华东师范大学软件工程学院实践报告

课程名称：计算机组成与实践	**年级：2023 级**	**上机实践成绩：**
指导教师：谷守珍	**姓名：**张建夫	**上机实践日期：**5/7~5/21
实践编号： 实验2	**学号：**10235101477	**上机实践时间：**4 学时

一、 实验名称

32位ALU设计实验

二、 实验目的

掌握定点数加减法溢出检测方法

理解算术逻辑运算单元ALU的基本构成

- 熟悉logisim中各种运算组间
  - 逻辑运算部件
  - 算术运算部件
- 熟悉多路选择器的使用
- 设计32位简单ALU

三、 实验内容

设计32位简单ALU

- 利用已完成的32位加减法器、其他运算组件构造
- 禁止使用logisim中内置的加法器、减法器

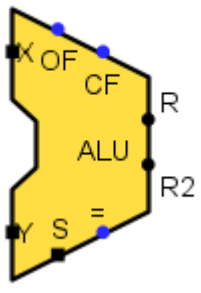
四、 实验原理

ALU原理：

本次实验的ALU主要支持以下操作：

ALU OP	十进制	运算功能
0000	0	R = X << Y 逻辑左移 （Y取低五位） R2=0
0001	1	R = X >> Y 算术右移 （Y取低五位） R2=0
0010	2	R = X >> Y 逻辑右移 （Y取低五位） R2=0
0011	3	R = (X * Y) [31:0] R2= (X * Y) [63:32]无符号乘法
0100	4	R = X/Y R2= X%Y 无符号除法
0101	5	R = X + Y （set OF/UOF）
0110	6	R = X - Y （set OF/UOF）
0111	7	R = X & Y 按位与
1000	8	R = X | Y 按位或
1001	9	R = X⊕Y 按位异或
1010	10	R = ~(X |Y) 按位或非
1011	11	R = (X < Y) ? 1 : 0 有符号比较
1100	12	R = (X < Y) ? 1 : 0 无符号比较

ALU的接口和示意图如下：



引脚	输入/输出	位宽	功能描述
X	输入	32	操作数X
Y	输入	32	操作数Y
S	输入	4	运算操作码 ALU_OP
R	输出	32	ALU运算结果
R2	输出	32	ALU运算结果第二部分，用于乘法运算高位或除法运算的余数位，其他运算时为0
OF	输出	1	有符号加减法运算溢出标记，其他运算为0
UOF	输出	1	无符号加减法运算溢出标记，其他运算为0
Equal	输出	1	Equal=(x==y)?1:0,对所有运算均有效

实验的ALU相比于课上讲的ALU多了一些接口，课上的溢出只关心有符号加法的溢出，此

处还需考虑减法和无符号的溢出，除此之外，本实验的ALU可以提供64位乘法结果。

五、 实验过程

我是按照ppt上提供的运算符顺序完成的，于是便将整个实验分为3个部分，第一部分（0~4）讲述前五个运算符的实现，第二部分讲述加法和减法（5~6）的实现，第三部分讲述剩余运算符（7~12）和各个输出接口的实现

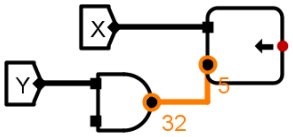
（1） 第一部分（移位操作与乘除法实现）

移位操作可以使用logism自带的封装电路实现，第一个移位操作是逻辑/算数左移，经过阅读需求发现：

R = X << Y 逻辑左移 （Y取低五位）

Y需要取低5位，防止X左移溢出（下面的移位操作同理）

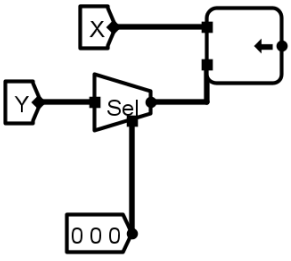
由于之前从未有将某个数取特定位的经历，我就在logism自带的封装电路电路里找类似的部件，我首先试的是与门，想着这应该能自动截断：



显然我错了，logism下方报了如下错误：

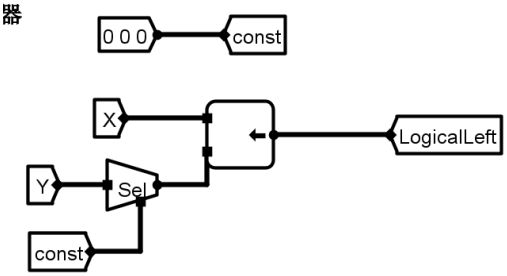
Incompatible widths

后面在Plexers里面找到一个Bit Selector，看起来是取特定位的元件，便拿来使用，但是由于不懂该元件的接口如何使用，便上网查了相关资料（后面想起来这部分的时候感觉自己蠢到家了，知道从网上找答案，却没有直接搜索如何在logism中将一个数的特定位取出来，我也是后面才发现有Splitter这个东西），最终成功实现逻辑左移：



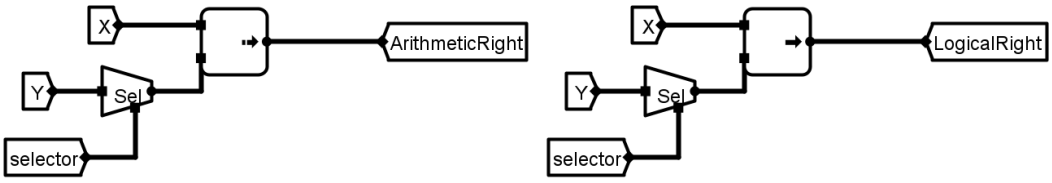
到这里，我稍做了停顿，我原本的想法是直接将每个运算结果接到result的选择器上，但是一个移位操作的电路就这么大了，到时候接result的时候电路肯定相当复杂，于是便使用了上节课学到的隧道，这样每个模块只执行一个功能，电路也更加清晰，修正版如下：

