中国科学院大学计算机组成原理实验课

实 验 报 告

学号： 2020K8009926006 姓名： 游昆霖 专业： 计算机科学与技术

实验序号： 1 实验名称： 基本功能部件设计

注1：撰写此Word格式实验报告后以PDF格式保存在~/COD-Lab/reports目录下。文件命名规则：prjN.pdf，其中“prj”和后缀名“pdf”为小写，“N”为1至4的阿拉伯数字。例如：prj1.pdf。PDF文件大小应控制在5MB以内。此外，实验项目5包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：prj5-projectname.pdf，其中“-”为英文标点符号的短横线。文件命名举例：prj5-dma.pdf。具体要求详见实验项目5讲义。

注2：使用git add及git commit命令将实验报告PDF文件添加到本地仓库master分支，并通过git push推送到GitLab远程仓库master分支（具体命令详见实验报告）。

注3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

1. 逻辑电路结构与仿真波形的截图及说明（比如关键RTL代码段{包含注释}及其对应的逻辑电路结构图、相应信号的仿真波形和信号变化的说明等）
2. reg\_file.v 代码说明
3. 宏定义及变量定义

|  |
| --- |
| `define TOP 32  **reg** **[**`DATA\_WIDTH **-**1 **:** 0**]** RF **[**`TOP**-**1 **:** 0**];** |

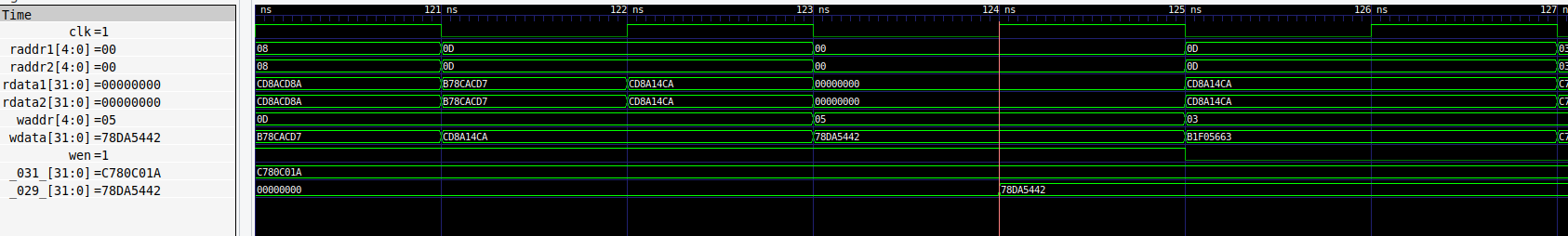
宏定义TOP表示寄存器个数，并设置32个32位寄存器变量。

(2)同步写数据

|  |
| --- |
| **always** **@** **(posedge** clk**)**  **begin**  **if(**wen **&** **(|**waddr**))** **begin** // |waddr is equal to waddr != 5'b0  RF**[**waddr**]** **<=** wdata**;**  **end**  **end** |

通过时序逻辑和非阻塞赋值实现同步写数据。同时写数据条件为写使能信号wen置高电平，且写地址waddr非零，注意此处可用数据按位或简便表示。

相应波形：



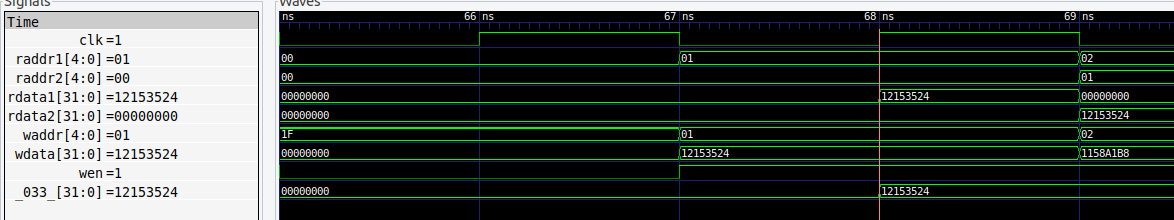
waddr=05时写5号寄存器，对应reference中\_029\_端口，123ns处waddr和wdata改变，但在124ns时钟上升沿处\_029\_所存的值才完成更新，说明了“同步写”，waddr=03时对应3号寄存器(reference中\_031\_端口)，由于126ns时wen为0，不满足写条件，故其值未更新。

