**实验5报告**

蔡润泽、付轶凡

箱子号：45

一、实验任务（10%）

本次实验任务是在流水线CPU中添加运算类指令，具体包括算术逻辑运算类指令 ADD、ADDI、SUB、SLTI、SLTIU、ANDI、ORI、XORI、SLLV、SRAV、SRLV，乘除运算类指令 MULT、MULTU、DIV、DIVU，以及乘除法配套的数据搬运指令 MFHI、MFLO、MTHI、MTLO。并通过func\_lab6的仿真和上板验证。

二、实验设计（40%）

（一）总体设计思路

添加运算类指令后的硬件设计图如下：

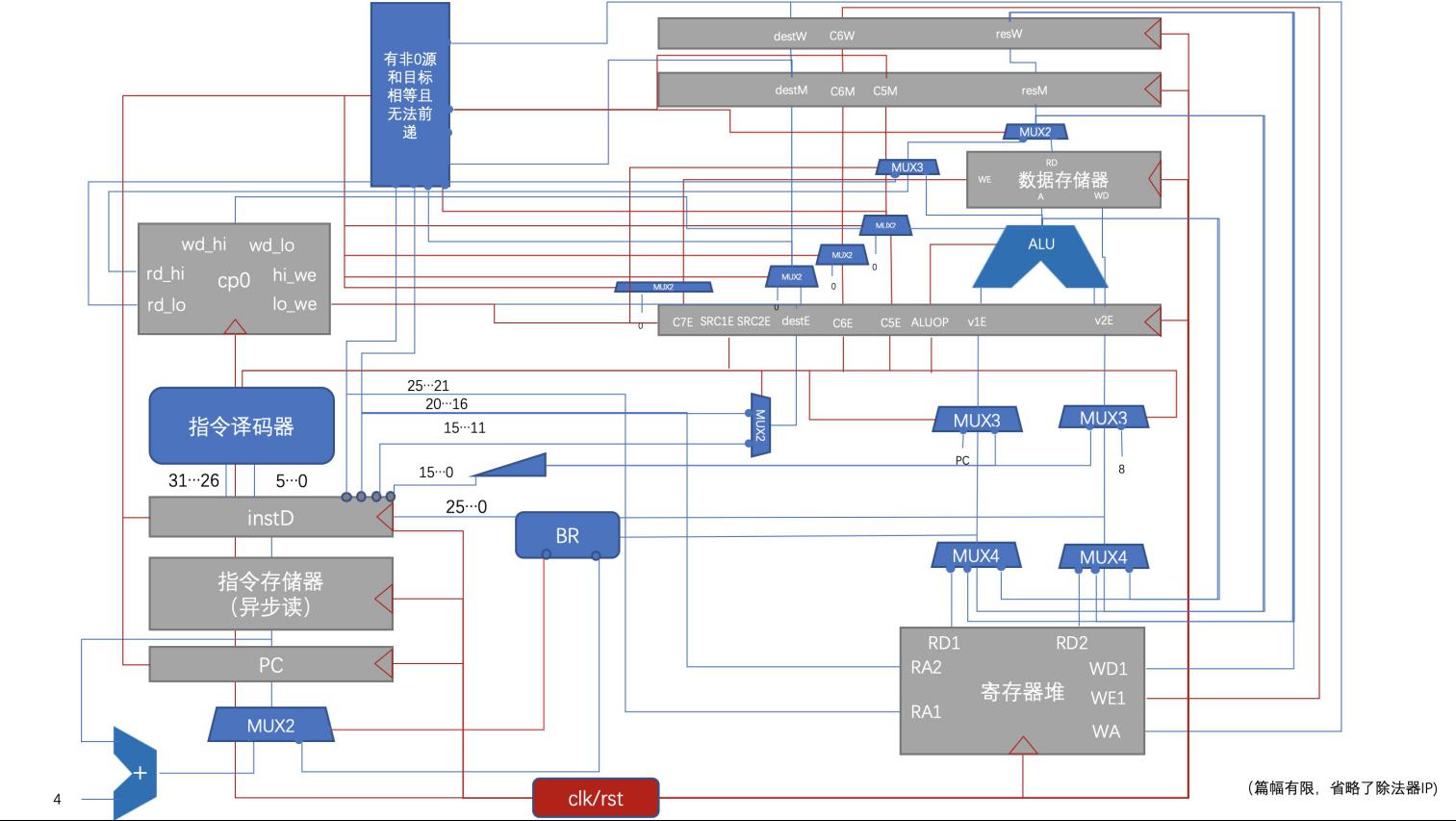


图1.1 硬件结构设计图

如图1.2的流水示意图，在代码设计中，主要有8个模块，包括五级流水、ALU、寄存器堆以及cp0协处理器（用来储存乘法指令的高32位、低32位，以及除法指令的余数、商）。该设计使用了四个IP，Inst\_RAM和Data\_RAM（采用同步RAM），mydiv和mydivu（分别计算有符号、无符号除法）。并且采用Tools模块进行译码。