器



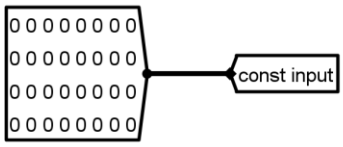
此处将选择器的选择信号用const表示，选择的是第0组（按每5位进行划分，32/5 = 7，选择信号要3位，低5位就是第0组），这样其他的移位模块就都可以使用这个const了

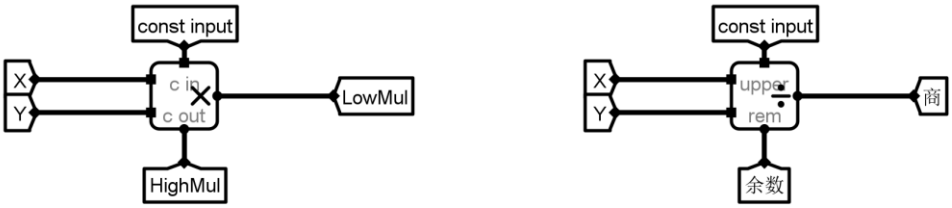
最终对其他两个移位模块如法炮制：



（此处的const改为selector了，和下面的乘除法区分）

对于乘法和除法，原本我以为要自己先实现乘法器和除法器（ppt上讲了实现方式），但发现有点困难后询问老师，老师说可以使用自带的封装电路实现，便按照先前实现移位器的经验同样实现了乘法器和加法器：

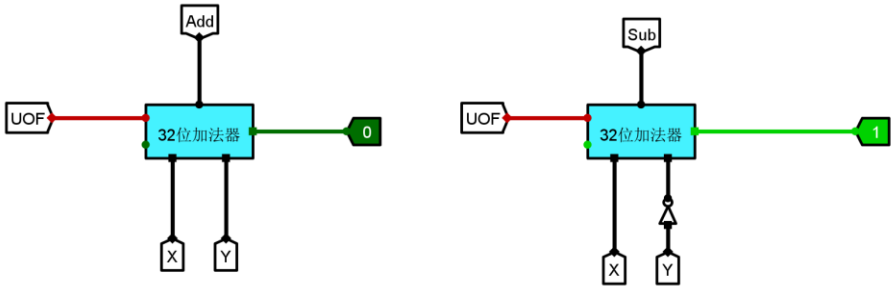




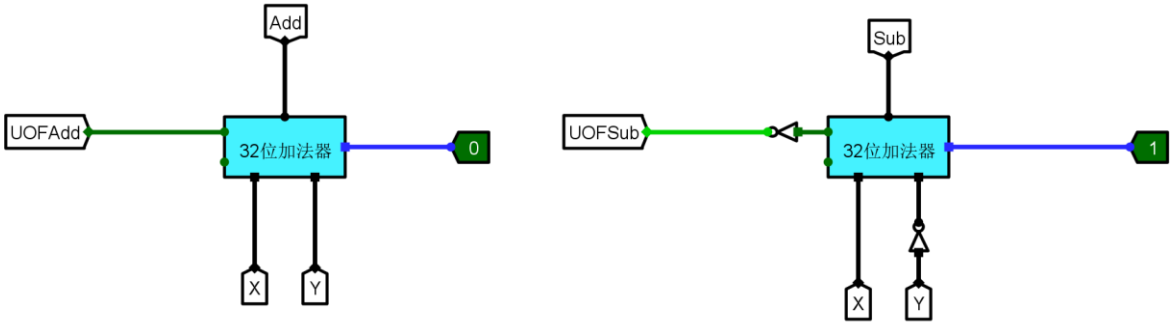
此处的const input是模仿前一部分移位器的const，这样就不用写两个32位的输入了，产生的结果也都使用了隧道来实现电路简洁。

（2） 第二部分（加减法的实现）：

该处的实现遇到了一些困难，根据ppt的提示，这两个操作需要给出有符号和无符号计算的溢出，针对无符号的溢出可使用32位加法器的C32进位来实现，但是有符号的溢出没有这么简单，于是忽略有符号的溢出，先实现基本的加减法功能：

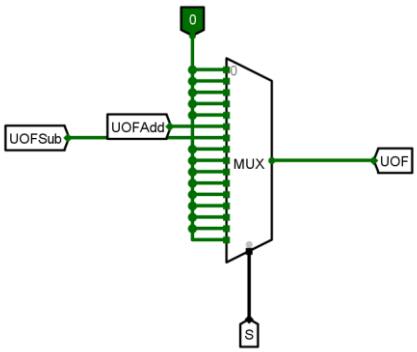


此处产生了一个错误，我想当然地将加法和减法的无符号溢出信号都直接接到结果UOF中，结果所有与UOF相连的线变成了红色，网上查资料显示说是结果不一致导致同一个隧道报错。除此之外，我无符号减法的溢出的计算也出错了，对于无符号减法，只可能产生下溢出（即结果小于0），而无符号的加法只能产生上溢出，因此，在对第二个操作数取反加1后，如果产生上溢，反而说明减法没有溢出，因此此处的无符号减法溢出要取反：



