**重庆大学课程设计报告**

课程设计题目： MIPS SOC设计

学 院： 计算机学院

专业班级： 信息安全1班、2班

年 级： 2021级

姓 名： 罗皙、 刘倬宇、 庞博、 杨佳俊

学 号：20215509、20215292、20215238、20215223

完成时间： 2024 年 1 月 12 日

成 绩：

指导教师： 吴长泽

重庆大学教务处制

综合设计指导教师评定成绩表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 项目 | 分值 | 优秀  (100>x≥90) | 良好  (90>x≥80) | 中等  (80>x≥70) | 及格  (70>x≥60) | 不及格(x<60) | 评分 |
| 参考标准 | 参考标准 | 参考标准 | 参考标准 | 参考标准 |
| 学习态度 | 15 | 学习态度认真，科学作风严谨，严格保证设计时间并按任务书中规定的进度开展各项工作 | 学习态度比较认真，科学作风良好，能按期圆满完成任务书规定的任务 | 学习态度尚好，遵守组织纪律，基本保证设计时间，按期完成各项工作 | 学习态度尚可，能遵守组织纪律，能按期完成任务 | 学习马虎，纪律涣散，工作作风不严谨,不能保证设计时间和进度 |  |
| 技术水平与实际能力 | 25 | 设计合理、理论分析与计算正确，实验数据准确，有很强的实际动手能力、经济分析能力和计算机应用能力，文献查阅能力强、引用合理、调查调研非常合理、可信 | 设计合理、理论分析与计算正确，实验数据比较准确，有较强的实际动手能力、经济分析能力和计算机应用能力，文献引用、调查调研比较合理、可信 | 设计合理，理论分析与计算基本正确，实验数据比较准确，有一定的实际动手能力，主要文献引用、调查调研比较可信 | 设计基本合理，理论分析与计算无大错，实验数据无大错 | 设计不合理，理论分析与计算有原则错误，实验数据不可靠，实际动手能力差，文献引用、调查调研有较大的问题 |  |
| 创新 | 10 | 有重大改进或独特见解，有一定实用价值 | 有较大改进或新颖的见解，实用性尚可 | 有一定改进或新的见解 | 有一定见解 | 观念陈旧 |  |
| 论文(计算书、图纸)撰写质量 | 50 | 结构严谨，逻辑性强，层次清晰，语言准确，文字流畅，完全符合规范化要求，书写工整或用计算机打印成文；图纸非常工整、清晰 | 结构合理，符合逻辑，文章层次分明，语言准确，文字流畅，符合规范化要求，书写工整或用计算机打印成文；图纸工整、清晰 | 结构合理，层次较为分明，文理通顺，基本达到规范化要求，书写比较工整；图纸比较工整、清晰 | 结构基本合理，逻辑基本清楚，文字尚通顺，勉强达到规范化要求；图纸比较工整 | 内容空泛，结构混乱，文字表达不清，错别字较多，达不到规范化要求；图纸不工整或不清晰 |  |

指导教师评定成绩：

指导教师签名： 年 月 日

**MIPS SOC设计报告**

罗皙、刘倬宇、庞博、杨佳俊

#### 一、设计简介

本次硬件综合实验课程，我们实现了具有57条指令、连接了SRAM接口、AXI接口、实现了基础cache的五级流水线的MIPS处理器。此次的设计，我们成功通过了89个功能测试点、性能测试的10个程序的仿真以及一系列需要上板的操作，并最终成功得到了相应的性能分数。我们的设计以计算机组成原理的lab4实验的标准代码为基础，通过添加指令、模块和数据通路，成功完成了本次课程的相关要求。

##### （一）小组分工说明

杨佳俊：添加逻辑运算指令、移位指令、性能测试与一系列的上板操作

庞 博：添加访存指令、数据移动指令、特权指令

刘倬宇：添加算术运算指令、添加异常处理模块与cpu0

罗 皙：添加分支跳转指令、封装SRAM与AXI接口、添加cache

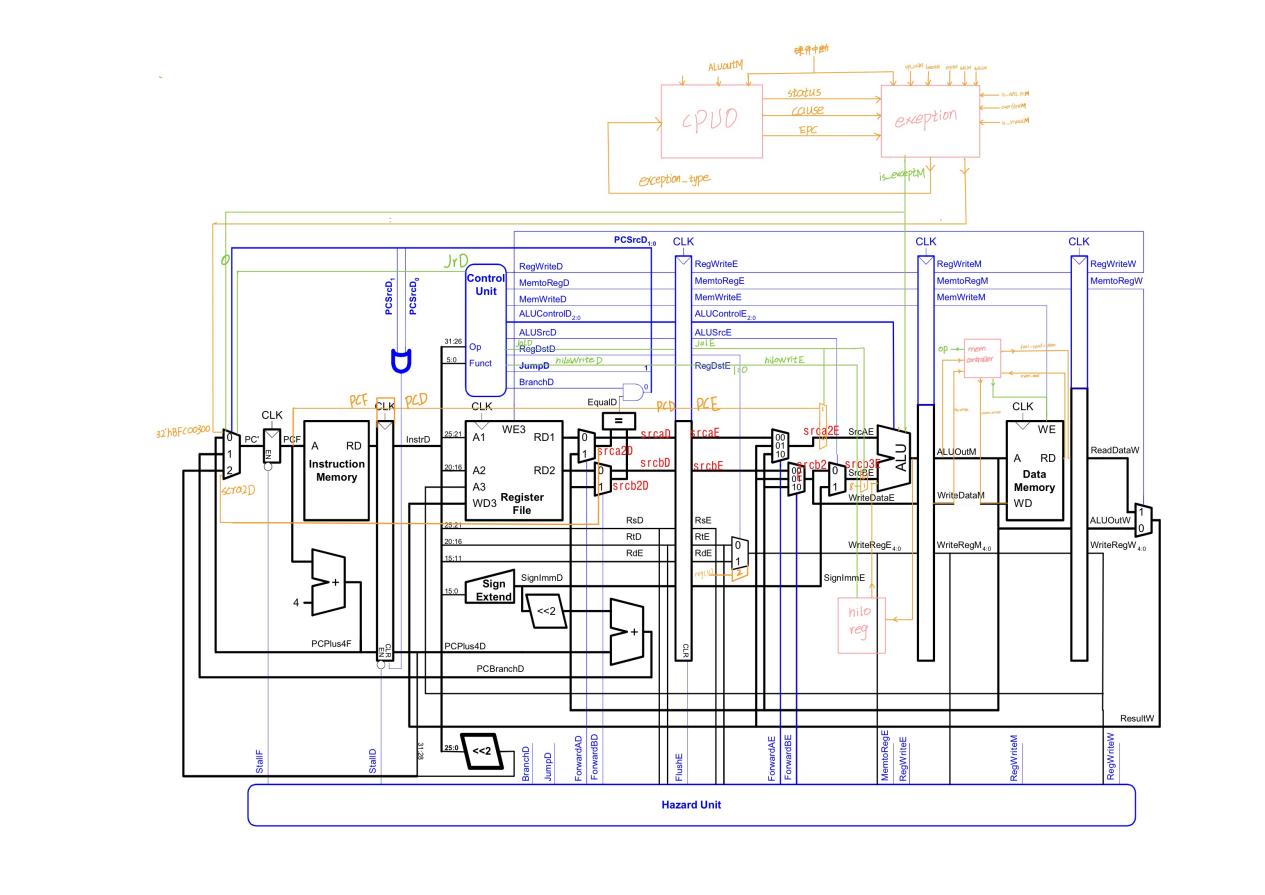
全体组员：对通路进行调试，对功能测试调试、跑通测试点

#### 二、设计方案（30%）

##### （一）总体设计思路

总体设计思路是先添加除特权指令的其他52条指令，然后封装SRAM接口跑通89个功能测试点，然后添加异常与特权指令，最后封装AXI接口与添加cache。其中5 级流水线 MIPS CPU共有取指、译码、执行、访存、回写五个阶段，处理器的工作通过从指令存储器中取值，通过译码产生指令的控制信号，用于控制指令按规定的的数据通路执行，执行阶段进行运算、访存阶段访问数据存储器、回写阶段写入寄存器堆。

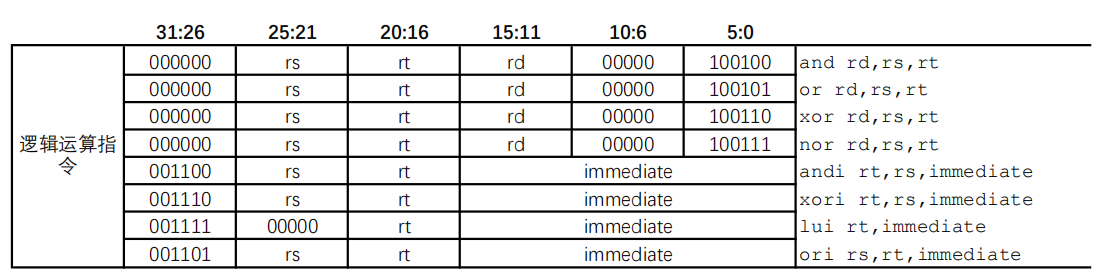
数据通路图：

为实现 57 条指令，需要完成的设计有控制信号的设计、数据通路的设计、 处理数

据冒险等。我们采用对指令分类，分模块逐步实现的方式，具体的设计详细描述如下：

###### 逻辑运算指令

* 1. 指令机器码



1.2 and,or,xor,nor指令

1.2.1 功能描述

寄存器 rs 中的值与寄存器 rt 中的值按位逻辑与，结果写入寄存器 rd 中。

1.2.2 数据通路调整

数据通路与lab4相同，无需调整，只需在alu中增加对应运算即可。

1.2.3 控制信号调整

扩展alucontrol信号种类，识别在alu中进行何种运算。

1.3 andi，ori，xori，lui指令

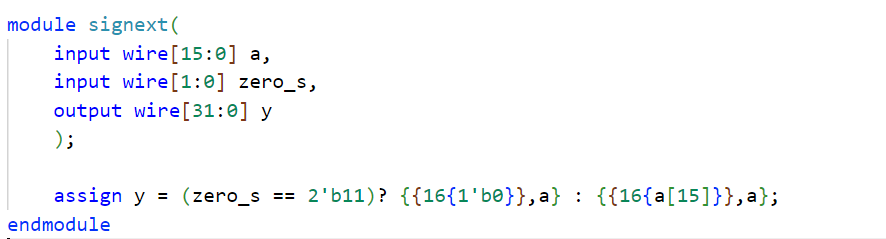
1.3.1 功能描述

ANDI ORI XORI：寄存器 rs 中的值与 0 扩展至 32 位的立即数 imm 按位逻辑运算，结果写入寄存器 rt 中。

LUI：将 16 位立即数 imm 写入寄存器 rt 的高 16 位，寄存器 rt 的低 16 位置 0。

1.3.2 数据通路调整

ANDI ORI XORI三条指令是零扩展，其他需要扩展的指令都是有符号扩展，而LUI指令如何扩展都不影响结果，因为LUI指令只需要低十六位。只需根据这三条指令的op码中间两位判断是否零扩展即可。将op码的中间两位传入signext模块，并修改signext模块如下：

