**实验 5：加法器和 ALU 设计**

姓名: 学号:

**一、实验目的**

1.理解存储器RAM和ROM的读写方法，掌握支持RV32I存取指令的数据存储器设计方法。

2.理解处理器读取指令的过程，掌握RV32I下取指令部件设计方法。

3.理解 RV32I运算指令功能，掌握指令译码、取操作数、运算、访存等执行不同阶段的实现方法。

4.理解 RV32I每条目标指令的功能和对应数据通路的关系，掌握单周期数据通路的设计方法。

**二、实验环境**

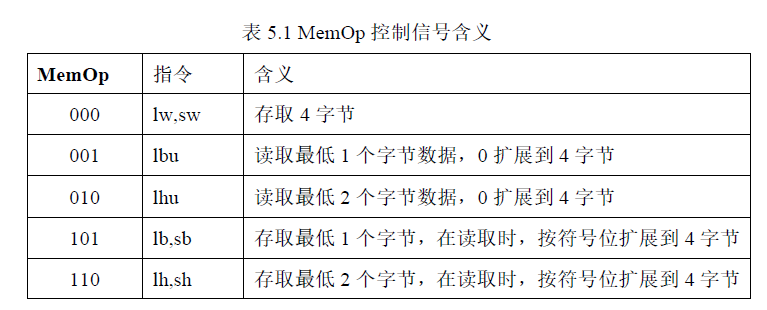
Logisim：https://github.com/Logisim-Ita/Logisim

RISC-V模拟器工具RARS：https://github.com/thethirdone/rars

**三、实验内容**

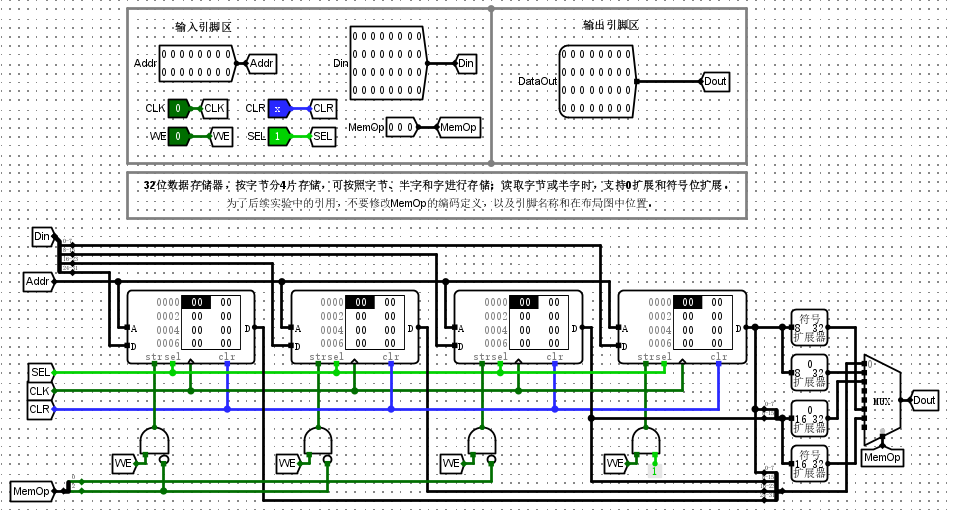
**1. 存储器读写实验**

在RV32I指令集中，访存指令要求数据存储器支持按照字节（byte）、半字（halfword）和字（word）的不同字节长度来进行数据读写，因此需要增加一个额外的控制信号MemOp来表示当前指令读写数据的字节长度，如表5.1所示定义了一种MemOp的编码方法。

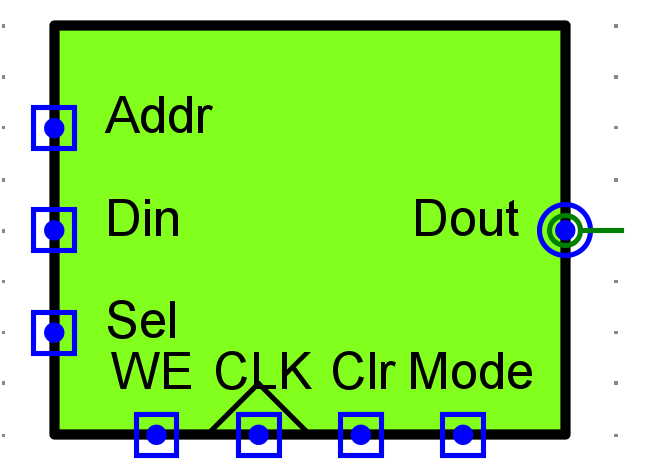


实验步骤如下：

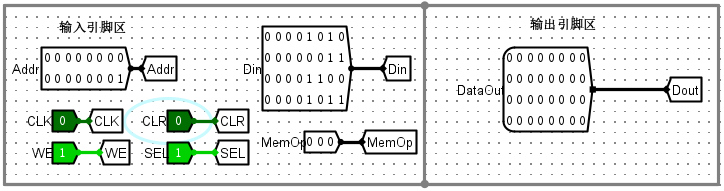
1）数据存储器实验。在Logisim中导航区中添加“数据存储器实验”的子电路，双击该子电路，在工作区中按图5.3所示的组件布局图放置输入输出引脚、RAM、隧道等组件。修改RAM属性，设置数据字长为8位，地址宽度为16位，数据接口模式设置为分离加载和存储端口模式，片选信号（Sel Acitve On）为高电平有效。设计电路如下：