我将无符号加法和无符号减法的溢出计算分开，后面加一个选择器即可产生正确的无符号计算结果，并给无符号减法的溢出取反。

对于无符号溢出结果的选择器，由于只有计算加减法（运算指令为5和6）时有效，其他情况就设为0：



对于有符号的溢出，我先花了一些时间解决理论的问题才开始画的，先看有符号加法，有符号的溢出主要取决于三个数，前两个是操作数的符号位（即最高位），再一个是次高位的进位（决定符号位是否会反转），显然，当两个操作数的符号不同时，根据二进制补码，是不可能发生溢出的，只有当二者操作数符号相同时，而且符号位在计算后发生改变才说明溢出，于是便可以得到如下的真值表：

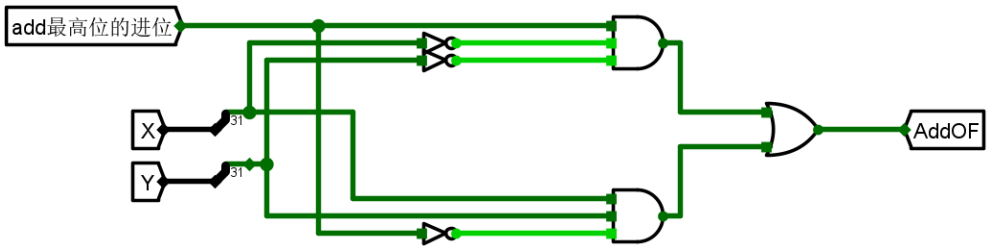
X最高位	Y最高位	次高位进位	是否溢出（1表示溢出）
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

从上面的真值表可以得出溢出的逻辑表达式：

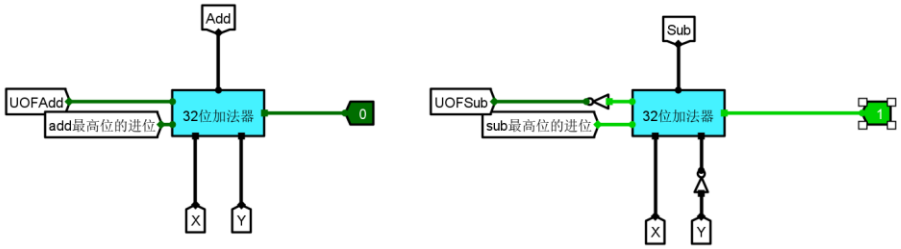
OF = （非X）（非Y）（C31）+ XY（非C31）

C31表示次高位进位。

于是可以画出如下电路图来得到有符号加法溢出：



其中“add最高位的进位”来源于32位快速加法器的C31接口：



在画此电路图时，我终于发现取出一个数的对应比特位的最好工具，使用Splitter就行，由于之前移位操作的selector也能用，便没有将selector换为splitter。

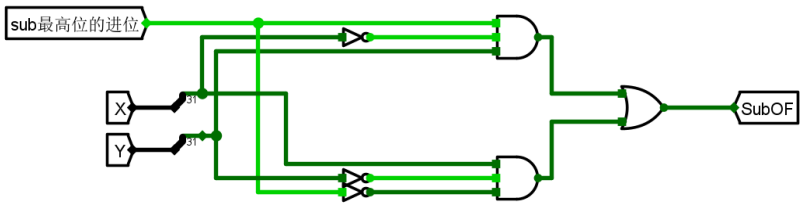
对于有符号减法也一样分析，可以得到如下真值表：

X最高位	Y最高位	次高位进位	是否溢出（1表示溢出）
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