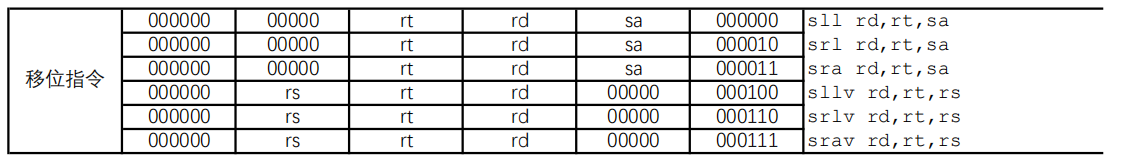


1.3.3 控制信号调整

扩展 alucontrol 信号的种类，用于识别并控制这些指令在 ALU 中执行何种操作。

###### 移位运算指令

2.1 指令机器码



2.2 立即数移位运算SLL SRL SRA

2.2.1 功能描述

SLL SRL: 由立即数 sa 指定移位量，对寄存器 rt 的值进行逻辑移位，结果写入寄存器 rd 中。SRA：由立即数 sa 指定移位量，对寄存器 rt 的值进行算术右移（即左边补上符号位），结果写入寄存器 rd 中。

2.2.2 数据通路调整

新增 sa 信号，从指令的[10:6]位直接连入 ALU 中。需在 ALU 中对应处理运

算。并在数据通路中将sa的值有ID阶段传递至EXE阶段：





2.2.3 控制信号调整

扩展 alucontrol 信号的种类，用于识别并控制这些指令在 ALU 中执行何种操作。

2.3 变量移位运算SLLV SRLV SRAV

2.3.1 功能描述

由寄存器 rs 中的值指定移位量，对寄存器 rt 的值进行逻辑移位，结果写入寄存器 rd 中。

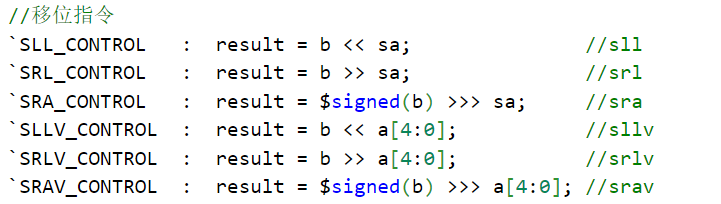
2.3.2 数据通路调整

数据通路与计组Lab4相同，无需调整。

2.3.3 控制信号调整

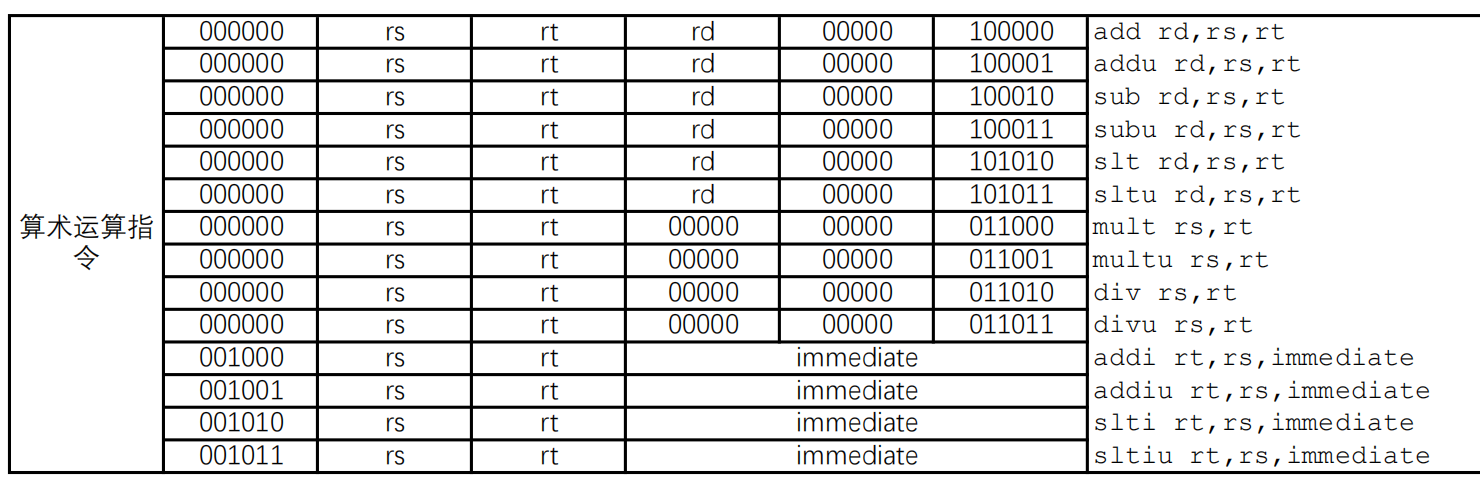
扩展 alucontrol 信号的种类，用于识别并控制这些指令在 ALU 中执行何种操作。

移位指令在alu中的运算如下：



###### 算术运算指令

3.1机器码



3.2指令实现过程

3.2.1指令功能

ADD：将寄存器 rs 的值与寄存器 rt 的值相加，结果写入寄存器 rd 中。如果产生溢出，则触发整型溢出例外；

ADDU：将寄存器rs的值与寄存器rt的值相加，结果写入rd寄存器中；

SUB：将寄存器rs的值与寄存器rt的值相减，结果写入rd寄存器中。如果产生溢出，则触发整型溢出例外；

SUBU：将寄存器rs的值与寄存器rt的值相减，结果写入rd寄存器中；

SLT:将寄存器rs的值与寄存器rt中的值进行有符号数比较，如果寄存器rs中的值小，则寄存器rd置1，否则寄存器rd置0；

SLTU:将寄存器rs的值与寄存器rt中的值进行无符号数比较，如果寄存器rs中的值小，则寄存器rd置1； 否则寄存器rd置0；

MULT：有符号乘法，寄存器rs的值乘以寄存器rt的值，乘积的低半部分和高半部分分别写入HILO寄存器的低32位和高32位；

MULTU:无符号乘法，寄存器rs的值乘以寄存器rt的值，乘积的低半部分和高半部分分别写入HILO寄存器的低32位和高32位；

DIV：有符号除法，寄存器rs的值除以寄存器rt的值，商写入HILO寄存器的低32位中，余数写入HILO寄存器的高32位中；

DIVU：无符号除法，寄存器 rs 的值除以寄存器 rt 的值，商写入HILO寄存器的低32位中，余数写入HILO寄存器的高32位中；

ADDI：将寄存器rs的值与有符号扩展至32位的立即数imm相加，结果写入rt寄存器中。如果产生溢出，则触发整型溢出例外；

ADDIU：将寄存器rs的值与有符号扩展至32位的立即数imm相加，结果写入rt寄存器中；

SLTI：将寄存器rs的值与有符号扩展至32位的立即数imm进行有符号数比较，如果寄存器rs中的值小，则寄存器rt置1；否则寄存器rt置0；

SLTIU：将寄存器rs的值与有符号扩展至32位的立即数imm进行无符号数比较，如果寄存器rs中的值小，则寄存器rt置1；否则寄存器rt置0。

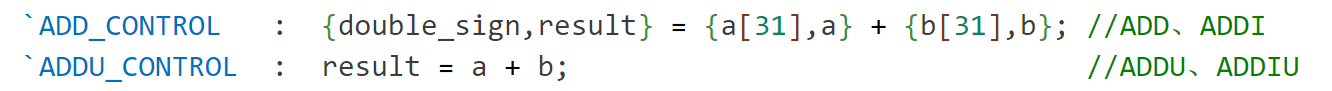
3.2.2实现过程

可以将这14条指令两两分组，ADD和ADDU、ADDI和ADDIU、SUB和SUBU、SLT和SLTU、SLTI和SLTIU、DIV和DIVU、MULT和MULTU。

每一组中的两条指令的功能是完全一样的，唯一的区别在于操作数是看成有符号还是无符号。这意味着在数据通路设计方面，可以复用一些已有的通路。

如果将ALU看成一个黑盒，那么不同组指令执行的操作是完全一样的，都是从寄存器或立即数中取得源操作数，送入ALU中进行运算，然后将结果写回rd或rt寄存器中，所以区别仅在于ALU内部执行的操作。

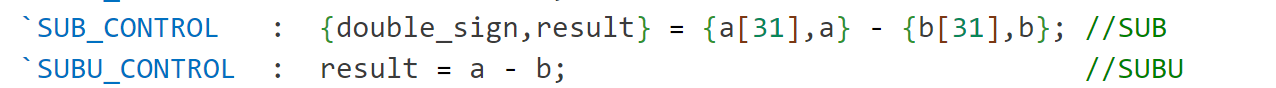
①ADD、ADDI和ADDU、ADDUI：



其中我们在对有符号的运算时使用了一个符号位double\_sign来作为最高位相加运算的结果，当操作为有符号的加或减时对double\_sign的值与运算结果的最高位进行异或运算，若结果为一，则产生溢出；若为零，则不产生溢出。

