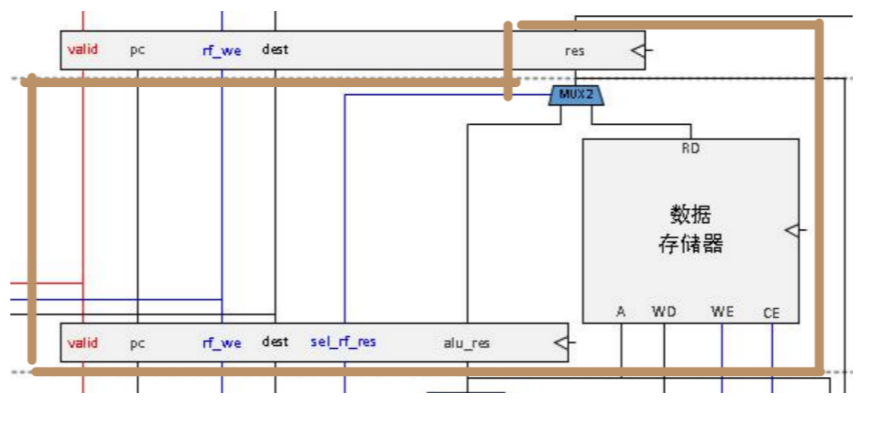


图16 MEM流水级结构示意图

* + 1. 接口定义

表4 MEM流水级接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| resetn | IN | 1 | 复位信号 |
| wb\_allowin | IN | 1 | WB模块允许MEM模块传入数据 |
| mem\_allowin | OUT | 1 | MEM模块允许EXE模块传入数据 |
| ex\_to\_mem\_valid | IN | 1 | 标记EXE模块传入MEM模块的数据是否有效 |
| ex\_to\_mem\_wire | IN | 104 | EXE模块传入MEM模块的数据 |
| mem\_to\_wb\_valid | OUT | 1 | 标记MEM模块传入WB模块的数据是否有效 |
| mem\_to\_wb\_wire | OUT | 70 | MEM模块传入WB模块的数据 |
| mem\_rf\_zip | OUT | 38 | MEM模块前递到ID模块的数据 |
| data\_sram\_rdata | IN | 32 | 数据ram读数据 |

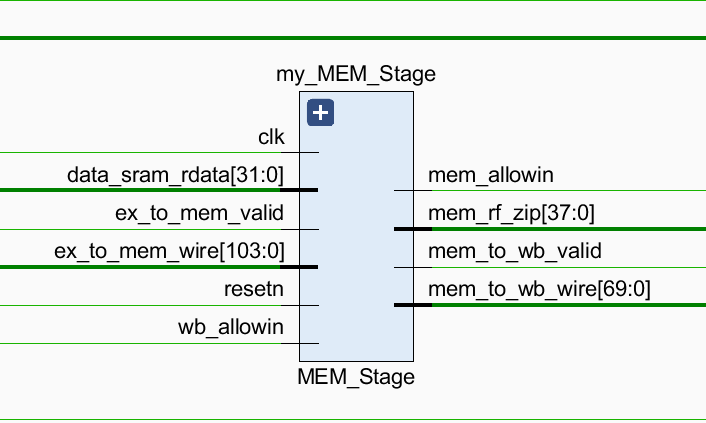
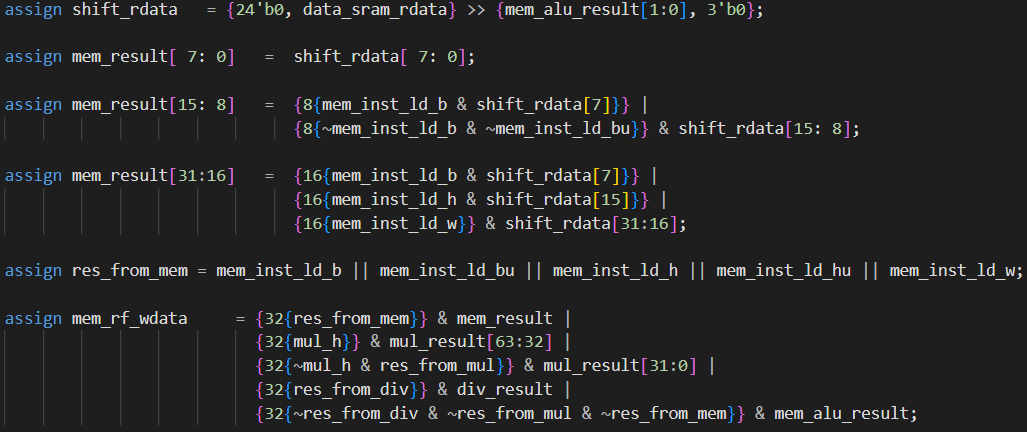


图17 MEM流水级接口

* + 1. 功能描述

根据EXE模块传递来的数据执行相应的访存操作。并将访存指令结果、写回控制信号、PC等信息在下一个时钟上升沿传递给WB模块。

内存读结果赋值逻辑如下：



先根据地址后两位进行移位，使rdata的最低位与应读数据的最低位对齐。然后根据指令的不同分别对rdata进行扩展，生成mem\_result。之后进行寄存器堆写回数据的选择。

