**实验 4：加法器和 ALU 设计**

姓名: 学号:

**一、实验目的**

1. 掌握先行进位加法器 CLA 和先行进位部件 CLU 的设计方法。

2. 掌握 32 位先行进位加法器的实现方法。

3. 掌握 ALU 的设计方法。

**二、实验环境**

Logisim：https://github.com/Logisim-Ita/Logisim

**三、实验内容**

**1. 4 位先行进位加法器 CLA 实验**

**实验目的：利用四位先行进位部件CLU设计四位快速加法器CLA**

先行进位部件原理图如下：



P和G分别为进位传递函数和进位生成函数：Pi= Xi+Yi，Gi= XiY

利用该部件实现4位快速加法器CLA

实验步骤：

1）设计CLU部件

·首先依据原理图，搭建四位先行进位器CLU；

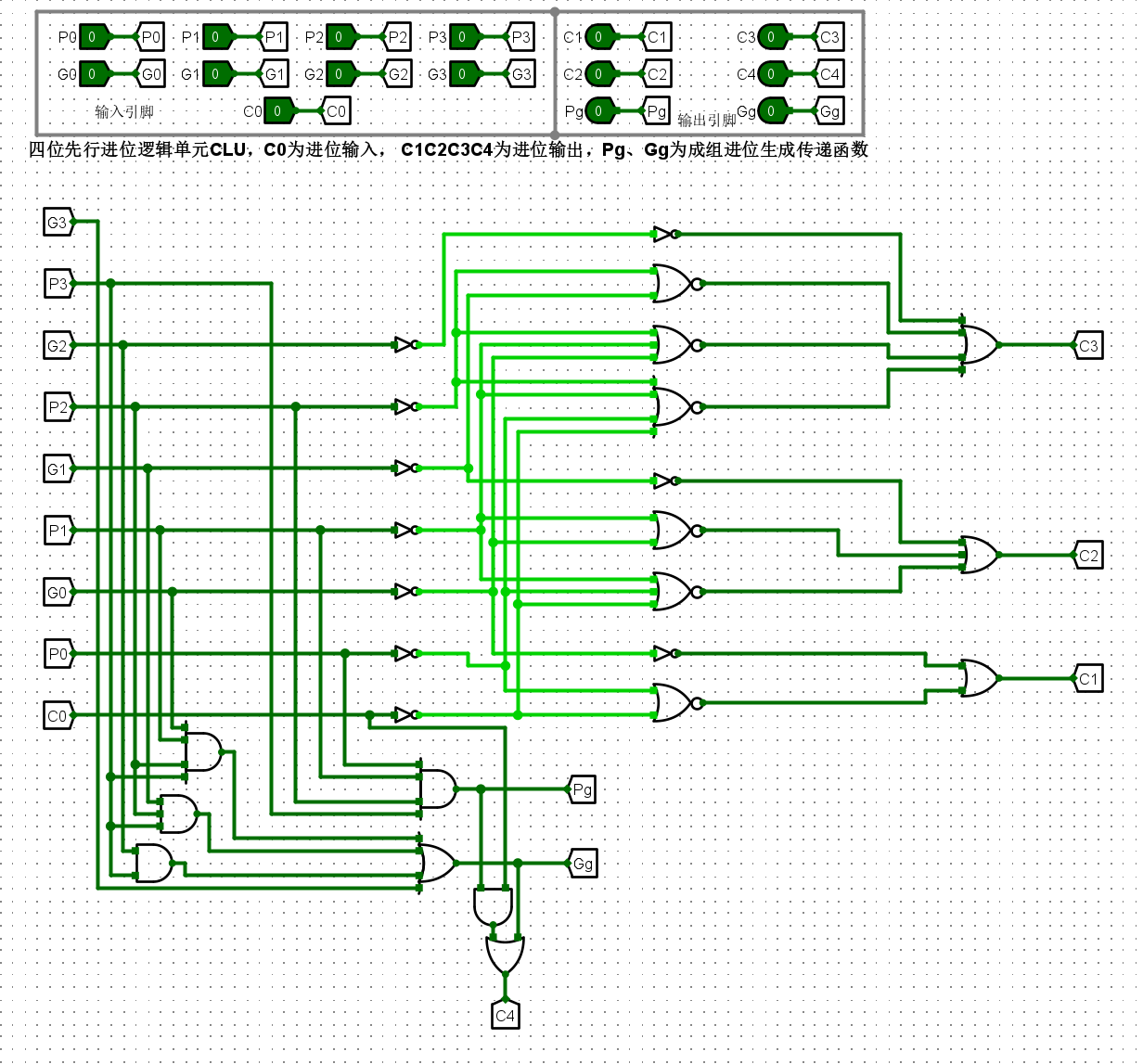
·然后根据组间并行的需要，输出组间进位生成函数Gg和组间进位传递函数Pg：

Pg= P3P2P1P0

Gg= G3+P3G2+P3P2G1+P3P2P1G0

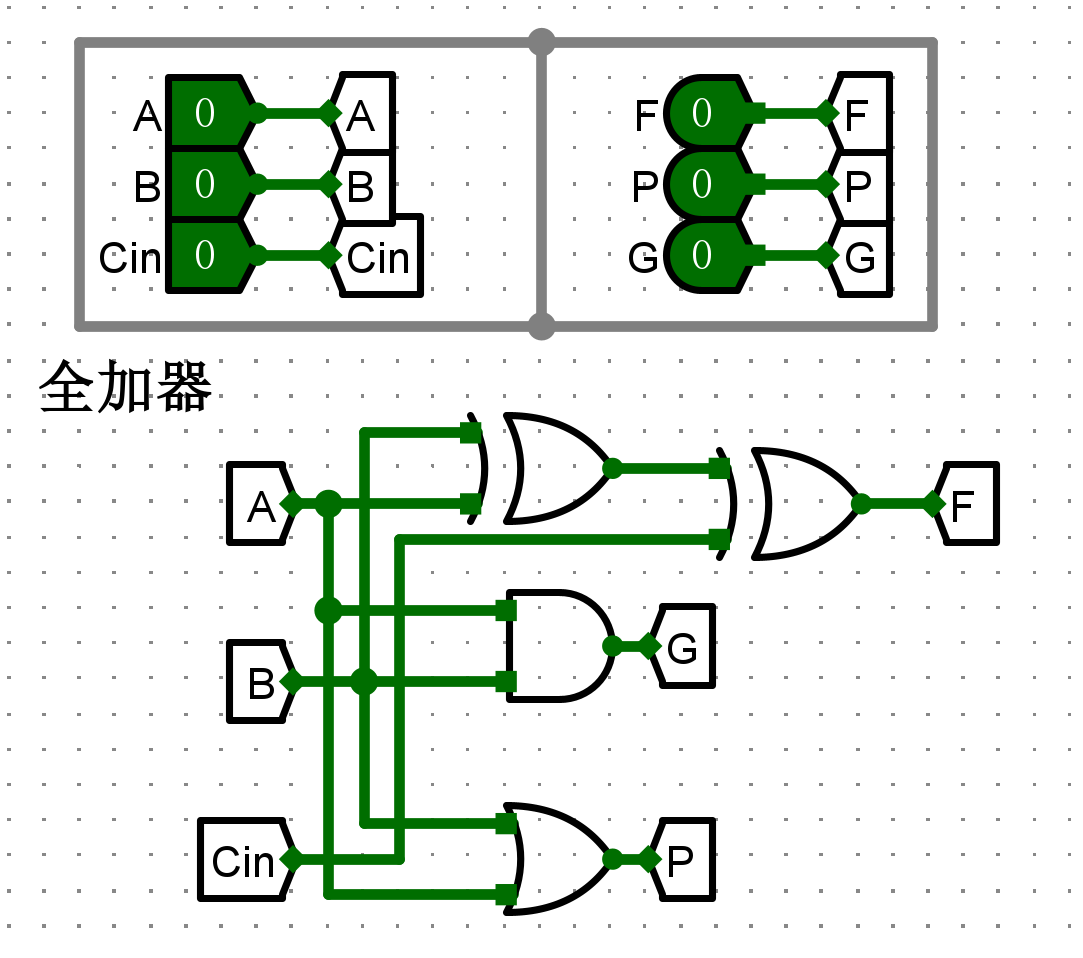
·将C4的逻辑表达式改写成为C4=Gg+PgC0实现简化，然后在CLU部件中增加两个输出 端口Pg和Gg。