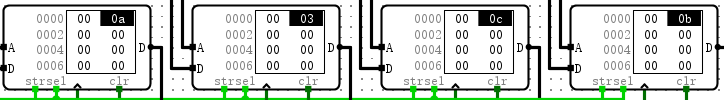


电路封装如下：

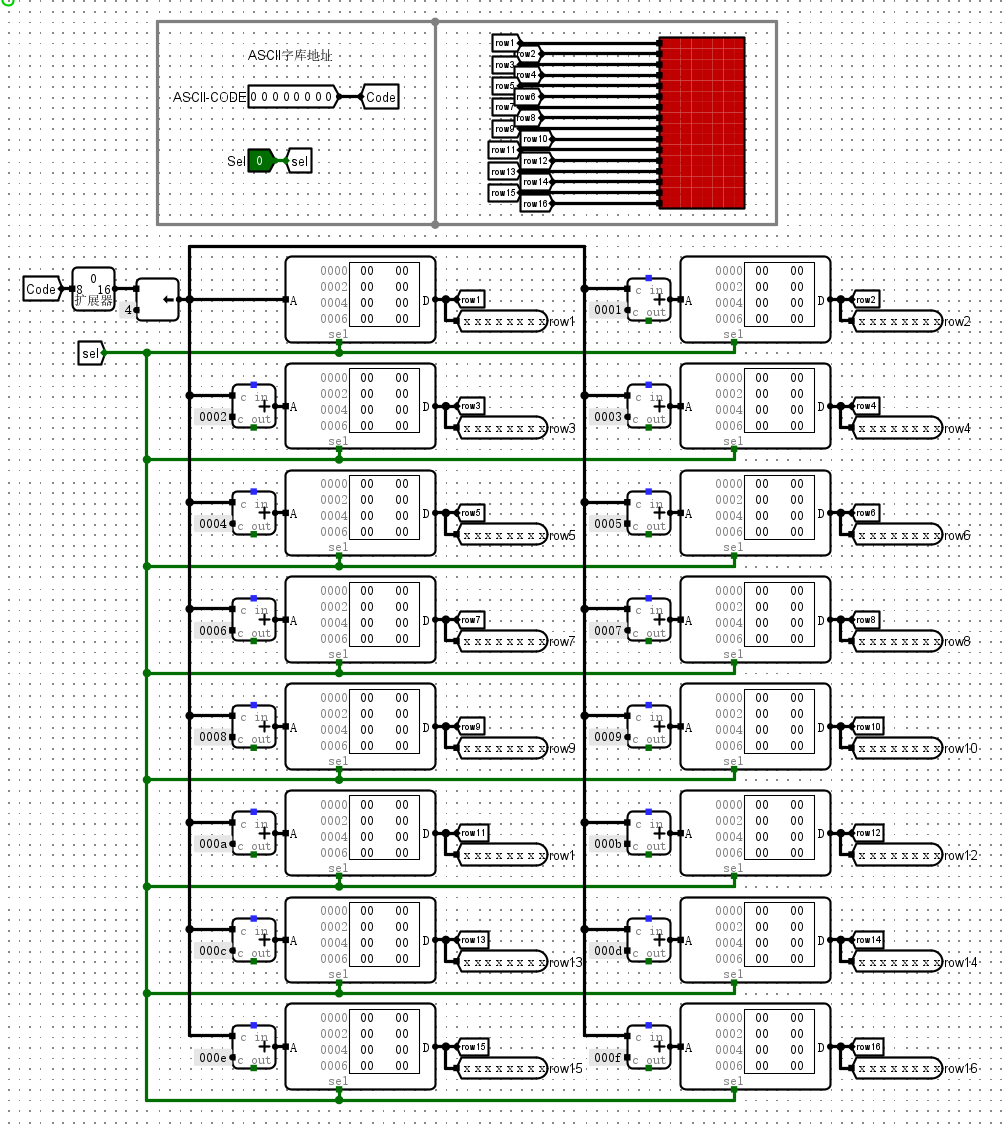


仿真实验：

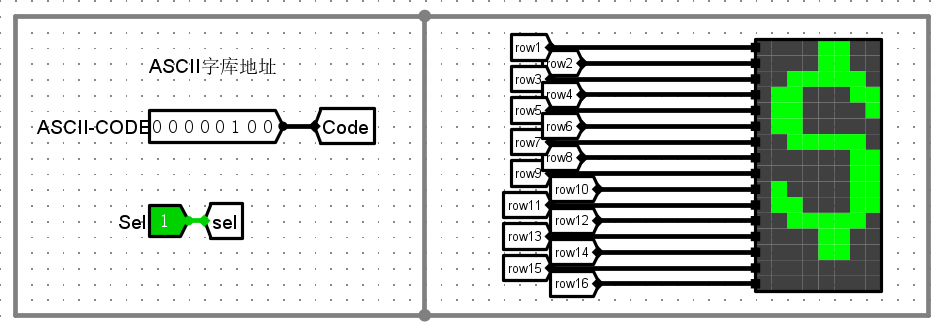




2）ASCII码显示实验。点阵字库文件按照ASCII码可显示字符顺序排列，从空格SP开始，到DEL结束共96个字符，每个字符使用8列16行的点阵表示字形。输入某个ASCII码的编码，在LED点阵组件上显示该字符形状。定义字长为8位ROM，地址宽度为16位，片选信号（Sel Acitve On）为高电平有效，加载ASCII码可见字符8\*16点阵字库文件ascii8-16.zk。在Logisim中导航区中添加“ASCII码显示实验”的子电路，双击该子电路，在工作区中放置输入输出引脚、ROM、隧道等组件。设计电路原理图如下：