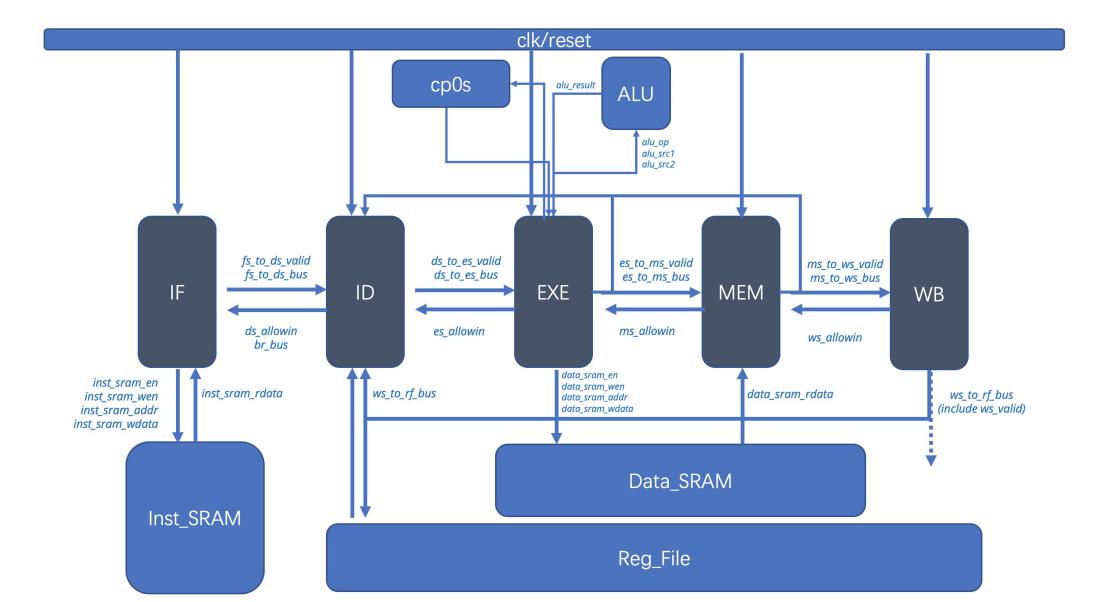


图1.2 流水示意图

（二）重要模块1设计：算数逻辑单元（ALU）模块

1. 工作原理

将CPU中的运算处理进行模块化，方便外界调用。同时模块化的ALU设计便于在其中增加新的运算功能，提高代码的扩展性。

1. 接口定义

input [15:0] alu\_op, //输入运算符

input [31:0] alu\_src1, //输入数据1

input [31:0] alu\_src2, //输入数据2

output [31:0] alu\_result //输出结果

1. 功能描述

采用16位的独热码对ALU进行控制，根据独热码进行16项（相较之前的实验增添了有符号、无符号乘除法）不同的算数逻辑运算操作，并将结果传回给exe阶段。

（三）重要模块2设计：寄存器堆（Reg\_File）模块

1. 工作原理

将32个32位宽的寄存器堆模块化，以实现两读一写，同步读异步写的操作。

1. 接口定义

input clk,

// READ PORT 1

input [ 4:0] raddr1,

output [31:0] rdata1,

// READ PORT 2

input [ 4:0] raddr2,

output [31:0] rdata2,

// WRITE PORT

input we, //write enable, HIGH valid

input [ 4:0] waddr,

input [31:0] wdata

1. 功能描述

当写使能信号为1时，在写回阶段对寄存器堆进行写入。同时，对于两个读端口信号，进行异步读取，将输出结果传递给ID阶段。

（四）重要模块3设计：取指阶段（IF\_stage）模块

1. 工作原理

将取指操作模块化，IF从inst\_ram中读出指令，并且在该周期模块之内处理PC的值，并且将指令传递给bus传递给ID模块，ID模块在下一周期再接收 。

1. 接口定义

表2.1 IF\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ds\_allowin | IN | 1 | ID模块允许接受IF传值 |
| br\_bus | IN | 33 | 输入是否跳转和branch的target |
| fs\_to\_ds\_valid | OUT | 1 | IF模块可以向ID模块传值 |
| fs\_to\_ds\_bus | OUT | 64 | IF模块向ID模块传递数据（指令码和地址） |
| inst\_sram\_en | OUT | 1 | Inst\_sram读使能 |
| inst\_sram\_wen | OUT | 4 | Inst\_sram写使能，此处恒为0 |
| inst\_sram\_addr | OUT | 32 | Inst\_sram目标地址 |
| inst\_sram\_wdata | OUT | 32 | Inst\_sram写数据 |
| inst\_sram\_rdata | IN | 32 | Inst\_sram读数据 |