1. 实验过程
   1. 实验流水账

贾城昊

2023年10月13日 19：00-21：00 阅读讲义

2023年10月14日 8：00-11：00 完成exp10的译码逻辑部分大致逻辑

2023年10月14日 12：00-14：00 完成exp10的对非乘除法指令的处理

2023年10月14日 19：00-23：00完成exp10中对乘法除法指令的处理以及对exp10的debug

李金明

2023年10月13日 19：00-21：00 阅读讲义

2023年10月14日 12：00-16：00 完成乘法器

2023年10月14日 17：00-19：00完成除法器

牛浩宇

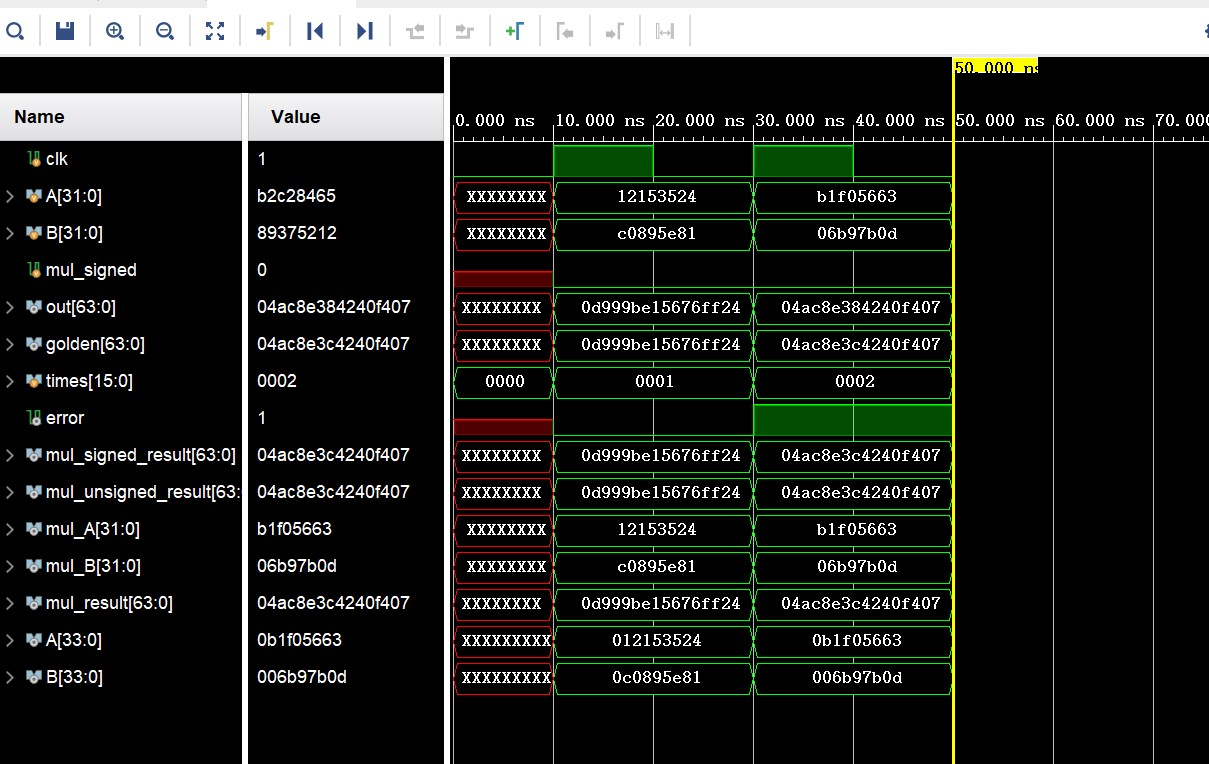
2023年10月12、13日晚上完成exp11大致逻辑

2023年10月15日下午-晚上完成exp11逻辑补充与debug

2023年10月16-19日，小组成员一起尝试对exp11的时序进行优化

* 1. 错误记录
     1. 错误1：exp10乘法器模块中booth算法输入错误
        1. 错误现象

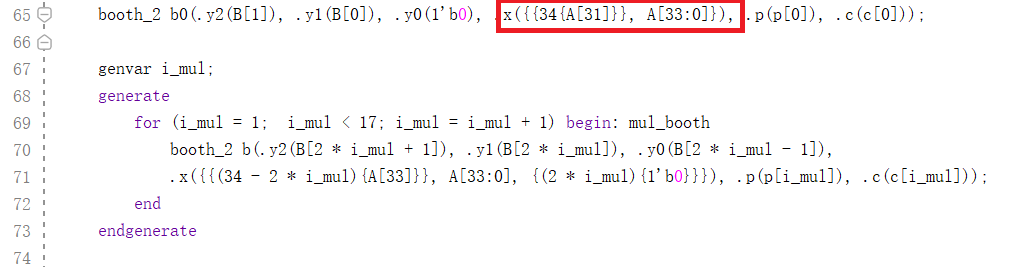
运行仿真后，一个结果是对的，一个结果是错的，它们都是无符号数相乘。



图x 错误1波形图

* + - 1. 分析定位过程

乘法器出错时较难依据信号定位问题所在。观察到出错结果和金标准相差在第34位，而这个乘法器模块是在体系结构理论课作业中已实现的32位定点补码乘法器改进来的，所以我认为出错的位置在于对操作数进行扩展时。但是阅读代码时发现以往写的64、32、16等数据该改的基本上都改了，经过一段时间才发现这一行代码出错：



图x 错误1错误代码

* + - 1. 错误原因

Booth编码一开始要将操作数A扩展为68位，这个扩展则是有符号扩展（因为被扩展的数是34位的操作数），该扩展需要补充34位操作数的符号位，符号位应当是A的第33位，而不是31位，这里没有修改导致booth编码扩展时出错。

* + - 1. 修正效果

将出错的那一行修改为：

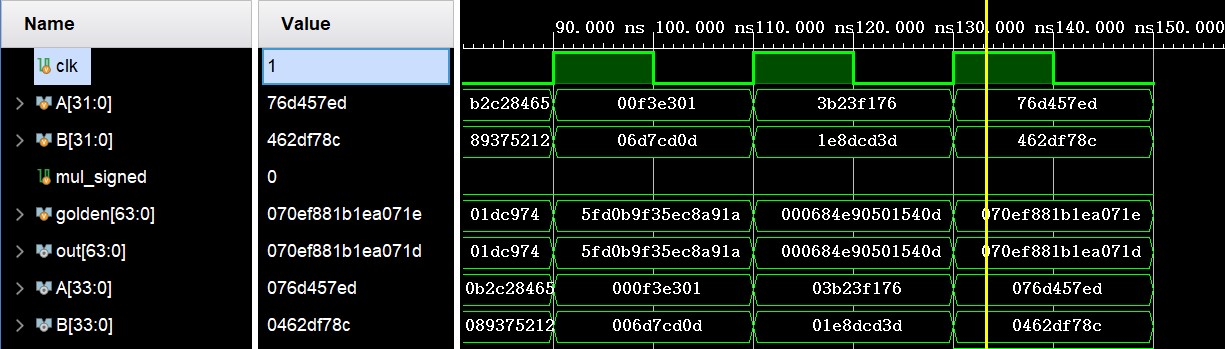


图x 错误1修改代码

该方法有效，来到下一个bug

* + 1. 错误2：exp10乘法器模块中booth算法输入错误
       1. 错误现象

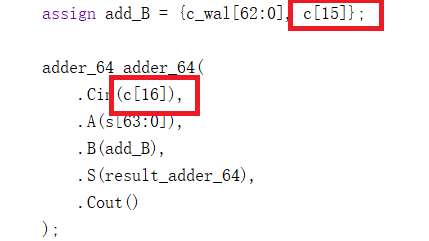
运行仿真后，一个结果是对的，一个结果是错的，它们都是无符号数相乘。



图x 错误2波形图

* + - 1. 分析定位过程

观察到出错结果和金标准相差在第0位，这种出错我认为是booth算法生成的C没有正确地传递给64位加法器导致的，查看相关代码：



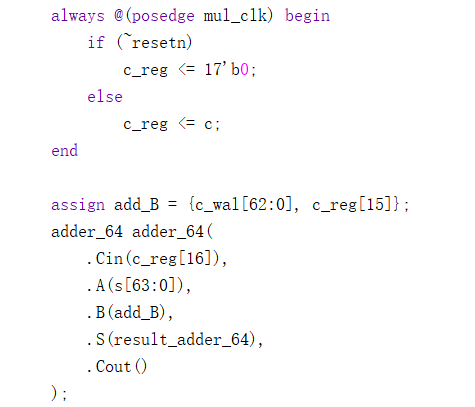
图x 错误2错误代码

* + - 1. 错误原因

Booth算法会产生17位的C，它们相当于是结果第0位的进位。其中0-14位通过华莱士树传递，第17位由于是扩展得到的，故一直为0，需要通过构造加数和进位加上的是第15、16位。但是华莱士数是一个二级流水，前一周期传入17个部分积，要在下一个周期才能拿到结果，即64位加法器的源操作数。而此时已经操作数，C也可能发生改变，因此需要使用一个寄存器存储C的第15位和第16位。