(3)异步读数据

|  |
| --- |
| **assign** rdata1 **=** **{**`DATA\_WIDTH **{|**raddr1**}}** **&** RF **[**raddr1**];**  **assign** rdata2 **=** **{**`DATA\_WIDTH **{|**raddr2**}}** **&** RF **[**raddr2**];** |

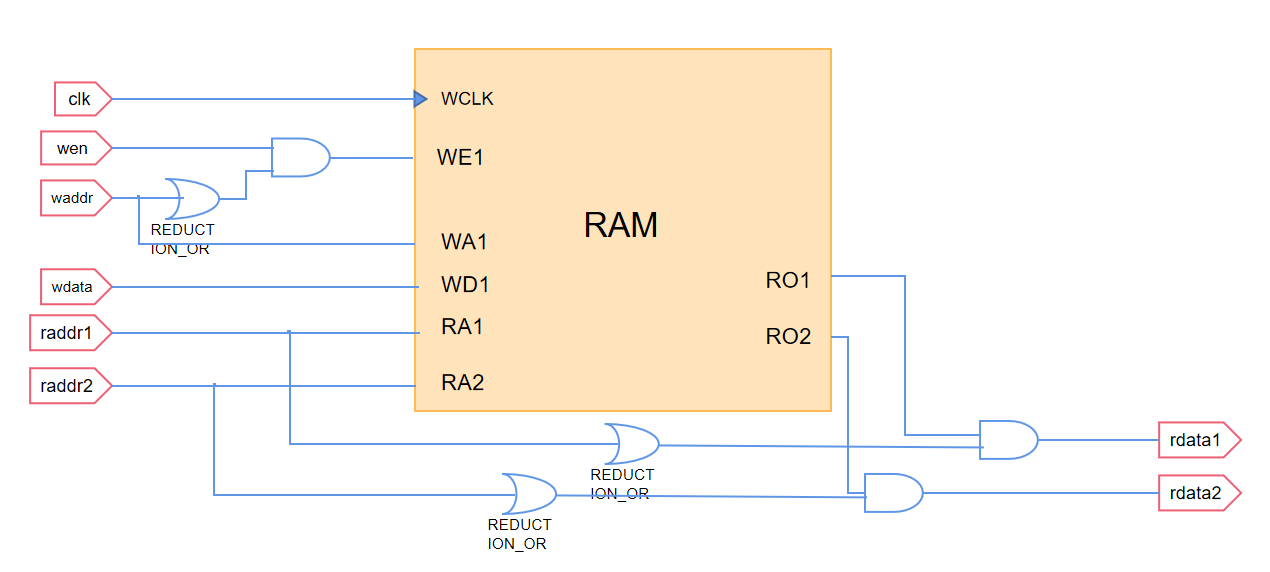
通过组合逻辑保证异步读数据。同时为保证读0号寄存器结果恒为0，通过|raddrN判断地址是否为0，并将其位拓展到32位，与从对应寄存器读取数据进行按位与，确保结果正确性。

相应波形：



68ns处，raddr2 = 00，对应0号寄存器，读出数据为全0；raddr1=01，对应1号寄存器，读出数据为12153524。Reference中的\_033\_端口对应1号寄存器，在67ns处waddr和wdata改变后，在68ns时钟上升沿才完成了数据更新，说明了“同步写”，数据更新后，rdata1立即读出数据，说明了“异步读”。