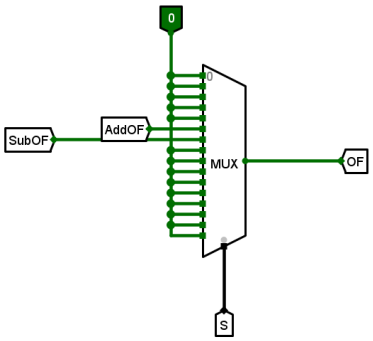
得到减法溢出的逻辑式：

OF = （非X）Y（C31） + X（非Y）（非C31）

画出电路图：



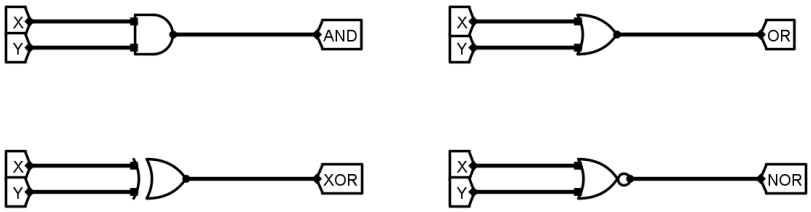
最后仿照无符号加减法做一个选择器：



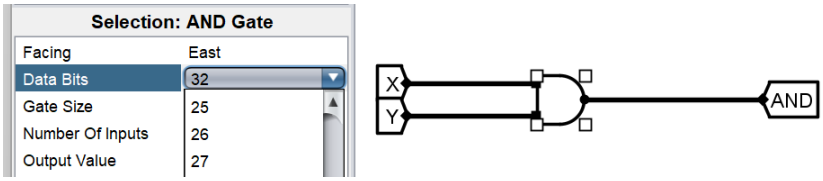
至此第二部分完成。

（3） 第三部分（剩余运算符和选择器的实现）：

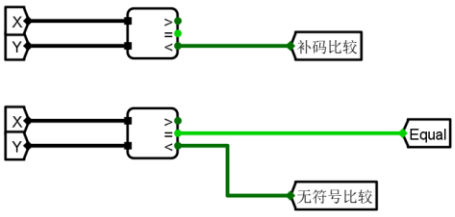
剩下的运算符实现都较为简单，运算符7~10为位运算，只要使用对应的门电路即可：



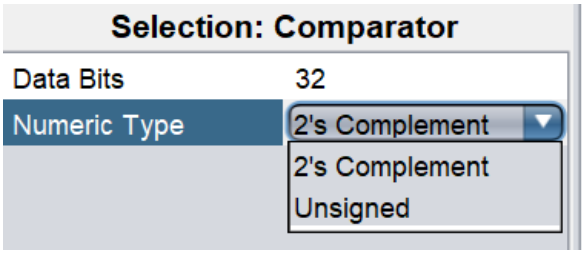
唯一需要注意的是要把门的data bits参数改为32位：



还有两个运算符是有符号和无符号的大小比较，可以使用logism内置的comparator（这次是直接网络上搜索出可用的元件，而不是一个一个在内置的库里面找了）：

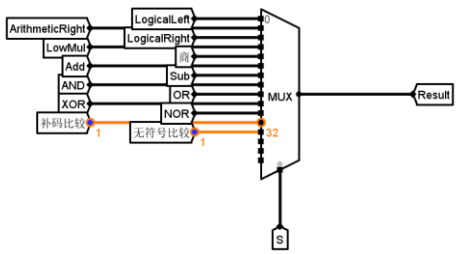


这两个比较器可以分别设置是补码比较和无符号比较：

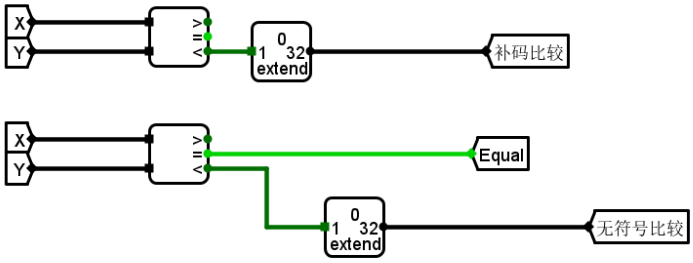


同时由于本实验的ALU还要求有一个equal接口，此处便将其中一个等号接口直接接到equal的输出上。

接下来画最后的选择器，用于结果的输出选择，由输入的操作码决定：

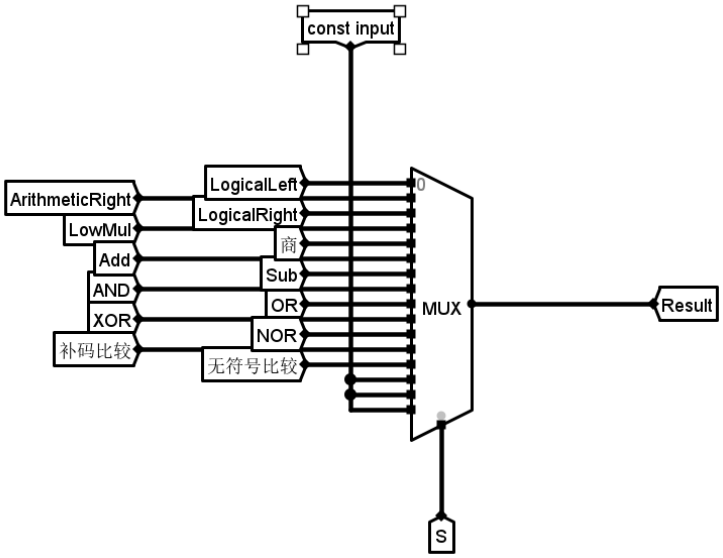


这里再次遇到了之前的位数不匹配的问题，需要将比较的结果进行位扩展，查到可以使用logism自带的bit extender进行位扩展，于是便小小调整了一下先前做好的大小比较：