* + - 1. 修正效果

使用一个寄存器存储C的第15位和第16位：



图x 错误2修改代码

该方法有效，乘法器单独验证通过。

* + 1. 错误3：exp10中ID与EXE流水级的寄存器信息解析出错
       1. 错误现象

运行仿真后，仿真一直在进行，但是PC值却没有变化，如下所示：：

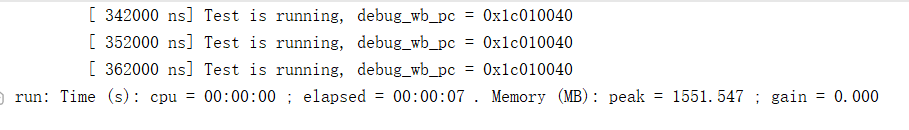


图 20 错误1 对应的Console报错

* + - 1. 分析定位过程

根据经验可知，是比对信号出现问题，导致一直没有与金标准进行比对，于是本人通过波形，观察是哪一个流水级首次出现错误，波形图如下：

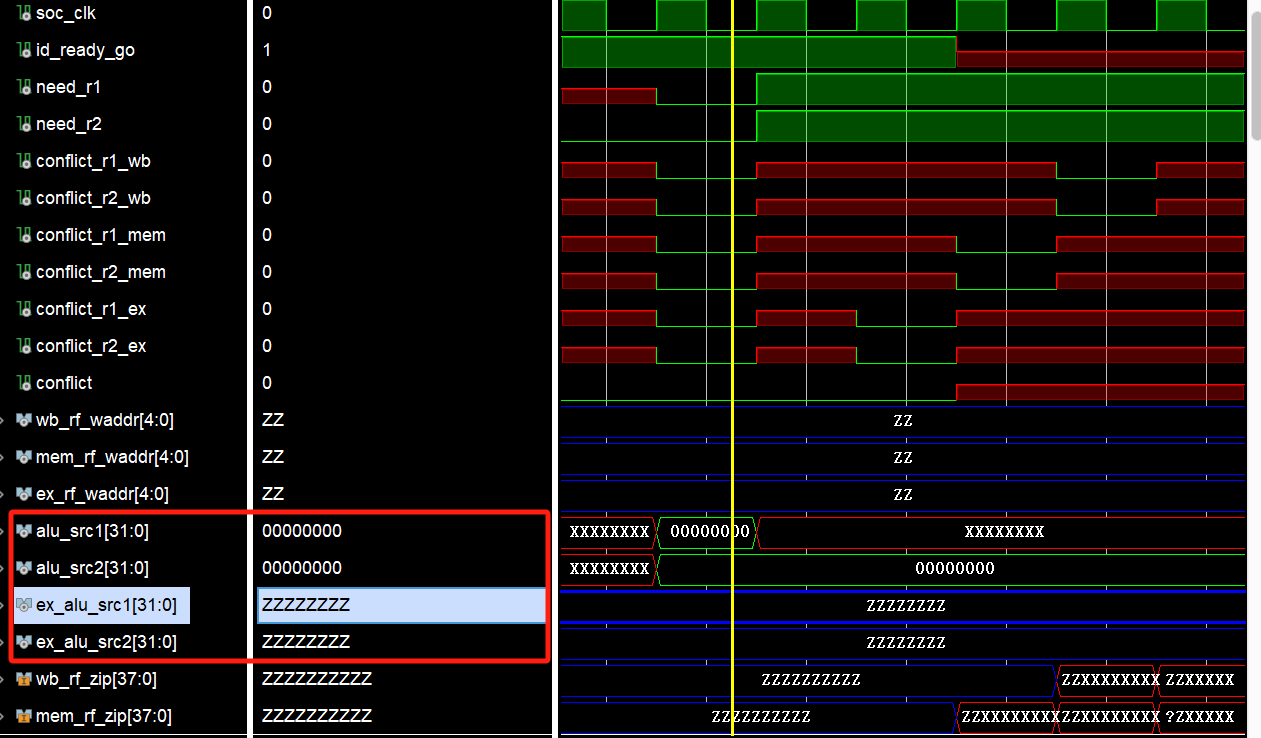


图21 错误1中ID和EXE传递数据不对应

可以看到，EXE流水级的ex\_alu\_src1和ex\_alu\_src2均为高阻态，所以本人查看了这两个信号的赋值，发现是出现了笔误，导致ID与EXE流水级的寄存器信息解析出错：

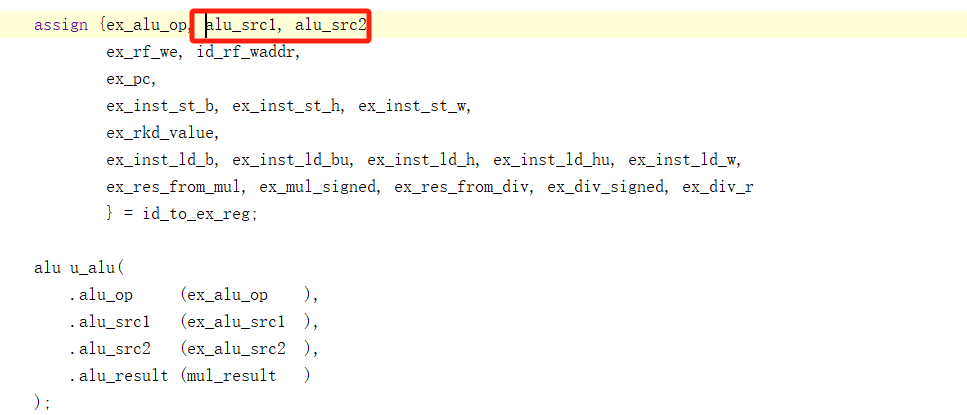
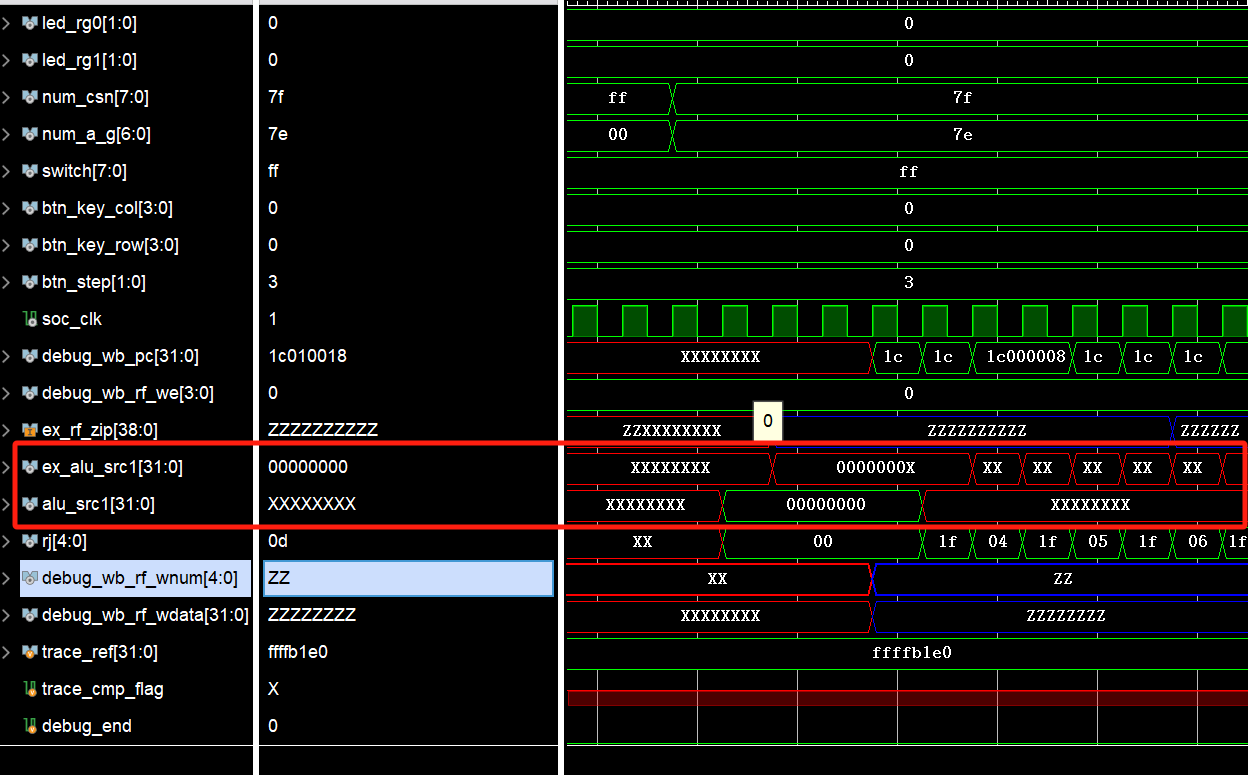
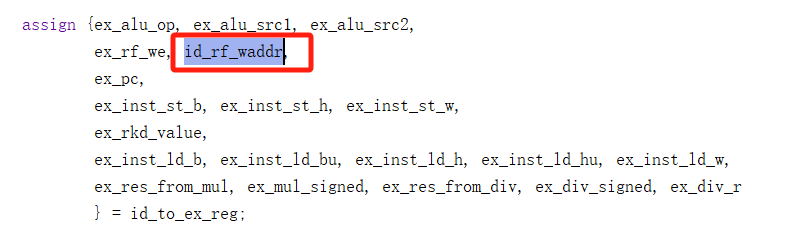


图22 错误1修改前代码

于是本人把alu\_src1和alu\_src2改为ex\_alu\_src1和alu\_src2，再次进行仿真，却仍然在这里出错，于是本人再次观察波形，发现如下结果：



可以看到，ex\_alu\_src1与ID的alu\_src1并不对应，所以看来对ID与EXE流水级的寄存器信息解析还是有问题，观察源码发现还是笔误，如下所示：



由于id\_rf\_waddr在EXE流水级没有定义，所以其默认位宽为1，才导致了上面波形图出现的ex\_alu\_src1与ID的alu\_src1不对应的问题。