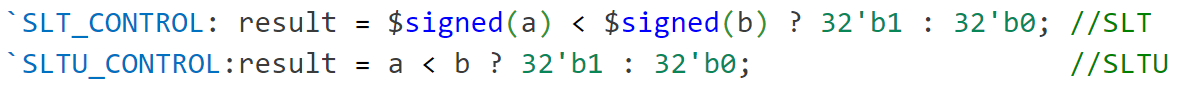


②SUB、SUBI和SUBU、SUBUI：



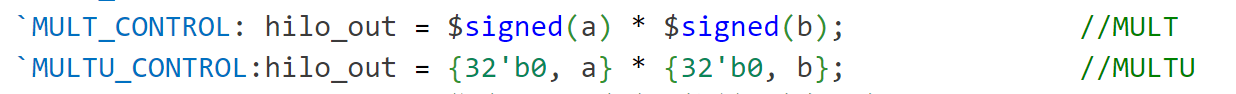
其中double\_sign与加法的用法一样。

③SLT和SLTU：



有符号的小于则置位使用了$sign()。$signed(c)是一个function，将无符号数c转化为有符号数返回，不改变c的类型和内容。

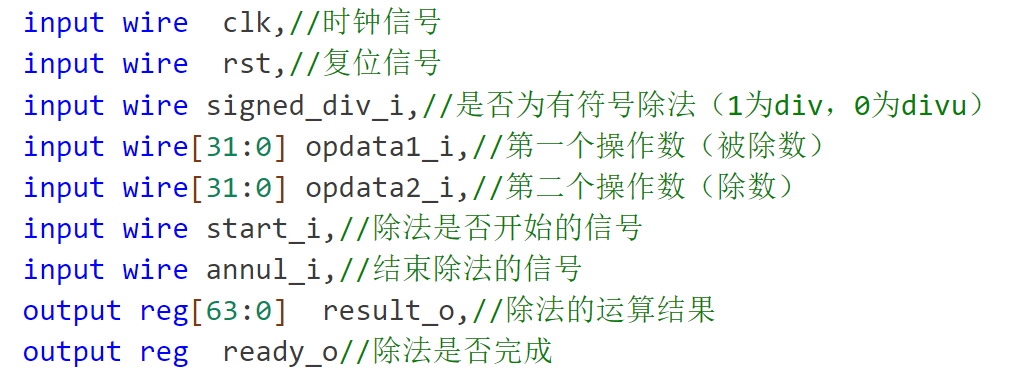
④MUL和MULTU：



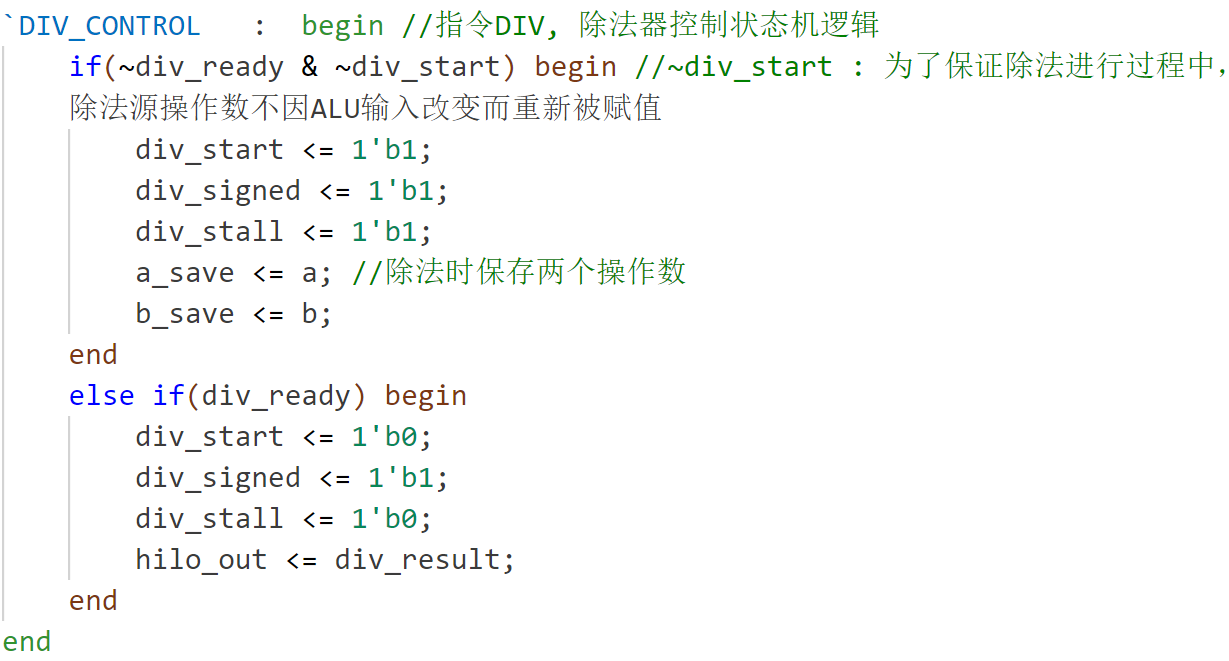
同样使用了$sign()函数来进行有符号乘法的运算。将运算的64位结果存入hilo\_out中，将这个信号传入hilo寄存器。

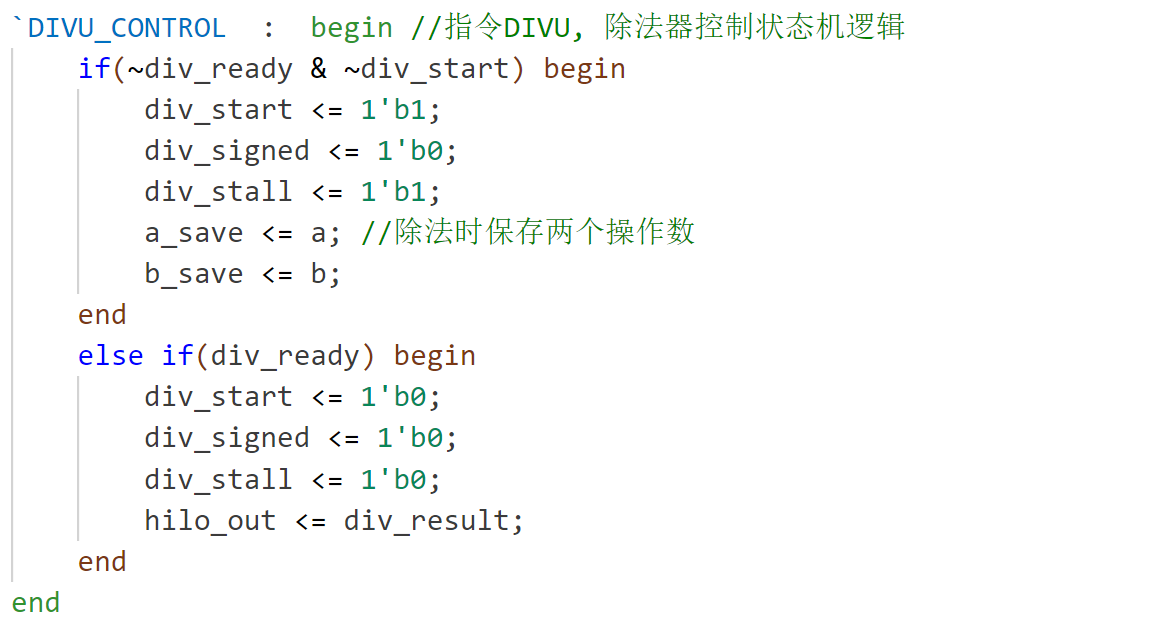
⑤：DIV和DIVU：

我们在ALU中接入了ref\_code中给出的div模块，这个模块的输入输出信号如下：

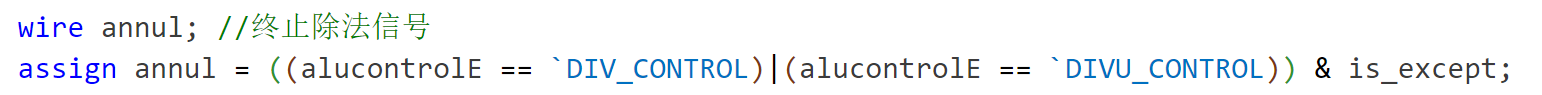


通过在ALU中对div模块传入输入信号，来对除法进行运算，ALU中相关信号的结果如下：





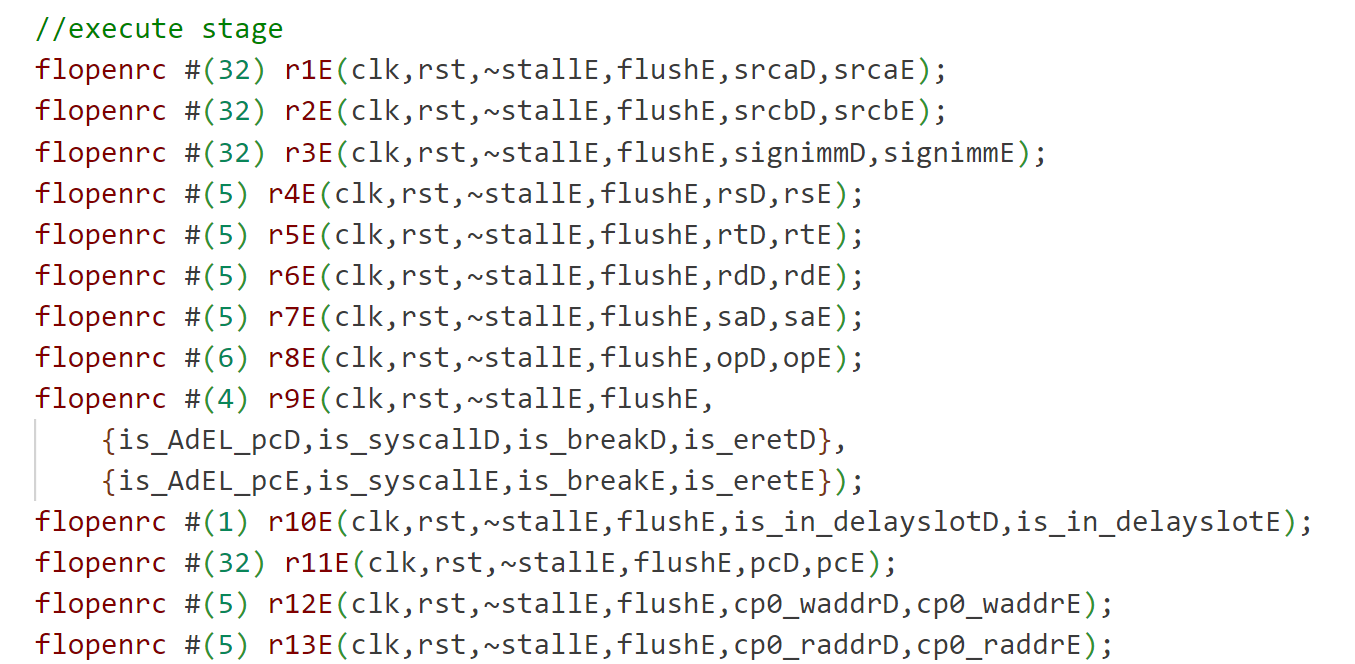
当检测到是DIV或DIVU运算时，要用a\_save和b\_save来保存操作数，这是为了避免alu中操作数的改变对除法运算的结果造成影响。



终止除法信号annul取决于is\_except的值，这个值是通过exception模块传入，是触发异常，会导致除法的刷新。

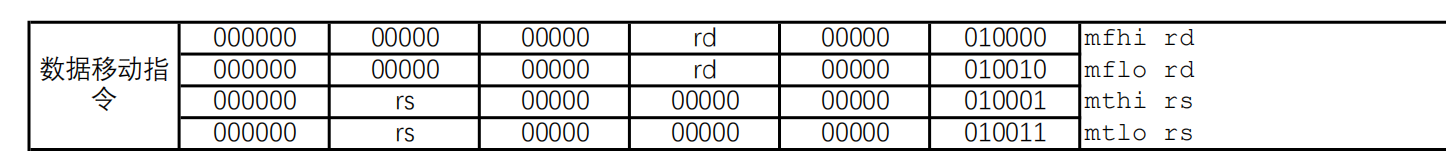
因为除法要36个时钟周期，所以要进行流水线暂停。新增stallE控制信号，控制译码-->执行阶段的流水线暂停，在除法进行时，拉高 stallE。

stallE 控制信号经由 hazard 模块生成，要接入 ID-->EX 阶段的流水线寄存器中（stallE 取反后接到使能端）。



###### 数据移动指令

4.1机器码



4.2指令功能实现过程

4.2.1指令功能

指令分别可以将hilo寄存器中的值移动到普通寄存器中，或是将普通寄存器中的值移动到hilo寄存器中。

4.2.2实现过程

为了能够让指令可以移动数据的位置，首先需要实现一个hilo寄存器，学校的实验代码中提供了hilo寄存器，稍微修改之后使用；

文本

描述已自动生成

实现了hilo寄存器之后，在maindec中添加新的控制信号hilo\_write，将其传递至执行阶段来控制hilo寄存器写入。并在aludec中添加新值控制alu。

实现MTHI、MTLO时，执行阶段写 hilo 寄存器，复用 ALU，ALU 的输出新增 64 位的 hilo\_out 信号，接到 hilo寄存器的输入端口，此时 hilo寄存器的写控制信号是拉高的。在时钟下降沿，写入 hilo寄存器。

实现MFHI、MFLO时，执行阶段读hilo寄存器，复用ALU，ALU新增64位输入值hilo\_in信号，运算后将其对应位置作为结果输出，传递到写回阶段后写入寄存器。