电路图如下：



2）设计设计支持进位生成函数和进位传递函数的全加器：

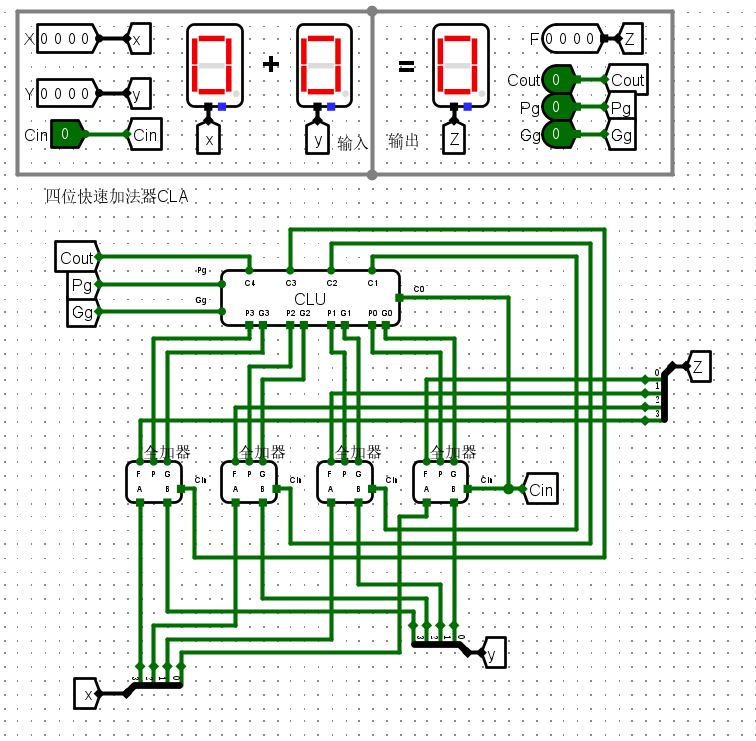
电路图如下：



3）设计4位快速加法器CLA：

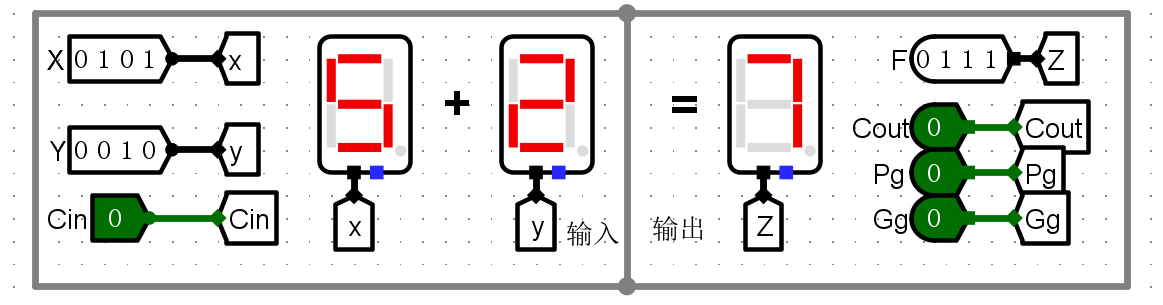
在子电路中，定义输入输出引脚和组件布局，其中包含输入输出引脚、隧道、4位先行进位部件子电路、全加器子电路、LED数码管、逻辑门电路等组件

电路图如下：

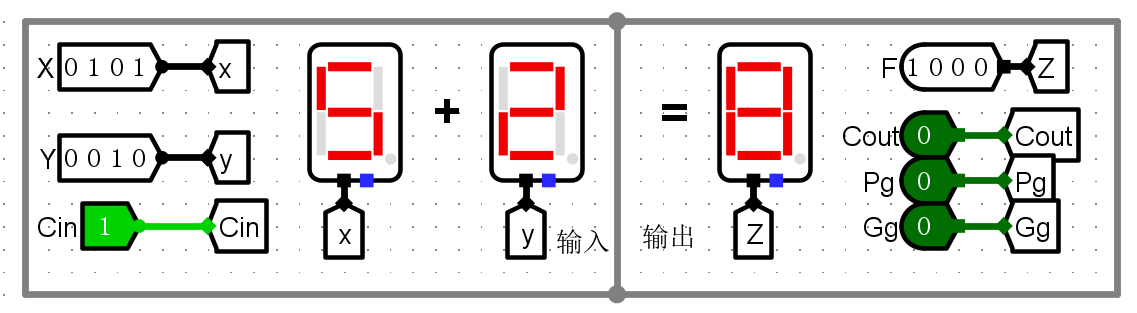


4）进行仿真测试

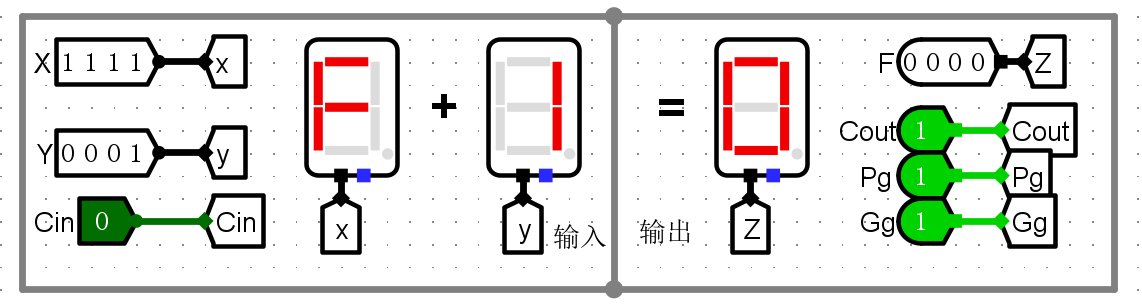
简单加法：



含低位进位：



高位进位：



**2.** 16位两级先行进位加法器实验

若将式(11-1)中进位 C4 的逻辑表达式改写成为 C4=Gg+PgC0，则 Pg、Gg 分别表示 4 位加法器的组间进位传递和组间进位生成输出变量，其逻辑表达式分别如下：

Pg= P3P2P1P0

Gg= G3+P3G2+P3P2G1+P3P2P1G0

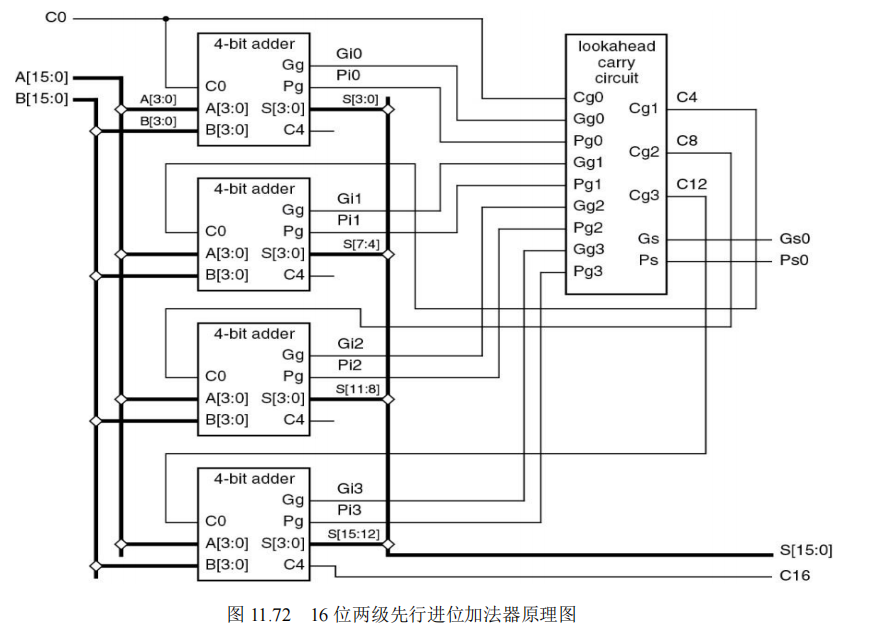
对于一个 16 位加法器，可以分成 4 组，每组用一个 4 位 CLA 实现，C4、C8、 C12、C16 分别为每一组向前一组的进位，即组间进位。将 Pg、Gg 用于生成 4 个组间 进位，则有如下逻辑表达式：

C4= Gg0 + Pg0C0

C8= Gg1 + Pg1C4 = Gg1 + Pg1Gg0 + Pg1Pg0C0

C12= Gg2 + Pg2C8= Gg2 + Pg2Gg1 + Pg2Pg1Gg0 + Pg2Pg1Pg0C0

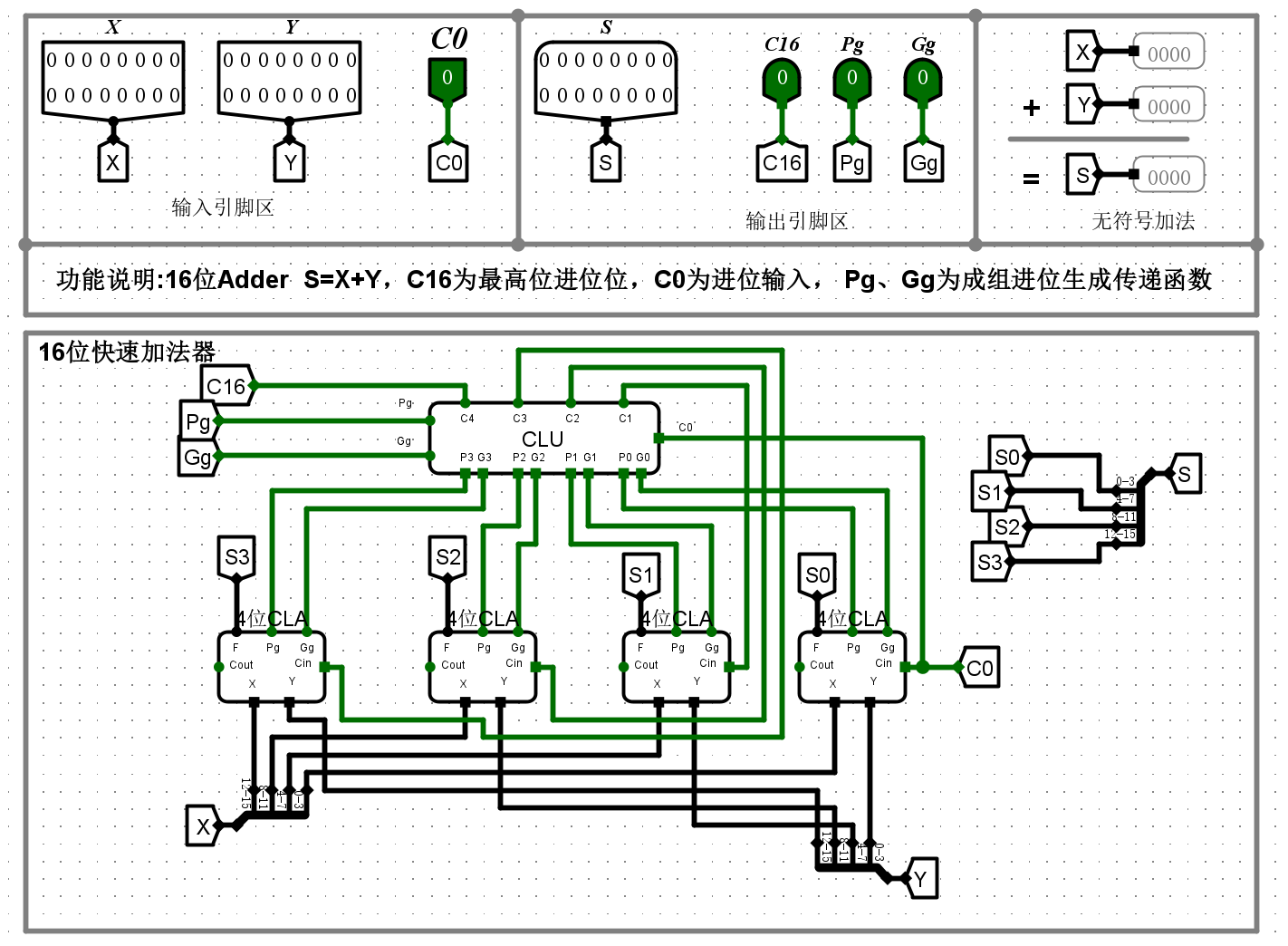
C16= Gg3 + Pg3C12 = Gg3 + Pg3Gg2 + Pg3Pg2Gg1 + Pg3Pg2Pg1Gg0+ Pg3Pg2Pg1Pg0C0