* + - 1. 错误原因

由于笔误，导致ID与EXE流水级的寄存器信息解析出错

* + - 1. 修正效果

将对ID与EXE流水级的寄存器信息解析的笔误修改过来即可，如下所示：

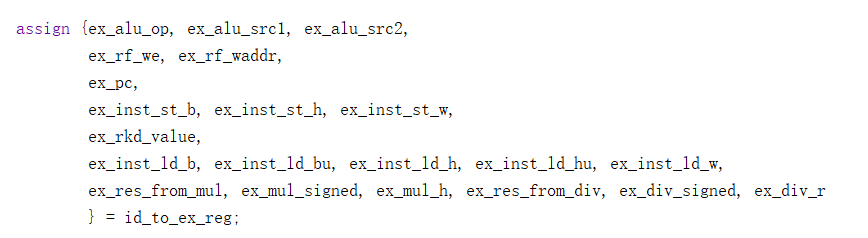


图23 错误1修改后代码

* + 1. 错误4：exp10中alu\_op译码出错
       1. 错误现象

Console报错如下：

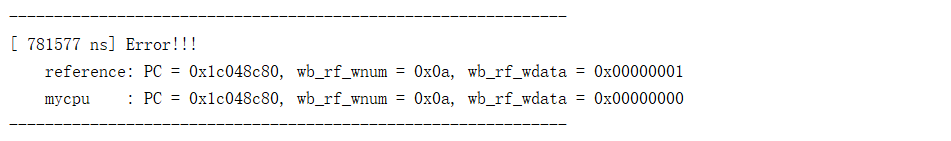


图 24 错误2 对应的Console报错

* + - 1. 分析定位过程

发现wb\_rf\_wdata错误，由于已经跑了很久的仿真，所以猜想是对新加的指令的处理出错。于是本人首先通过行为仿真，定位出错位置的相关信号，波形图如下：

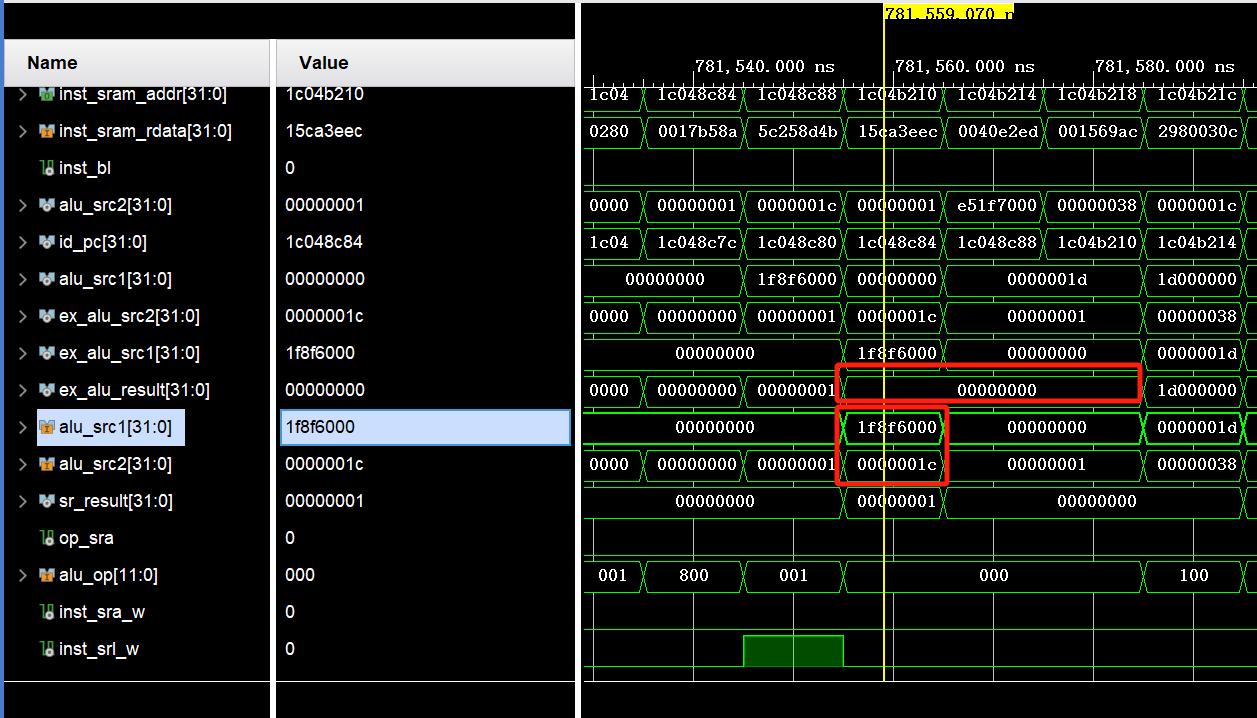
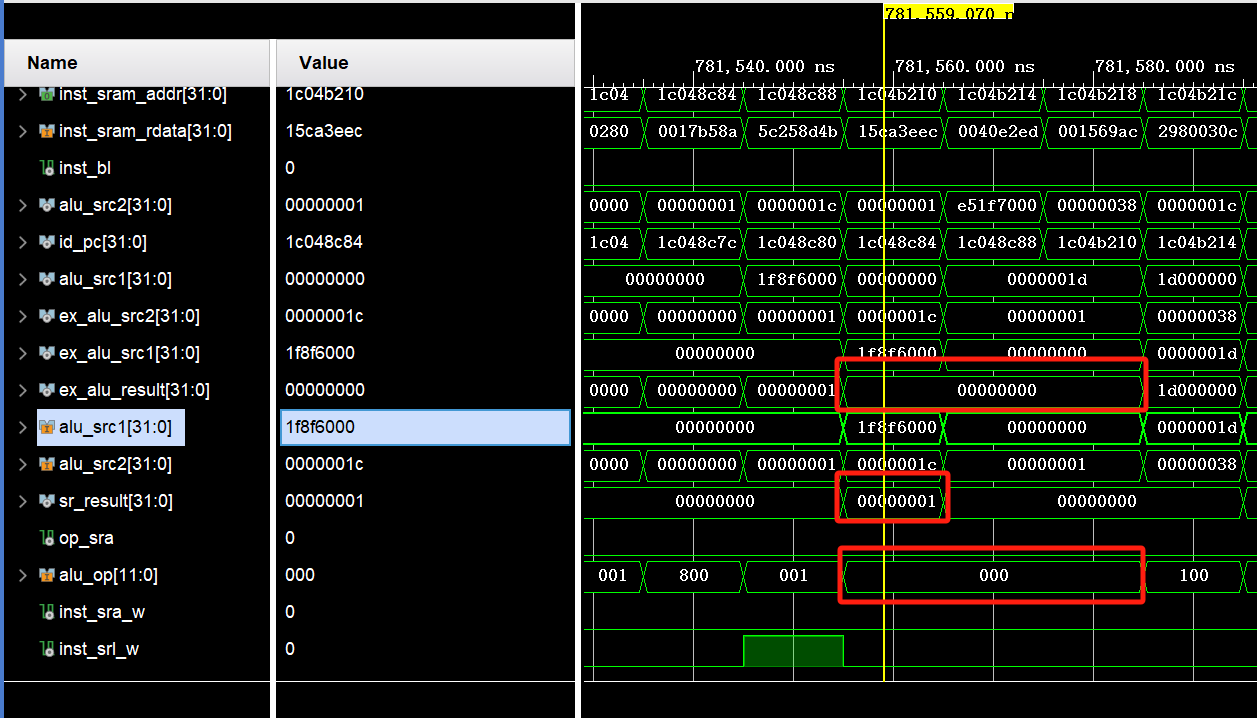


图25 错误2中数据RAM的读数据晚了一个周期

此时的alu模块的alu\_src1和alu\_src2以及alu\_result如上所示，本人紧接着查看反汇编代码，发现此时为srl\_w指令。然而手动验算发现，在该alu\_src下，alu\_result本就应该为0x00000001，所以本人猜测是alu的处理出错。进一步查看波形图信号，可以发现不对的地方，如下所示：



可以看到，此时的alu模块的sr\_result是正确的，但与alu\_result不符，进一步查看，可以发现，此时的alu\_op居然为000，可以看出是alu\_op的译码出现了问题，于是本人查看了alu\_op的译码逻辑，发现了错误，相关代码如下所示：