逻辑电路结构图：



1. alu.v代码说明 (只展示关键代码及对应逻辑电路结构)
2. 宏定义及变量定义

|  |
| --- |
| //ALUop operators  `define AND 3'b000  `define OR 3'b001  `define ADD 3'b010  `define SUB 3'b110  `define SLT 3'b111  ...  //wire virables for direct results  **wire** **[**`DATA\_WIDTH **-**1 **:**0**]** R\_AND**;**  **wire** **[**`DATA\_WIDTH **-**1 **:**0**]** R\_OR**;**  **wire** **[**`DATA\_WIDTH **-**1 **:**0**]** R\_ADD**;**  **wire** **[**`DATA\_WIDTH **-**1 **:**0**]** R\_SUB**;**  **wire** **[**`DATA\_WIDTH **-**1 **:**0**]** R\_SLT**;**  //wire virable for carryout  **wire** flag**;**  **wire** Inverse**;**  //wire virable for overflow  **wire** Compare**;**  //add 1 more bit for completement form  **wire** **[**`DATA\_WIDTH **:**0**]** A\_tmp**;**  **wire** **[**`DATA\_WIDTH **:**0**]** B\_tmp**;** |

宏定义五种操作对应的名称及操作码。变量定义通过不同逻辑得到的五种结果。（注意，只在对应操作下保证所得结果正确。）变量定义求carryout过程的中间变量flag和Inverse，求Overflow过程中间变量Compare及操作数A、B对应的转化数。

1. A、B转化数说明

|  |
| --- |
| //completement of A / B :  **assign** Inverse **=** ALUop**[**2**]** **;** //sub or slt  **assign** A\_tmp **=** **{**1'b0**,**A**};**  **assign** B\_tmp **=** **{**1'b0**,** Inverse **?** **~**B **:** B **};**  //note that the inverse result exclude the highest bit |

当加法操作时A、B转化数均在最高位位拓展一位0以得到进位信号；减法或比较操作时B转化数除最高位外取反并加1，（将加1的操作转移到转化数加法器，详见部分二问题4）得与B相加为0X1\_0000\_0000的数，（注意其与补码有一定区别，详见部分二问题2），下述求借位信号逻辑。

相应逻辑电路结构图见五种操作赋值说明（加减法操作）部分。

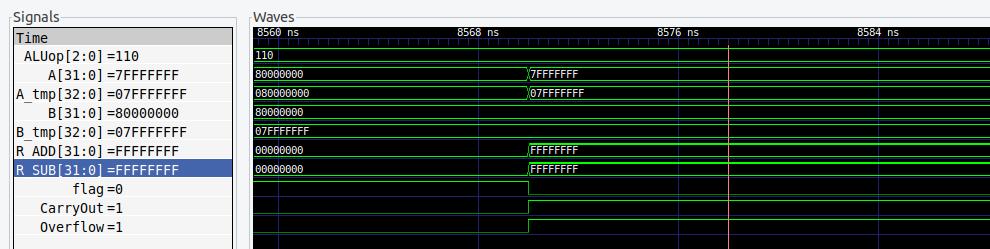
1. CarryOut、Overflow信号说明

|  |
| --- |
| **assign** **{**flag **,** R\_ADD**}** **=** A\_tmp **+** B\_tmp **+** Inverse **;**  ...  //CarryOut -> unsigned num  //ADD: carryout is the same as actual  //SUB: when A=B the result is 0x1 0000 0000  //when A < B, Completement of B < completement of A, so the highest bit of A-B is 0, however borrow actually happen, vice versa;  //so carryout = ~flag  //Overflow -> signed num  //overflow cases:  // If A > 0 and B > 0 but A + B < 0  // If A < 0 and B < 0 but A + B >= 0  // If A > 0 and B < 0 but A - B < 0  // If A < 0 and B > 0 but A - B >= 0  //Overflow only consider ADD or SUB, but we want to use result for slt, so set compare  **assign** CarryOut **=** **(**ALUop **==** `ADD**)?** flag **:** **~**flag**;**  **assign** Compare **=** **(~**A**[**`DATA\_WIDTH**-**1**]** **&** **~**B**[**`DATA\_WIDTH**-**1**]** **&** R\_ADD**[**`DATA\_WIDTH**-**1**]** **&** ALUop**==** `ADD**)|**  **(**A**[**`DATA\_WIDTH**-**1**]** **&** B**[**`DATA\_WIDTH**-**1**]** **&** **~**R\_ADD**[**`DATA\_WIDTH**-**1**]** **&** ALUop**==** `ADD**)|**  **(~**A**[**`DATA\_WIDTH**-**1**]** **&** B**[**`DATA\_WIDTH**-**1**]** **&** R\_SUB**[**`DATA\_WIDTH**-**1**]** **&** **(**ALUop**==**`SLT**|**ALUop**==** `SUB**))|**  **(**A**[**`DATA\_WIDTH**-**1**]** **&** **~**B**[**`DATA\_WIDTH**-**1**]** **&** **~**R\_SUB**[**`DATA\_WIDTH**-**1**]** **&** **(**ALUop**==**`SLT**|**ALUop**==** `SUB**));**  **assign** Overflow **=** **(**ALUop **==** `ADD **|** ALUop**==**`SUB **)** **&** Compare**;** |