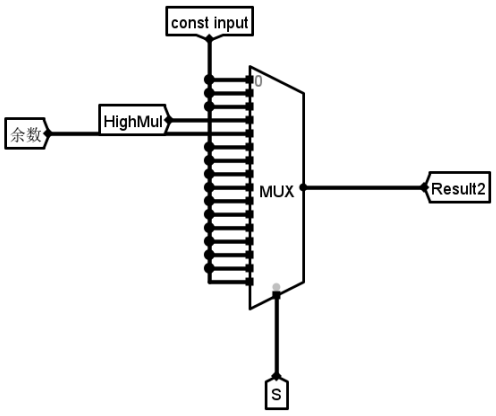
这里发现所有超过一位的隧道都是黑色的线，感觉不是很方便？

改完这个之后选择器就正确了：



这里使用了0（const input，之前用到过，表示32位的0）填充其他未定义操作码，来防止引脚悬空。

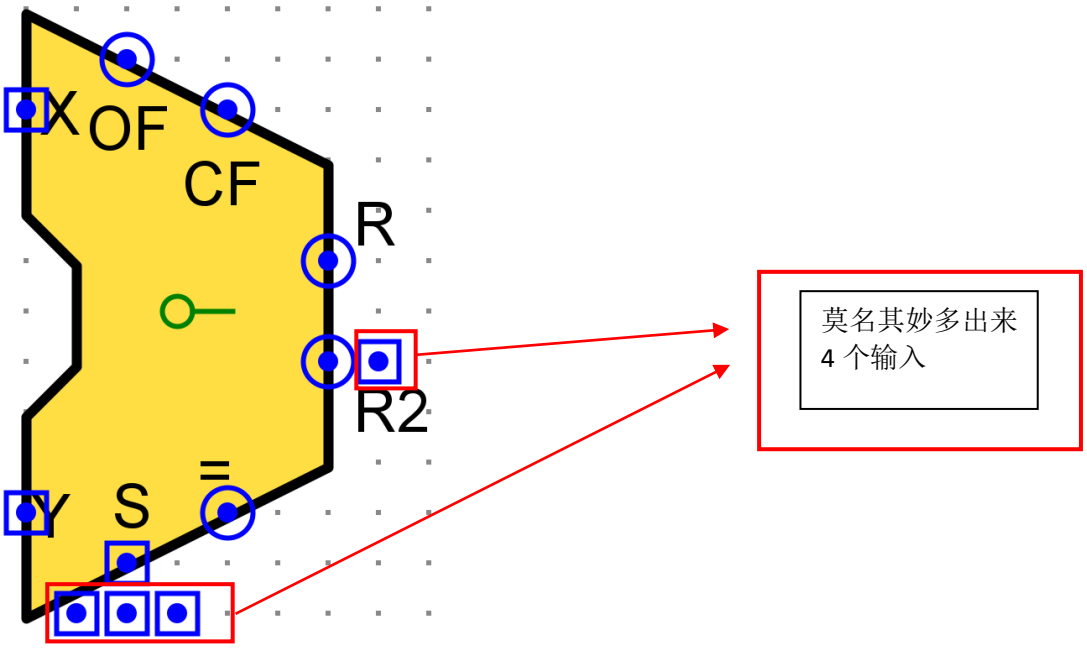
最后还需要给result2（乘法中的高32位，除法中的余数）做一个选择器，与上面的类似，只有操作码（3和4）有输入，其他默认输入为0（同样使用const input表示）：



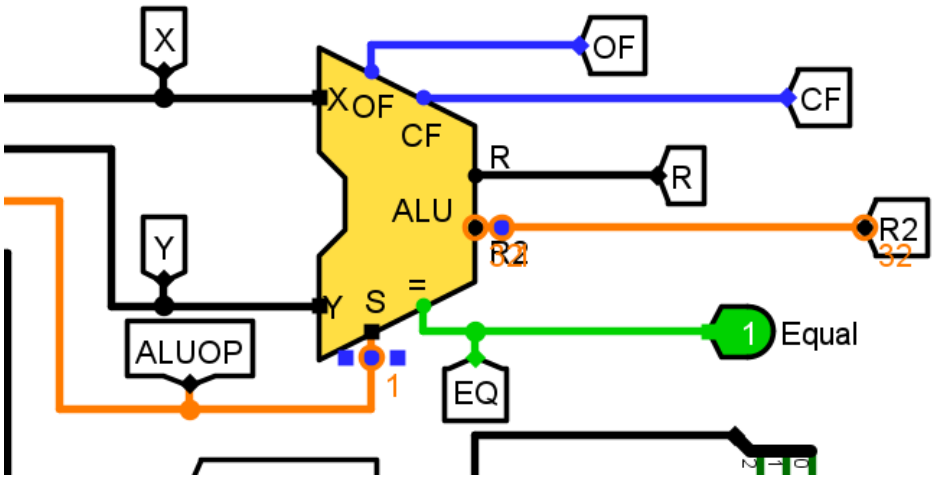
至此第三部分完成。

但是真的完成了吗？

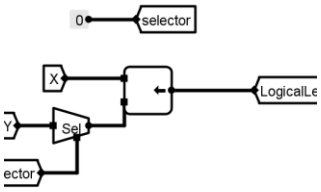
并没有，前面我一直犯了一个测试阶段才会发现的错误（对logism还是不熟悉），所有所谓的常量（const input， selector等不是真的常量，logism认为是用户要给输入的）导致了我的ALU封装变成了这个样子：

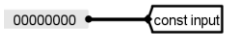


然后自动测试里面的ALU也乱了：



由于这些莫名其妙的接口导致了令人匪夷所思的错误，当时我一度怀疑是logism卡bug，后面发现自己定义的常量不是常量，而是接口，便将所有用到常量的部分换成logism里面真正的常量：