 ·16 位两级先行进位加法器原理图：

实验步骤：

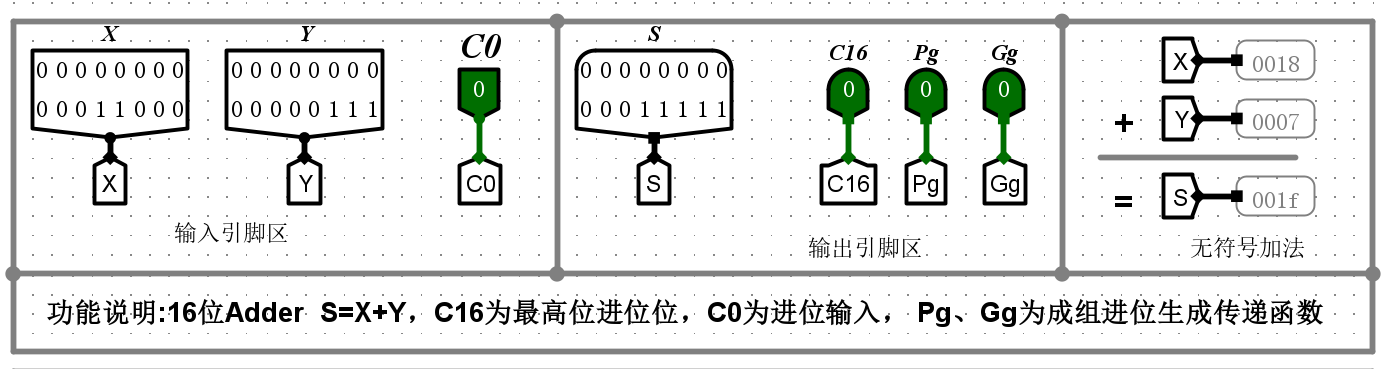
1）在Logisim中添加子电路：16位先行进位加法器。在Logisim工作区中按图4.9所 示，定义输入输出引脚和组件布局，其中包含输入输出引脚、隧道、4位快速加法器、 4位先行进位部件子电路、逻辑门电路等组件。参照图4.8的电路原理图，连接组件， 实现电路。

