**实验 5：存储器及数据通路设计**

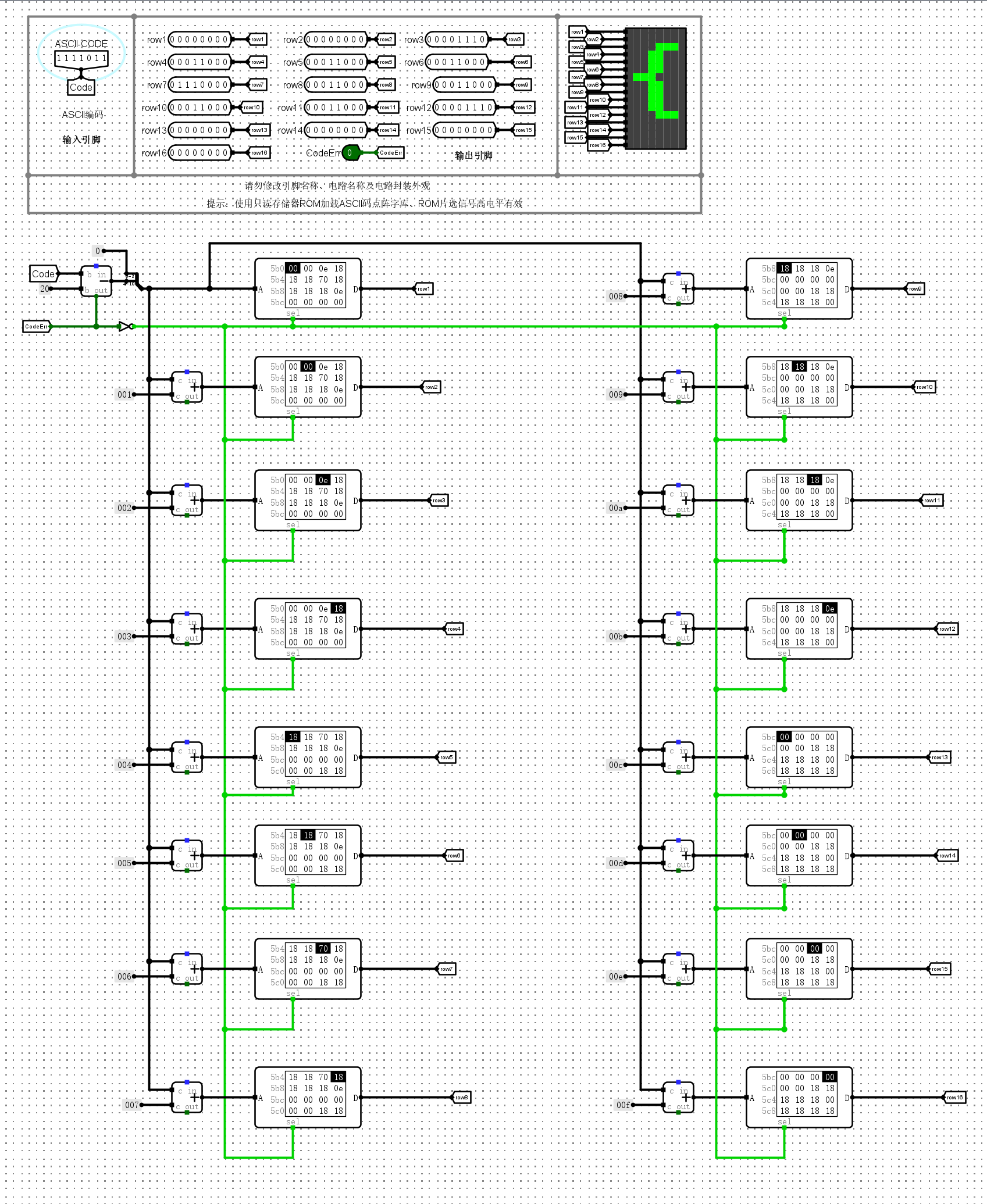
221275027 喻思文

**5.1 只读存储器实验**

（1）原理

根据hex文件排布，对于数据的一行，代表了一个字符的点阵数据。一行就是16个8bit的数据，分别对应了LED点阵的每一行数据，因此分别取出，电路实际上就是对地址不断移位。