4.3提示

由于hilo寄存器无论是读还是写都是在执行阶段进行的，因为多数情况下四条指令都是紧跟在乘除法后执行的，所以同一阶段写入读取不存在数据冒险。

###### 访存指令

5.1指令机器码

表格

描述已自动生成

5.2指令功能实现过程

5.2.1指令功能

LW：根据地址，取一个字存入对应寄存器中；

LH：根据地址，取一个半字有符号扩展后存入对应寄存器中；

LHU：根据地址，取一个半字无符号扩展后存入对应寄存器中；

LB：根据地址，取一个字节有符号扩展后存入对应寄存器中；

LBU：根据地址，取一个字节无符号扩展后存入对应寄存器中；

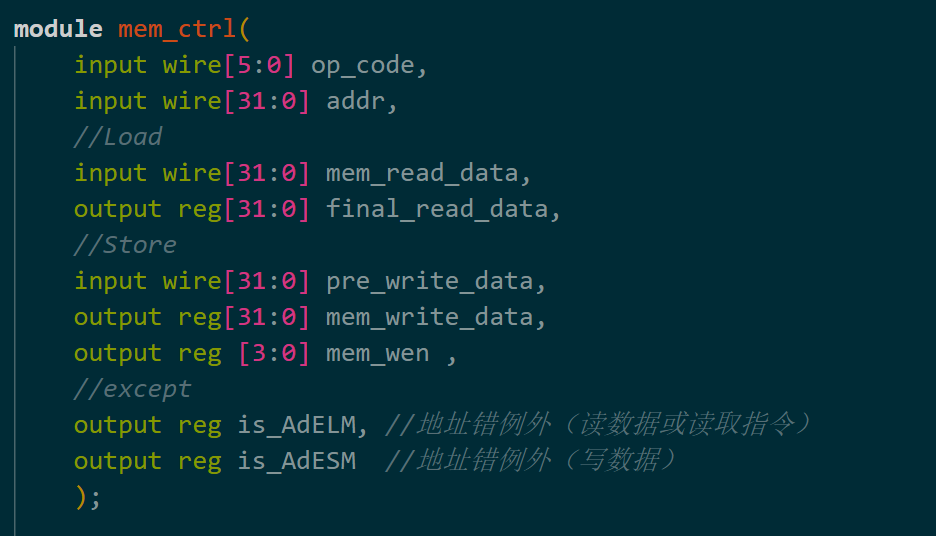
SW：根据地址，将对应寄存器中的值存入内存中；

SH：根据地址，将对应寄存器中的低十六位值存入内存中；

SB：根据地址，将对应寄存器中的低八位值存入内存中；

5.2.2实现过程

为了实现对寄存器和数据存储器中的值进行处理，添加了新的模块mem\_ctrl来根据指令op以及地址对数据进行处理；



①Load指令：

load指令可以复用实验四的lw指令的数据通路及控制信号，但需要对从数据存储器中取出的值进行进一步处理；

load类指令每次从数据存储器取出一个字取出值发送到mem\_ctrl模块中，然后再根据指令类型处理出最后要回写的值：对于LB、LBU指令，需要根据偏移量选择1个字节；LH、LHU指令需要选择低2字节或高2字节。

上面选择出来的内容可能是8位或16位，需要扩展到32位才能写回到寄存器中。LB、LH进行有符号扩展，LBU、LHU进行无符号扩展。

②Store指令：

store指令可以复用实验四的SW指令的数据通路，但需要对从寄存器中取出的值进行进一步处理；

store类指令每次从寄存器中取出值发送到mem\_ctrl模块中，然后再根据指令类型处理出最后要写入内存的值：对于SW指令，直接将所有值写入； 对于SH指令，将低2字节扩展后写入；对于SB指令，将低1字节扩展后写入。

由于store指令写入数据存储器的值的长度是不同的，所以需要新增一个4位mem\_wen信号来控制数据存储器写入位置，当信号对应位为1时才会写入对应位置，为0时会丢弃，即mem\_wen信号为0处的数据值不重要。mem\_wen的值由地址偏移决定。

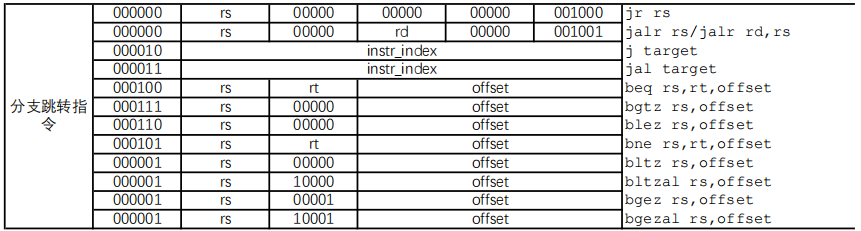
比如SH指令地址计算后低两位为00，则mem\_wen为0011，写入数据存储器时只有数据低两字节会被真正写入。

6.3提示

Load指令是在写回阶段将数据写回的，若下一条指令需要使用写回寄存器的数据，则会产生数据冒险，我们的处理方法是暂停流水线，将数据写回后再继续执行。

###### 分支跳转指令

6.1机器码



6.2指令功能实现过程

6.2.1指令功能

指令名称格式 指令功能简述

BEQ rs, rt, offset 相等转移

BNE rs, rt, offset 不等转移

BGEZ rs, offset 大于等于 0 转移

BGTZ rs, offset 大于 0 转移

BLEZ rs, offset 小于等于 0 转移

BLTZ rs, offset 小于 0 转移

BLTZAL rs, offset 小于 0 调用子程序并保存返回地址

BGEZAL rs, offset 大于等于 0 调用子程序并保存返回地址

J target 无条件直接跳转

JAL target 无条件直接跳转至子程序并保存返回地址

JR rs 无条件寄存器跳转

JALR rd, rs 无条件寄存器跳转至子程序并保存返回地址

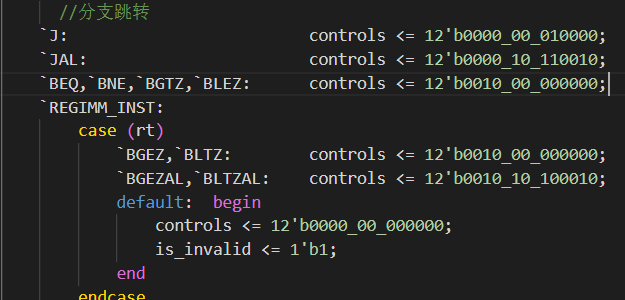
6.2.2实现过程

①BNE、BGEZ、BGTZ、BLEZ、BLTZ

这5条指令功能和BEQ类似，区别在于它们判断是否跳转的方式不一样。

数据通路方面，只需要对生成是否跳转信号的逻辑进行扩展即可，其余功能可直接复用BEQ的通路。

控制信号方面，要分别生成对应的控制信号，用于控制比较逻辑，可参照BEQ指令。

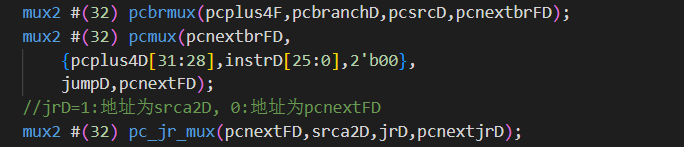


②JR

JR指令的功能与J指令完全相同，都是无条件跳转到目标地址，区别在于JR指令的跳转目标地址来自通用寄存器堆的第rs项。

数据通路只需新增寄存器堆的rdata1端口连接到生成下一个PC的多选器的输入。

要添加新的控制信号，用于控制下一个PC的选择。



③JAL

根据指令规范文档可知，JAL指令完成跳转目标地址的生成，并将PC+8写入到 31号 寄存器中。跳转这部分功能可以复用J指令的数据通路，而保存返回地址的操作在原有的基础上无法完成，需要增加新的数据通路。

写寄存器涉及到写端口的地址和写数据，所以对于写地址，需要调整通用寄存器堆写端口的地址输入waddr的生成逻辑，增加一个固定数值31作为新的输入；而对于写数据，需要得到PC+8，可以复用ALU的加法器，只不过第一个源操作数原来是仅来自寄存器堆的第一个输出端口，需要添加一个二选一部件，使其可以选择PC作为第一个源操作数，第二个需要将原有的二选一部件调整为三选一部件，8作为第三个输入。



④JALR

JALR指令的功能类似于JR和JAL的结合。对于跳转操作，可以复用JR指令对应的数据通路。对于Link操作，其返回地址的计算还是PC+8，这与JAL指令是一样的，不过计算的结果不再固定写到31号通用寄存器中，而是写入第rd项寄存器。因此JALR指令可以复用JAL指令计算PC+8并传递的数据通路，无需新增，只是写回级控制信号有所区别。

⑤BLTZAL、BGEZAL

这两条指令可以看成是BLTZ、BGEZ指令与JAL指令的结合。对于跳转操作，下一个PC的生成逻辑等同于BLTZ、BGEZ；对于Link操作，与JAL相同，将PC+8写入31号寄存器。因此，这两条指令不需要增加数据通路。

###### 特权指令

7.1机器码

表格

描述已自动生成

7.2指令功能实现过程

7.2.1指令功能

特权指令可以在异常发生时对对应的异常进行处理：

BREAK：发生断点异常，无条件地将控制权转到异常处理程序；

SYSCALL：发生系统调用异常，无条件地将控制权转到异常处理程序；

ERET：在中断、异常或错误处理完成时返回中断指令。ERET不执行下一条指令；

MTC0：将对应寄存器中的值存入cp0寄存器中；

MFC0：将cp0寄存器中的值存入对应寄存器中。

7.2.2实现过程

①BREAK：

新增标记信号is\_breakD，并将此标记信号传递到访存阶段。在译码阶段通过判断指令的高6位以及最低6位来产生此标记信号。

②SYSCALL：

新增标记信号is\_syscallD，并将此标记信号传递到访存阶段。在译码阶段通过判断指令的高6位以及最低6位来产生此标记信号。

③ERET：

新增标记信号is\_eretD，并将此标记信号传递到访存阶段。这个指令取值是唯一的，直接用对应信号来判断。

④MTC0：

新增写CP0寄存器的地址信号cp0\_waddrD，并将此地址信号传递到访存阶段。 扩展alucontrol信号的种类，用于识别并控制指令在ALU中执行何种操作。新增CP0寄存器的写控制信号cp0\_write，并将此地址信号传递到访存阶段。在译码阶段通过判断指令的高6位以及最低6位来产生这些信号。

⑤MFC0：

新增读CP0寄存器的地址信号cp0\_raddrD，并将此地址信号传递到访存阶段。 扩展alucontrol信号的种类，用于识别并控制指令在ALU中执行何种操作。在译码阶段通过判断指令的高6位以及最低6位来产生这些信号。

7.3提示

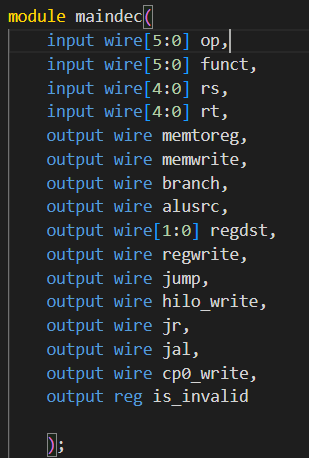
我们在执行阶段读取cp0，在访存阶段写cp0，所以可能产生数据冒险，解决方法是添加一个二选一多路选择器来进行数据前推。