1. 功能描述

PC在收到reset信号时设为偏移量32'hbfbffffc，并且在该周期模块之内处理PC的值，PC值的变化根据br\_bus取出来决定是否跳转还是加4。IF模块当inst\_sram\_en读使能信号为1时，将处理后的next\_pc作为地址传递给inst\_sram, 并从inst\_ram中读出指令。IF模块将取出的指令和地址传递给ID模块，ID模块在下一周期再接收 。

（五）重要模块4设计：译码阶段（ID\_stage）模块

1. 工作原理

将从IF模块获取的指令进行译码，获得指令格式类型、ALU操作类型、是否需要加载、写回内存和参与运算数据的值、是否需要加载协处理器cp0的hi和lo寄存器、是否需要写回协处理器cp0的hi和lo寄存器、跳转的目标PC。判断PC是否需要跳转，并将结果返还给IF模块。将译码后的数据和控制信号传递给EXE模块,EXE模块在下一时钟周期接受。另外，写回阶段的数据也通过该模块传递给寄存器堆。

CPU数据通路增加旁路设计，来让前面的指令直接把已经生成出来的结果直接转给后面的指令。在本设计中，采用了“流水级组合逻辑的结果传递到译码级寄存器读出处”的方案。并通过后续阶段的valid信号和gr\_we信号来控制ID模块中，rs\_value和rt\_value的值。

另外对于ready\_go信号，当译码级的指令和处在执行级的LW指令相关时，需要设置成“0”。

1. 接口定义

表2.2 ID\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| es\_allowin | IN | 1 | EXE模块允许接受ID传值 |
| ds\_allowin | OUT | 1 | 允许IF模块向ID模块传递数据 |
| fs\_to\_ds\_valid | IN | 1 | IF模块可以向ID模块传值 |
| fs\_to\_ds\_bus | IN | 64 | IF模块向ID模块传递数据（指令码及地址） |
| ds\_to\_es\_valid | OUT | 1 | 允许ID模块向ES模块传递数据 |
| ds\_to\_es\_bus | OUT | 145 | ID模块向EXE模块传递数据 |
| br\_bus | OUT | 33 | 输出是否跳转和branch的target给IF模块 |
| ws\_to\_rf\_bus | IN | 38 | WB模块向ID模块传递的需要写回REG FILE的信息 |
| es\_to\_ms\_bus | IN | 71 | EXE模块向MEM模块传递数据 |
| ms\_to\_ws\_bus | IN | 70 | MEM模块向WB模块传递数据 |
| es\_to\_ms\_valid | IN | 1 | EXE模块可以向MEM模块传值 |
| ms\_to\_ws\_valid | IN | 1 | MEM模块可以向WB模块传值 |
| out\_es\_valid | IN | 1 | 接收es\_valid是否为1 |
| out\_ms\_valid | IN | 1 | 接收ms\_valid是否为1 |