2）连接电路如下：

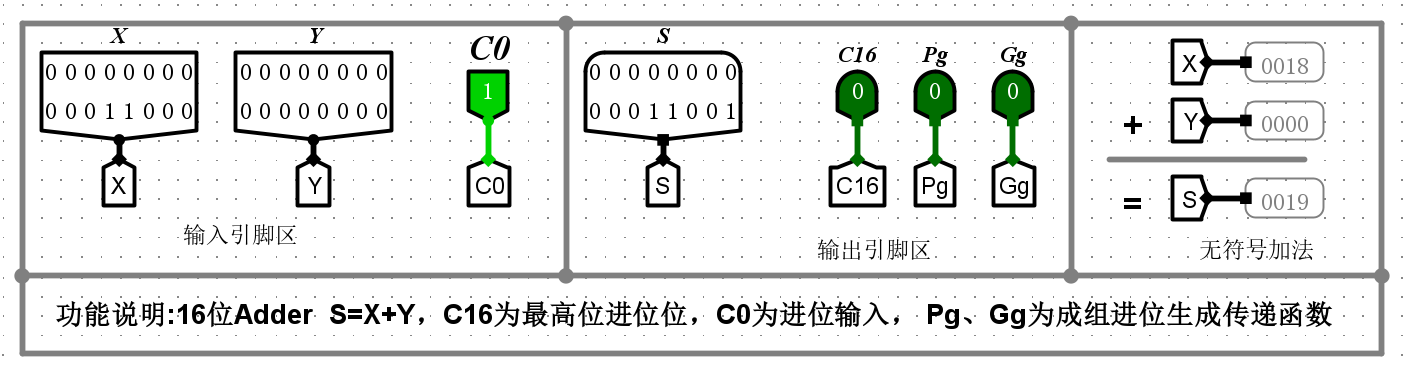


3）仿真实验：

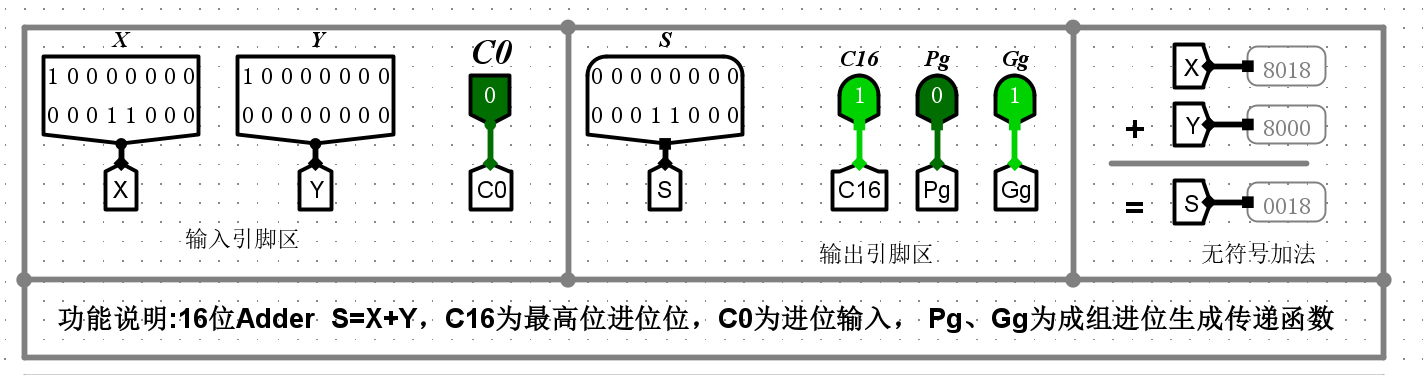
简单加法：



低位进位：

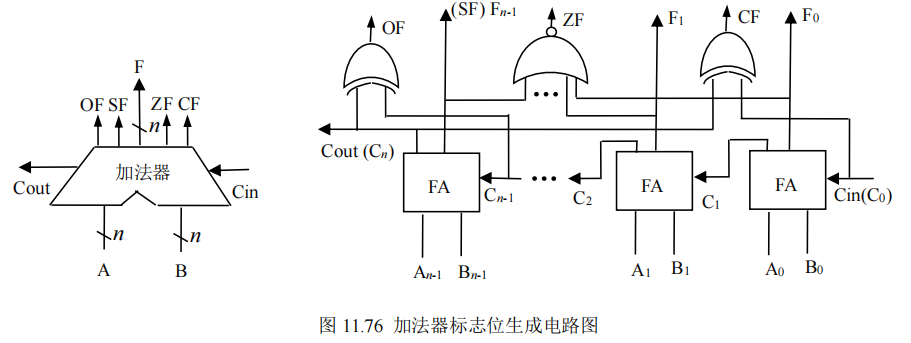


高位进位（组间进位）：



**3. 32 位加法器构建实验**

通过将两个 16 位两级先行进位加法器串行级联构建一个 32 位加法器，并根据给 出的标志位生成电路，在 32 位加法器中生成 CF、SF、OF、ZF 标志位。

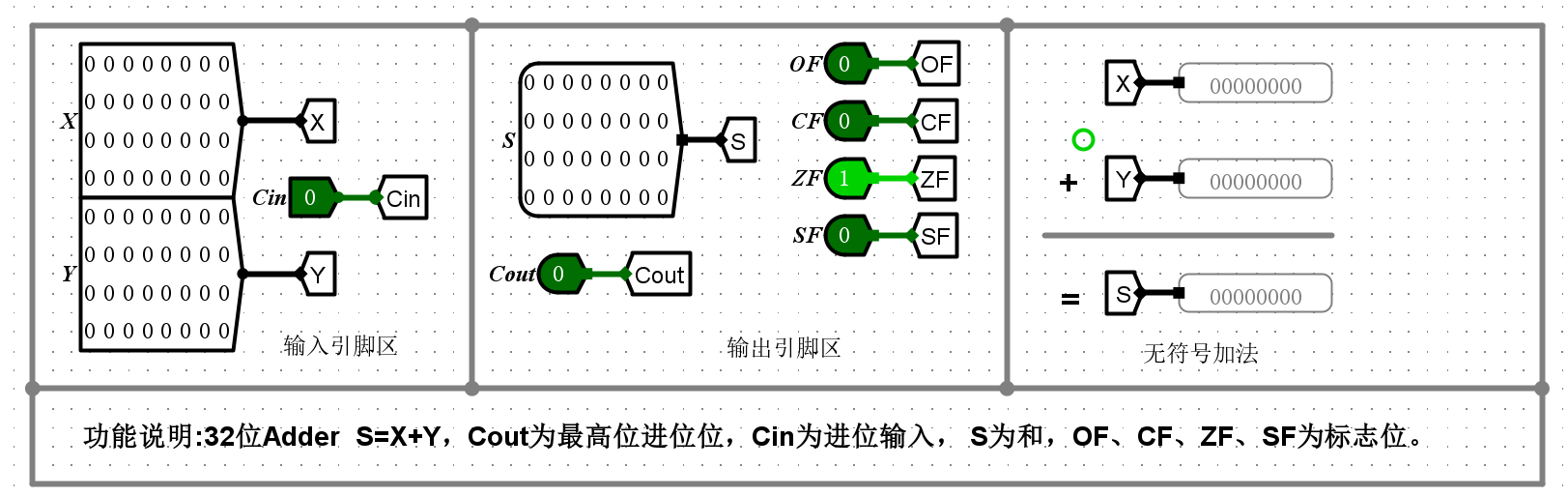
原理图：

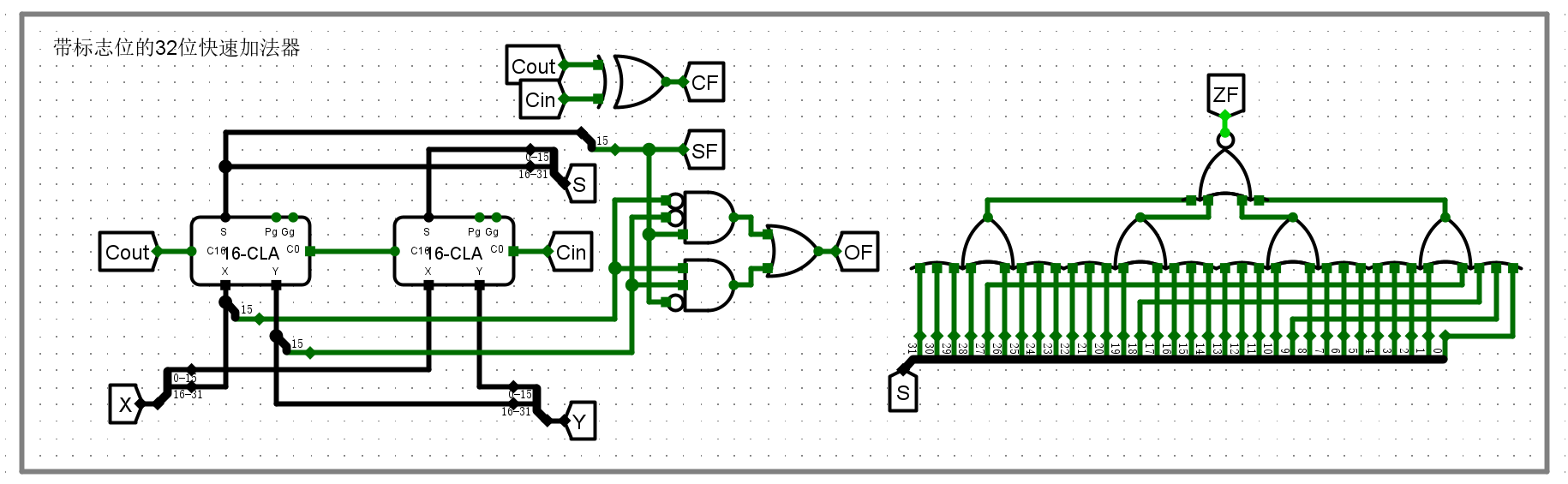
实验步骤：

1）创建子电路。在Logisim中添加一个名为“32位加法器”的子电路，并双击该子电 路名称，在右侧工作区中构建相应电路。

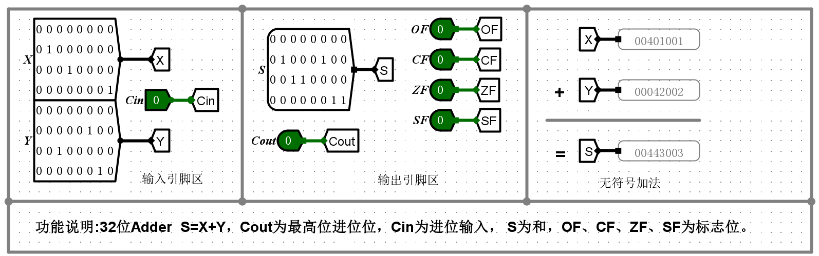
2）设计子电路并进行功能测试。如图4.12所示，在工作区中添加两个16位先行进位 加法器、逻辑门、分线器、输入/输出引脚、探针等部件，将它们布局到适当位置，进 行线路连接，添加标识符和电路功能描述文字。通过设置加数A、B和低位进位CIN 的输入值，观察相加和F、标志位和向高位的进位Cout等输出值。