1. Logisim电路图



1. 实验验证

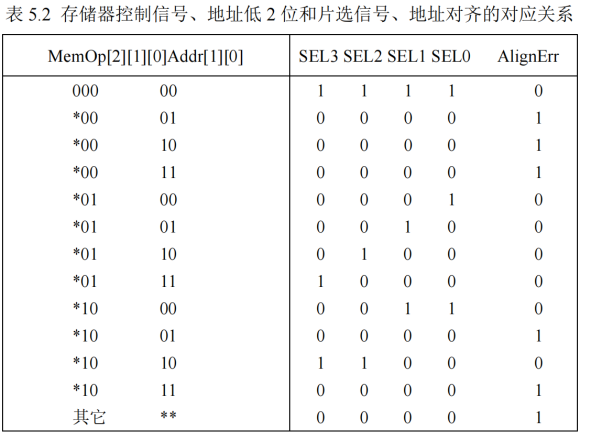
见上图和思考题1，已通过。’{‘的ascii值为7b成立。

**5.2 数据存储器实验**

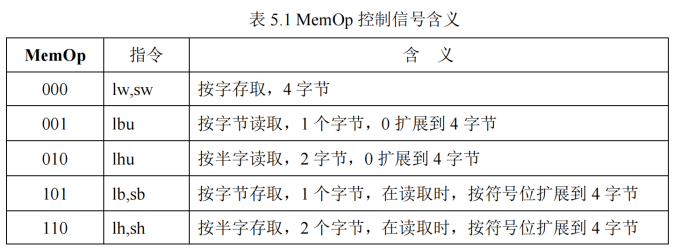
（1）原理

为了实现读和写，使用RAM。考虑到每个RAM存储8位，分成4个一起存储。

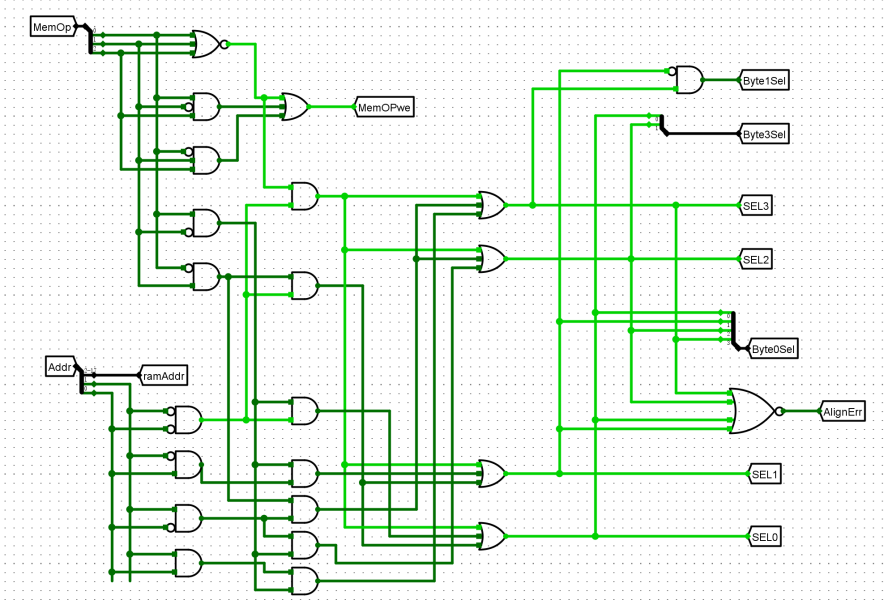
根据手册给的真值表，



再考虑到MemOp控制信号有下面含义，我们合并为三大类：按字存取、按半字存取和按字节存取，控制信号取000、\*10、\*01（上表也已经分类）

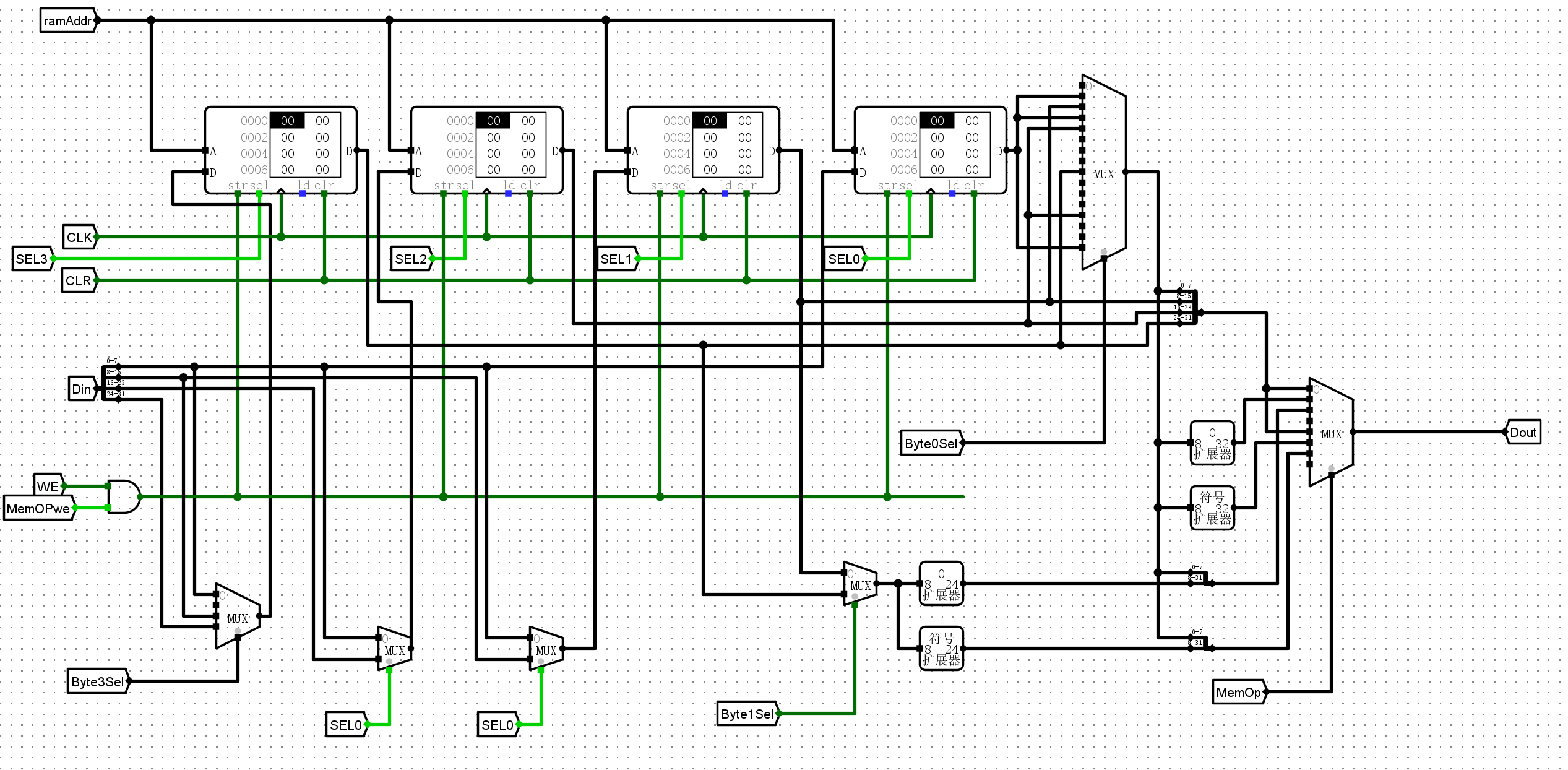


因此控制信号应该如下：



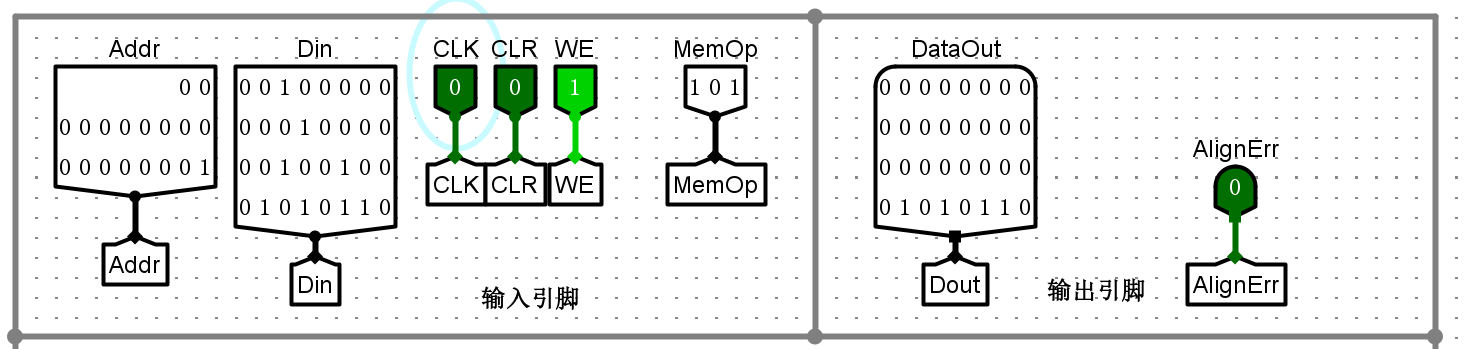
根据片选信号确定读取数据时每个字节的来源，例如当片选信号为 1111、0001、0011 时，最低字节的数据来自 RAM0 的数据输出。RAM的状态由上面的片选信号SEL和控制信号ByteSel共同控制。