##### （二）Maindec模块设计

###### 1.功能描述

该模块对指令进行译码，根据指令的 op、funct、rs 和 rt 识别所有指令，生成各个控制信号。

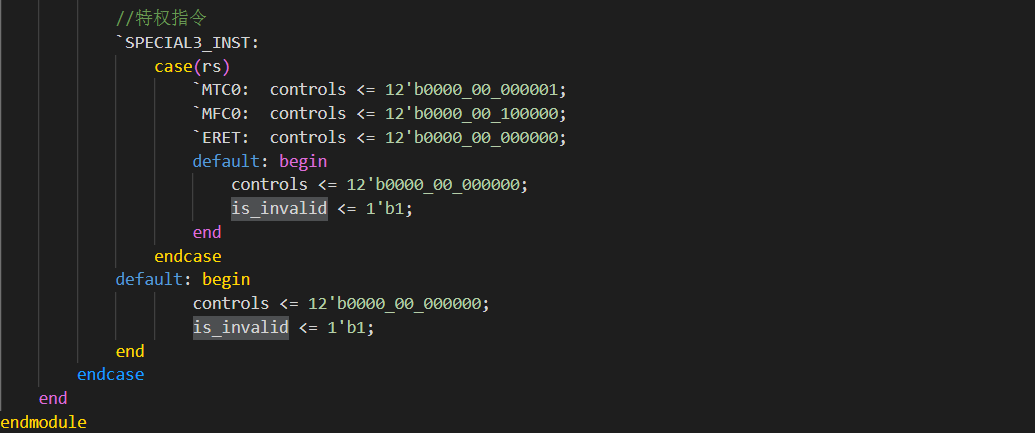
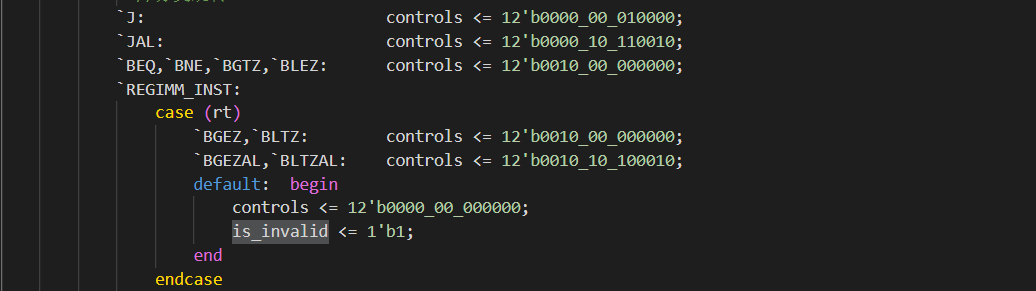
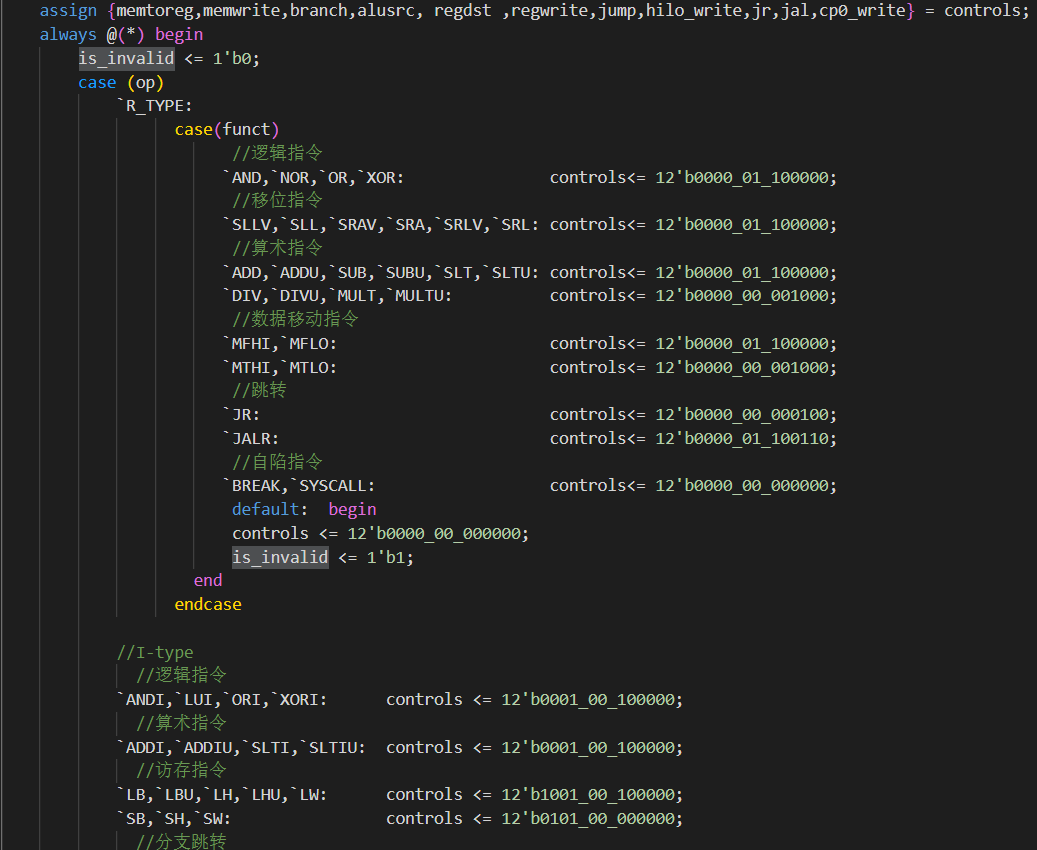
###### 接口定义



|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | 输入/输出 | 功能 |
| Op | 6bit | input | 指令的 opcode |
| funct | 6bit | input | 指令的 funct |
| rs | 5bit | input | 指令的 rs |
| rt | 5bit | input | 指令的 rt |
| memtoreg | 1bit | output | 回写的数据来自于  ALU 计算的结果还是  存储器读取的数据 |
| memwrite | 1bit | output | 是否需要写数据存储  器 |
| branch | 1bit | output | 是否为 branch 指令 |
| alusrc | 1bit | output | 送入 ALU B 端口的值是立即数的 32 位扩展还是寄存器堆读取的值 |
| regdst | 2bit | output | 写入寄存器堆的地址  是 rt 还是 rd 还是  31 |
| regwrite | 1bit | output | 是否需要写寄存器堆 |
| jump | 1bit | output | 是否为 jump 指令 |
| hilo\_write | 1bit | output | 是否需要写 HI、LO  寄存器 |
| jr | 1bit | output | 是否是 jr 指令 |
| jal | 1bit | output | 是否是 Link 类型指令 |
| cp0\_write | 1bit | output | 是否需要写 CP0 寄存器 |
| is\_invalid | 1bit | output | 是否是无效指令 |

###### 逻辑控制

根据 op 区分指令是 R-Type 还是 I-Type 还是 J-Type 还是特权指令。R-Type指令根据 funct 再作区分；I-Type 指令中的 REGIMM 指令根据 rt 再作区分；特权指令根据 rs 再作区分。然后再为不同指令生成各个控制信号。

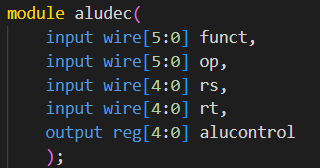


##### （三）Aludec模块设计

###### 1.功能描述

该模块对指令进行译码，根据指令的 op、funct、rs 和 rt 识别所有指令，生成 ALU 的控制信号 alucontrol。

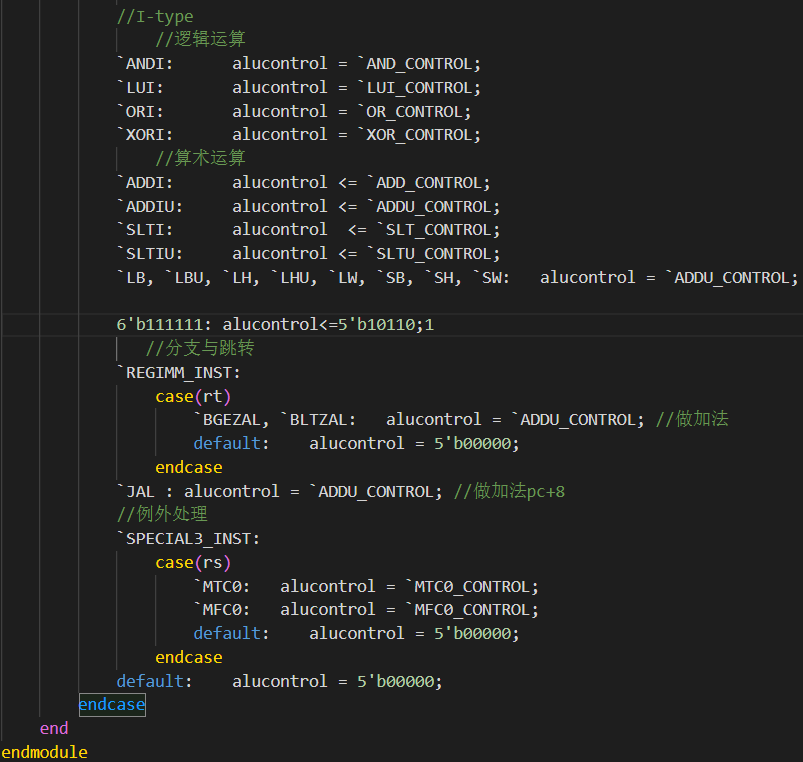
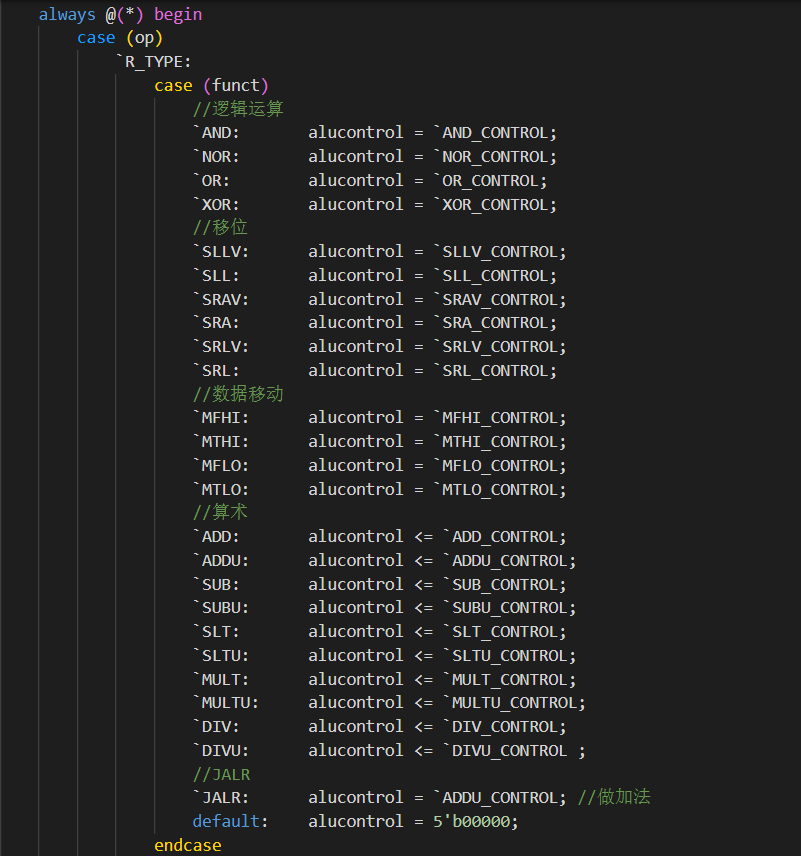
###### 接口定义



|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | 输入/输出 | 功能 |
| Op | 6bit | input | 指令的 opcode |
| funct | 6bit | input | 指令的 funct |
| rs | 5bit | input | 指令的 rs |
| rt | 5bit | input | 指令的 rt |
| alucontrol | 5bit | output | ALU 控制信号，代表  不同的运算类型 |