1. 功能描述

将从IF模块获取的指令进行译码，获得指令格式类型、ALU操作类型、是否需要加载、写回内存和参与运算数据的值、是否需要加载协处理器cp0的hi和lo寄存器、是否需要写回协处理器cp0的hi和lo寄存器、跳转的目标PC。判断PC是否需要跳转，并将结果返还给IF模块。将译码后的数据和控制信号在传递给EXE模块,EXE模块在下一时钟周期接受。另外，写回阶段的数据也通过该模块传递给寄存器堆。并且采用前递的方式来减少CPU的阻塞，缩短运行时间。

（六）重要模块5设计：执行阶段（EXE\_stage）模块

1. 工作原理

将从ID模块获取的指令相应的执行。将执行后的ALU结果和前阶段传递的通用寄存器写使能、写地址控制信号、PC传通过总线传递给MEM模块,MEM模块在下一周期接收新值。另外，load指令的发出读信号处理也在EXE阶段完成，EXE模块将数据传递给MEM模块，在下一周期WB阶段进行写回。输出数据RAM的写信号和数据。

此阶段在进行算数逻辑运算时，需要调用ALU模块。

在本次任务中，除了对ALU模块需要新增有符号、无符号乘法外，还需要在此阶段调用我们生成的mydiv和mydivu两个除法器IP完成新增的有符号、无符号除法运算，以及对除法器中的控制信号进行相应的设置。

1. 接口定义

表2.3 EXE\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ms\_allowin | IN | 1 | MEM模块允许接受EXE传值 |
| es\_allowin | OUT | 1 | EXE模块允许接受ID传值 |
| ds\_to\_es\_valid | IN | 1 | ID模块可以向EXE模块传值 |
| ds\_to\_es\_bus | IN | 145 | ID模块向EXE模块传递数据 |
| es\_to\_ms\_valid | OUT | 1 | EXE模块可以向MEM模块传值 |
| es\_to\_ms\_bus | OUT | 71 | EXE模块向MEM模块传递数据 |
| data\_sram\_en | OUT | 1 | data\_sram读使能 |
| data\_sram\_wen | OUT | 4 | data\_sram写使能 |
| data\_sram\_addr | OUT | 32 | data\_sram目标地址 |
| data\_sram\_wdata | OUT | 32 | data\_sram写数据 |
| out\_es\_valid | OUT | 1 | 将es\_valid的值传递给ID模块 |

mydiv和mydivu除法器IP数据通道定义如下：

s\_axis\_divisor\_tdata[31:0]; //除数

s\_axis\_divisor\_tready; //除数应答信号

s\_axis\_divisor\_tvalid; //除数请求信号

s\_axis\_dividend\_tdata[31:0]; //被除数

s\_axis\_dividend\_tready; //被除数应答信号

s\_axis\_dividend\_tvalid; //被除数请求信号

m\_axis\_dout\_tdata[63:0]; //商和余数，其中高32位为余数、低32位为商

m\_axis\_dout\_tvalid; //结果有效信号，表示除法已经算完，可以取得结果

aclk; //时钟信号

在除法指令处于执行流水级且没有对除法器成功输入数据的时候，同时将 s\_axis\_dividend\_tvalid 和s\_axis\_divisor\_tvalid 置为 1，向除法器IP发送调用除法器的请求。当发现s\_axis\_dividend\_tready 和 s\_axis\_divisor\_tready 反馈为 1 后，tvalid和tready成功握手，并需要在下一拍将s\_axis\_dividend\_tvalid和s\_axis\_di vior\_tvalid清为0，以此保证一个除法操作只调用一次除法器IP，避免除法器给CPU送多个结果从而导致出错。成功握手后，数据传入除法器的各个数据通路，8拍后，除法器产生结果并将m\_axis\_dout\_tvalid信号置1，表示可以取得除法结果。