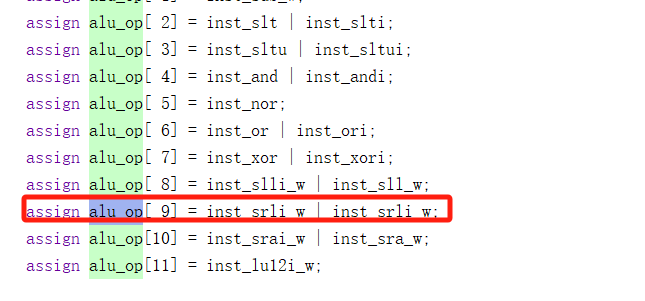


图26 错误2修改前代码

* + - 1. 错误原因

在ID流水级对alu\_op译码的过程中，由于笔误，导致alu\_op[9]的赋值逻辑出错。

* + - 1. 修正效果

将ID流水级的alu\_op[9]的赋值逻辑修正过来即可，如下所示：

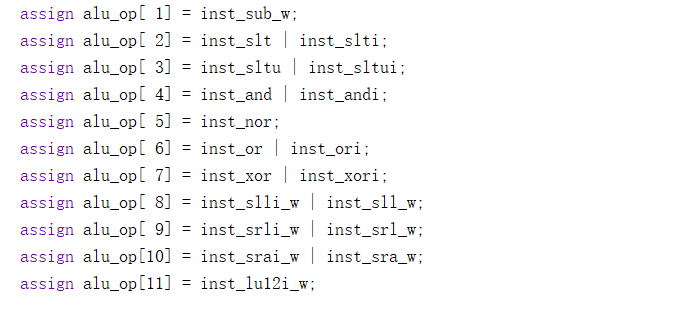


图27 错误2修改后代码

* + 1. 错误5：exp10中传递给除法器模块的信号赋值逻辑出错
       1. 错误现象

Console报错如下：

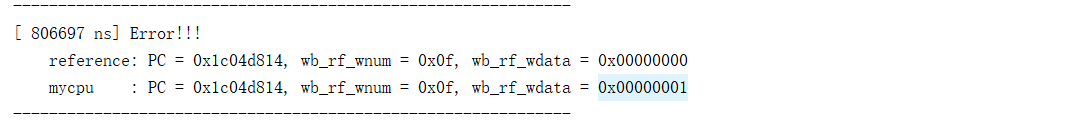


图 28 错误3 对应的Console报错

* + - 1. 分析定位过程

发现wb\_rf\_wdata错误，由于已经跑了很久的仿真，所以猜想是对新加的指令的处理出错。于是本人首先查看了反汇编代码，发现此时是div.w指令，看来大概率是除法指令处理出错，于是紧接着通过行为仿真，定位出错位置的相关信号，波形图如下：

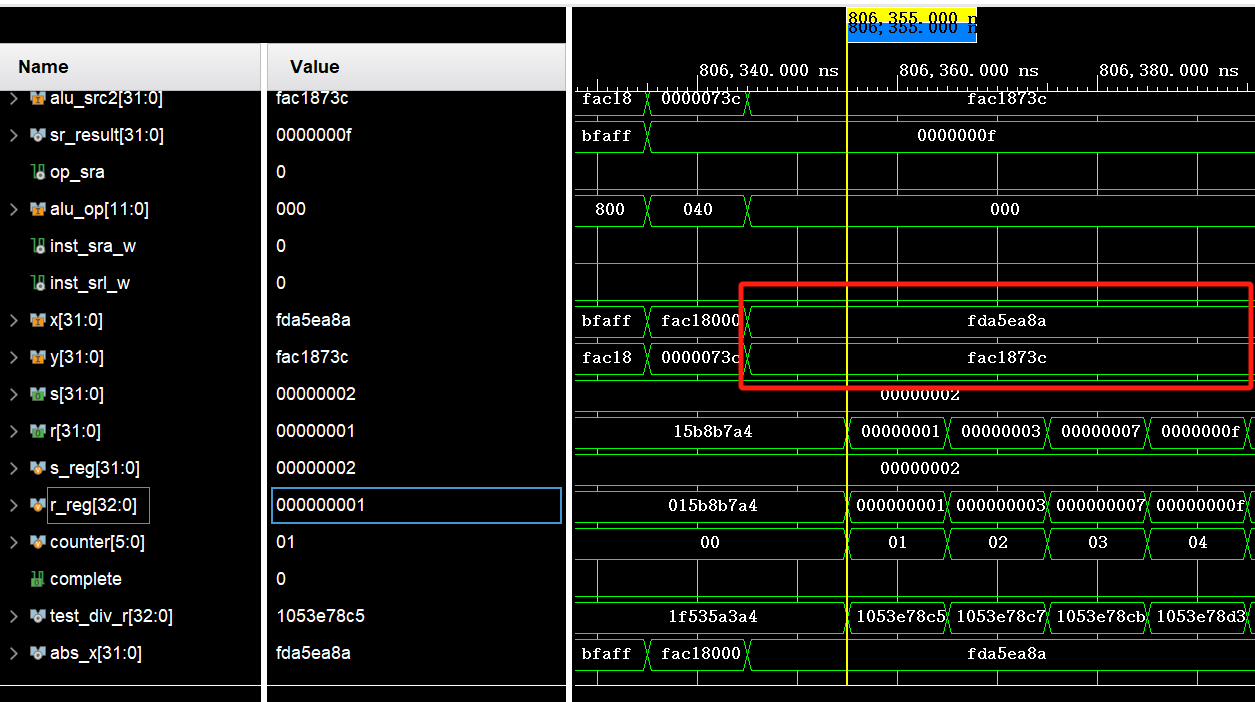
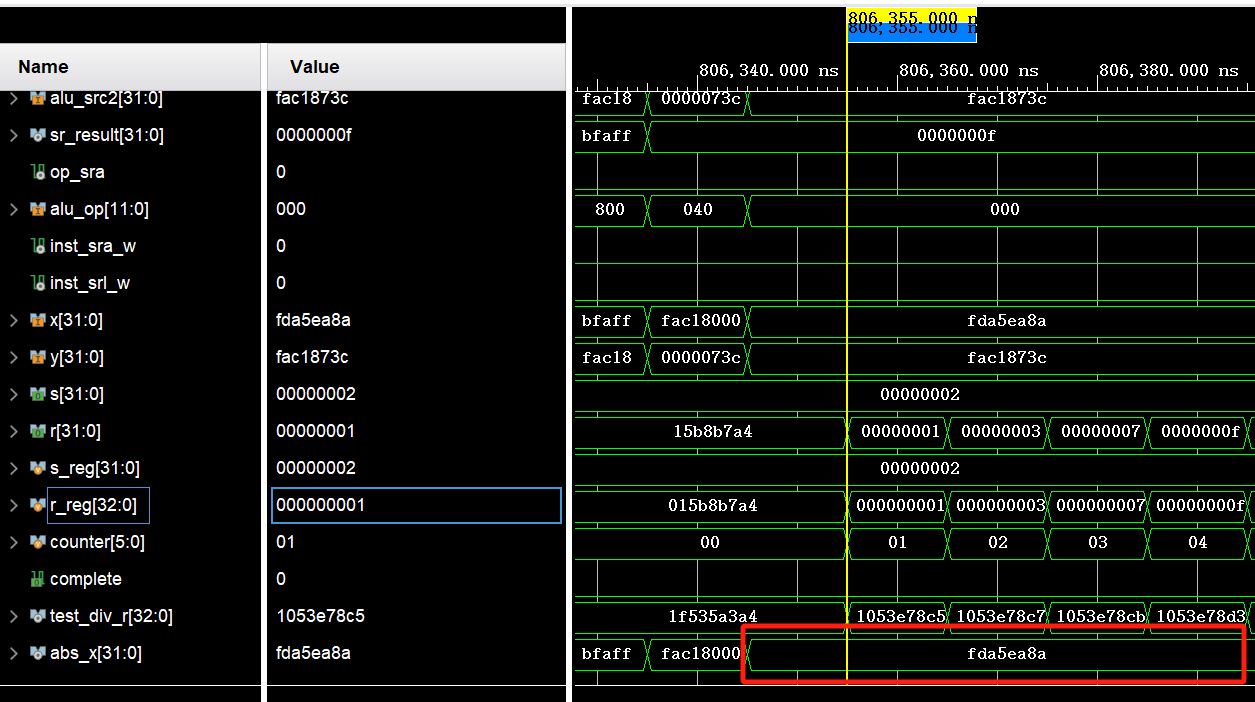


图25 错误2中数据RAM的读数据晚了一个周期

此时的除法器模块的x和y如上所示，本人通过计算，发现计算结果本就应该为0x00000000，但输出是0x00000001，于是经过讨论，本组成员首先认为是除法器的实现出错，但是通过给除法器单独写的test\_bench可以看出，除法器的功能并没有出错。

于是本组成员将目光又聚焦到出错位置的除法器信号，通过较长时间的排查，发现了除法器模块中信号不对的地方：



除法器模块中abs\_x代表的是x的绝对值，但显然对于div.w指令来说，此时x为有符号数，但此时abs\_x明显为负数，查看除法器模块对于abs\_x的处理如下：