###### 逻辑控制

根据 op 区分指令是 R-Type 还是 I-Type 还是 J-Type 还是特权指令。R-Type 指令根据 funct 再作区分；I-Type 指令中的 REGIMM 指令根据 rt 再作区分；特权指令根据 rs 再作区分。然后再为不同指令生成 ALU 控制信号 alucontrol。



#### 三、实验过程（40%）

###### （一）设计工作日志

2023.12.26

全体组员配置实验需要的环境，了解任务与要求，观看视频。

2023.12.27——2023.12.28

回顾计算机组成原理实验四、阅读文档与资料。

2023.12.29

讨论并分配成员任务，确认总体设计通路

罗皙开始编写前8条分支跳转指令，刘倬宇开始编写算术运算指令，庞博开始编写数据转移指令，杨佳俊开始编写逻辑指令。

2023.12.30——2023.12.31

罗皙完成所有分支跳转指令，刘倬宇完成算术指令的编写，庞博完成数据转移与访存指令的编写，杨佳俊完成逻辑指令与移位指令的编写。

2024.1.1

对所有成员编写的指令进行整合，完善并统一数据通路与变量的设定，并成功调通逻辑与移位指令的仿真测试结果。

2024.1.2

对算术指令，数据移动指令，跳转指令和访存指令进行调试，成功得到正确的仿真结果。

2024.1.3

罗皙编写SRAM接口，进行功能测试，通过前面8个测试点。其余人进行调试。

2024.1.4

所有人继续调试功能测试，更改数据通路，最后成功跑通前面64个功能测试点。

2024.1.5

刘倬宇编写例外、异常处理模块，庞博编写5条特权指令，罗皙继续完善数据通路和补充数据通路图。杨佳俊辅助前面两人的编写并汇合到一起。

2024.1.6

所有人将全部的57条指令进行功能测试，进行调试与调整，最后成功通过89个测试点。

2024.1.7

进行上板测试，发现数码管显示0x44，不是89个测试点都能通过上板测试。所有人继续对通路进行调试与改进，最后上板数码管显示0x59,成功通过上板测试。

2024.1.8

所有人查看文档与视频，罗皙编写AXI接口并完成，进行功能测试，无法通过89个测试点。所有人开始进行调试与debug。

2024.1.9

罗皙查看文档与视频开始编写cache，其余人继续进行调试并最终成功通过axi接口的89个功能测试点。

2024.1.10

罗皙完成对cache的添加，刘倬宇与庞博对整个完整的处理器设计进行调试发现没有问题后，由杨佳俊开始性能测试仿真。

2024.1.11

杨佳俊通过更改频率等调试，成功通过了10个性能测试仿真。最后所有人一起进行性能的上板测试，最终等到相应的性能分。

###### （二）主要的错误记录

1、错误1

（1）错误现象

在进行inst\_ram导入时，无法正确识别指令。

（2）分析定位过程

观察pc的值看出pc没有正确自增，所以指令存储器读取错误，通过观察datapath模块中的指令选择器，发现是由于pcsrcD的值不正确导致，pcsrcD的值是由跳转指令部分决定的。

（3）错误原因

跳转指令还未编写完成，单独测试无法提供pcsrcD的值，导致无法正确读取指令。

（4）修正效果

将pcsrcD的值暂时先设置为1，之后可正确获取指令。

2、错误2

（1）错误现象

编写完分支跳转指令后出现无法仿真的错误

1. 分析定位过程

观察仿真图和对比检查代码，查看错误文件

1. 错误原因

发现在eqcmp模块中在非寄存器类型的变量（wire） y 上进行了过程赋值

1. 修正效果

将y的类型改为reg.成功仿真

3、错误3

（1）错误现象

测试算术指令的除法运算时，发现只有第一次除法运算的结果

1. 分析定位过程

观察仿真图，通过观察不同周期，寄存器对中相应寄存器的值和相应模块的输出

1. 错误原因

除法运行的周期不够，导致除法运算不能正常完成

1. 修正效果

在控制模块中对执行阶段的流水线加入stall暂停，当进行除法运算时就暂停直至完成除法。

4、错误4

（1）错误现象

进行功能测试的时候，第9个测试点错误

1. 分析定位过程

查看报错发现是pc值为0xbfc0f918的地址发生错误，查看功能测试文件中该地址执行的指令，为jalr型指令。再次更换指令测试的跳转指令的test1的coe文件，观察仿真图，发现指令执行一段时间后重复执行。

1. 错误原因

jalr指令没有正确保存pc的值，查看代码发现是aludec模块没有添加jalr指令应该执行的操作

1. 修正效果

在aludec模块中加入执行jalr指令时，进行加法的操作。成功通过改测试点

5、错误5

（1）错误现象

测试算术指令时，在观察仿真图时发现有符号减法和加法的结果有问题。

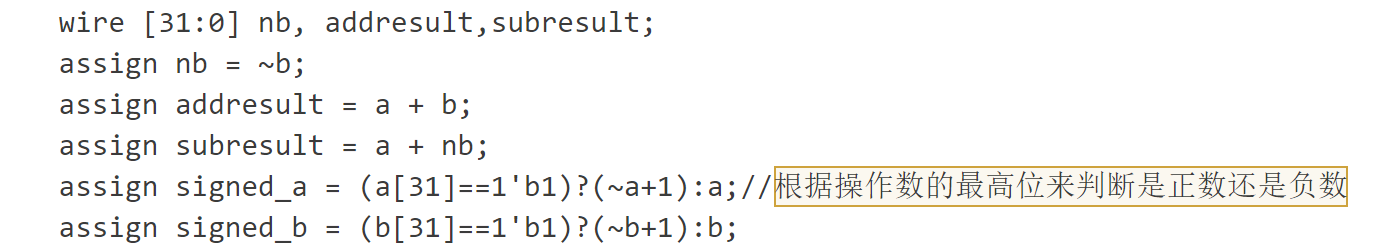
1. 分析定位过程

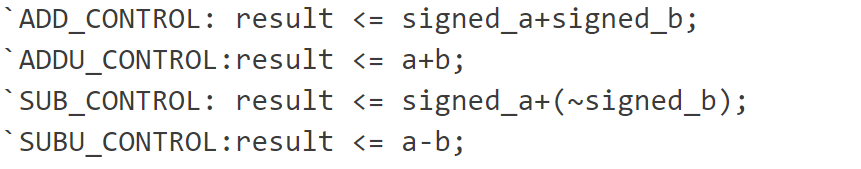
通过对其他算数指令的运算的结果仿真图的观察，发现其余的算术指令运算的结果没有问题，所以我们考虑应该是运算指令有问题。

1. 错误原因

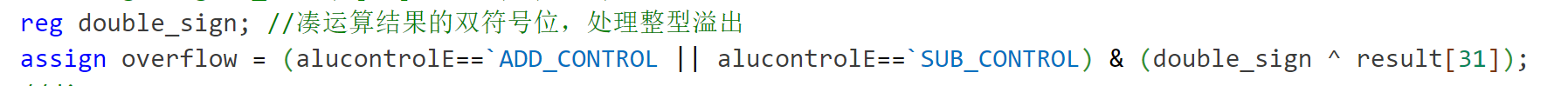
在进行有符号加减法运算时，我们通过最高位来判断操作数是正数还是负数，如果是负数的话通过取反加一来作为新的操作数。后来我们改变了思路，考虑到有符号加减法和无符号加减法的区别在于是否有溢出，我们直接对操作数进行运算，但是增加了一个double\_sign位来和运算结果的最高位一起来判断是否有溢出。

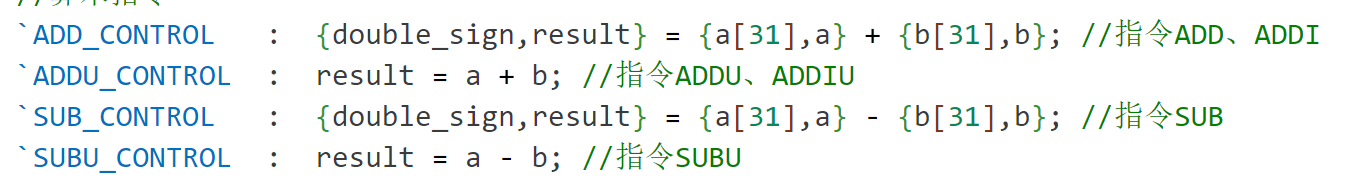
原来的代码：





修改后的代码：





（4）修正效果

在修改了代码之后，仿真图上的结果正确。

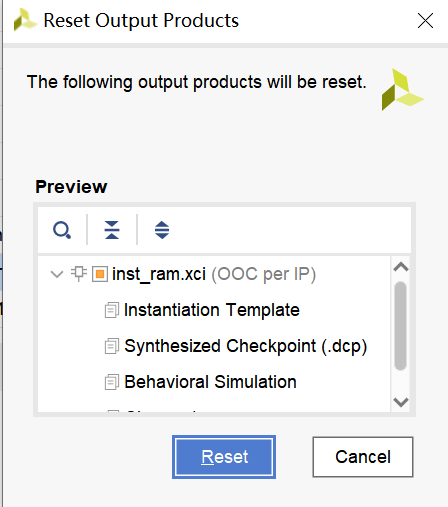
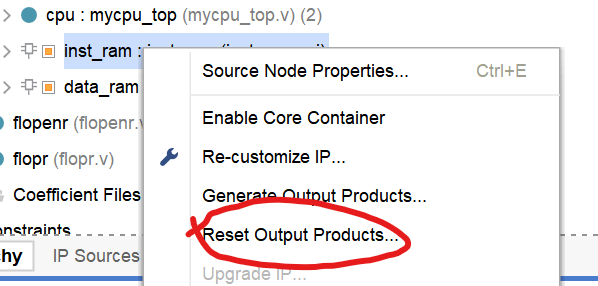
6、错误6

（1）错误现象

在测试时无法根据新的coe文件测试，提示：Generate of output products did not run again as all output products were previously generated and up-to-date.