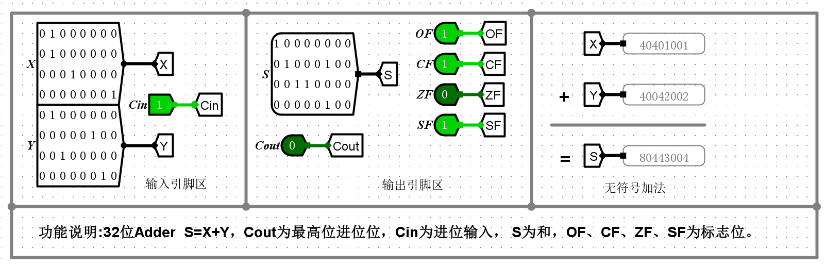
电路图如下：

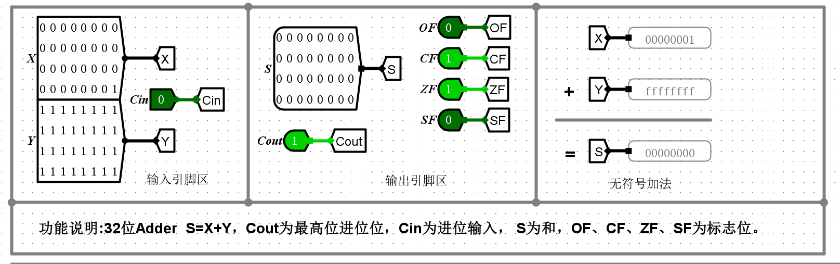




3）仿真实验：

 简单加法：

 低位进位与溢出：

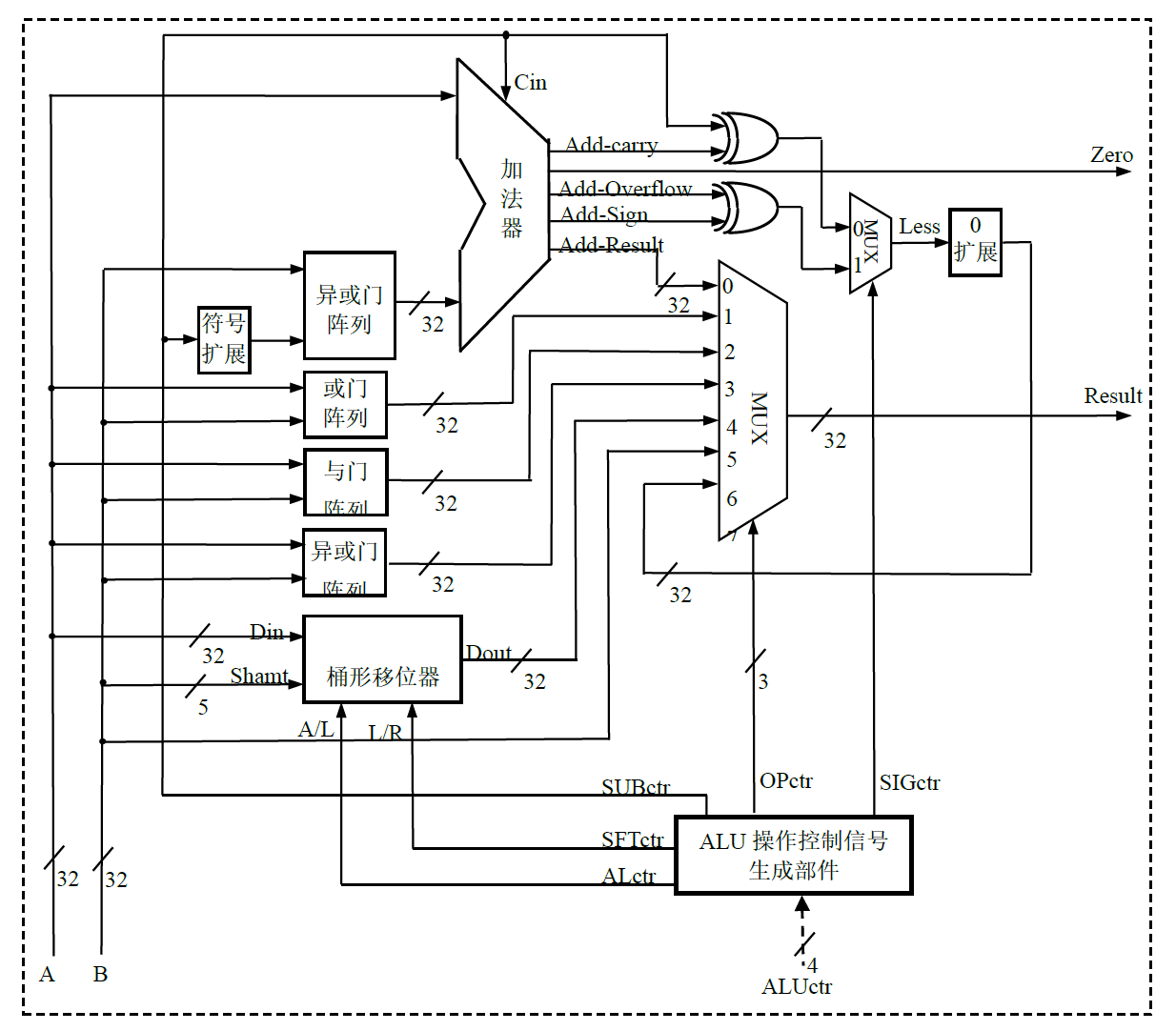
 高位进位：

**4. ALU 设计实验**

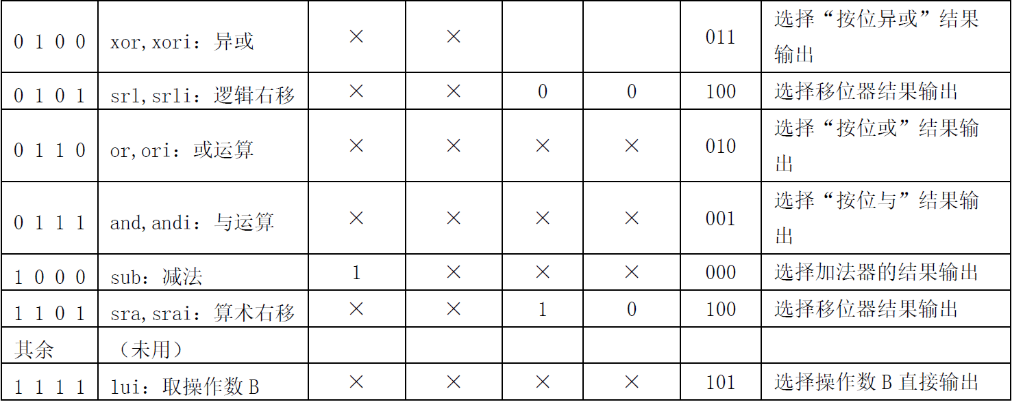
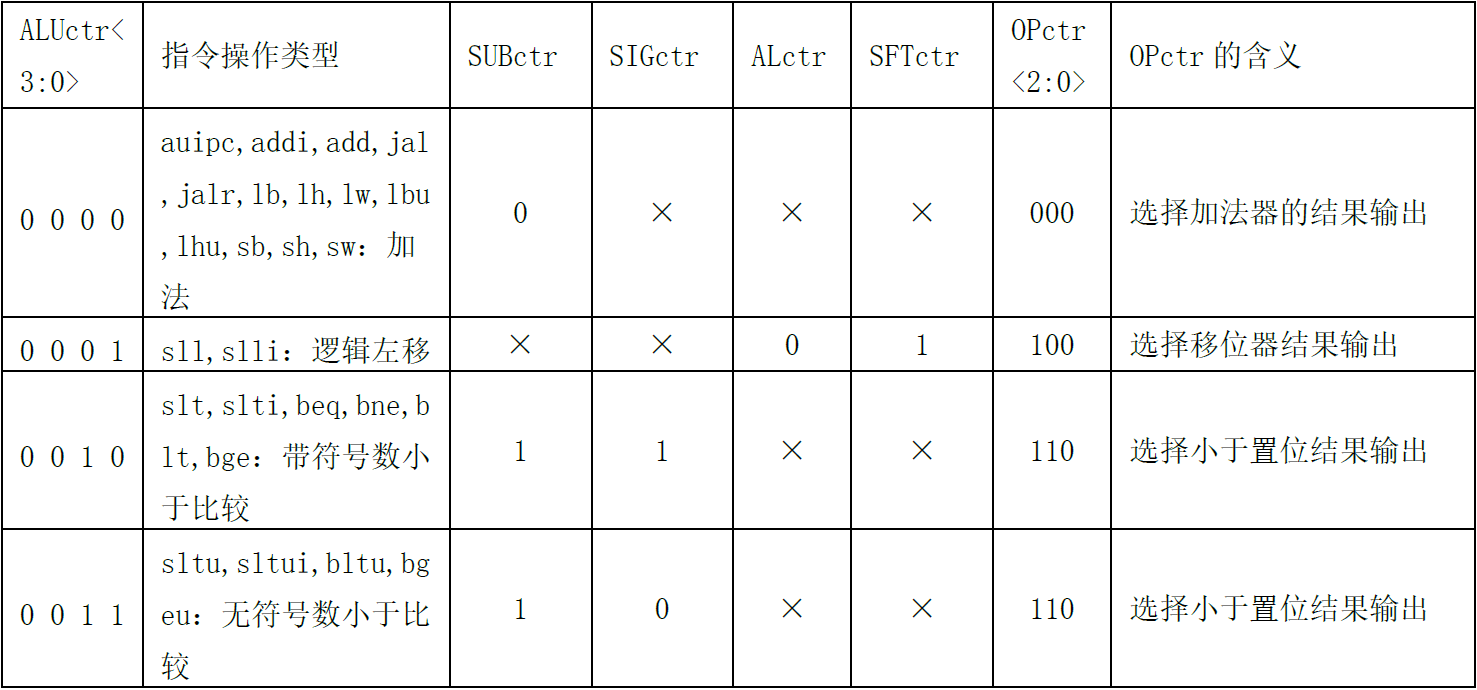
RV32I 包括4种ALU指令：比较运算、移位运算、逻辑运算和算术运算。比较运 算通过操作数相减然后判断标志位来生成结果；移位运算通过桶形移位器来实现左移和 右移；逻辑运算可直接通过逻辑门阵列来实现；算术运算通过加法器以及生成的标志位 来实现。ALU在操作控制信号ALUctr的指示下，执行相关运算，Result作为ALU运 算的结果被输出，零标志Zero被作为ALU的结果标志信息输出。

ALU的操作需要通过输入ALUctr信号生成ALU控制信号来确定，控制信号包括： SUBctr用来控制ALU执行加法还是减法运算，当SUBctr=1时，做减法，当SUBctr=0 时，做加法；SIGctr信号控制ALU是执行“带符号整数比较小于置1”还是“无符号数 比较小于置1”功能，当SIGctr=1，则执行“带符号整数比较小于置1”，当SIGctr=0， 则执行“无符号数比较小于置1”；SFTctr用来控制桶形移位器移位方向，当SFTctr=0， 则执行右移操作，当SFTctr=1，则执行左移操作；ALctr用来控制桶形移位器执行算术 移位还是逻辑移位，当ALctr=0，则执行逻辑移位，当ALctr=1，则执行算术移位； OPctr[2:0]用来控制选择哪种运算的结果作为Result输出，因为有加法器和、按位或、 按位与、按位异或、移位器输出、操作数B输出、小于置1等7种运算，所以输出结果 的多路选择器的选择信号OPctr需要3位二进制数。

ALU原理图如下：



编码与控制信号的关系表如下：

\*注：与运算和或运算，原理图和关系表不匹配，下按关系表实现，和原理图有出入

控制信号的最小项表达式如下：

SUBctr =∑m(2,3,8)

SIGctr =m2

ALctr =m13

SFTctr =m1

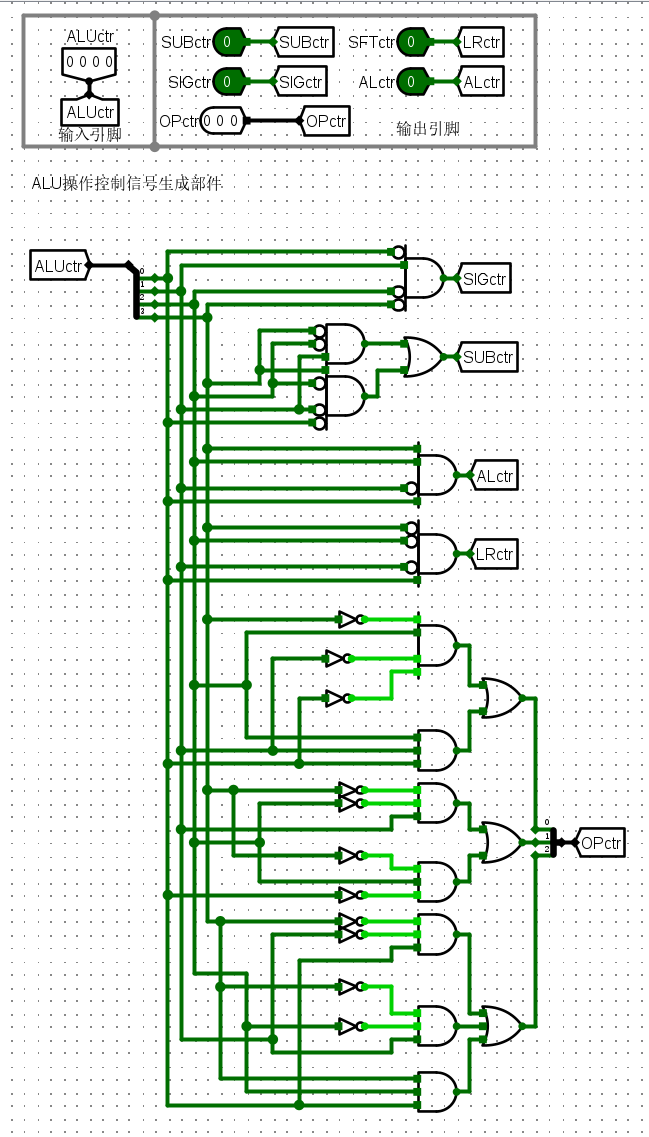
Opctr[2]=∑m(1,2,3,5,13,15)