可见其逻辑没有出错，那么出错的地方只可能是div\_signed信号出现问题了，查看ID阶段的div\_signed信号赋值逻辑，果然发现了问题：

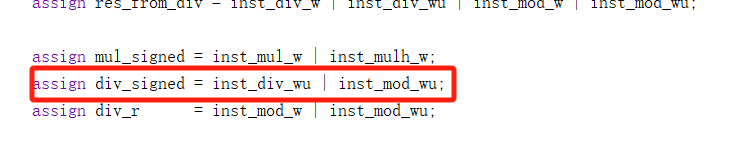


图26 错误2修改前代码

最后复盘发现，是因为写译码逻辑的同学与写除法器模块的同学的信号约定没有商量好，导致出现了歧义，所以才有了上面的代码。

* + - 1. 错误原因

写译码逻辑的同学与写除法器模块的同学的信号约定没有商量好，导致出现了歧义，所以导致传递给除法器模块的信号出错。

* + - 1. 修正效果

将div\_signed赋值逻辑进行如下修改即可：

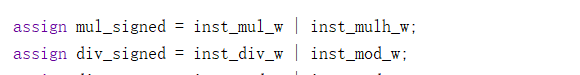


图30 错误3修改后代码\

* + 1. 错误6：exp10中乘法器结果未进行高位和低位选择
       1. 错误现象

Console报错如下：

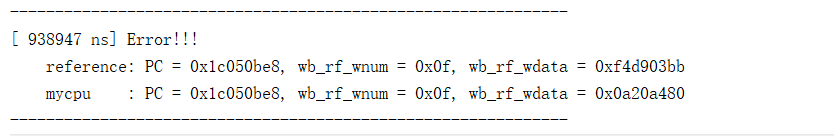


图 28 错误4对应的Console报错

* + - 1. 分析定位过程

发现wb\_rf\_wdata错误，由于已经跑了很久的仿真，所以猜想是对新加的指令的处理出错。于是本人首先查看了反汇编代码，发现此时是mulh.w指令，看来大概率是乘法指令处理出错，于是紧接着通过行为仿真，定位出错位置的相关信号，波形图如下：

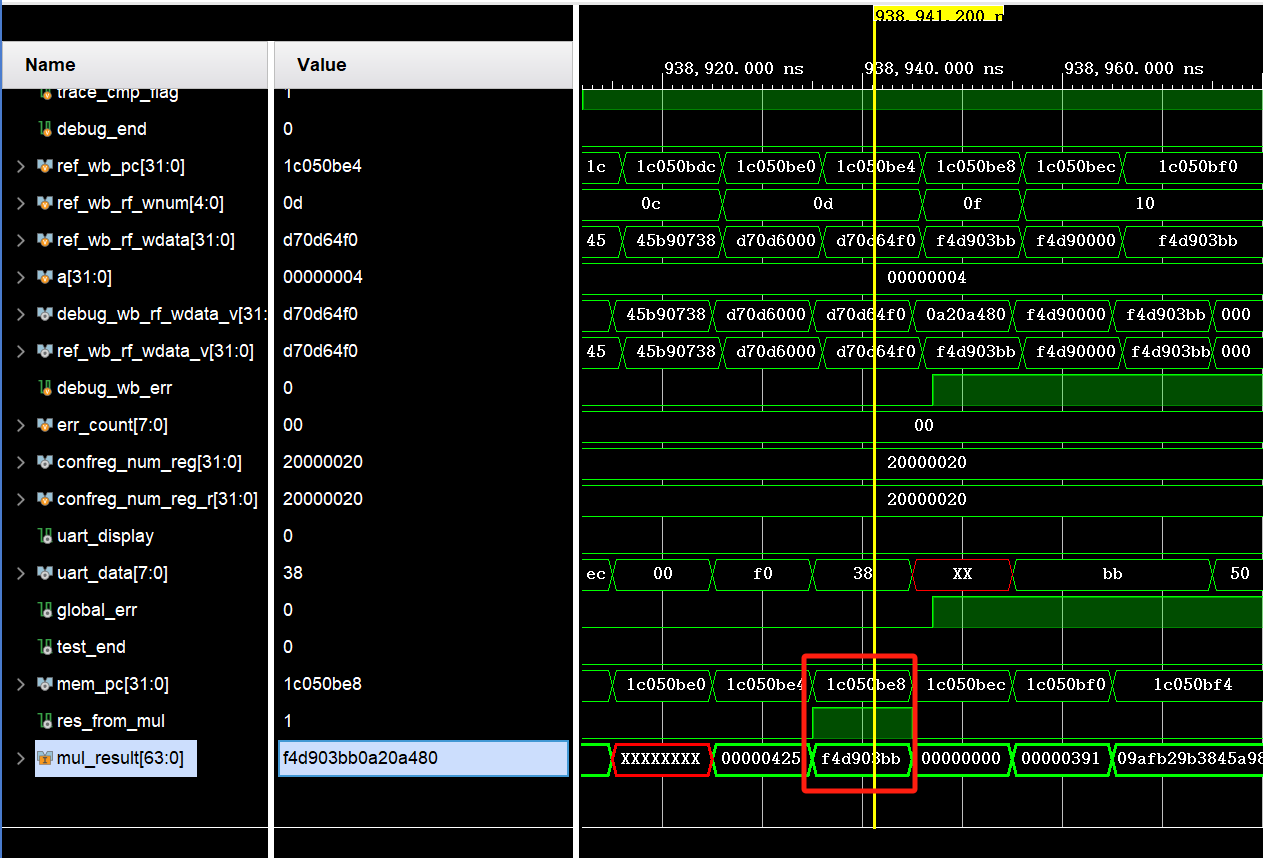
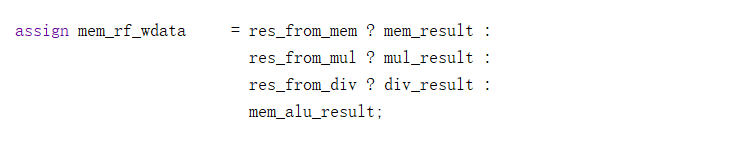


图25 错误4中数据RAM的读数据晚了一个周期

此时MEM流水级的的相关信号如上所示，可以看出mul\_result计算并未出错(其高位为0xf4d903bb)，但写回结果是0x0a20a480。所以很显然，最终写回结果选择了乘法器计算结果的低位，然后查看相关代码，发现问题所在：



可以看出，对最终结果的选择。并没有考虑选择乘法器的高位还是低位，而mem\_rf\_wdata位宽为32位，所以默认选择了乘法器计算结果的低位。

* + - 1. 错误原因

MEM流水级没有考虑选择乘法器的高位还是低位，而mem\_rf\_wdata位宽为32位，所以默认选择了乘法器计算结果的低位。

* + - 1. 修正效果

修改数据通路的位宽，添加mul\_h信号，其拉高代表选择乘法器计算结果的高位，并把MEM流水级写回数据的赋值逻辑作如下修改即可：

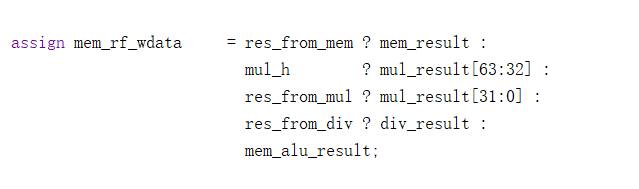
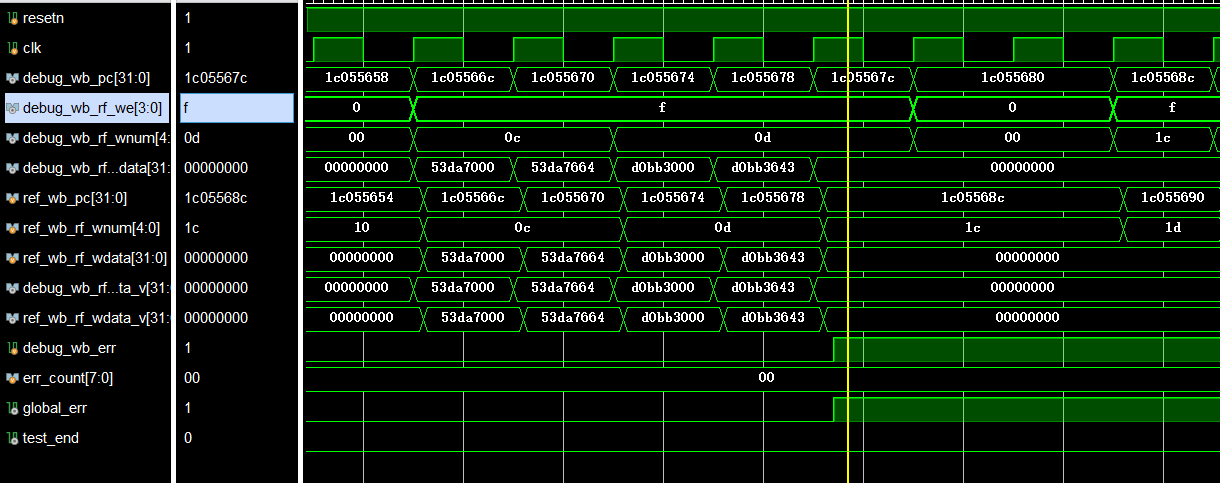