仿真实验：



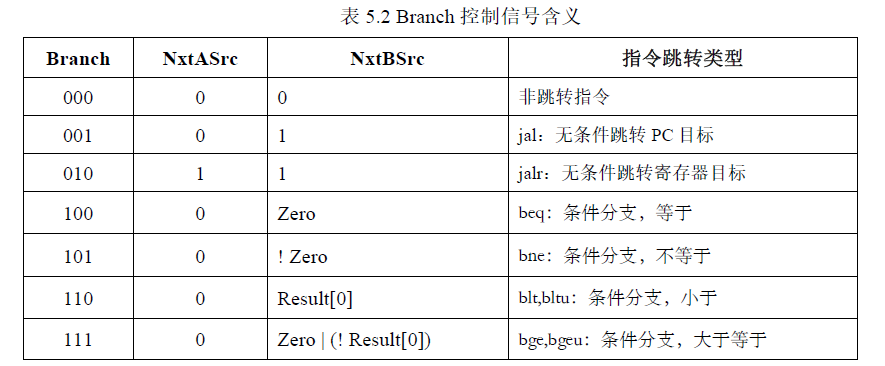
**2. 取指令部件IFU实验**

取指令部件就是处理器将指令从指令存储器由程序计数器PC值指定地址中读取出来的过程。取指令部件示意图如下：



RISC-V32I体系架构按字节编址，指令采用32位定长指令字格式，每条指令用4个字节表示。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，利用了跳转控制部件Branch生成加法器输入端的选择信号。NxtASrc控制PC加法器输入端A的信号，为0时选择PC寄存器的值，为1时选择Rs1寄存器值BusA。NxtBSrc 控制PC加法器输入端B的信号，为0时选择常量4，为1时选择立即数Imm。

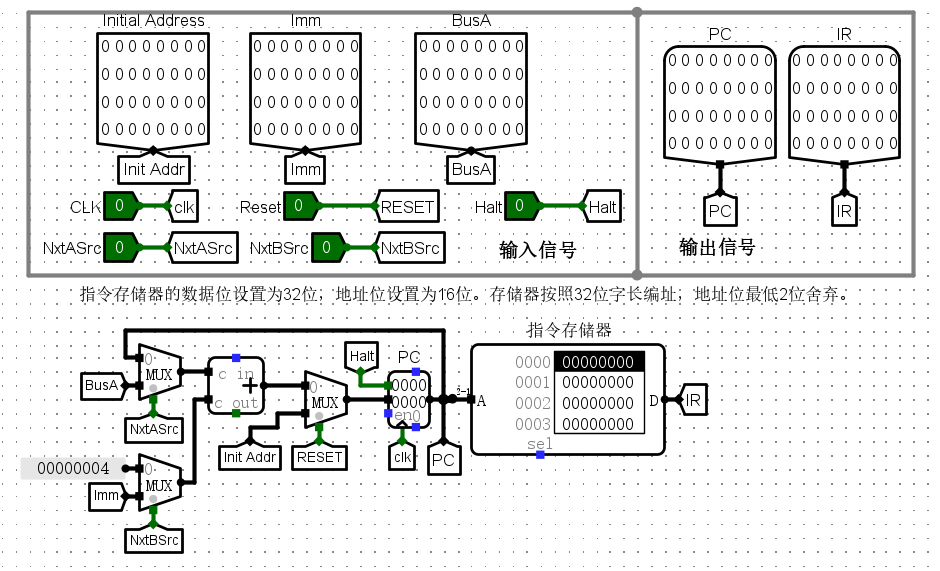
跳转控制模块根据控制信号Branch和ALU输出的Zero及Result[0]信号来决定NxtASrc和NxtBSrc，其中控制信号Branch的定义来自于跳转指令，编码定义如下：



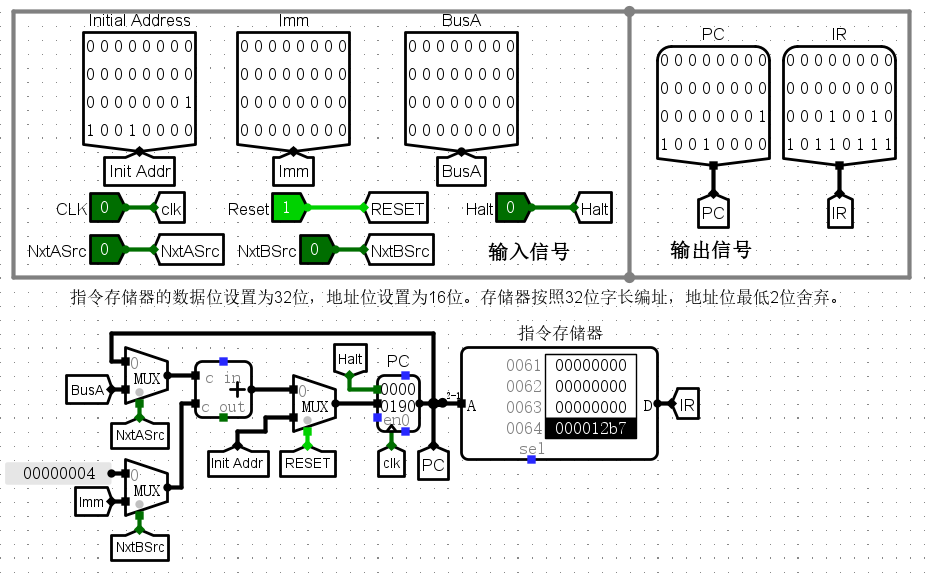
取指令部件中包括初始可执行程序地址Init Address，当Reset复位信号有效时，从该地址开始执行。Halt信号有效时，PC寄存器暂停输出指令地址。Imm输入表示程序跳转的偏移量，BusA表示寄存器Rs1中的数据。