1. Logisim电路图

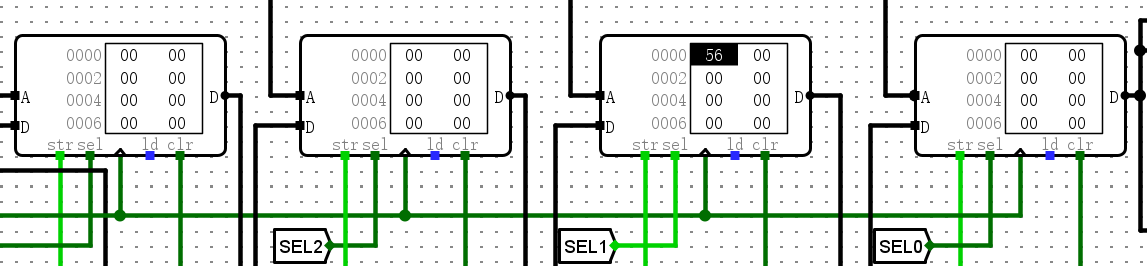


1. 实验验证

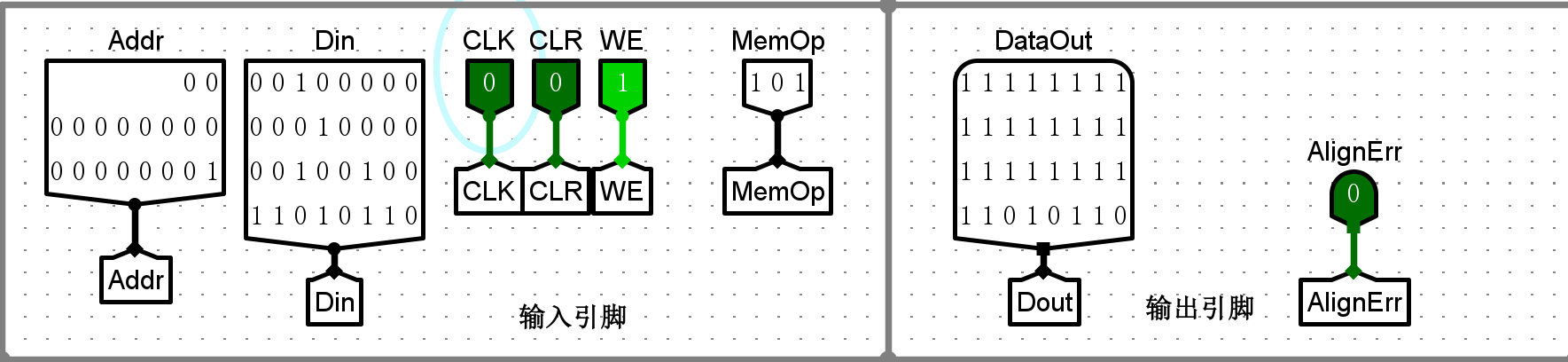
下面是按字节存取的一个实例：



可以看到，1号存储器读入低8位：



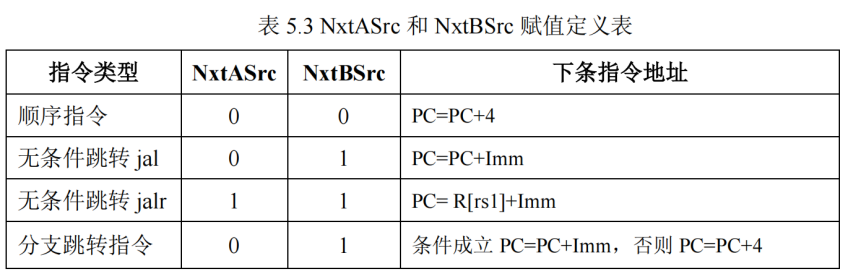
按符号拓展验证如下：



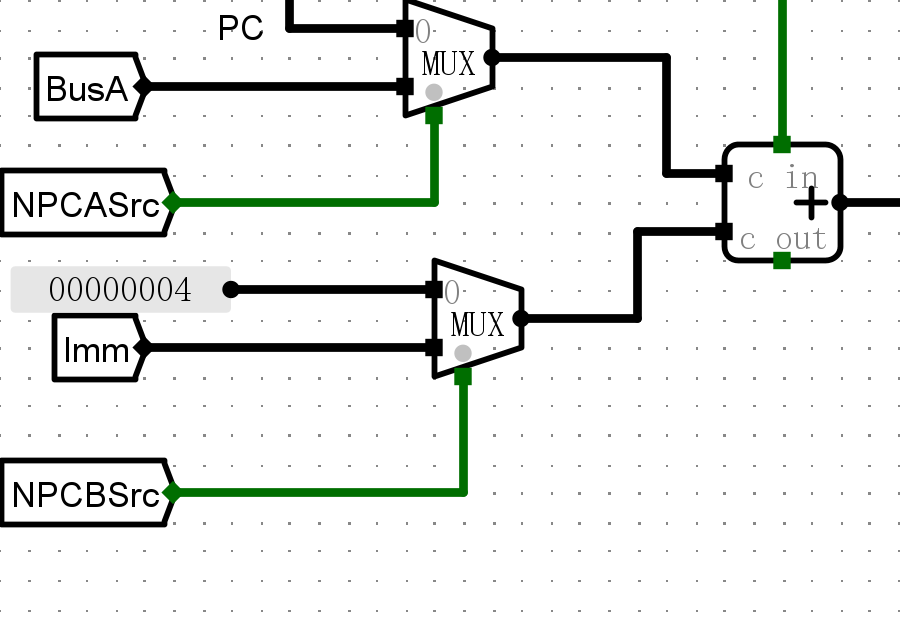
**5.3 取指令部件实验**

（1）原理

在下地址逻辑设计中可以使用专用加法器来进行计算下一条指令的地址，NxtASrc和 NxtBSrc控制信号分别表示专用加法器输入端A和B的输入数据。下面给出指令赋值定义：