为了避免除法指令与之后的指令产生“写后读”相关，考虑到除法器IP需要8拍才能拿到结果，所以如果执行流水级这一拍执行的是除法指令，需要将执行级阻塞住（即修改es\_ready\_go信号），拿到除法结果后再释放，因为通常程序中的除法指令较少，这样的阻塞设计不会太影响整体的性能。

1. 功能描述

将从ID模块获取的指令相应的执行。将执行后和前阶段传递的数据控制信号通过数据总线在下一时钟周期更新传给MEM模块。另外，load指令的发出读信号处理也在EXE阶段完成，EXE模块将数据传递给MEM模块，在下一周期进行写回。输出数据RAM的写信号和数据。

（六）重要模块6设计：访存阶段（MEM\_stage）模块

1. 工作原理

将从EXE模块获取的访存指令相应的执行。根据es\_to\_ms\_bs中的是否数据来自数据RAM信号确定是否有访存取出的数据，并将相应指令的最终结果和前阶段传递的通用寄存器写使能、写地址控制信号、PC传通过总线在下一时钟周期更新给WB模块。

1. 接口定义

表2.4 MEM\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ws\_allowin | IN | 1 | WB模块允许接受MEM传值 |
| ms\_allowin | OUT | 1 | MEM模块允许接受EXE传值 |
| es\_to\_ms\_valid | IN | 1 | EXE模块可以向MEM模块传值 |
| es\_to\_ms\_bus | IN | 71 | EXE模块向MEM模块传递数据 |
| ms\_to\_ws\_valid | OUT | 1 | MEM模块可以向EXE模块传值 |
| ms\_to\_ws\_bus | OUT | 70 | MEM模块向WB模块传递数据 |
| data\_sram\_rdata | OUT | 32 | data\_sram读出的数据 |
| out\_ms\_valid | OUT | 1 | 将ms\_valid的值传递给ID模块 |

1. 功能描述

将从EXE模块获取的访存指令相应的执行。确定是否有访存指令，并将相应指令的数据和前阶段传递的数据控制信号传通过总线在下一时钟周期更新给WB模块。

（七）重要模块7设计：访存阶段（WB\_stage）模块

1. 工作原理

将从MEM模块获取的写回指令相应的执行。确定是否有写回指令，并进行相应的操作。同时，该模块将PC、寄存器堆写使能、地址、和写回结果传递给debug模块，用于调试CPU的正确性。

表2.5 WB\_stage接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ws\_allowin | OUT | 1 | WB模块允许接受MEM传值 |
| ms\_to\_ws\_valid | IN | 1 | MEM模块可以向WB模块传值 |
| ms\_to\_ws\_bus | IN | 70 | MEM模块向WB模块传递数据 |
| ws\_to\_rf\_bus | OUT | 40 | WB模块向寄存器堆模块（通过ID模块）传递数据（包括ws\_gr\_we、ws\_valid等） |
| debug\_wb\_pc | OUT | 32 | debug显示PC |
| Debug\_wb\_rf\_wen | OUT | 4 | debug显示寄存器堆写使能 |
| Debug\_wb\_rf\_wnum | OUT | 5 | debug显示寄存器堆写地址 |
| Debug\_wb\_rf\_wdata | OUT | 32 | debug显示寄存器堆写数据 |

1. 功能描述

将从MEM模块获取的写回指令相应的执行。确定是否有写回指令，并进行相应的操作。同时，该模块将PC、寄存器堆写使能、地址、和写回结果传递给debug模块，用于调试CPU的正确性。