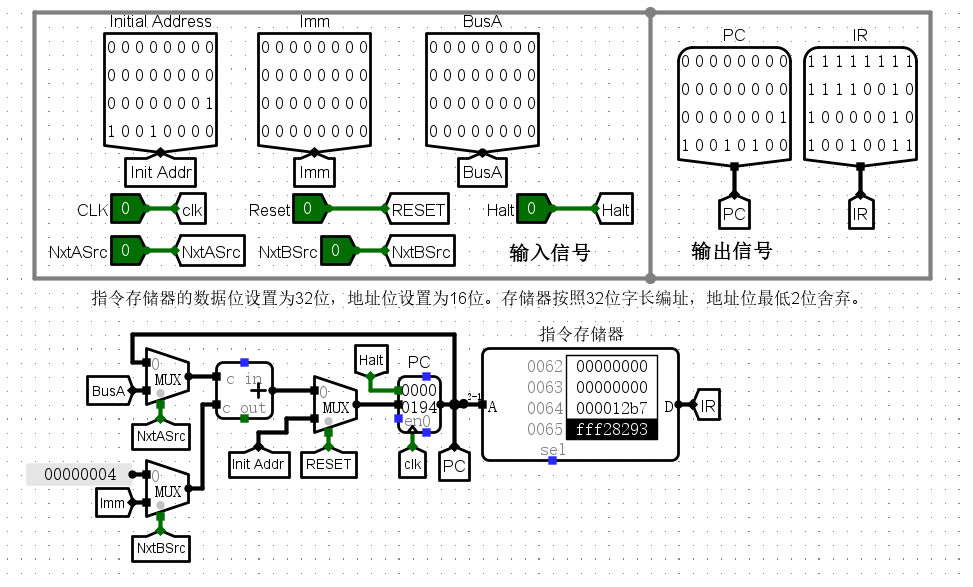
实验步骤如下：

1）在Logisim中导航区中添加“IFU实验”的子电路，双击该子电路，在工作区中按图5.8所示的组件引脚布局图放置输入输出引脚、指令存储器ROM、隧道等组件。修改ROM属性，设置数据位宽为32位，地址宽度为16位，片选信号（Sel Acitve On）为高电平有效。按照上述要求设计电路原理图如下：

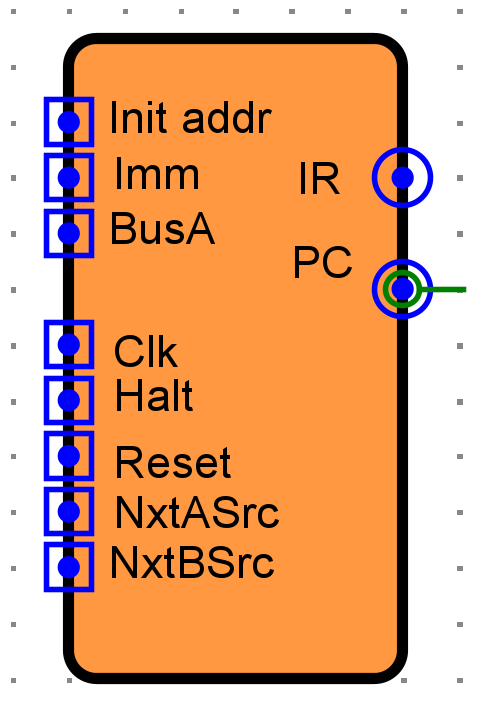


2）在ROM存储器存储单元0064处，写入：000012b7、fff28293、05de1863、288000ef、00112623等数据，设置初始地址数据，进行仿真实验：





3）编辑电路封装如下：

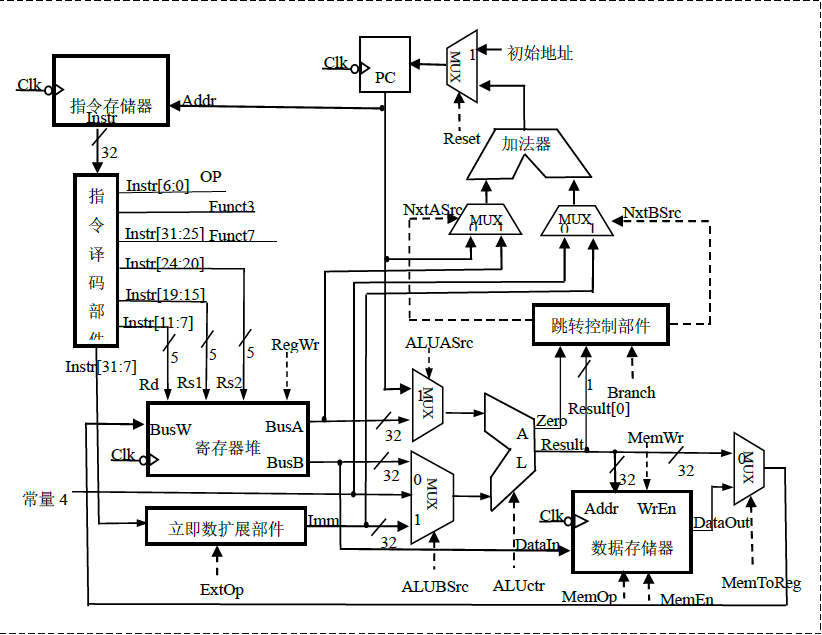


\*这里注意IFU内含指令寄存器以及IR引脚，应用于后续实验时可无视IR输出口

**3.数据通路实验**

数据通路是具体完成数据存取、运算的部件。单周期CPU的数据通道是指获取到指令之后，根据指令内容，读取操作数，进行操作，得到结果并写回的过程。不同类型的指令，数据传输过程并不一致。大致可分为取操作数IDU、执行EX、访存M和写回WB等阶段。