下面是该部分的原理（提供的接口不太一样）：

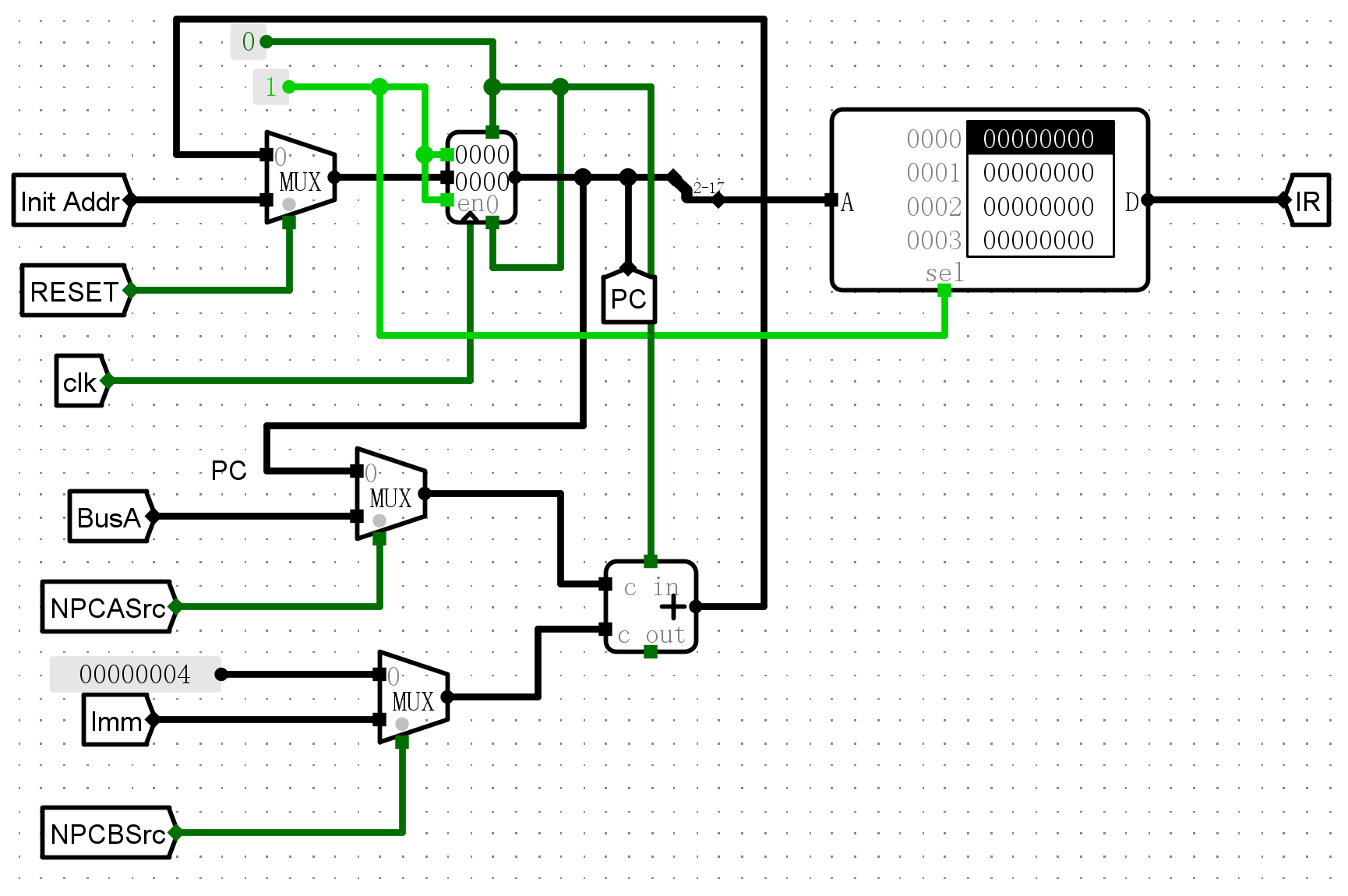


过程为：

·取出PC，并在指令存储器中根据PC取出指令。

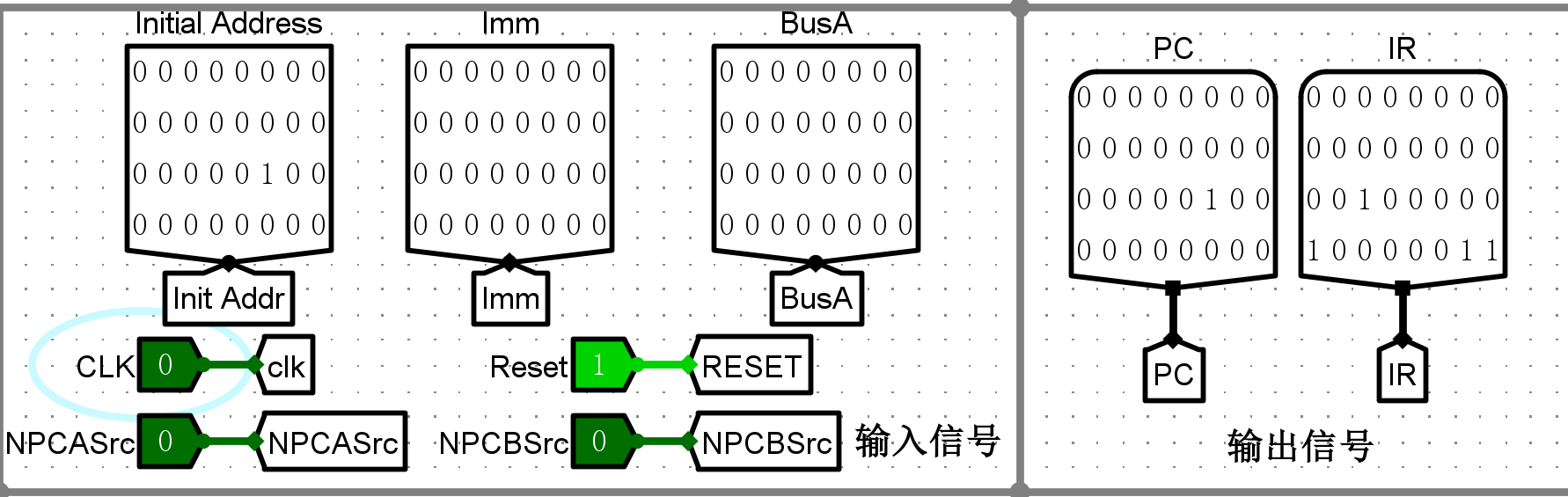
·根据NxtASrc和NxtBSrc计算下⼀个PC的值，作为PC寄存器的新输⼊。