Opctr[1]=∑m(2,3,4,6)

Opctr[0]=∑m(4,7, 15)

根据图4.14所示的ALU原理图可知，组成32位ALU的基本部件除了基本的逻辑门、 多路选择器和扩展器外，较复杂的子电路包括一个32位加法器、32位移位器和一个ALU 操作控制部件。前述实验已经设计了一个32位加法器并封装生成了相应的子电路，因 此，还需要设计一个ALU操作控制部件和32位桶形移位器，并将其封装成子电路。

实验步骤：

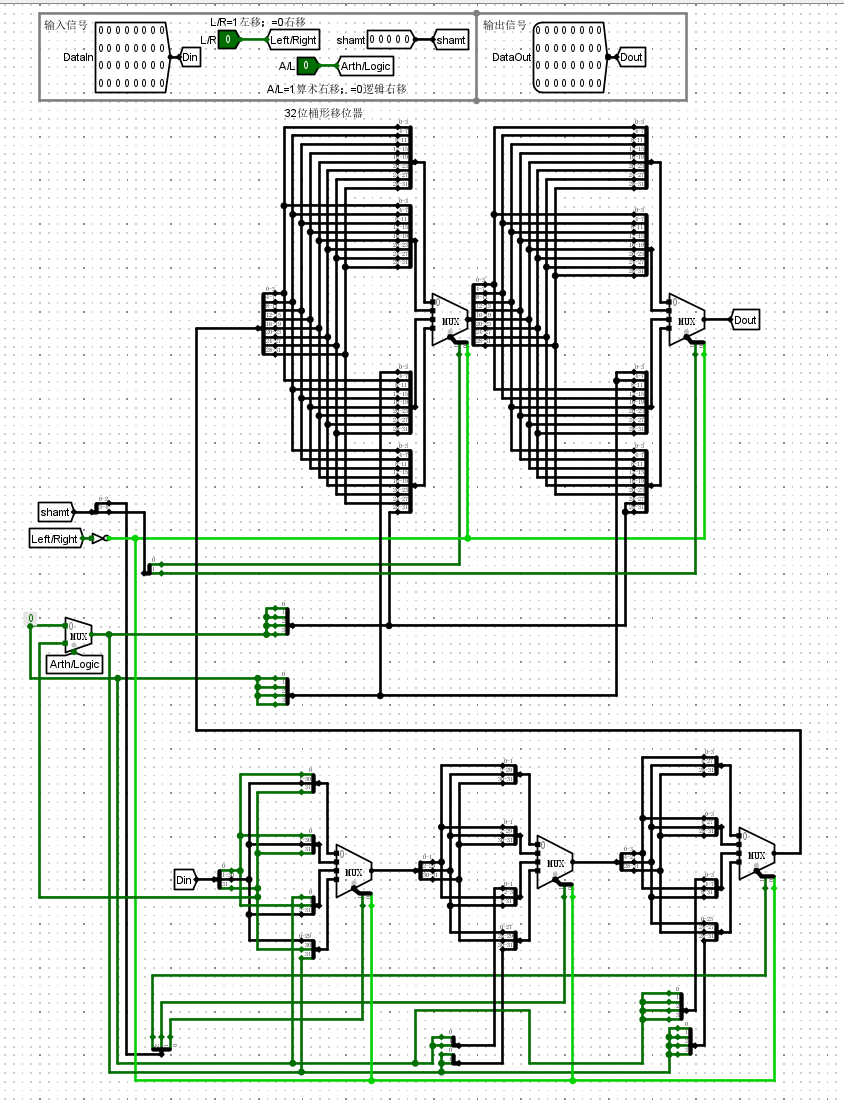
 1）ALU操作控制部件实验。在Logisim中添加一个名为“ALU操作控制部件”的子电 路，根据控制信号逻辑表达式，按照图4.15的组件布局图，在工作区中放置合适的逻辑 门电路、多路选择器等组件。设置相应属性，进行线路连接，添加标识符和电路功能描 述文字。电路图如下：

依据关系图验证输入输出正确

2）设计32位桶形移位器部件

考虑使用实验2中已完成的32位桶形移位器，简单修改为shamt为5位的版本。

电路图如下：



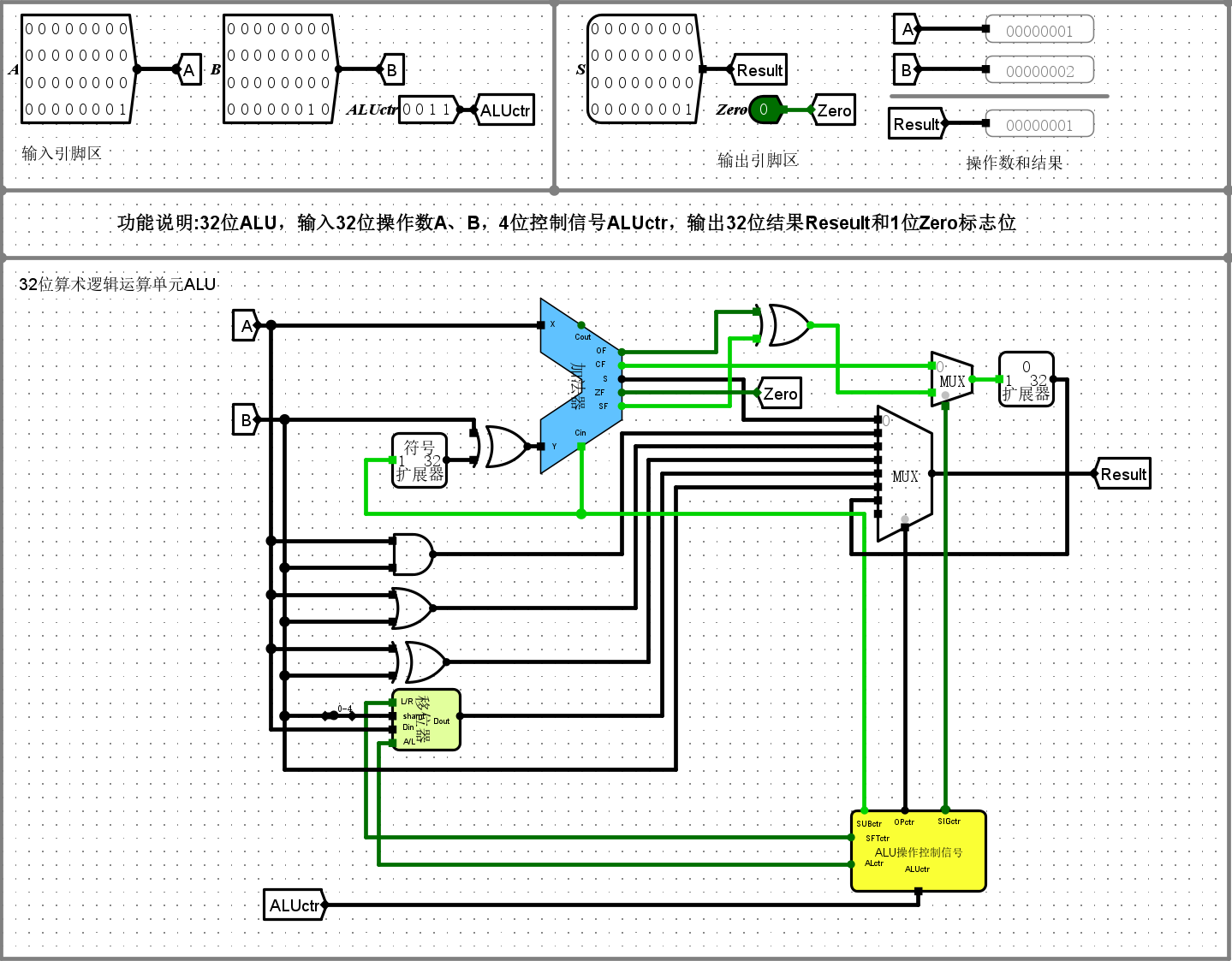
简单验证功能正常

3）实现32位ALU

在Logisim中添加一个名为“32位ALU”的子电路，在右侧工作区中构建相应电路。参照图4.14所示的ALU原理图，按照图4.19所示的ALU组件布局图，在工作区中放置所需的32位加法器、ALU操作控制部件、32位移位器、逻辑门阵列、多路选择器、位扩展器、探针等组件，设置相应属性，进行线路连接，添加标识符和电路功能描述文字，移位器的移动位数由操作数B的最低5位数决定。