综合考虑上述不同类型的指令数据通路的结构，可得到如下所示的完整单周期数据通路：



图中所有加下划线的都是控制信号名，控制信号线用虚线表示。指令执行结果总是在下个时钟到来时开始保存在寄存器、数据存储器或PC中。

RISC-V指令格式如下图所示。其中，opcode为操作码字段，funct3和funct7为功能码字段，imm为立即数字段，rs1和rs2为源操作数寄存器编号，rd为目的寄存器编号。



除了R型指令外，其它5中指令都带有立即数，立即数是ALU一个数据输入源。这5种指令格式中的立即数编码方式各不相同，立即数扩展器需要根据指令生成正确的立即数。5种指令的立即数扩展格式如下：

immI = {20{Instr[31]}, Instr[31:20]};

immU = {Instr[31:12], 12'b0};

immS = {20{Instr[31]}, Instr[31:25], Instr[11:7]};

immB = {19{Instr[31]}, Instr[7], Instr[30:25], Instr[11:8], 1'b0};

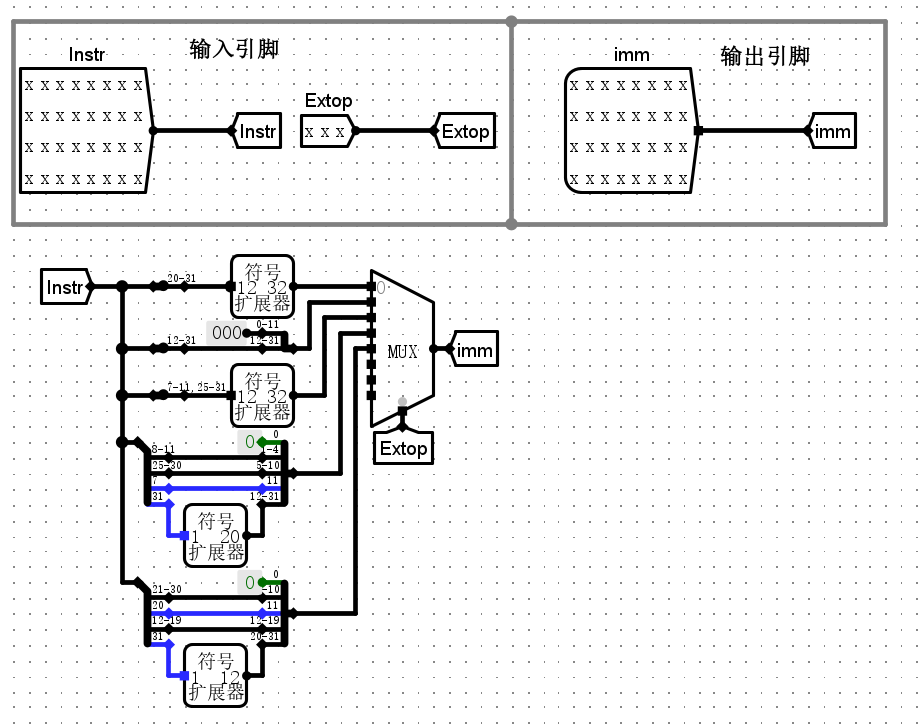
immJ = {11{Instr[31]}, Instr[19:12], Instr[20], Instr[30:21], 1'b0};

其设计示意图如下所示，通过控制信号ExtOp来选择不同立即数编码类型以及在扩展器中进行的扩展操作：

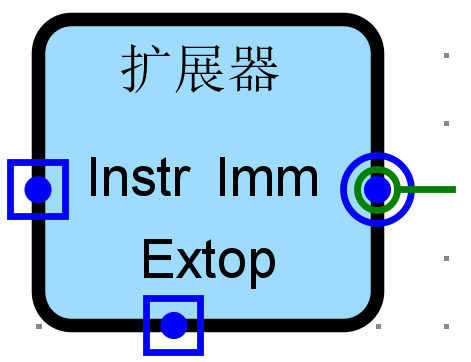


实验步骤如下：

1）设计“立即数扩展器”子电路。在Logisim中添加一个名为“立即数扩展器”的子电路，双击该子电路名称，在右侧工作区中构建相应电路。在工作区中添加指令输入引脚、扩展器、分线器、多路选择器、输出引脚和隧道等组件；修改组件属性，进行线路连接，多路选择器的控制信号ExtOp为0、1、2、3、4时，分别进行I-型、U-型、S-型、B-型、J-型指令的立即数扩展。设计电路如下：