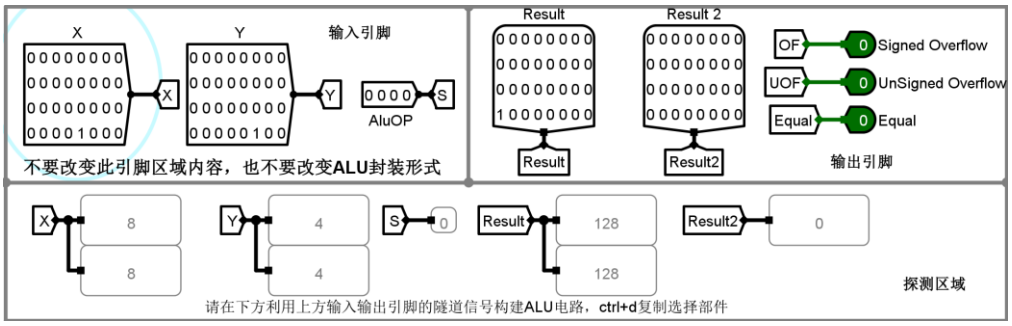
另外两个都是单比特常量，此处不再展示。

至此第三部分完成。（真的完成了，没有转折了）

六、 实验结果及分析

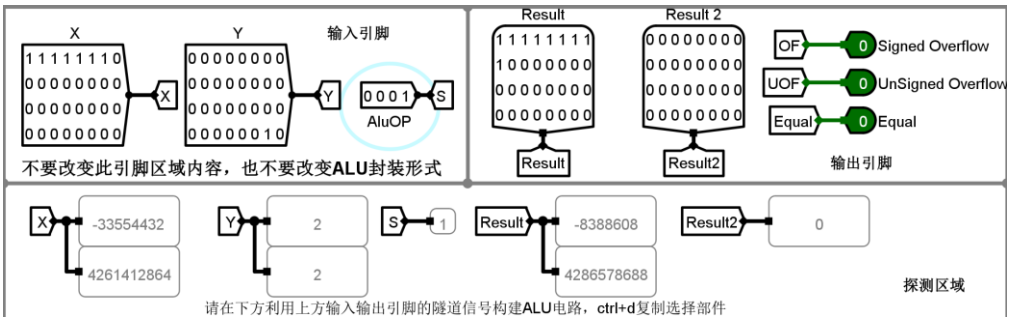
我将为每一个操作做一个自己的测试用例，其中加减法的操作会多做几个用于溢出符号的测试