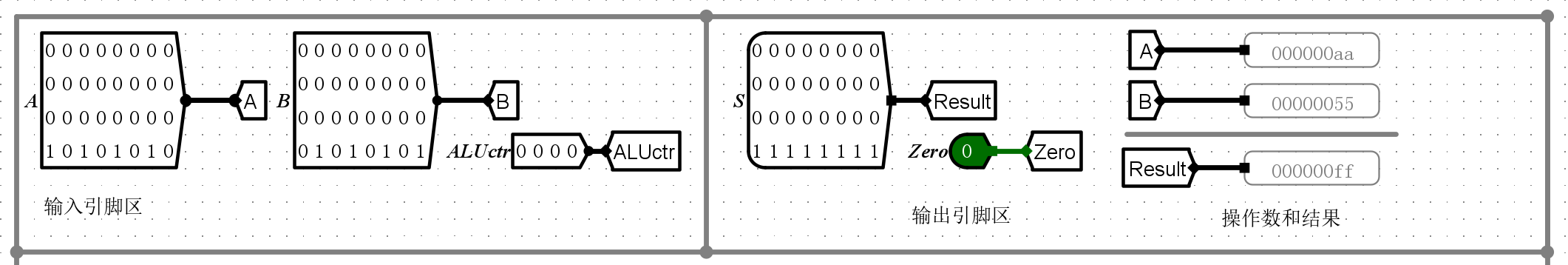
ALU输入信号包括两个32位操作数A、B和一个4位ALU控制信号ALUctr，输出信号包括一个32位结果数据S和一个1位结果为0的标志位Zero。由于32位加法器直接输出了CF标志位，因此不需要将CIN和Adder-carry异或来生成CF，而是直接将加法器的CF输出端连到2选1多路选择器的输入端。