（八）重要模块8设计：协处理器cp0模块

1、工作原理

记录乘法指令生成的完整的64位乘积，以及除法指令生成的各32位的商和余数。

表2.7 cp0接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| rd\_hi | OUT | 32 | 从cp0模块中读取hi寄存器的值 |
| rd\_lo | OUT | 32 | 从cp0模块中读取lo寄存器的值 |
| hi\_we | IN | 1 | 写hi寄存器使能 |
| lo\_we | IN | 1 | 写lo寄存器使能 |
| wd\_hi | IN | 32 | 写hi寄存器数据，本实验中为乘法指令生成结果的高32位以及除法指令生成结果的余数 |
| wd\_lo | IN | 32 | 写lo寄存器数据，本实验中为乘法指令生成结果的低32位以及除法指令生成结果的商 |

2、功能描述

类似于寄存器堆regfile模块，实现对hi、lo寄存器的同步写和异步读。

三、实验过程（50%）

（一）实验流水账

10月12日 20：00-20：30 阅读讲义

10月12日 20：30-22：00 设计代码（除去除法指令）

10月13日 10：00-11：00 设计除法指令部分代码

10月13日 14：30-18：00 调试bug

10月13日19：30-22：30 撰写实验报告

（二）错误记录

1、错误1：andi、xori、ori指令立即数进行了符号扩展

（1）错误现象

运行仿真，比对trace，在PC为0xbfc54cf0处，参考的wb\_rf\_wdata为0x00008800，而mycpu得到的结果为0xea428800，如图3.1所示：

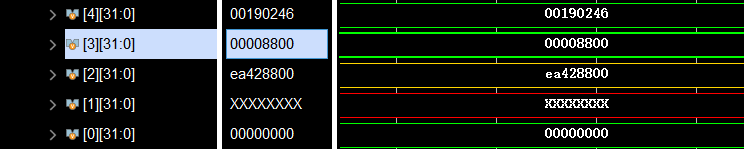


图3.1 当前指令写值出错

（2）分析定位过程

如图3.1，wb\_rf\_wdata的值出现错误，PC为0xbfc54cf0，在test.s汇编文件中找到对应的指令为“andi v0,t0,0x8d3e”，关注汇编文件中该指令之前的与t0寄存器（该指令的源操作数）相关的指令如图3.2。

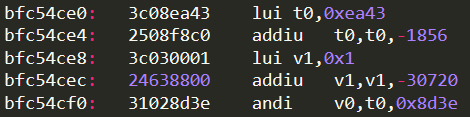


图3.2 相关指令

分析指令，其中lui、addiu两条指令将t0寄存器置为0xea42f8c0，在andi指令中t0与立即数0x8d3e按位与，应得到的结果为0x00008800，而我们得到的结果为0xea428800，立刻想到andi指令的立即数进行了符号扩展从而导致得到的结果高位没有清零。从andi指令出错，立刻想到xori、ori指令也会出现同样的错误，于是一并修改。

（3）错误原因

andi、xori、ori指令的立即数进行了符号扩展。

（4）修正效果

在ID流水级增加src2\_is\_uimm信号，表示第二个源操作数是零扩展的立即数，译码级增加“assign src2\_is\_uimm = inst\_andi | inst\_ori | inst\_xori;”，将信号添加到ds\_to\_es\_bus中传给执行级的es\_src2\_is\_uimm，并在执行级中es\_alu\_src2增加一个零扩展的选择“es\_src2\_is\_uimm? {{16{1'b0}}, es\_imm[15:0]}:”，这样就完成了这三条指令立即数的零扩展。修改后进行仿真，该问题被解决。

2、错误2：除法指令商与余数储存的寄存器弄反

（1）错误现象

运行仿真，比对trace，在PC为0xbfc3095c处，参考的wb\_rf\_wdata为0x00000002，而mycpu得到的结果为0x15b8b7a4，如图3.3所示：

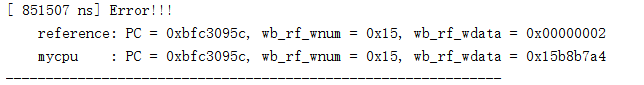


图3.3 wb\_rf\_wdata出错

（2）分析定位过程

如图3.3，wb\_rf\_wdata的值出现错误，PC为0xbfc3095c，当前PC为“mflo s5”，上一个PC进行了有符号除法操作div，抓取es\_dout\_tdata到波形图中如图3.4所示：