flag记录A、B转化数相加后的最高位。由于CarryOut针对无符号数，当操作为加法时，flag自然表示进位；当操作为减法或比较时，简记此时B\_tmp为△B，详见(2)。由于A+△A=0X1\_0000\_0000，则当A≥B时，有△A≤△B，即A+△B≥0X1\_0000\_0000，最高位为1，然而从真值看实质未发生借位，当A<B时情况类似，因此操作为减法或比较时进位信号为flag的反。

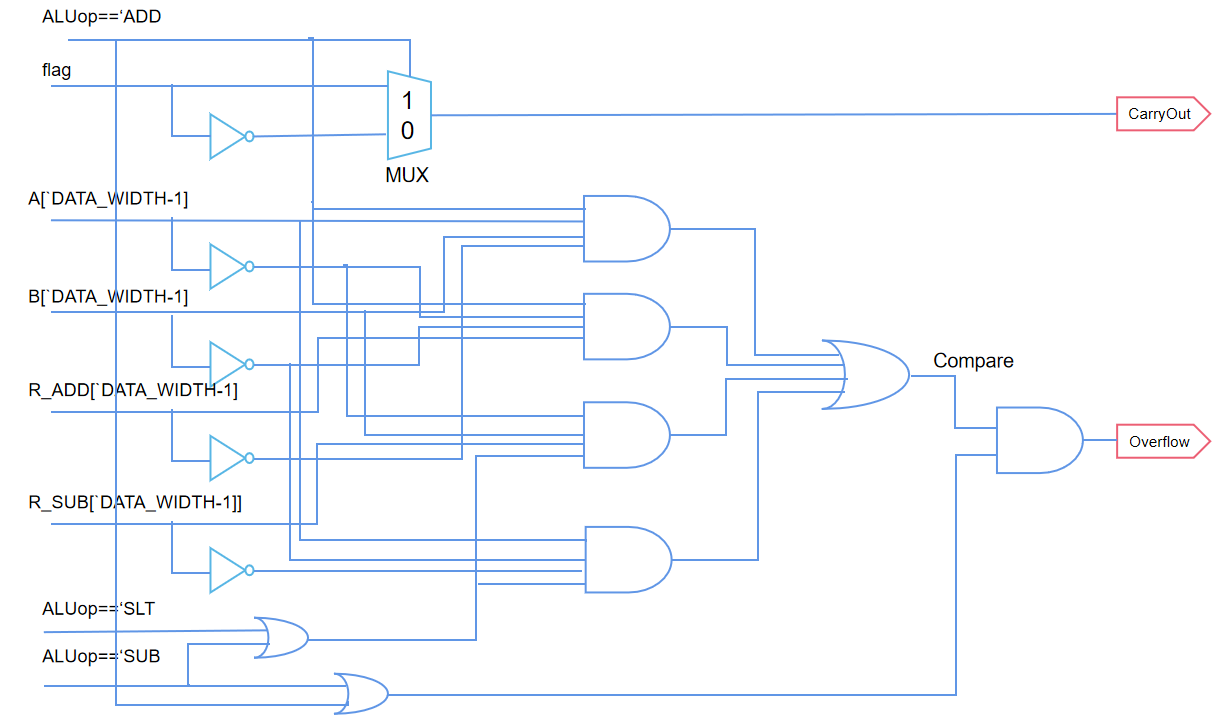
Overflow针对有符号数的加减法操作。由于该结果在比较操作时也需要使用，且判定逻辑与减法类似，因此设置变量Compare表示两数加减不符合预期的情形，四种情形如注释所示，通过与或操作获得Overflow的值。