1. 解决办法

通过在网上查阅资料，我们找到了解决办法：右键单击列表中的IP，点击Reset Output Products.在弹出的小窗中点击Reset，即可重新修改并生成IP。



7、错误7

（1）错误现象

在测试跳转指令时，发现应存入$31的PC+8的值始终为X。

（2）分析定位过程

经过仿真调试，我们加入了aluout的值以及一些信号观察，发现aluout的值其实是写入了$31的，但只是当regdst为11（即写入到$31）时的aluout始终为XXXXX，所以才会出现这样的情况，而别的时候aluout的值是正确的。由此我们考虑到可能是有关pc的某个信号出现了问题。

（3）错误原因

我们发现是pcE未进行初始化，于是添加wire [31:0] pcE，然后再进行仿真，发现$31有正确的PC+8的值了。

（4）修正效果

通过仿真，应存入$31的PC+8的值显示正确。

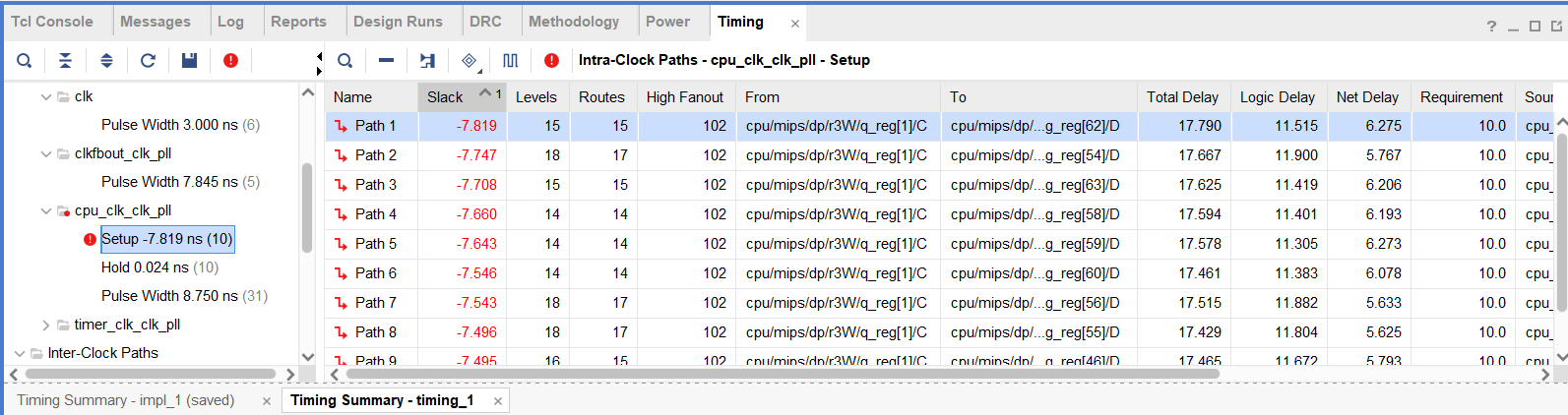
8、错误8

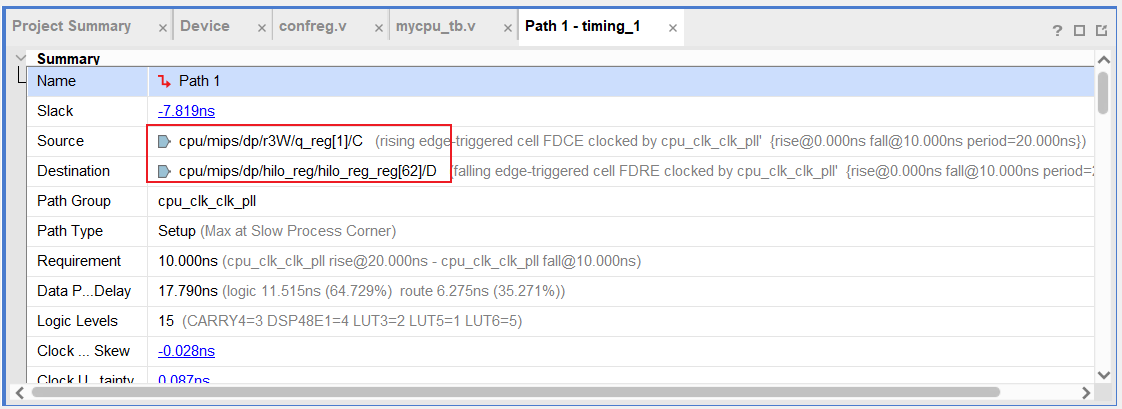
（1）错误现象

在sram上板测试时，板子上显示2b而不是59（通过89条指令）

（2）分析定位过程

通过对文档《A10\_FPGA在线调试说明\_v1.00》的阅读，我们对上板异常的原因进行了排查，在检查到时序报告时通过对intra clock path中出现报红的path进行查看，发现所有10个path的Destnation都是hilo\_reg寄存器的某位的值，于是我们猜想或许是alu运算时有关hilo\_reg的赋值的时序出现问题。





1. 错误原因

查看alu文件后发现乘除法运算时使用的是非阻塞赋值，于是我们将其改为阻塞赋值。

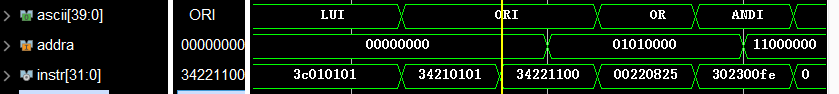
（4）修正效果

修改之后再上板测试，七段数码管成功显示为59（即89个测试全部通过）。

9.错误9

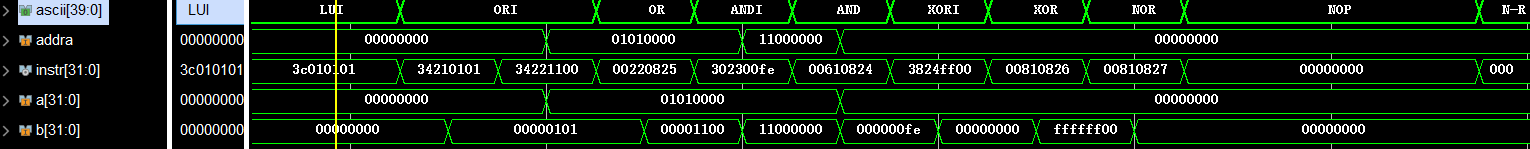
1. 错误现象

编写逻辑运算指令仿真时只有第一条LUI指令运算结果正确



1. 分析定位过程

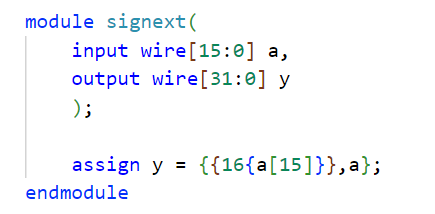
观察alu单元两个输入的仿真结果，发现只有LUI传入的输入正确，a是00000000，b是00001010，运算结果是10100000，ORI传入的输入a是正确的10100000，b却是00001100，b应该是原来的00001010才正确。



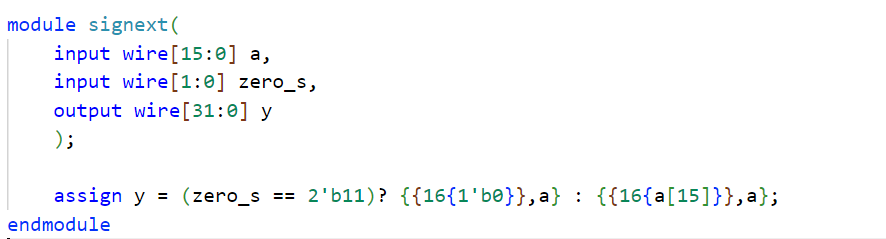
1. 错误原因

起初以为是传给alu的控制信号有问题，在miandec部分反复修改，结果还是不成功；最后感觉可能是符号扩展出了问题，此前用的是https://co.ccslab.cn/basic/extend\_52/中的方法一，改为方法二，在符号扩展部分添加2位信号进行无符号扩展。