图3.4 es\_dout\_tdata的值

从图中可以看出mydiv除法器IP生成的结果为0x0000000215b8b7a4，即商为0x00000002，余数为0x15b8b7a4，当前PC对应的指令为“mflo s5”，即加载lo寄存器，lo寄存器存储除法结果的商，应为0x00000002，而我们的结果为0x15b8b7a4，显然是商和余数的存储位置反了。

（3）错误原因

除法指令商和余数的存储位置设置反了。

（4）修正效果

重新设置商和余数的储存位置如下图代码所示：

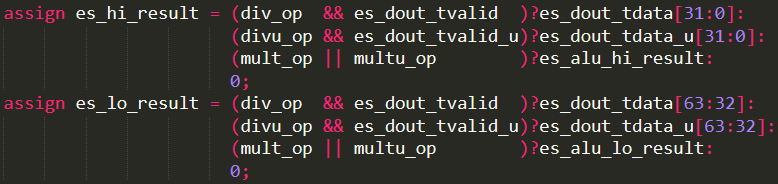


图3.5 写hi、lo寄存器值的设置

重新仿真后，该问题被解决。

3、错误3：请求信号valid时序逻辑有误

（1）错误现象

运行仿真，比对trace，虽然一直没有报错，但是仿真停在了PC值为0xbfc275d8的位置，如下图：

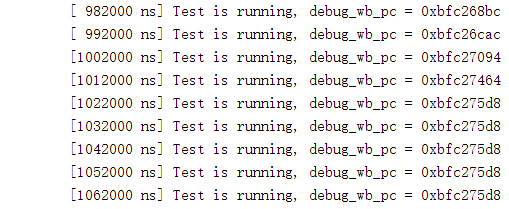


图3.6 PC值停在0xbfc275d8

（2）分析定位过程

执行级正在执行的指令是“divu zero,t0,t1”，如图3.7，观察波形图，发现此时除法被阻塞在执行级且始终没有向除法器IP发送请求信号es\_divisor\_tvalid\_u和es\_dividend\_tvalid\_u：