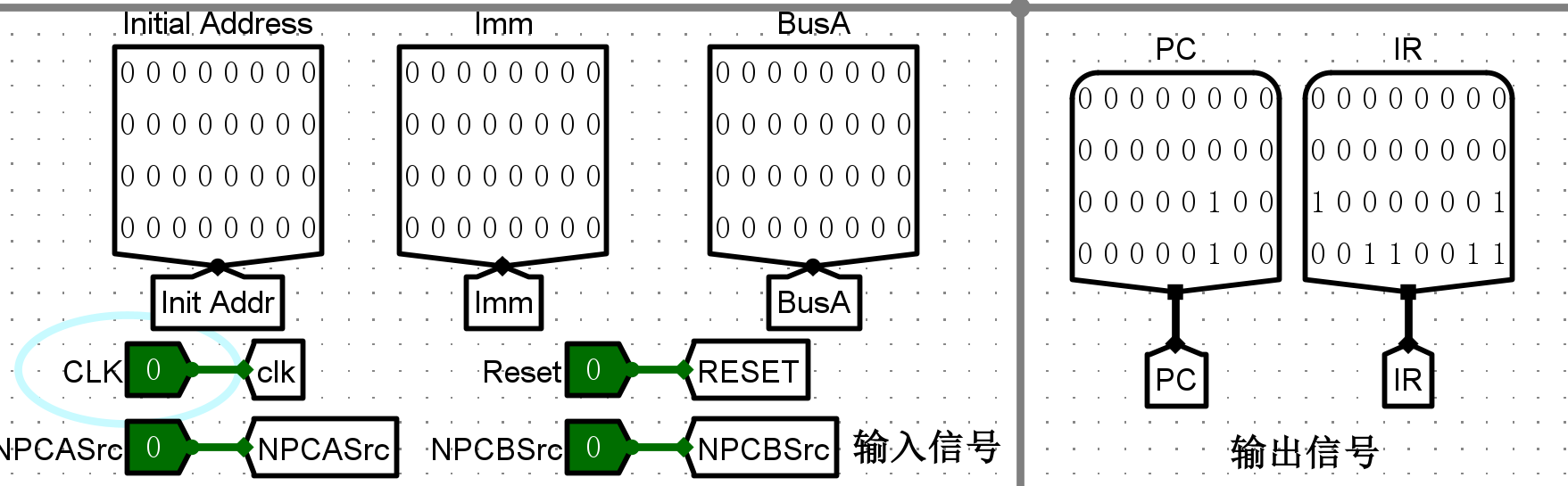
1. Logisim电路图



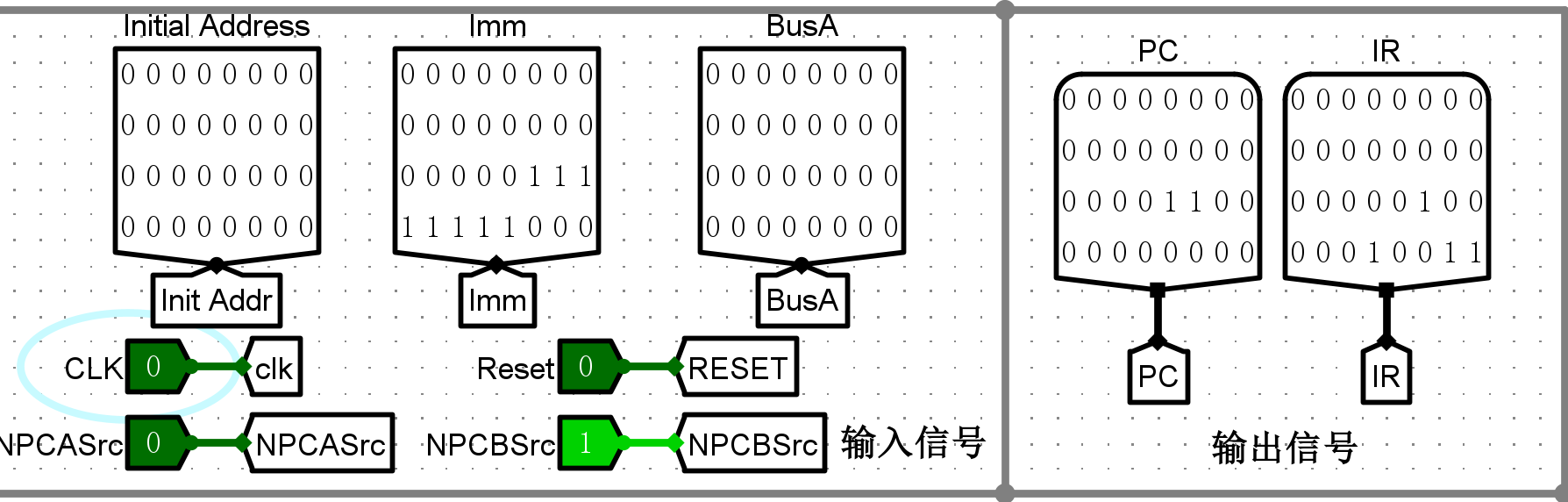
1. 实验验证

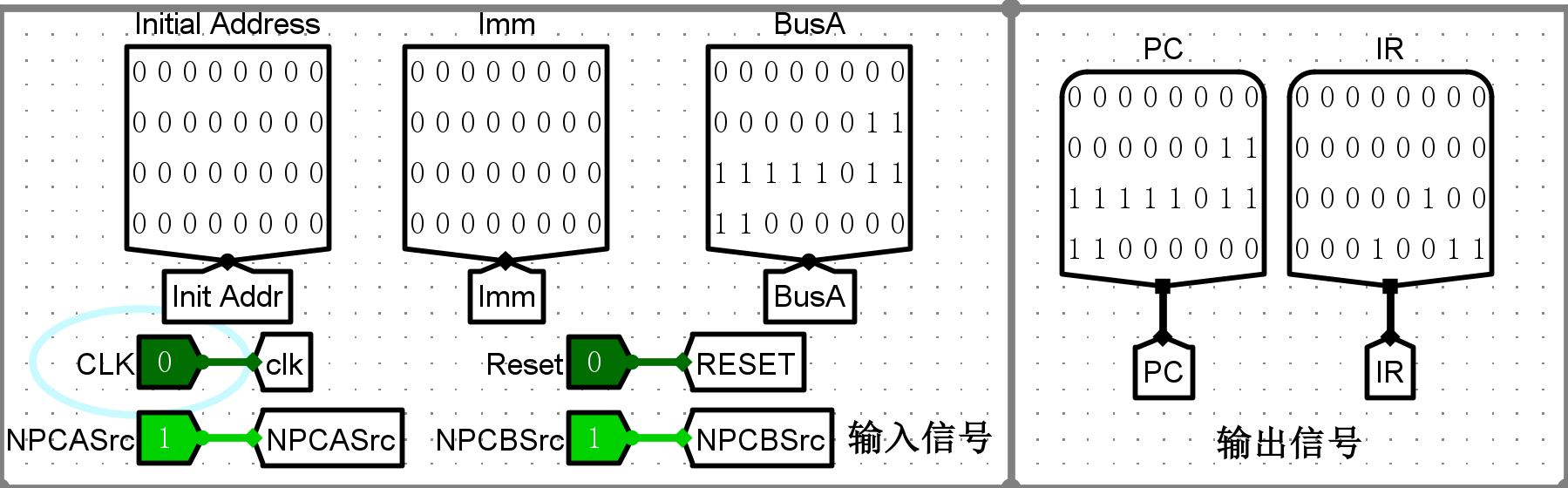
加载lab5.3.hex文件，连续执行手册中部分指令输出如下（存储器取指令位置见图右）：