相应波形：



上图红线所示周期，A为0X7FFFFFFF，B为0X80000000，操作为SUB。由于CarryOut针对无符号数，Overflow针对有符号数。当A、B均为无符号数时，B取反并拓展1位零与A相加的拓展位flag为0，CarryOut为1，产生借位。当A、B均为有符号数时，相减的真值应当为正，但是R\_SUB表示为负，因此产生Overflow信号。

逻辑电路结构图：



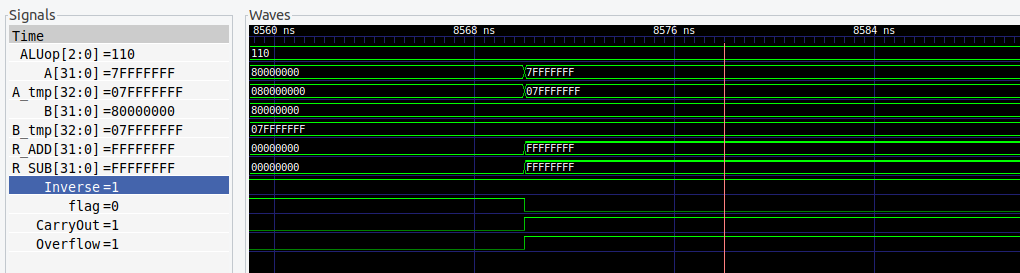
1. 五种操作对应赋值说明

|  |
| --- |
| //result of OP AND / OR  **assign** R\_AND **=** A **&** B**;**  **assign** R\_OR **=** A **|** B**;**  ...  //result of OP ADD / SUB  **assign** **{**flag **,** R\_ADD**}** **=** A\_tmp **+** B\_tmp **+** Inverse **;**  **assign** R\_SUB **=** R\_ADD**;**  ...  //result of OP SLT  //Sigend num: compare means the result is the inverse of expected, so we use xor to get result  **assign** R\_SLT**[**0**]** **=** Compare **^** R\_SUB**[**`DATA\_WIDTH**-**1**];**  **assign** R\_SLT**[**`DATA\_WIDTH**-**1 **:**1**]** **=** 31'b0**;** |

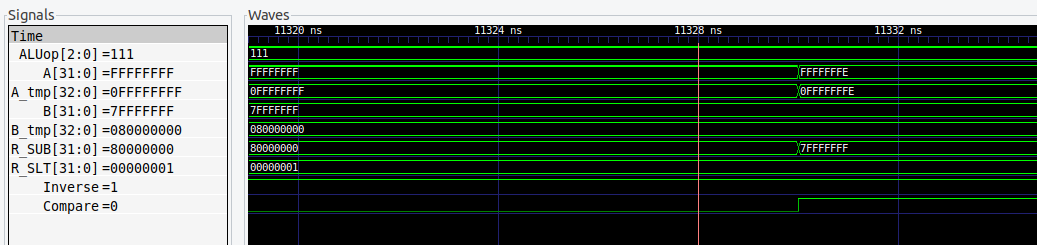
对于AND/OR/ADD/SUB操作，结果只需考虑32位的直接操作结果，特别的，不需考虑有符号数加减法的溢出情况。由于减法时会先将B取补得到B的转化数，同样使用加法器运算，因此转化数运算逻辑与ADD相同，且直接赋值可以减少一个加法器的使用。

对于SLT操作，只需利用减法逻辑判断即可，A<B也即相减结果真值为负。Compare信号为高表示减法结果符号与真值不同。因此当Compare为0、R\_SUB最高位为1 或 Compare为1，R\_SUB最高位为0时真值为负，SLT最低位置1。从结果上考虑只需将SLT最低位赋值为Compare和R\_SUB最高位的异或值，其余位取零即可。