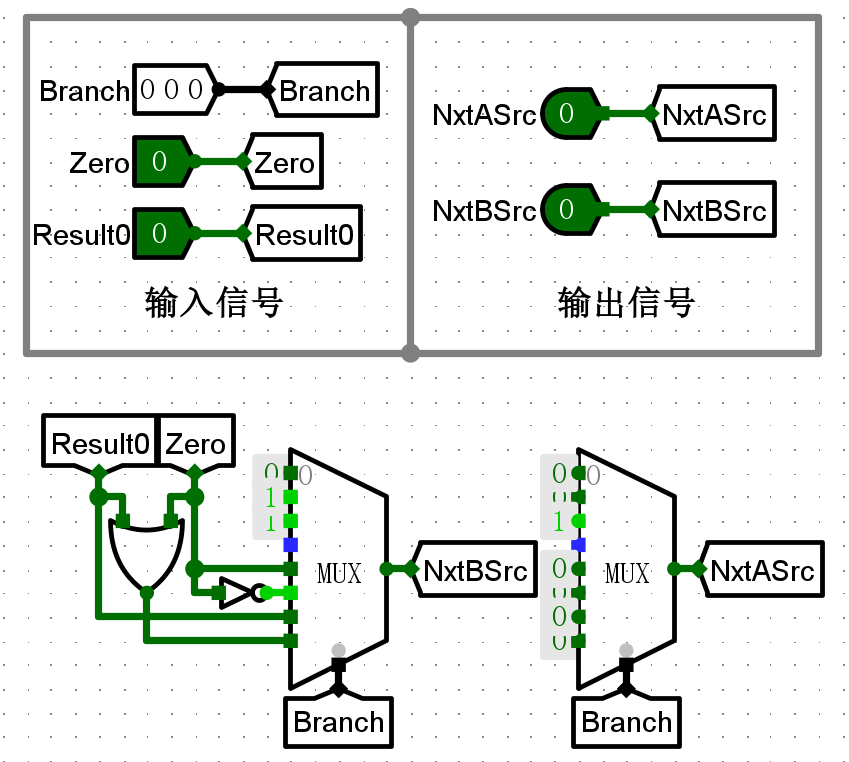


封装电路如下：

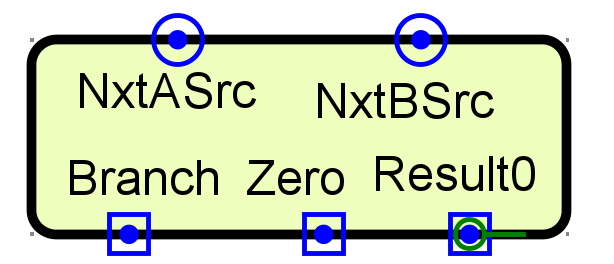


2）设计跳转控制器子电路。在Logisim中添加一个名为“Branch”的子电路，双击该子电路名称，在右侧工作区中构建相应电路。在工作区中添加指令输入引脚、分线器、多路选择器、逻辑门输出引脚等组件；修改组件属性，根据输出信号逻辑表达式进行线路连接。