 0x0100

 0x0101

再执行一次（手误） 0x0102

 0x0300

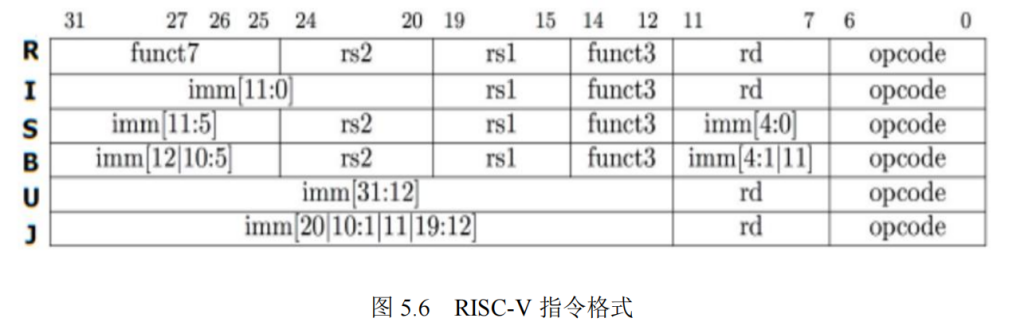
 0xfef0

可以看出，基本功能已经实现。

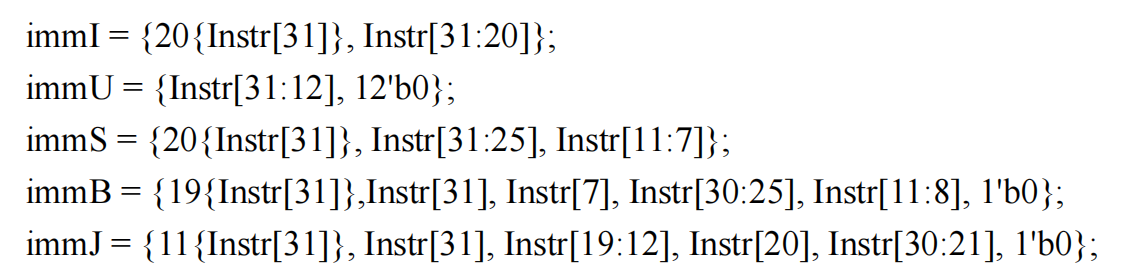
**5.4 取操作数部件IDU实验**

（1）原理

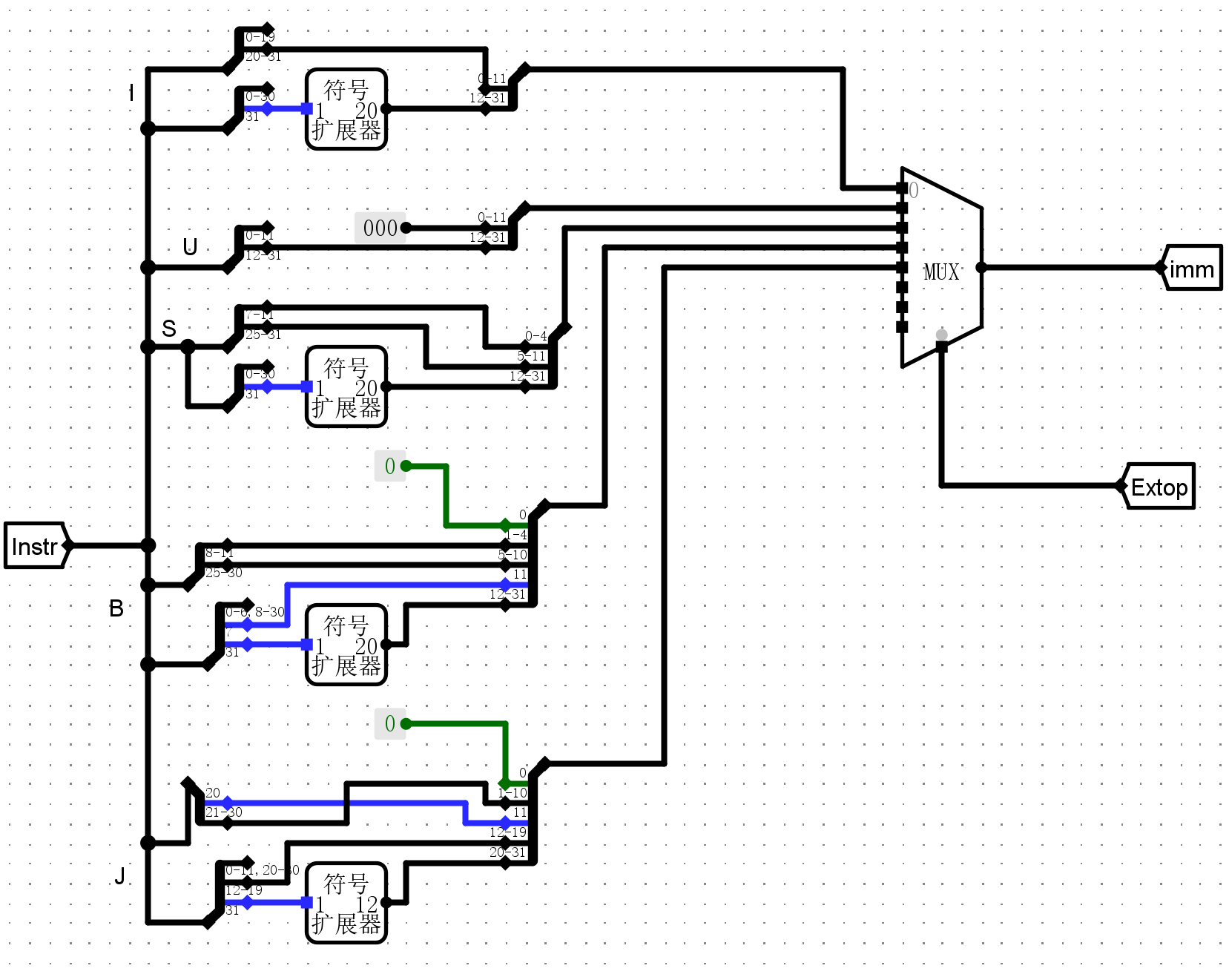
RISC-V指令体系如下：



根据规则取出imm，下面是拓展规则（B那块应该是20，J应该是12）：



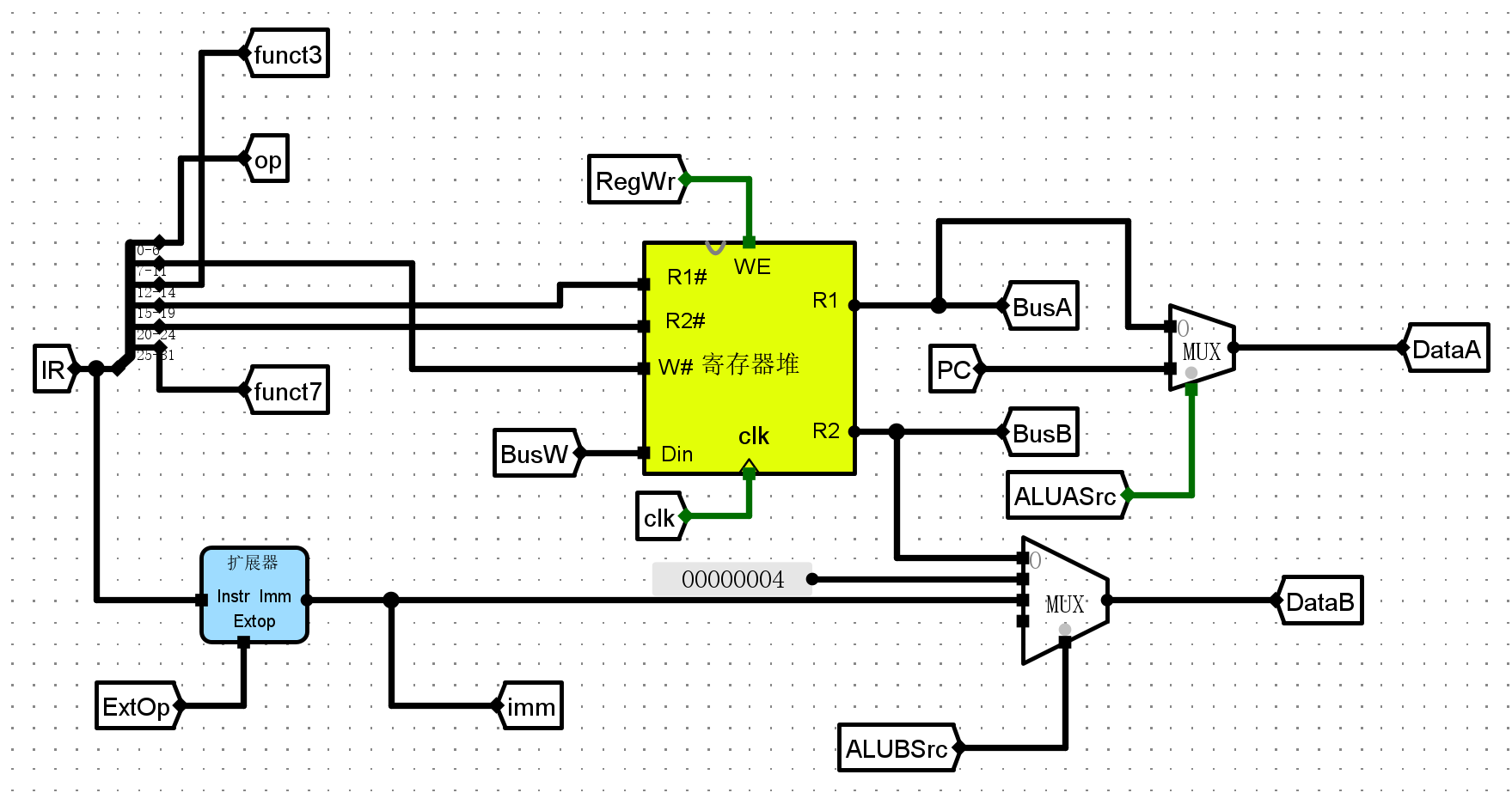
根据拓展规则执行立即数拓展原理如下：



寄存器堆注意使得0号寄存器始终为0。

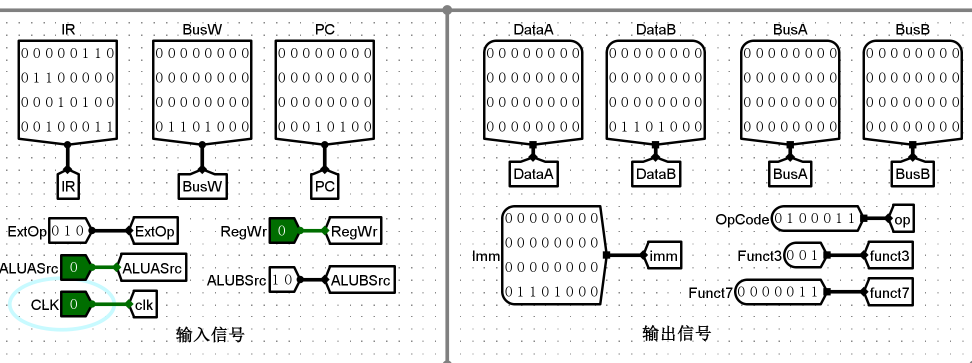
将输入指令寄存器IR按照上表RISC-V规则分解出 opcode、rd、funct3、rs1、rs2和funct7等字段，并通过立即数扩展器得到32位的立即数，根据rs1和rs2的读取寄存器堆中相应编号寄存器中的数据，输出到BusA和BusB两个端口。DataA和DataB行为与5.3上的表一致。