相应波形（部分）：

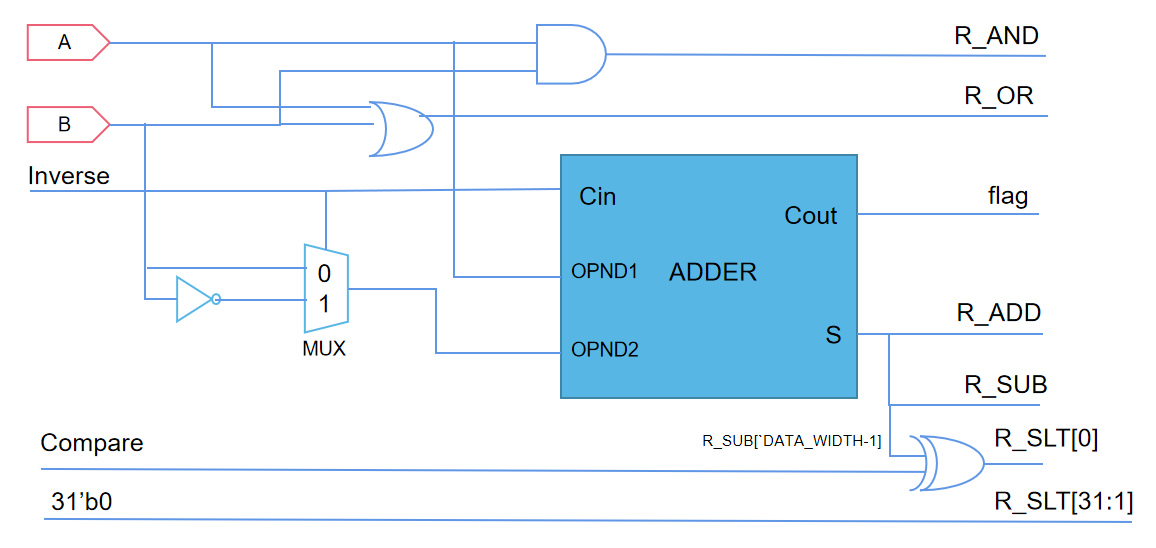


红线所示周期处操作为SUB，R\_SUB为A、B转化数相加结果（其中+1独立为+Inverse）的最后32位，与预期结果一致。



图中所示操作为SLT，针对有符号数比较。11328ns处A<B，相减结果也为负，与预期相符，Compare为0；11332ns处A<B，但相减结果为正，与预期不符，Compare为1，上述两个周期对应R\_SLT最低位均为1。