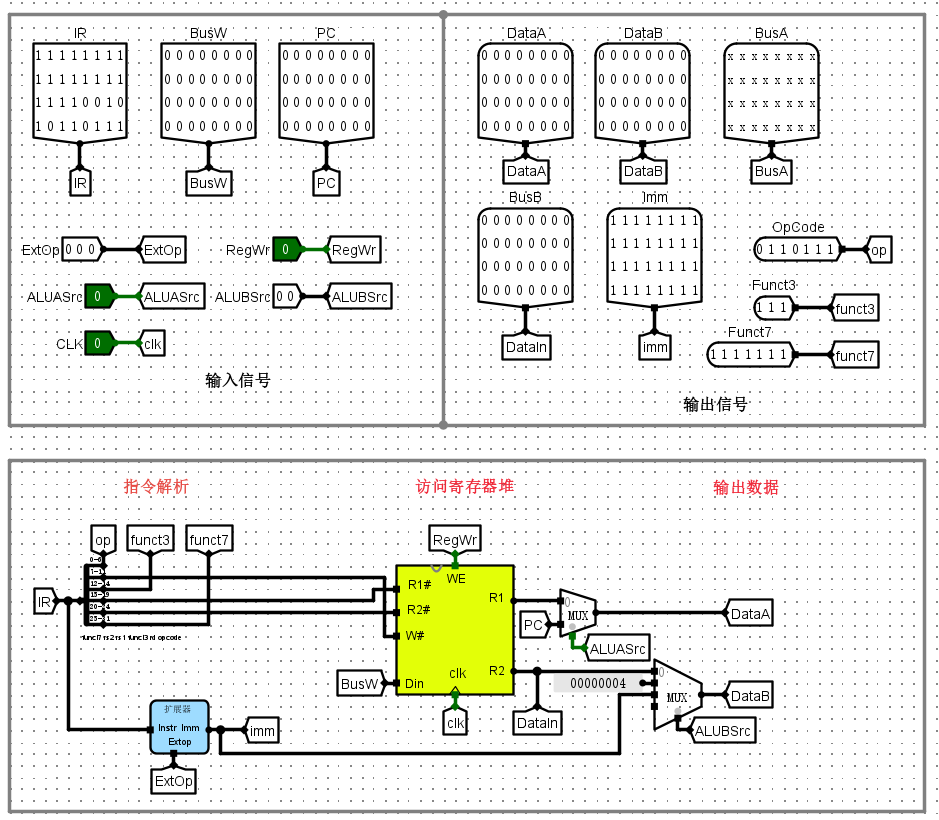
设计电路如下：



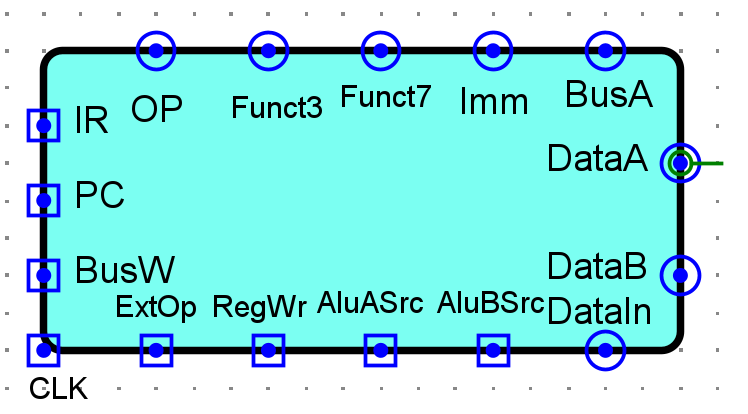
封装电路如下：



3） 设计“IDU”子电路。首先根据上文所示的指令字段的位置分布，将输入指令分解出opcode、rd、funct3、rs1、rs2和funct7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，并根据控制信号ALUAsrc和ALUBsrc选择ALU的两个操作数。ALUASsrc宽度为1位，选择ALU输入端A的来源，为0时选择BusA，为1时选择PC。ALUBSrc宽度为2位，选择ALU输入端B的来源。为00时选择BusB，为01时选择常数4，为10时选择立即数Imm。在Logisim中添加一个名为“IDU”的子电路，双击该子电路名称，在右侧工作区中构建相应电路如下：

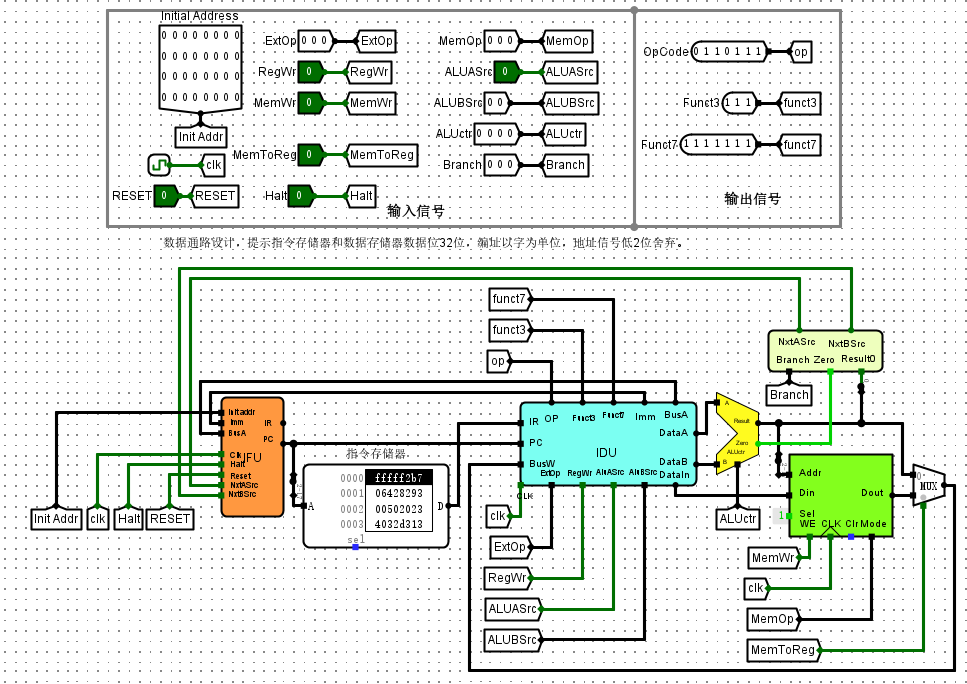


编辑电路封装如下：



4） 数据通路实验。在Logisim中添加一个名为“DataPath”的子电路，双击该子电路名称，在右侧工作区中构建相应电路。电路原理图如图5.10所示，引脚布局如图5.19所示，在工作区中添加IFU取指令部件子电路、IDU子电路、ALU子电路和数据存储器子电路、Branch子电路，输入输出引脚、隧道和探针等组件；

修改组件属性，指令存储器和数据存储器片选信号设置为高电平有效并通过Reset信号进行控制，进行线路连接，完成电路图如下：



5）数据通路测试

指令存储器第0单元开始，顺序写入表5.2中的14条RV32I指令的机器代码，或者加载程序代码镜像文件lab5.o。加载指令代码后，设置初始地址，依次读出指令存储器中的14条指令，读出指令后，设置不同的控制信号值（如下表列出，未列出的均为0）。