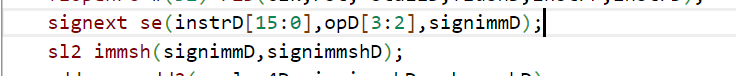
修改前的signext：



修改后的signext：

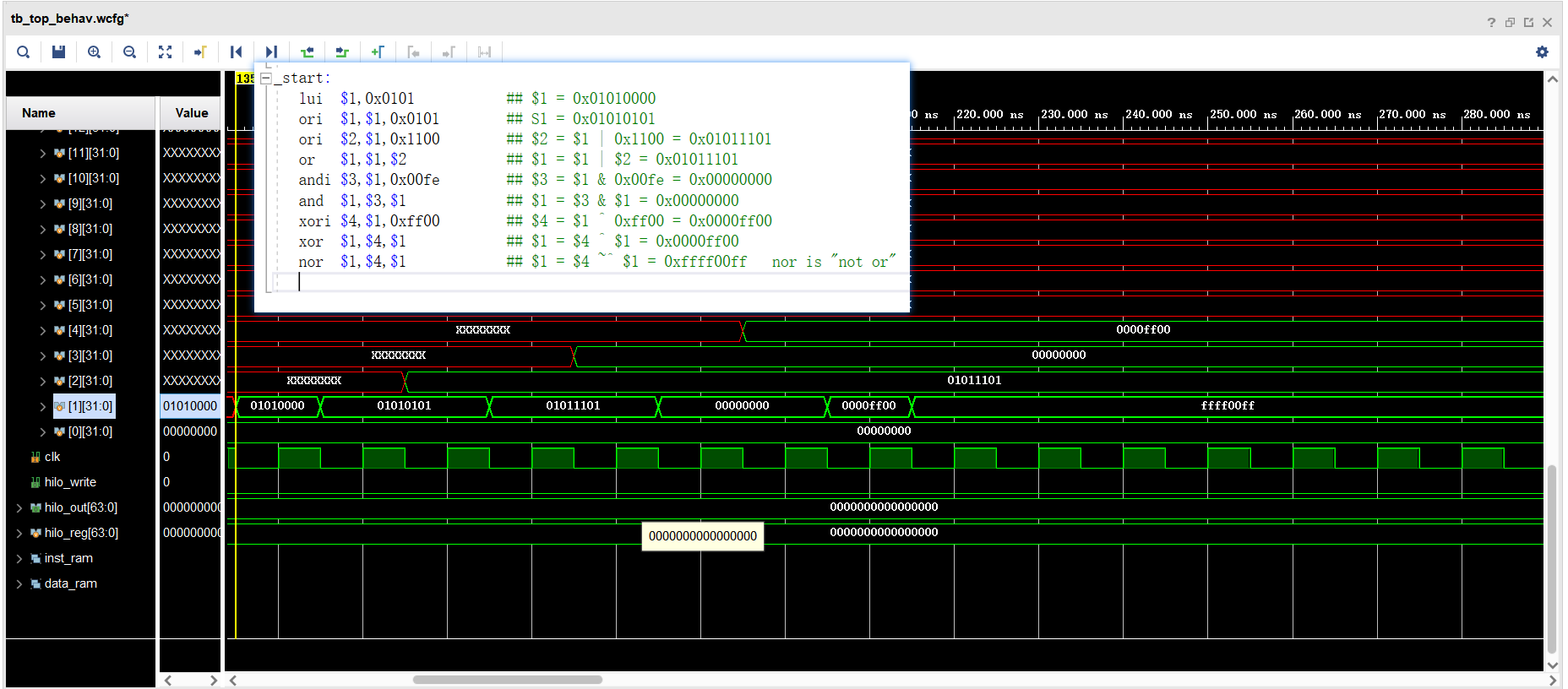


同时修改数据通路实例化signext部分，将op的第三位和第四位作为控制信号，因为立即数指令的op三四位都是1。



1. 修正结果

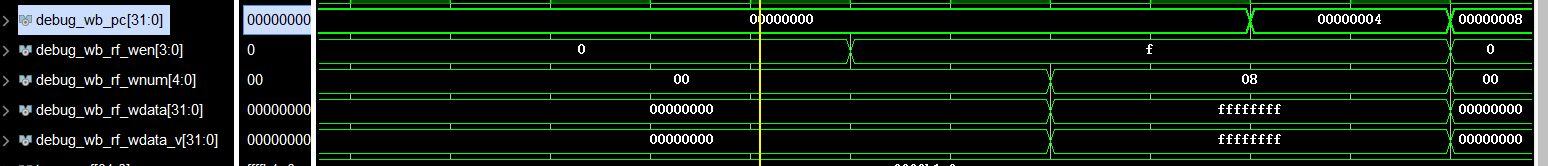
运算的正确结果都写入对应的寄存器当中。



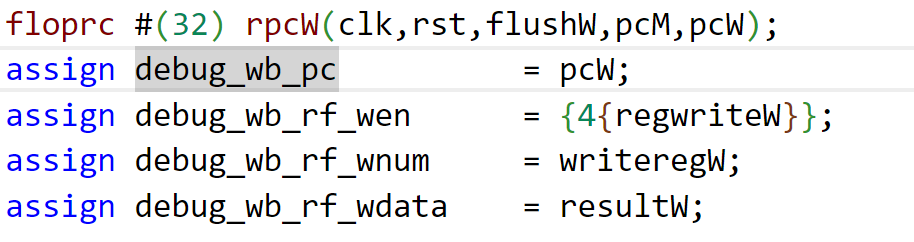
10.错误10

1. 错误现象

Debug\_wb\_pc从0开始增长



1. 分析定位过程



根据流水线传递值的过程，Debug\_wb\_pc的值受原pc的影响，从0开始应该是因为PC复位地址是0。

1. 错误原因

PC复位地址是0。

（4）修正结果

修改pc模块中pc复位值，并添加mmu地址转换模块后，Debug\_wb\_pc值从bfc00000开始增长。

#### 四、设计结果

##### （一）设计交付物说明

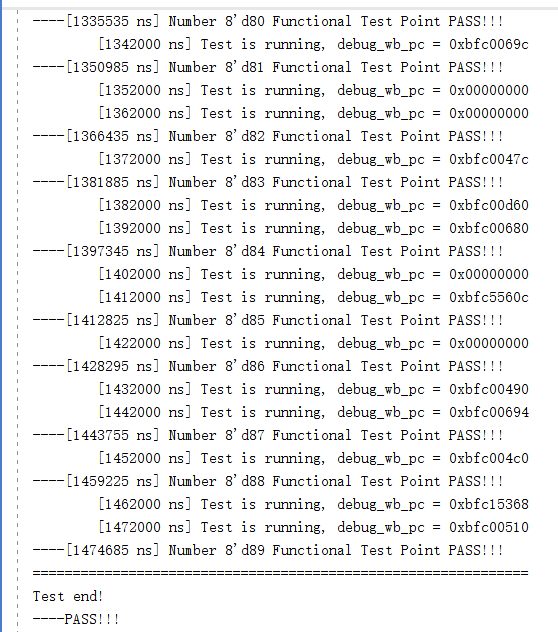
/submit/mycpu\_sram 文件夹下，是我们实现的sram接口的 MIPS 处理器。

/submit/mycpu\_axi 文件夹下，是我们实现的axi接口和添加了cache的 MIPS 处理器。

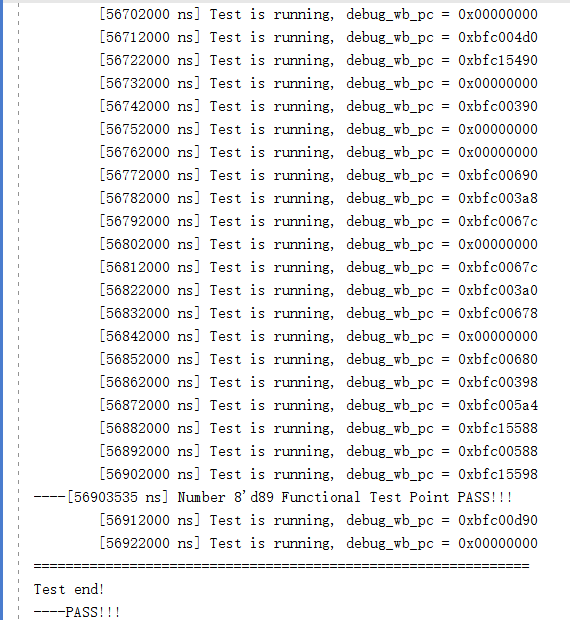
若要进行仿真，需要把整个 mycpu\_axi或mycpu\_sram 文件夹导入实验资料包相应的工程中。即可仿真、综合、上板。

##### （二）设计演示结果

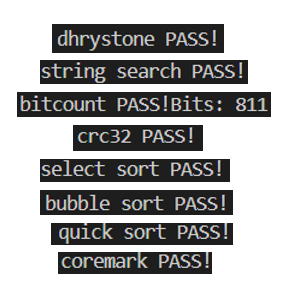
1.sram接口的功能测试



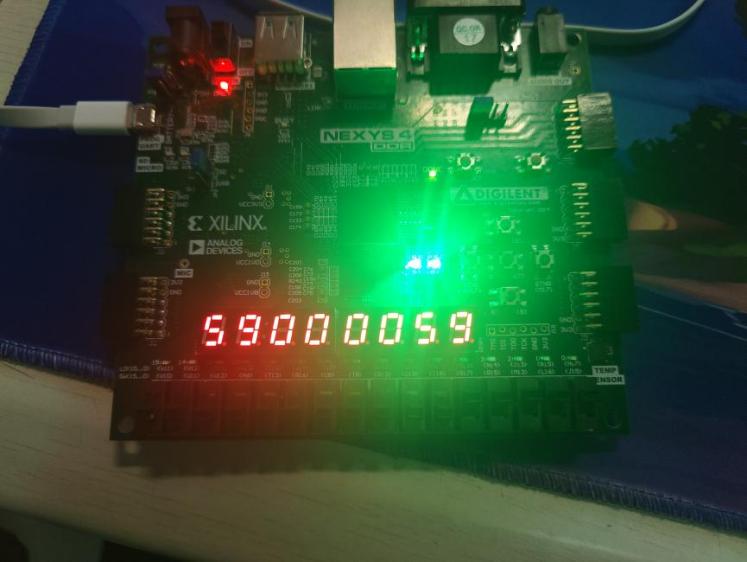
2.axi接口的功能测试



3.性能测试仿真



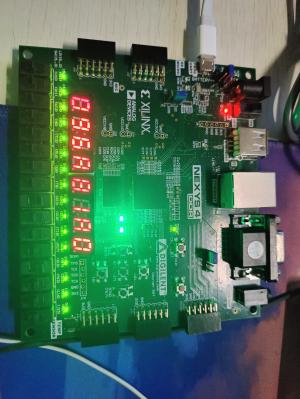
4.功能测试上板



5.性能测试上板

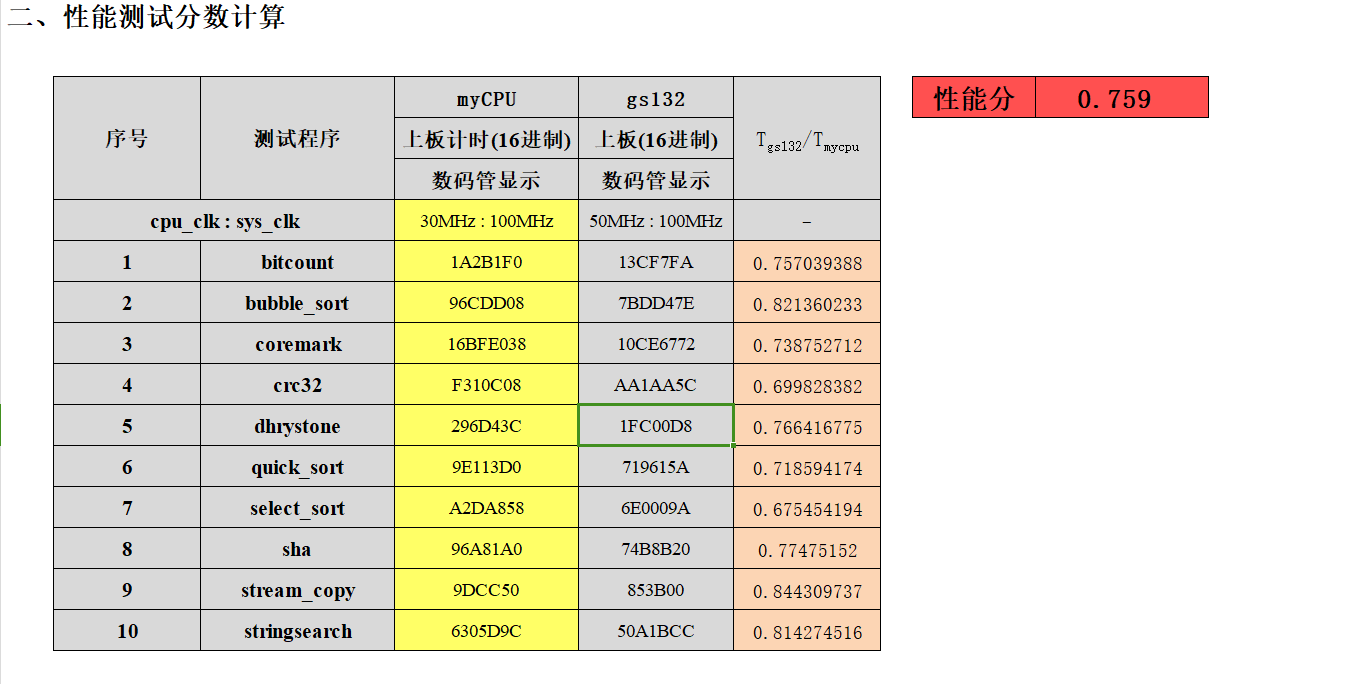








6.性能分



#### 五、参考设计说明

本次硬件综合模块引用如下：

1，计算机组成原理 lab4 标准代码：实验资料包提供。

2，defines2.vh 宏定义：实验资料包提供。

3，div.v 除法模块：实验资料包提供。

4，cp0\_reg.v：实验资料包提供。

5，基础 cache:实验资料包提供。

6，d\_sram2sraml.v、i\_sram2sraml.v 类 SRAM 转换：引用自重庆大学硬件综合设计实验文档中2020视频中。

7，转接桥 bridge\_1x2.v、bridge\_2x1.v: 引用自计算机系统结构实验二资料包。

8，mmu.v: 实验资料包提供。

9，cpu\_axi\_interface.v: 实验资料包提供。

#### 六、总结

（一）组员：罗皙

对我而言，这次硬件综合设计实验整体来说还是比较困难的，不过好在经过三周的共同努力，终于成功完成了它。通过这次项目，我们积累了硬件综合设计的经验，也拥有了一些团队协作能力和问题解决能力。过程虽然痛苦，但结果是好的。

（二）组员：刘倬宇

在实验过程中我们遇到了很多错误，通过在仿真图中对信号的加入和观察和小组成员们的互帮互助，我们逐一解决了这些问题。这次硬件综合设计课程锻炼了我们的团队协作能力和沟通能力，小组成员们都积极讨论和热心互相帮助，解决问题。感谢每个小组成员的辛勤付出！

（三）组员：杨佳俊

此次硬件综合设计难度较大，但通过与组员共同坚持，最终还算是圆满完成。非常感谢队友们的热心帮助，特别是刚开始做的时候不知道如何下手，是队友的耐心指导才让我有做下去的动力！最后，祝老师新年快乐，祝罗皙、刘倬宇和庞博新年快乐！

（四）组员：庞博

经过这次硬件综合设计，使我更加了解了一个cpu运行的过程，也让我明白了要完成一个cpu的设计有多么困难，这让我更加敬佩那些设计cpu的工作者们，希望未来我国的科研工作者能够在这方面取得更大的进步！

#### 七、参考文献

[1]《计算机组成原理实验指导书》,重庆大学计算机学院编.

[2]《A03\_“系统能力培养大赛”MIPS指令系统规范\_v1.01》.

[3]《功能测试说明》.

[4]《性能测试说明》.

[5]《自己动手写 CPU》,雷思磊著,电子工业出版社.

[6]重庆大学硬件综合设计实验文档[EB/OL].[2023-01-12].https://co.ccslab.cn/

[7]《指令及对应机器码\_2018》