图30 错误4修改后代码

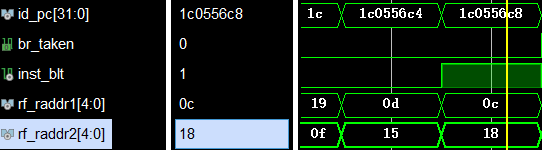
* + 1. 错误7：exp11中译码逻辑补充不全
       1. 错误现象

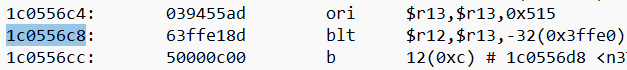
此错误发生两次，包含以下现象：

跳转指令的rf\_we错误拉高：



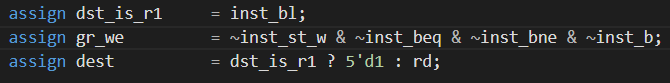
blt指令未考虑raddr2应为rd寄存器：





* + - 1. 分析定位过程

第一次，检查rf\_we的赋值逻辑，发现没有将新增指令的rf\_we拉低：



第二次，发现raddr2地址不对，检查src\_reg\_is\_rd信号，发现没有加入新增的分支与内存写信号：

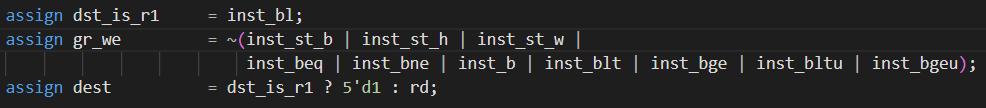
2-2

* + - 1. 错误原因

添加指令时对译码逻辑考虑不完全。

* + - 1. 修正效果

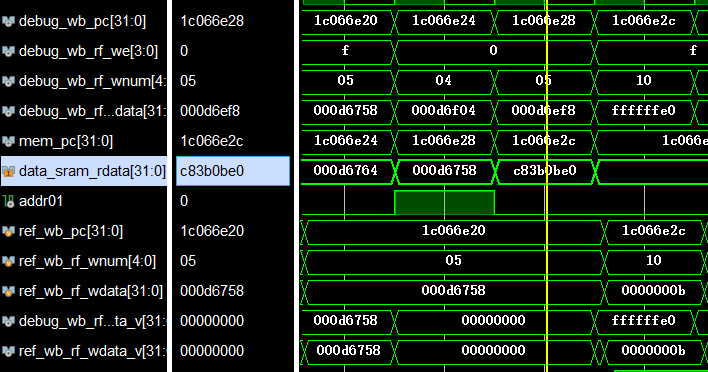
以上两次错误修正如下：



2-3

之后在ID流水级对已有的分支和访存指令进行搜索，将需要补充新增指令的地方都进行补充。

* + 1. 错误8：exp11中内存读未对齐4字节
       1. 错误现象



alu计算得到的后两位地址为01，这里应该选择0b为读数据，但实际选择了最低字节的e0。

* + - 1. 分析定位过程

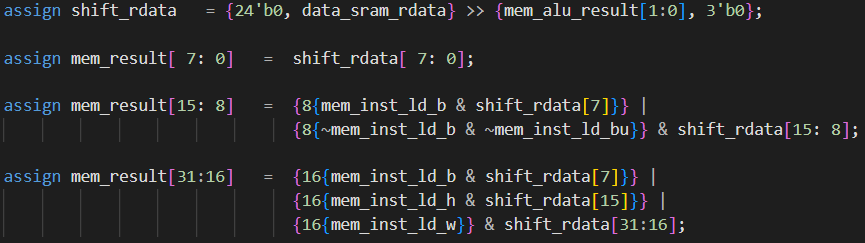
检查内存读数据产生逻辑，即发现在MEM阶段处理读数据时没考虑到地址后两位未对齐的问题。