（1） 逻辑/算术左移：



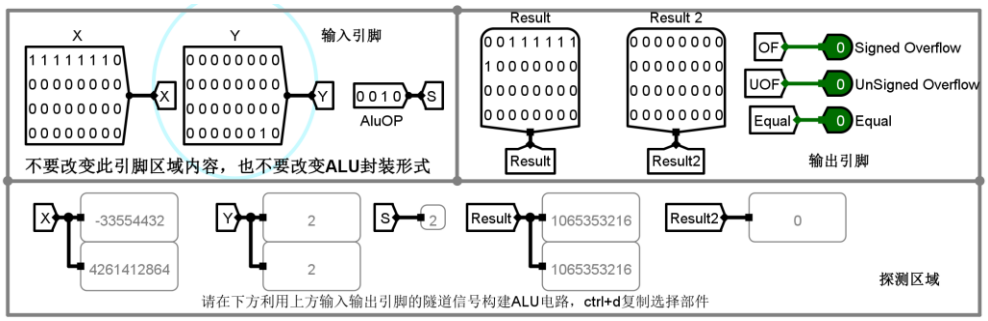
8左移四位，乘以16(2^4)，得128，故正确。

（2） 算术右移：



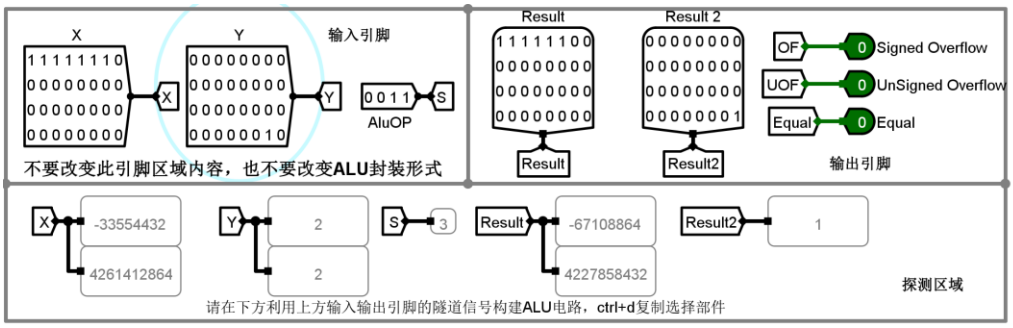
从result可以看到x带符号地右移了两位，故正确。

（3） 逻辑右移：



可以看到负数右移变成正数，result中的结果右移了两位，故正确。

（4） 64位无符号乘法：

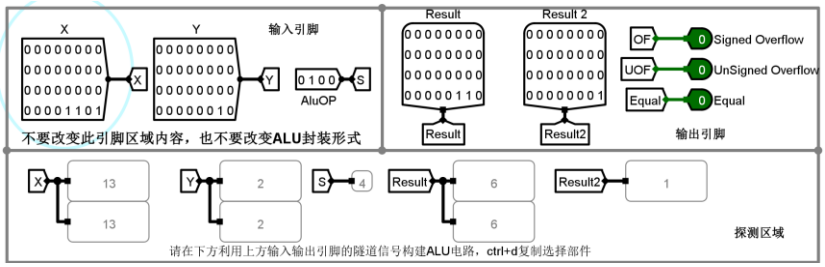


此处计算结果验证如下：

（此处是用结果倒推x的值）

高位（result2）为1，故有一个2^32,加上低位的部分再除以2（乘数为2），应该是x的值，故正确。

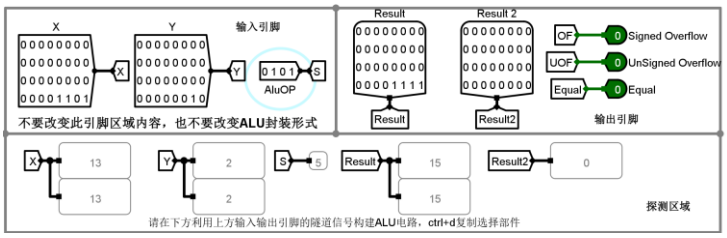
（5） 32位无符号除法：



13 除以 2 得 6 余 1，故正确。

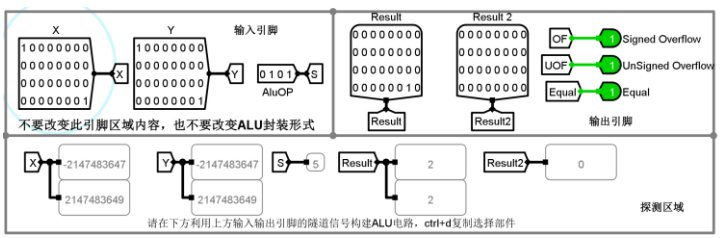
（6） 有/无符号加法：

普通加法：



13 + 2 = 15，故正确。

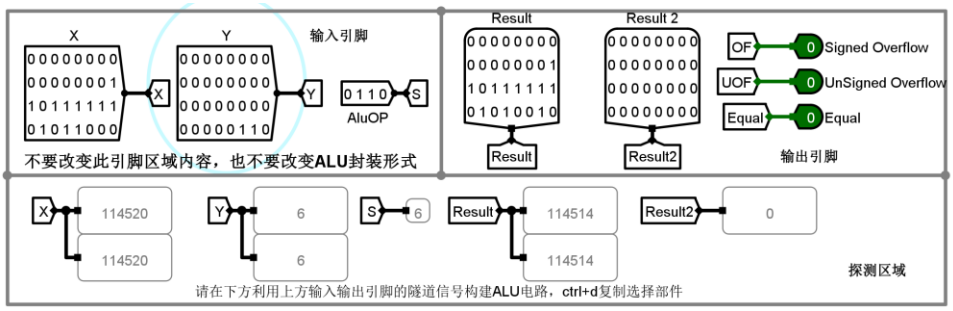
加法溢出：



可以看到x和y最高位的1分别相加溢出，最低位的1相加得到2，结果正确，溢出符号全亮（此处也顺带验证了equal输出，x和y是一模一样的。）

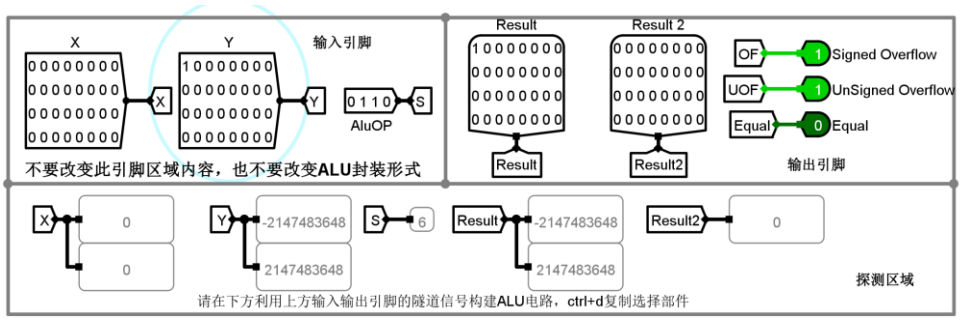
（7） 有/无符号减法：

正常减法：



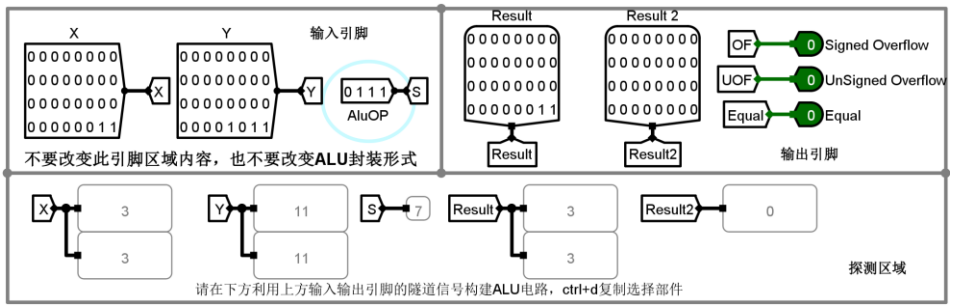
114520 - 6 = 114514，正确。

减法溢出：



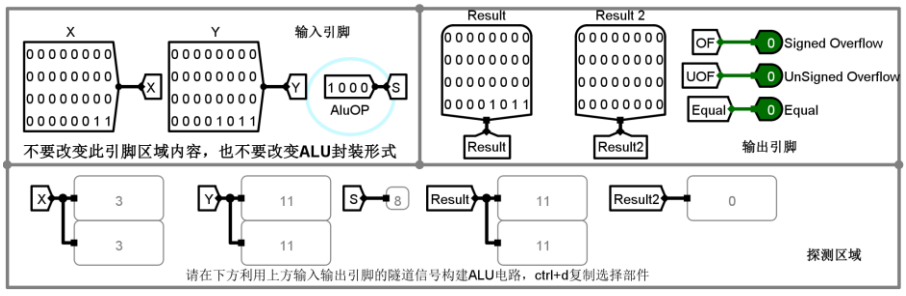
此处使用补码算法可以得知 -Y 和 Y 一样，故结果应该和Y一样，且发生了溢出，故正确。