1. Logisim电路图

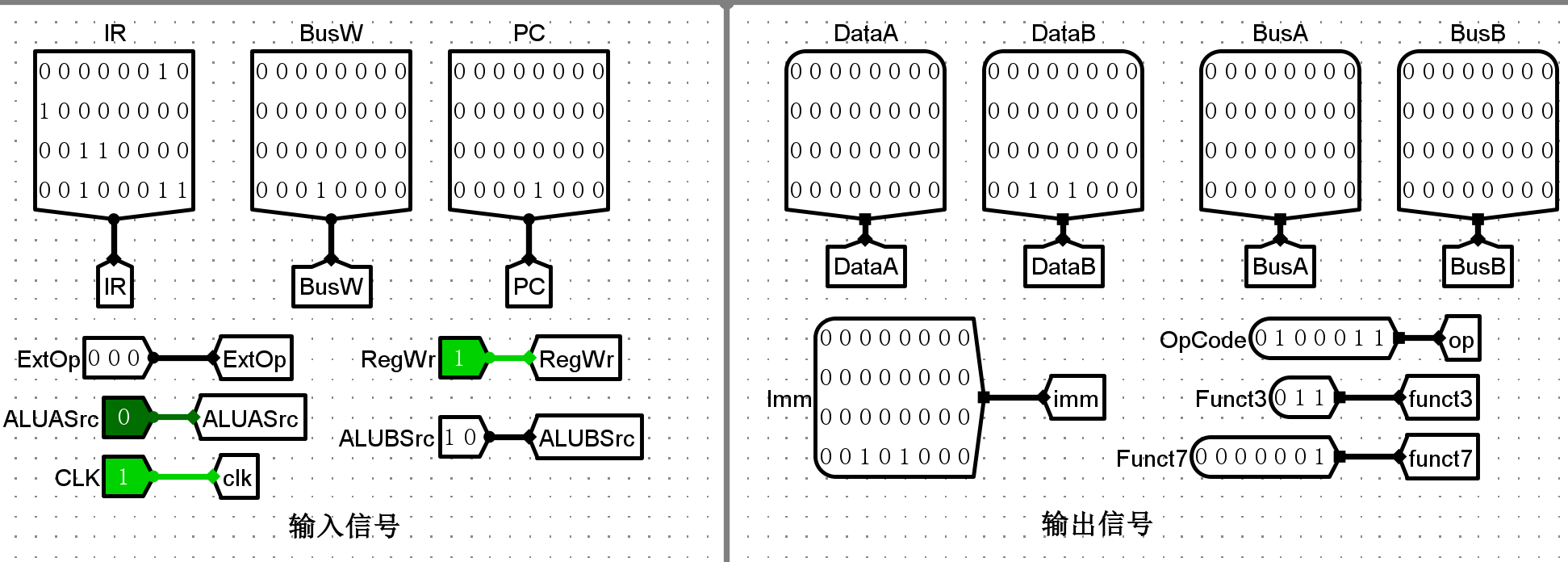


1. 实验验证

用手册表中部分测试点验证。PC=14时，结果如下：



PC=8的点结果如下：

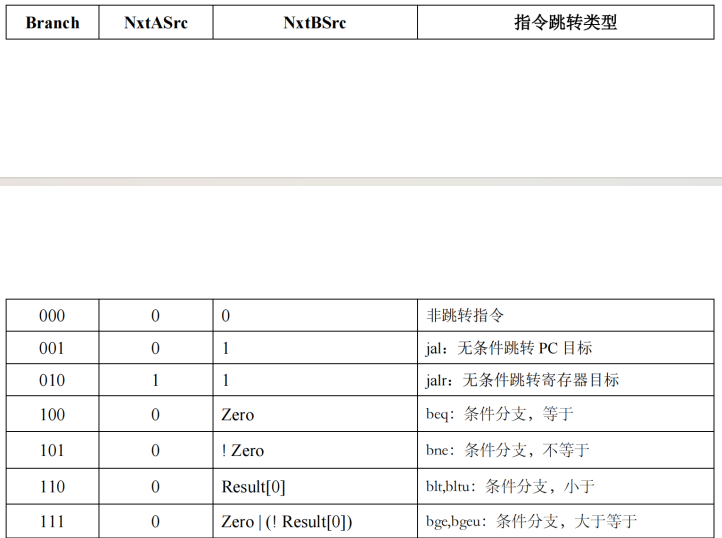


可以看出结果符合预期。

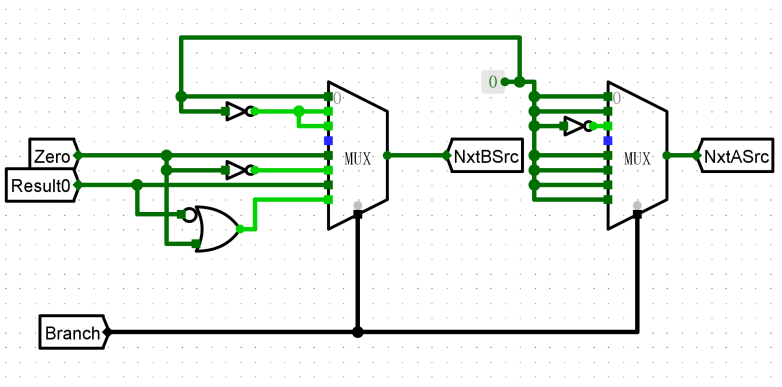
**5.5 数据通路实验**

（1）原理

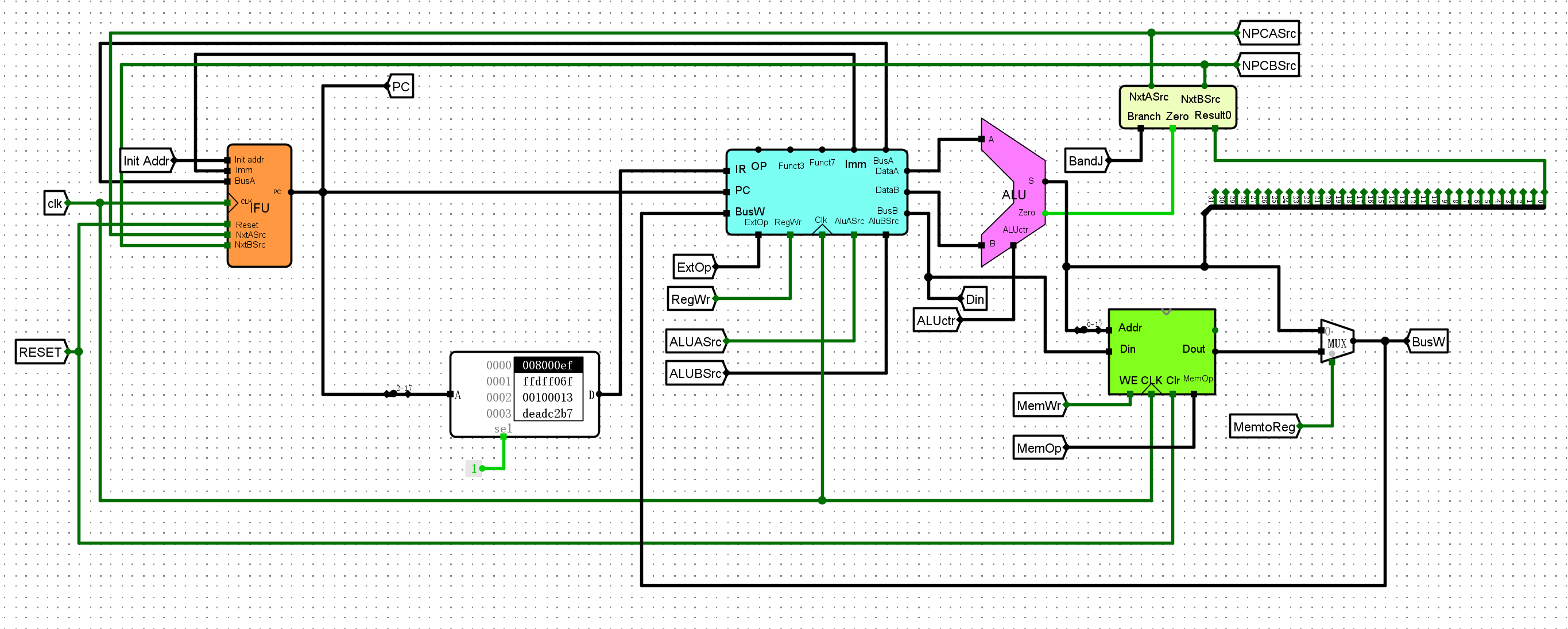
基本上就是加载前面涉及到的部件，注意将指令存储器移出IFU子电路，单独部署在数据通路中。自己要增加的是跳转控制器。跳转控制器根据控制信号Branch和ALU输出的Zero及Result[0]信号来决定NxtASrc和NxtBSrc，其中控制信号Branch的定义来自于跳转指令，真值表如下：



因此，该部分原理图如下：



1. Logisim电路图



1. 实验验证

最终验证测试使用[线下验收表](实验五 线下验收表.pdf)，如下：

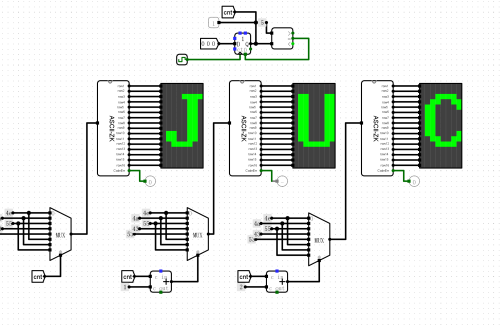
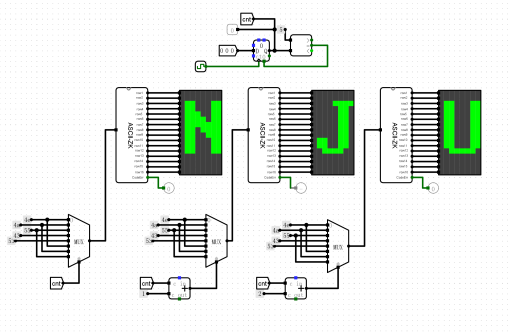


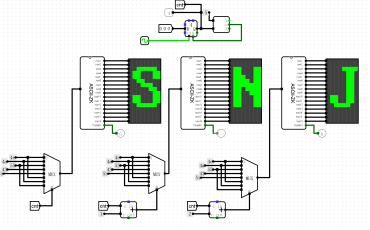
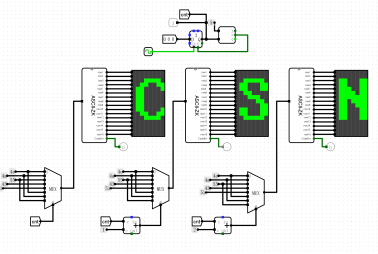
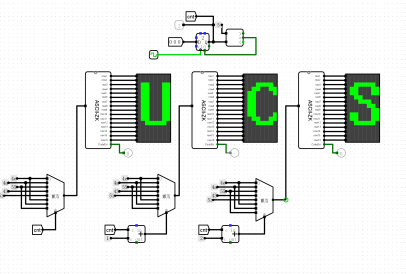
最终输出信号值验收通过。