* + - 1. 错误原因

在MEM阶段处理读数据时没考虑到地址后两位未对齐的问题。

* + - 1. 修正效果

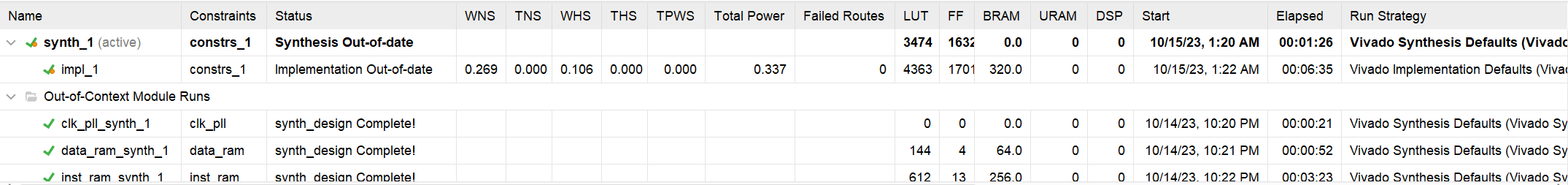
增加了按照地址后两位处理读数据的逻辑：



（逻辑经过优化）

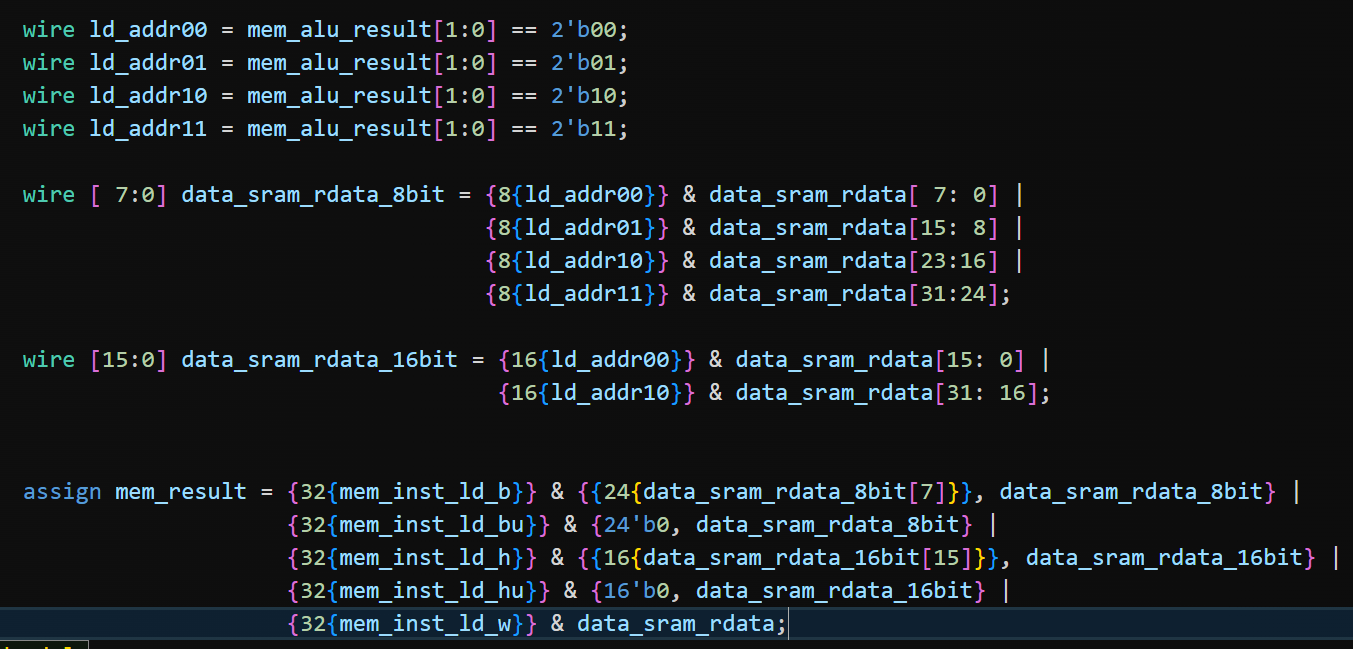
* 1. 对exp11的时序进行优化

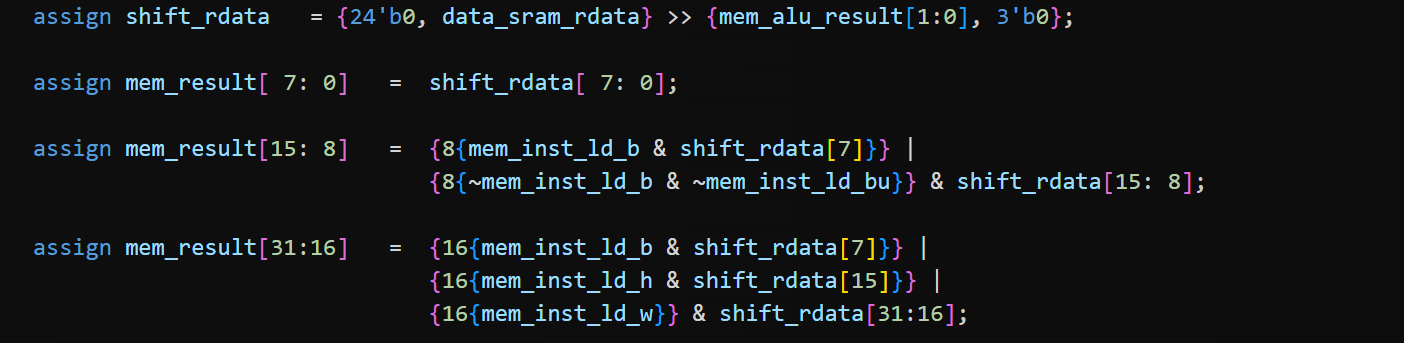
在exp10中，本小组对乘法器切成了两级流水，所以exp10中的时序较好



但在exp11中，由于访存指令的处理逻辑变为更加复杂，跳转的判断也更加复杂，导致exp11中的时序出现了问题，如下所示：

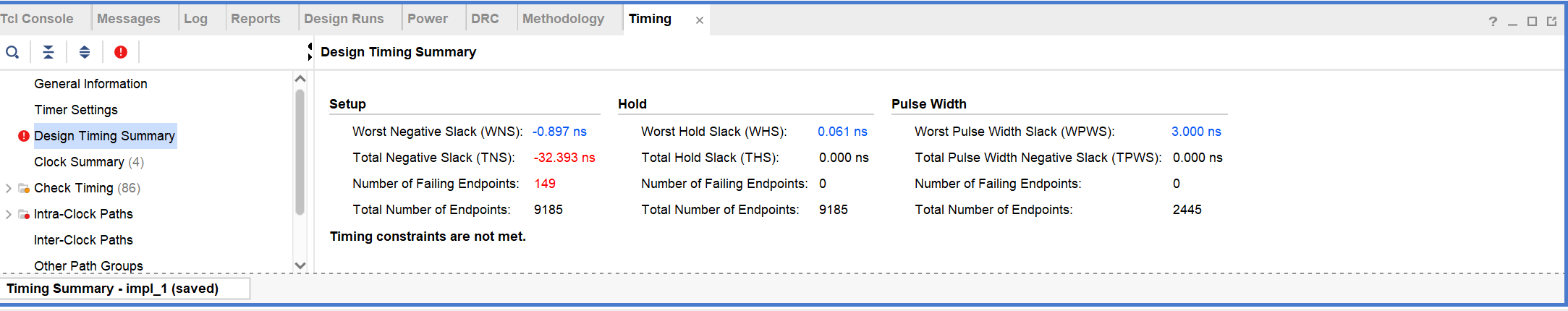
通过观察最差时序的路径，发现是访存指令的处理路径（访存指令的结果通过前递到ID流水级的延迟过高），于是本小组对最初的访存数据的处理逻辑进行了优化，如下所示：



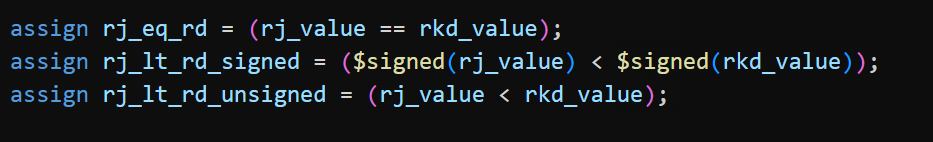


但是时序结果仍然不好：

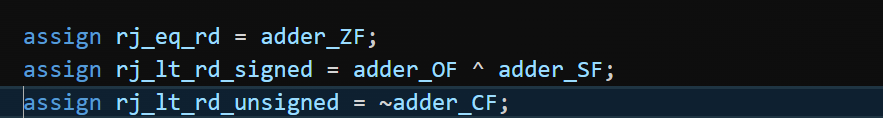
通过对代码进行分析，本小组认为可能是mem\_rf\_data使用了很多个多路选择器，可能导致延迟较高，于是尝试改为独热码进行与或操作，时序结果得到了改善，如下所示：



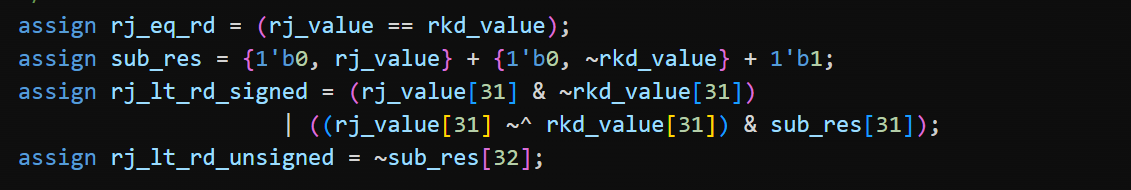
修改后，时序结果较最开始好了很多：继续对时序最差的路径进行分析，发现是br\_taken的判断逻辑的问题，于是本小组成员开始对br\_taken进行优化，最初本组成员的br\_taken的判断逻辑直接使用了 < 进行判断，如下所示：



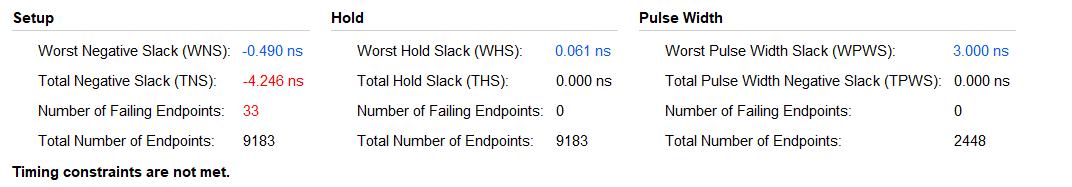
于是本小组尝试自己实现32位的块间并行，块间并行的先行进位加法器，通过CF,OF,SF等进行判断，如下所示：



但最后发现时序并不如直接使用 < ，于是最后本小组成员尝试直接使用 + 而不是自己写的加法器进行处理，逻辑如下：



最后发现时序变好了：



后续本小组尝试对其它地方进行修改，如把华莱士数从7层改为6层，把乘法器流水切成三级，均没有让时序变得更好（可能此时主要问题是跳转分支的判断逻辑）。然后把其它的多路选择器改为独热码与或逻辑也没有改善。由于本小组成员对时序优化并不太熟悉，且其它课程的作业压力较大，最后没有进行更进一步的尝试了。

1. 实验总结

本实验添加了算术逻辑运算、乘除法运算、转移、访存四类指令，基本上都可以复用现有的数据通路，只需对控制信号作一些修改，但也有一些地方需要设计新的通路和器件，比如在ALU中添加乘法器和除法器。

通过本次实验，本小组成员还对git的使用进一步熟悉，对如何使用git进行合作有了更深的理解。