（8） 按位与：



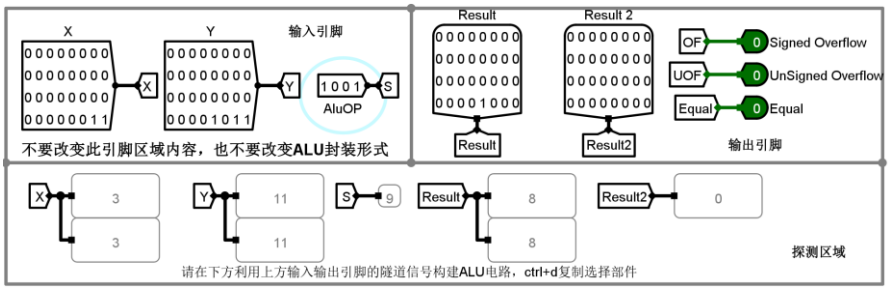
11（1011）与上 3（0011）结果为3，故正确。

（9） 按位或：



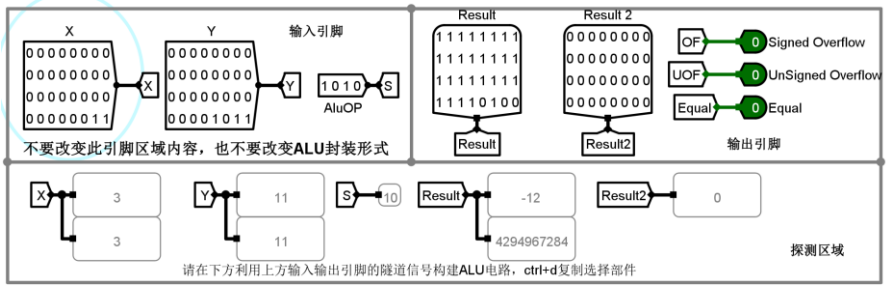
11（1011）或上 3（0011）结果为11，故正确。

（10） 按位异或：



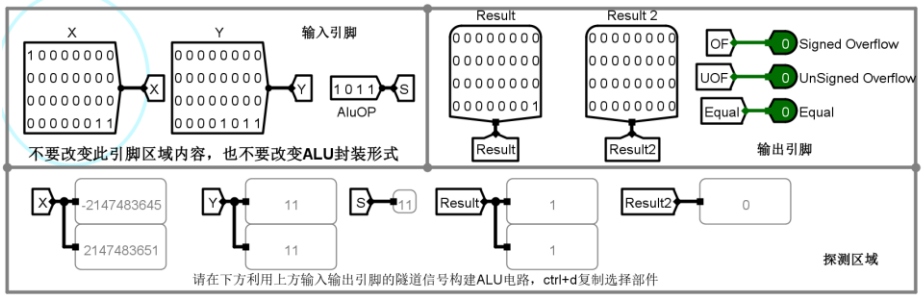
11（1011）异或 3（0011）结果为8（1000），故正确。

（11） 按位或非：



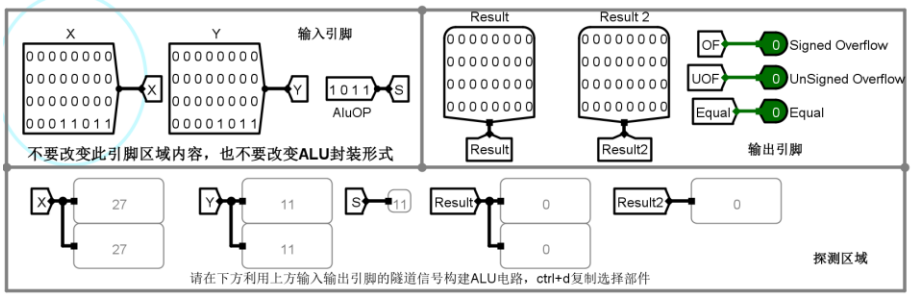
11（1011）或非 3（0011）结果为 -12，可以和或的结果比较，确实是取反的。

（12） 有符号比较：



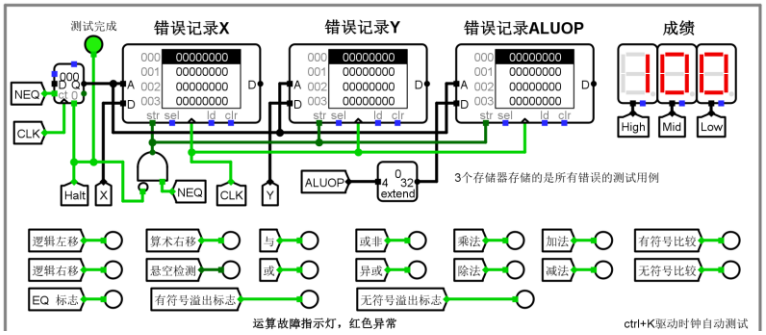
此处X明显小于Y，故正确。

（13） 无符号比较：



27 > 11，故正确。

（14） 自动测试：



所有测试点均通过。

七、 心得体会

本次实验体验了一把制作ALU的过程，虽然不是使用实体的硬件和门电路进行搭建，整个ALU的构建过程算是搞得比较清楚了，其中让我印象最为深刻的是加减法溢出的门电路搭建，这个部分是实验里面最困难的地方，需要列出真值表进行分析，并根据分析结果画出门电路，否则只单纯思考的话，很难第一次就思考出来。

另外一个我觉得学习到的东西是logism的深度用法，里面常量，多选器，位分组选择器，位扩展器等我都是在本次实验中第一次用到，因此对logism的使用也更加熟练。

最后一点吐槽一下logism有时候会抽风，需要重启软件才能正常工作，不是很利于调试。