电路图如下：

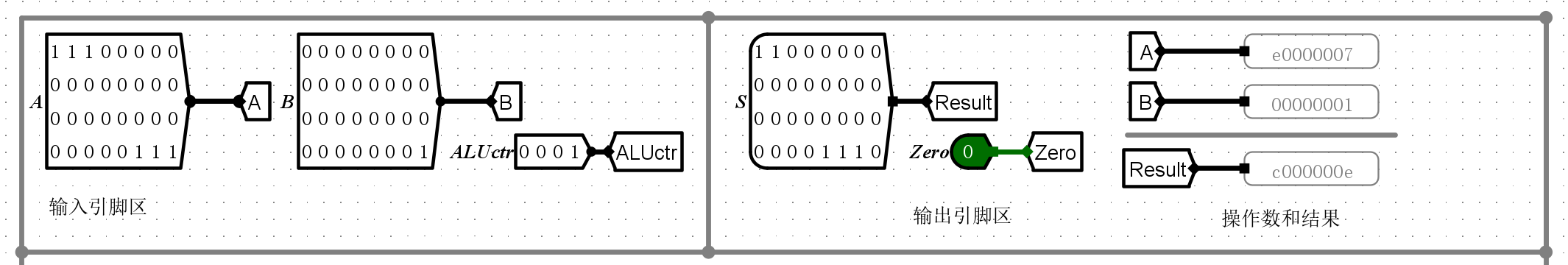


4）仿真实验，验证每种控制信号的功能

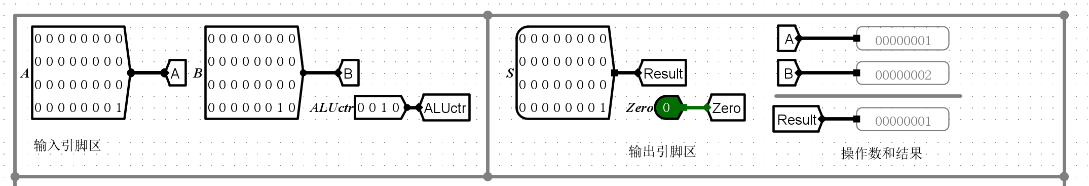
0000-加法：

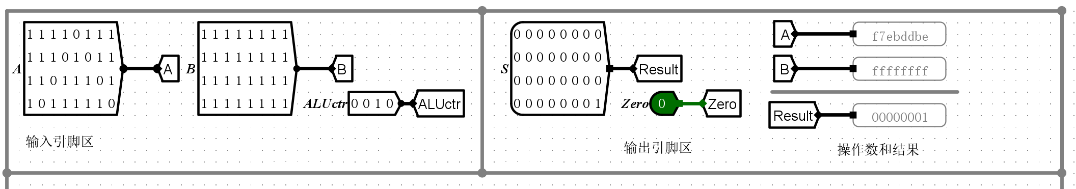


0001-逻辑左移（B第五位为位移位数）：

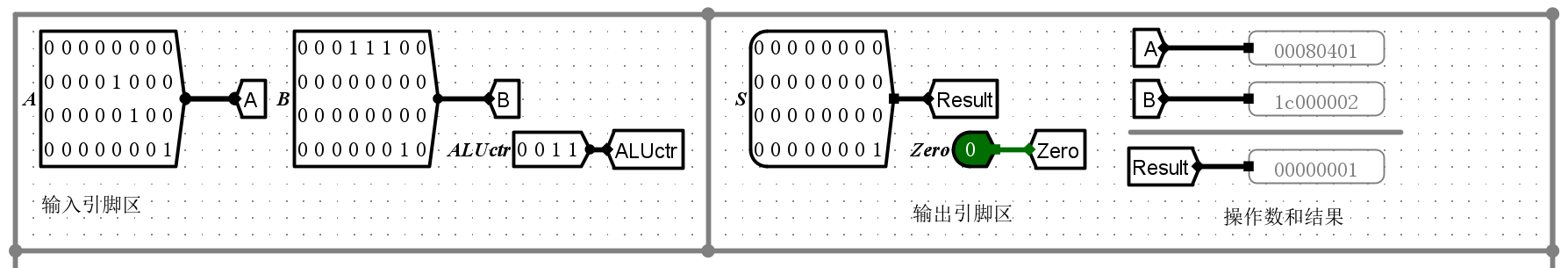


0010-带符号小于比较：

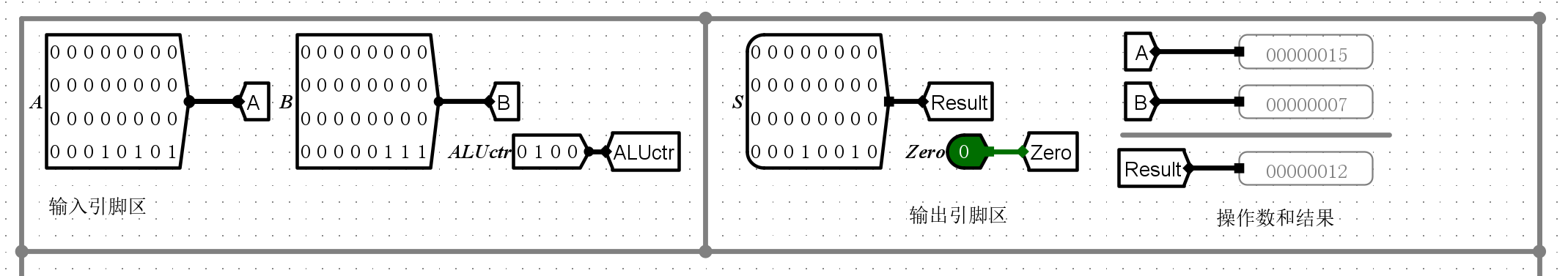
 正数：

 负数：

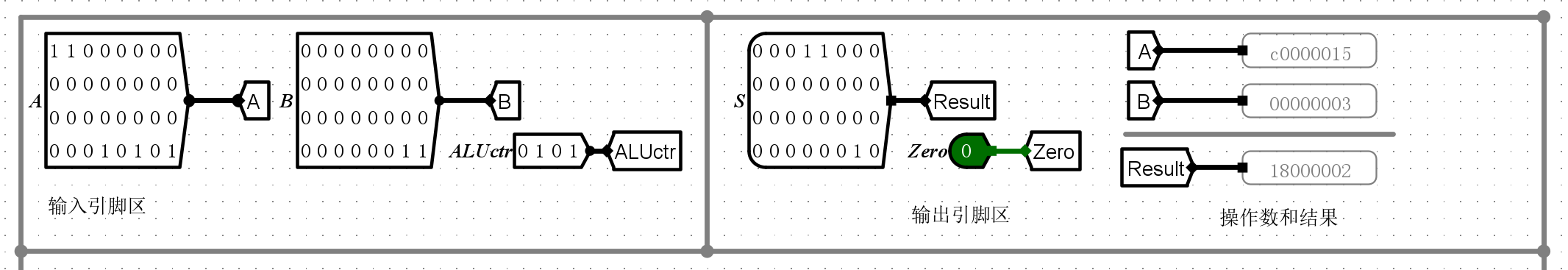
0011-无符号小于比较：



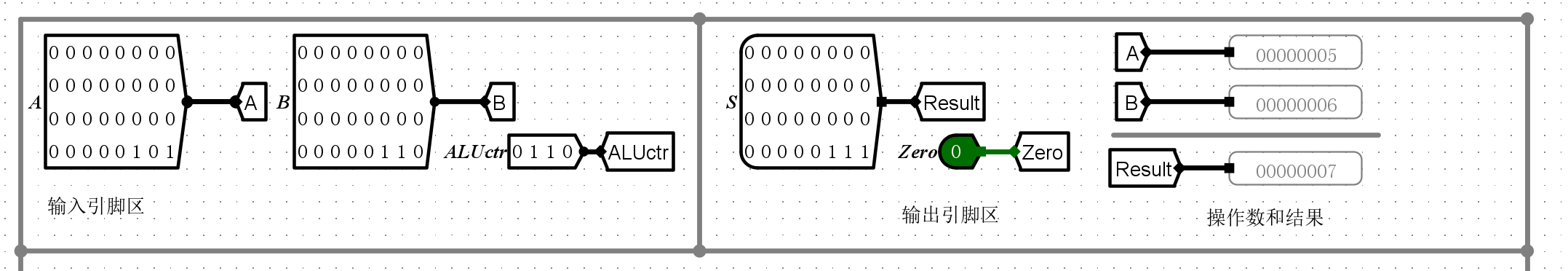
0100-（按位）异或：



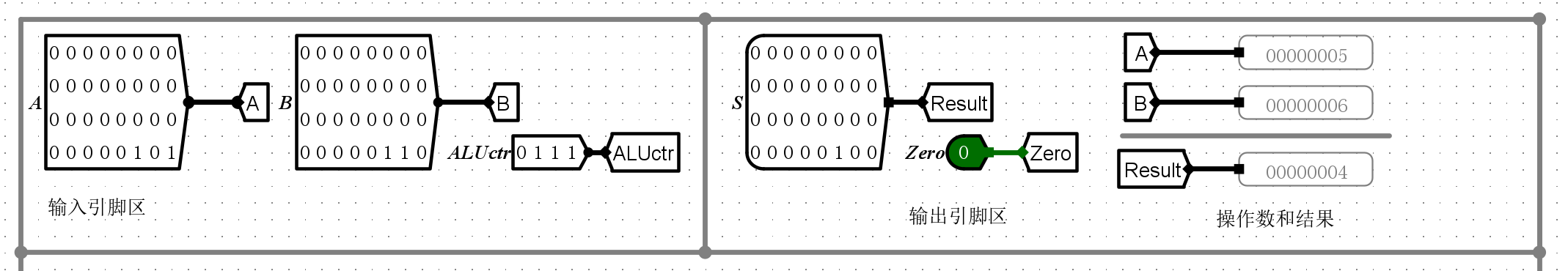
0101-逻辑右移：

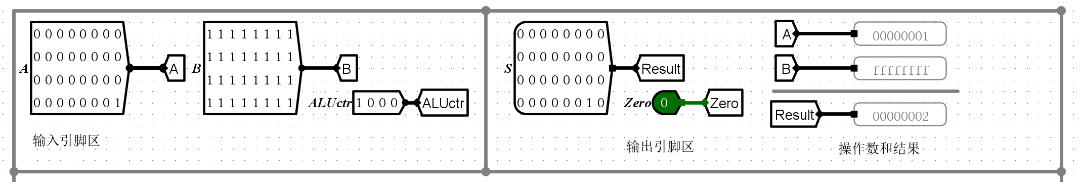


0110-（按位）或：

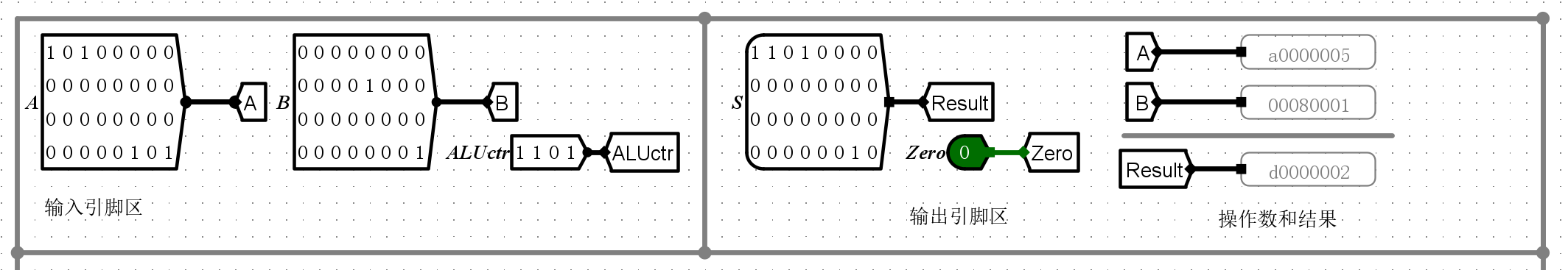


0111-（按位）与：

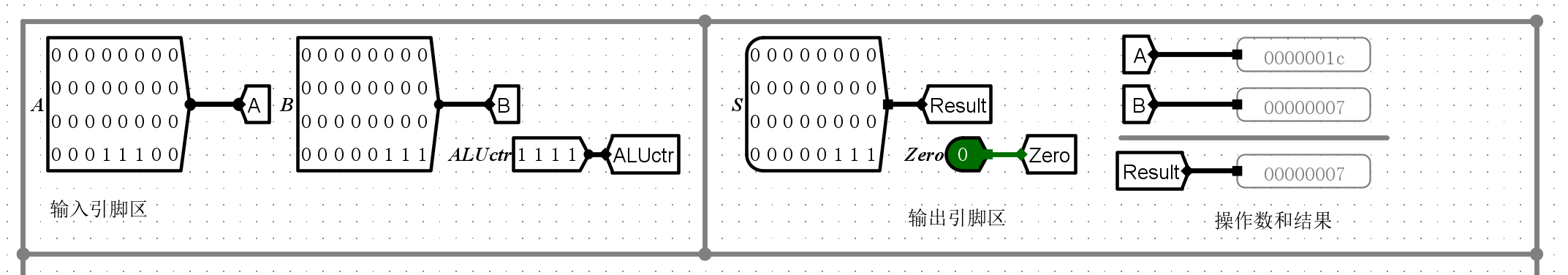


 1000-减法：

1101-算数右移：



1111-取数B：



**四、思考题**

**1. 利用设计的ALU，举例说明如何实现一个32位整数和一个常量的乘法运算。**

考虑使用移位和加减法的组合。首先将结果寄存器初始化为0。然后将常量表示为若干个2的幂次方的数b1,b1…bn的和，bi为2的k次方，就将目标数左移k位，然后加到结果上。或者，我们也可以把常量这样拆解：用比它大一点的2的幂次方的数，再减去差值。

举例来说，假设常量是13，那么把13拆解为8+4+1。先把目标数左移3位加到结果寄存器中；然后把目标数左移2位加到寄存器中；最后把目标数本身加到寄存器中，这样寄存器中就储存了这个数和13的乘积。

而如果常量是61，我们考虑使用64-2-1。先把目标数左移6位加到结果寄存器中，然后用结果寄存器中的数减去目标数左移1位的结果，再减去目标数本身，就得到了目标数和61的乘积。

\*另外，如果不是这样的常量乘法，而是对两个变量进行乘法，我们考虑使用booth算法，使用ALU的移位和加法功能即可，这需要搭配额外的乘积寄存器，而且需要对输入ALU的数据做进一步处理。

**2. 查阅资料，说说除了先行进位加法器以外还有哪些高速并行多位二进制数加法算法，至少简要说明一种算法的实现流程。**

布谷鸟加法器（Cuckoo-Filter-based Adder）是一种基于Cuckoo Filter的高速并行多位二进制数加法算法。它采用了哈希表的数据结构，能够在具有高并行性和高效率的同时，保持较低的硬件成本和低延迟。

具体来说，Cuckoo Filter中的每个位置都可以存储一个二进制数，布谷鸟加法器将每一个加数拆分成若干个二进制数，然后将它们插入到两个Cuckoo Filter中。如果某个位置已经存储了一个二进制数，那么新的二进制数就会顶替掉原来的二进制数，并被存储在另一个Cuckoo Filter中。（类似于布谷鸟巢间的飞跃）。

在所有的二进制数都插入到Cuckoo Filter中之后，布谷鸟加法器将这两个Cuckoo Filter逐个比较，并将相同位置的二进制数进行相加。由于Cuckoo Filter的哈希函数能够保证元素的唯一性，因此不会出现哈希冲突的情况。

最终，布谷鸟加法器可以在两个Cuckoo Filter之间进行多次哈希、查找和相加操作，从而实现高速并行多位二进制数加法。

**3. 简要分析加法运算、比较运算和移位运算三种运算中操作数在ALU中经过的逻辑门级数。**

首先考虑先行进位全加器的门级数。ALU部件具有3级门（P、G函数本身有1级），所以产生进位有3级门。全加器有两个异或门，但是第一个异或门延迟恰在刚才的3级门延迟完成（异或门视为3级门延迟），于是全加器只需要额外的3级门延迟。所以一个先行进位加法器4级逻辑门（6级门延迟）。

现在考虑16位和32位加法器。16位由4个4位和CLU构成，易知只需要增加2级门；32位由16位串行得到，所以门级数乘2，现有12级逻辑门（至于标志位，如果后续步骤不需要标志位，只需要S，则不需要考虑标志位的门延迟。标志位门延迟如下：CF：0（早于S），SF：0（等同于S），OF：2，ZF：5（32位门阵列））。

最后考虑ALU部分。

加法运算：操作数B需要和SUB信号异或，32位门阵列有1级门。8路多路选择器为6级门。故加法运算的门级数为19级门。

比较运算： 在加法运算的基础上，OF信号是2级门（SF是0级门），再异或，现额外需要3级门。2路选择器是2级门（CF是0级门，不会产生影响），0扩展器是0级门（不需要逻辑运算）。因此比较运算会比加法运算多5级门，合计24级门。

最后是移位运算：直接用32位桶形移位器的门级数加上多路选择器的6级门即可。其

中桶形移位器最大位移5位，我的实现采用了5个4路选择器，合计20级门。故移位运算

的门级数为26级门。