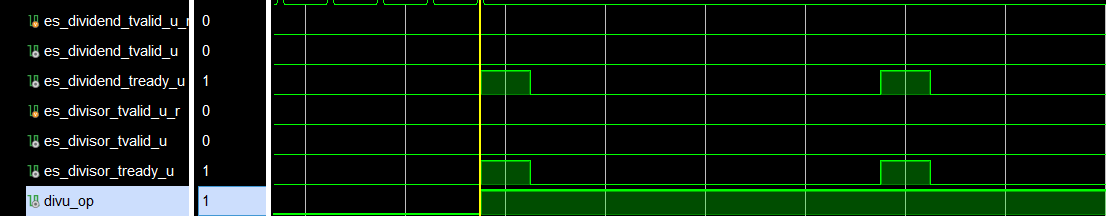


图3.7 es\_divisor\_tvalid\_u和es\_dividend\_tvalid\_u始终为0

于是关注两个valid信号的时序逻辑代码，如图3.8：

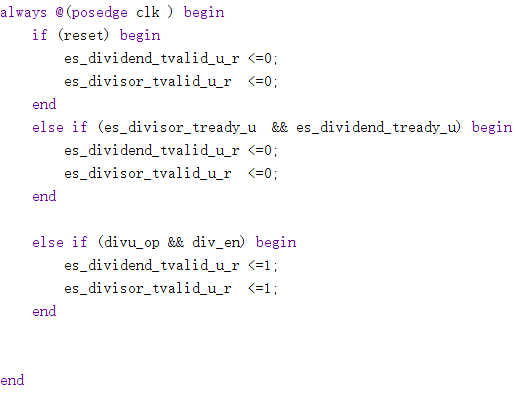


图3.8 valid信号时序逻辑

如图3.8，我们之前设置代码段的想法是，一旦执行级进行除法操作，就在下一拍将两个valid信号拉高，向除法器发出请求，同时令div\_en信号为0，这就保证了在同一次除法操作中，valid信号不会拉高第二次；另外，一旦接受到两个ready信号的高电平反馈，就撤销两个valid信号，但是我们写的时候把这两种情况的优先级弄反了，这样就出现了上面的问题，由于除法器每八拍向CPU发一次ready信号，在这一拍刚好除法信号被拉高，按照原代码的判断逻辑的优先级，就会在下一拍将两个valid信号置为0，同时div\_en也置为0，导致两个valid信号再也无法拉高，从而出错。

（3）错误原因

Valid信号的时序逻辑出错。

（4）修正效果

只需将这一部分代码优先级调换即可，如图：

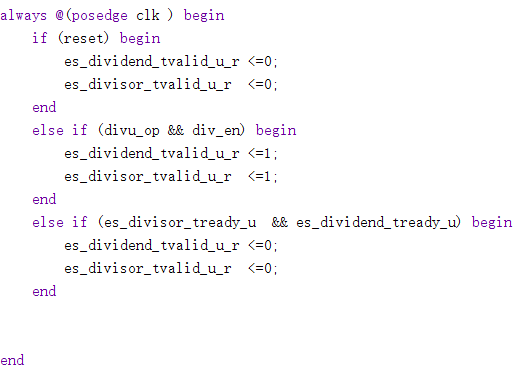


图3.9 修改后的valid时序逻辑

重新仿真后，该问题被解决。

4、错误4：乘法指令的alu设置出错

（1）错误现象

运行仿真，比对trace，在PC为0xbfc57c5c处，参考的wb\_rf\_wdata为0x0a20a480，而mycpu得到的结果为0x0XX0XXX0，如图3.10所示：

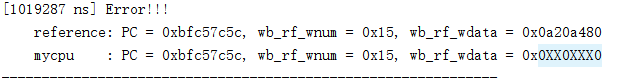


图3.10 wb\_rf\_wdata出错

（2）分析定位过程

如图3.6，wb\_rf\_wdata的值出现错误，PC为0xbfc57c5c，当前PC为“mflo s5”，上一个PC进行的有符号乘法操作mult，抓取mult\_result到波形图中如图所示：

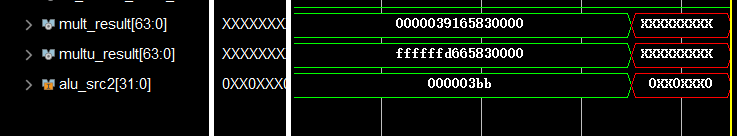


图3.11 mult\_result的值

抓取alu\_src2到波形发现正是alu\_src2出错，但再抓取相关的信号却无法定位到底是哪里错了，于是再仔细地看了一遍代码梳理乘法操作的流程，在alu模块中发现有符号乘法和无符号乘法的操作写反了，如下图：



图3.12 alu中的mult\_result和multu\_result

（3）错误原因

Alu模块有符号、无符号乘法操作写反了。

（4）修正效果

修改alu模块的这两个操作如下：



图3.13 修改后的mult\_result和multu\_result

重新仿真后，该问题被解决并成功通过所有测试。

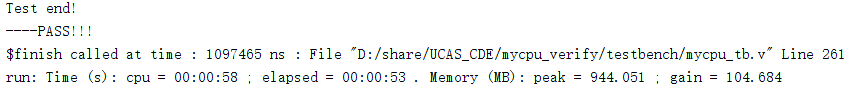


图3.10 测试通过

四、实验总结（可选）

本次实验需要自己通过查看MIPS手册来添加一些新的指令。在调试代码的过程中，感触最深的一点是我们对于指令手册的前期阅读不够仔细，导致出现了指令理解出错，例如符号位扩展，商和余数如何放置等问题。希望在后期的实验过程中，在设计代码之前，能够更加细心的思考清楚指令如何实现，而不是粗略的读一遍任务书或者指令手册，这样后期浪费更多的时间就有点得不偿失了。