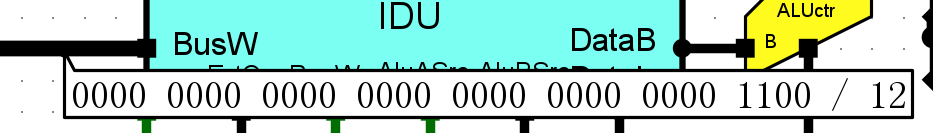


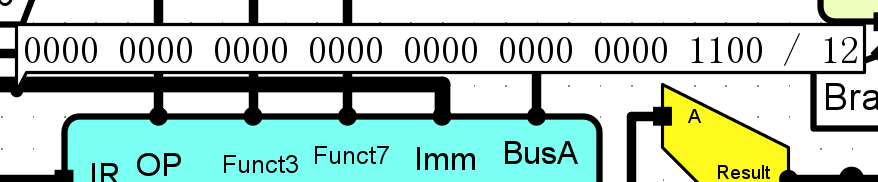
依指令推出各步骤关键数据表如下：

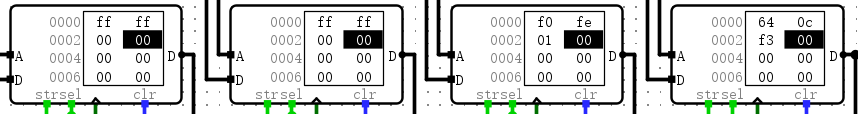
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 汇编指令 | PC | BusW | 立即数 | 寄存器堆 | 数据存储器 |
| 1 | lui x5,-1 | 0 | 0xfffff000 | 0xfffff000 | X5=0xfffff000 |  |
| 2 | addi x5,x5,100 | 4 | 0xffff064 | 0x0000064 | X5=0xfffff064 |  |
| 3 | sw x5,0(x0) | 8 | 0x00000000 | 0x00000000 | X5=0xfffff064 | 0: 0xfffff064 |
| 4 | srai x6,x5,3 | 0c | 0xfffffe0c | 0x00000403 | X6=0xfffffe0c |  |
| 5 | sw x6,4(x0) | 10 | 0x00000004 | 0x00000004 | X6=0xfffffe0c | 1: 0xfffffe0c |
| 6 | lh x7,4(x0) | 14 | 0xfffffe0c | 0x00000004 | X7=0xfffffe0c |  |
| 7 | xori x8,x7,-1 | 18 | 0x000001f3 | 0xffffffff | X8=0x000001f3 |  |
| 8 | sw x8,8(x0) | 1c | 0x00000008 | 0x00000008 | X8=0x000001f3 | 2: 0x000001f3 |
| 9 | slt x9,x8,x7 | 20 | 0x00000000 | 0x00000007 | X9=0x00000000 |  |
| 10 | sw x9,12(x0) | 24 | 0x0000000c | 0x0000000c |  | 3:0x00000000 |
| 11 | bne x9,x0,label2 | 28 | 0x00000000 | 0xffffffd8 |  |  |
| 12 | jalr x10,x0,48 | 2c | 0x00000030 | 0x00000030 | X10=0x00000030 |  |
| 13 | sw x10,16(x0) | 30 | 0x00000010 | 0x00000010 |  | 4: 0x00000030 |
| 14 | auipc x11,100 | 34 | 0x00064034 | 0x00064000 | X11=0x00064034 |  |

从简起见，报告内仅呈现部分验证结果：

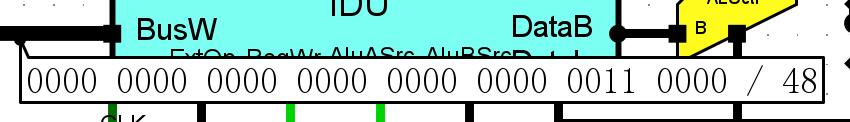
第10条：

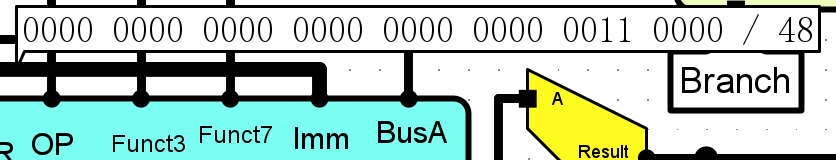
BusW为0x0000000c

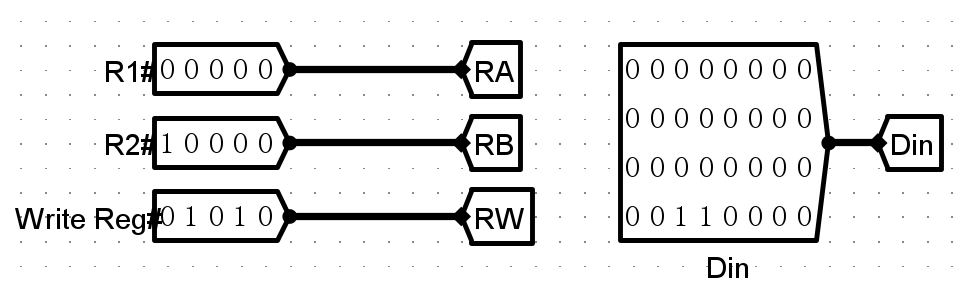
Imm为0x0000000c

3号储存被写为0x00000000

第12条：

BusW为0x00000030

Imm为0x00000030

X10即将被写为0x00000030

四、思考题

1.如何拓展ROM实验实现跑马灯的功能，在3个LED点阵中，滚动显示5个ASCII字符，如“NJUCS”?

参见子电路“跑马灯”

2.表5.2给出的第11条测试指令中标号label2所表示的偏移地址是多少（用真值表示）？

12位立即数imm是0xfd8也就是111111011000，根据前面编写的立即数扩展器，这里的imm已经是SEXT[imm[12:1]<<1]。（可见立即数扩展器实际上是立即数处理器）

那么偏移地址就是0xfd8，对应真值-40。

3.在Risc-V架构中，举例说明什么是伪指令？伪指令如何实现？

在 RISC-V 架构中，伪指令是一类在汇编级别提供便利的指令。它们并不是真正的硬件指令，而是由汇编器在汇编过程中被替换成一系列真正的指令来实现其功能。伪指令主要用于简化编程和提供语法上的方便性。

举例来说，RISC-V 中的伪指令 li（load immediate）用于将立即数加载到寄存器。实际上，汇编器会将 li 伪指令转换为LUI（Load Upper Immediate）和 ADDI（Add Immediate）指令的组合。

另一个例子是 la（load address）伪指令，用于加载某个标签或符号的地址到寄存器。实际上，汇编器会根据标签或符号的位置和上下文，将 la 伪指令转换为LUI、AUIPC（Add Upper Immediate to PC）和偏移量修正指令（例如 ADDI、SUB）的组合。

4.在指令执行过程中，如何实现程序结束后，指令不再继续执行？

就数据通路而言，将IFU中的Halt信号设为1，PC停止输出信号。

如果是讨论程序设计，我们可以在程序末尾使用ecall这样的指令将控制权交给操作系统。