逻辑电路结构图：

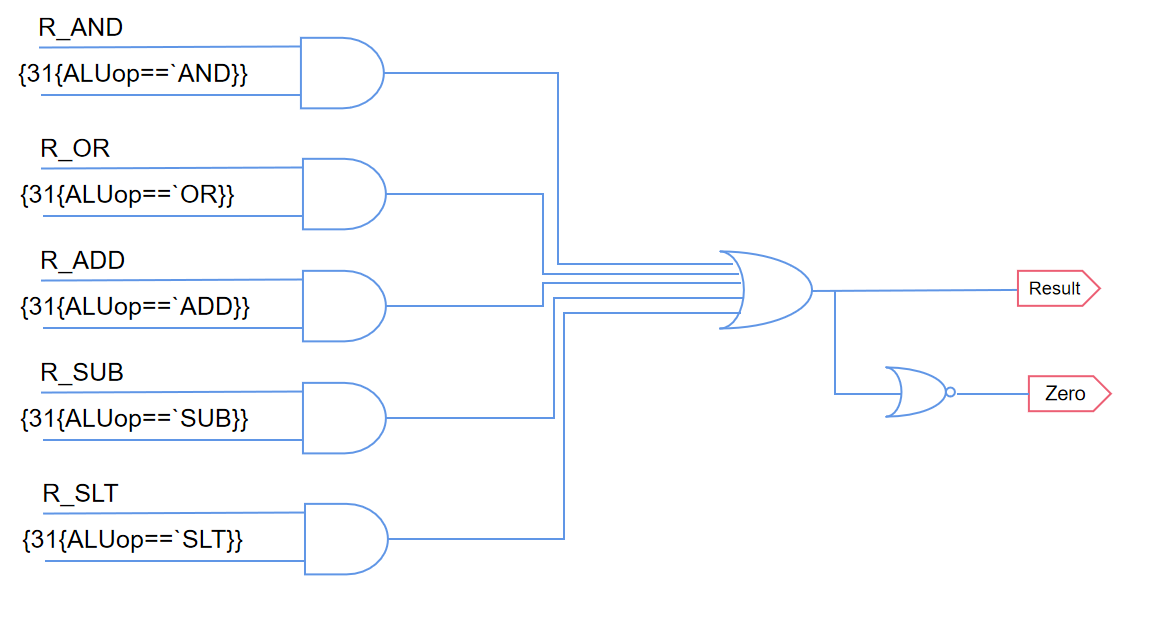


1. 生成最终输出结果及Zero信号

|  |
| --- |
| //final result and zero check  **assign** Result **=** **({**32**{**ALUop **==** `AND**}}** **&** R\_AND **)** **|**  **({**32**{**ALUop **==** `OR **}}** **&** R\_OR **)** **|**  **({**32**{**ALUop **==** `ADD**}}** **&** R\_ADD **)** **|**  **({**32**{**ALUop **==** `SUB**}}** **&** R\_SUB **)** **|**  **({**32**{**ALUop **==** `SLT**}}** **&** R\_SLT **);**  **assign** Zero **=** ~|Result**;** |

通过操作数选择不同操作逻辑所得结果，并通过Result与0比较得到Zero信号。

逻辑电路结构图：



1. 实验过程中遇到的问题、对问题的思考过程及解决方法（比如RTL代码中出现的逻辑bug，逻辑仿真和FPGA调试过程中的难点等）

问题1：ALU中位拼接位宽错误

查看云平台语法检查log，发现在进行位拼接，未指定1位0的位数，vivado默认未指定整数位宽为32，经检查后指定位宽，解决错误。

问题2：ALU中位拼接求值错误

为进行减法及求Overflow等值，设置B\_tmp，减法或比较运算时表示与B相加为0X1\_0000\_0000的值（以下简记为△B）。但在检查波形时发现，{1’b0, ALUop[2] ? ~B : B} + ALUop[2] 与 {1’b0, ALUop[2]? ~B+1 :B} 二者在对0求△B时，前者符合预期值0X1\_0000\_0000，而后者得到0X0\_0000\_0000，不符合预期。这是因为当+1发生在位拼接过程中时，对32’0进行取反加1时高位进位会因位数限制被舍去，因此应当采用前一种写法。

问题3：运算优先级及括号嵌套问题

编写代码过程中同级运算容易造成理解歧义，例如&和|运算看似同级，但是&优先执行，因此应当增加括号规范逻辑顺序。同时由于多层括号嵌套容易降低代码可读性，应当通过适当缩进，使同级逻辑缩进相同，从而凸显逻辑层次，并便利错误查找。

问题4：电路资源消耗的精简

在考虑对B求补的转化数加法中如果将+1置于转化数生成过程，将产生额外的一个加法器，消耗资源。在经过实验课助教老师的讲解后，将其作为A、B转化数的加法器的Cin，节约了一个加法器，减少电路资源消耗。

1. 对讲义中思考题（如有）的理解和回答

本次实验无思考题。

1. 在课后，你花费了大约\_\_\_\_\_5\_\_\_\_\_\_小时完成此次实验。
2. 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

实验难度适中，在与同学交流实验项目心得时发现部分同学不理解ALU部分输出信号的意义，建议在实验讲解中适当结合后续指令应用ALU输出信号进行判断的例子，（例如分支指令中对于SLT结果，Zero、Overflow等信号的应用），有助于同学们更好理解ALU的作用。