**5.6 思考题**

1. 如何利用ROM实验实现滚动显示的功能，在3个LED点阵矩阵中，左右滚动显示5个ASCII 字符，如“NJUCS”。

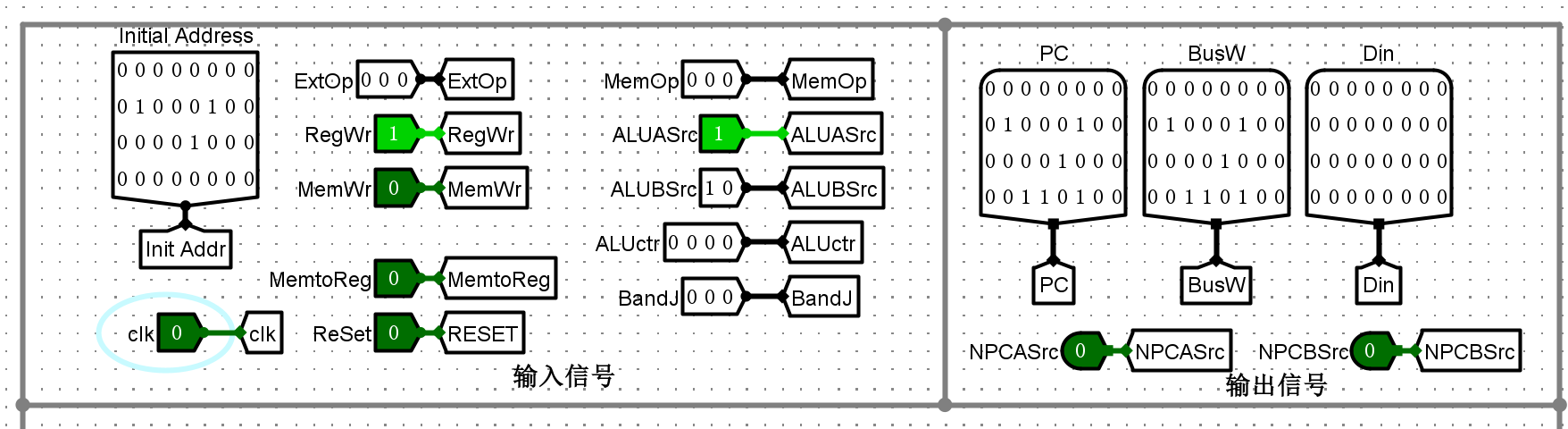
循环计数，对3个点阵，出现字母顺序分别为NJUCS, JUCSN, UCSNJ，相当于左移，这样后两位多路选择器控制端对cnt+1和+2，输出字母用5.1的main封装电路。下面演示，具体电路见lab5.1.circ中的Marquee电路。

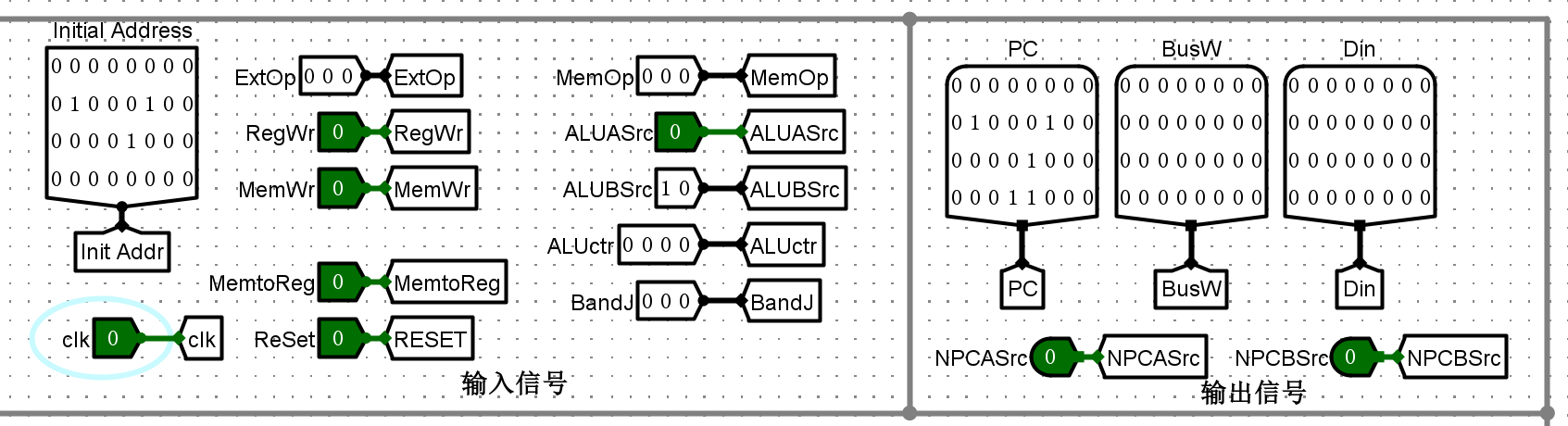


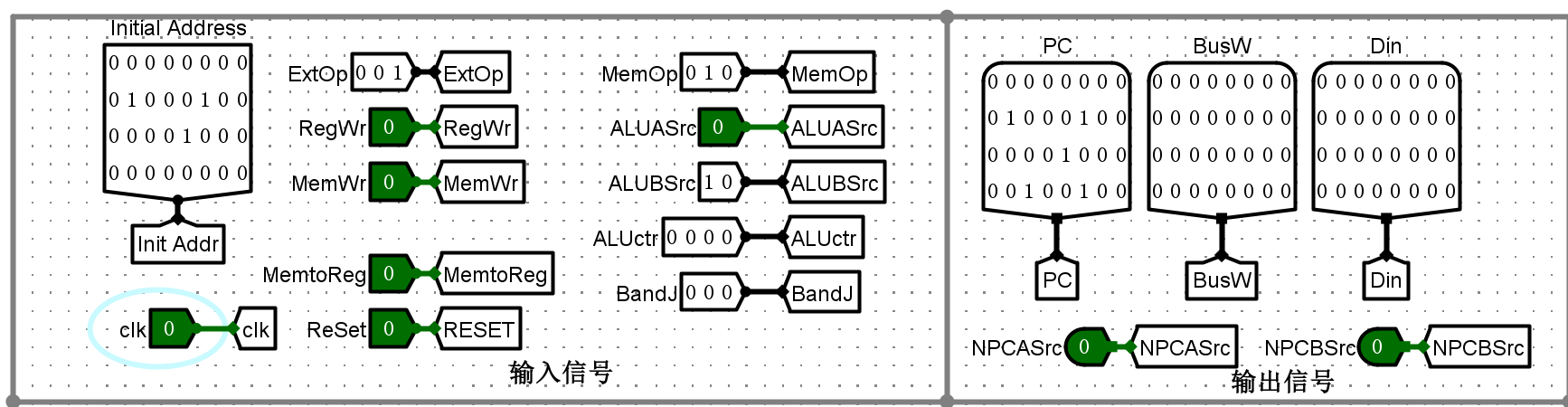


1. 分析说明如果寄存器堆写入数据时是下降沿触发有效，而PC寄存器和数据存储器写入时是上升沿触发有效，则对程序执行结果有什么影响？

自己简单实验发现在时钟变化内，给予初始地址，试了几个指令，但是只有PC=PC+4，而BusW和Din始终为0（当然，也出现了BusW=PC的情况）







理论分析如下：会导致导致时序上的不匹配

· 数据不一致：例如，如果一个指令需要在同一个时钟周期内读取和写入同一个寄存器，那么写入操作可能不会看到读取操作的结果，因为它们发生在时钟周期的不同阶段。

· 指令执行错误：在某些情况下，这种时序不匹配可能导致指令地址指向混乱。例如，如果一个指令需要在写入数据到寄存器堆之前先更新PC寄存器，那么由于PC寄存器在上升沿更新，它可能不会指向正确的下一条指令，导致程序流程出错。

· 寄存器堆、PC和数据存储器写入时序：依赖寄存器堆数据的功能可能在时钟周期的前半部分就需要这些值，依赖PC或者数据存储器的可能在时钟周期的后半部分就需要值。

· 潜在的竞态条件：不同的写入时序可能导致竞态条件，即指令的执行结果可能取决于时钟周期的特定时序，这行为不可预测。

1. 在 CPU 启动执行后，如何实现在当前程序结束后，CPU不再继续执行指令？
2. 程序计数器：CPU总是按照PC的指向对指令序列进行取指、译码和执行，若修改PC值，指向结束的位置或指令之外的地址，从而终止CPU运行。
3. 异常或中断：在发生异常情况（比如除以0或者非法指令）下，触发异常条件，导致CPU中断运行，操作系统会根据中断号找到对应的处理程序并执行。处理程序会进行必要的处理，比如I/O操作、错误处理等。
4. 直接调用：程序加显式指令直接停止运行，如RISC-V架构中的"ecall"指令结束程序。（实验6有所体现，opcode=73给halt指令）