实验 5：存储器及数据通路设计

姜帅 221220115

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

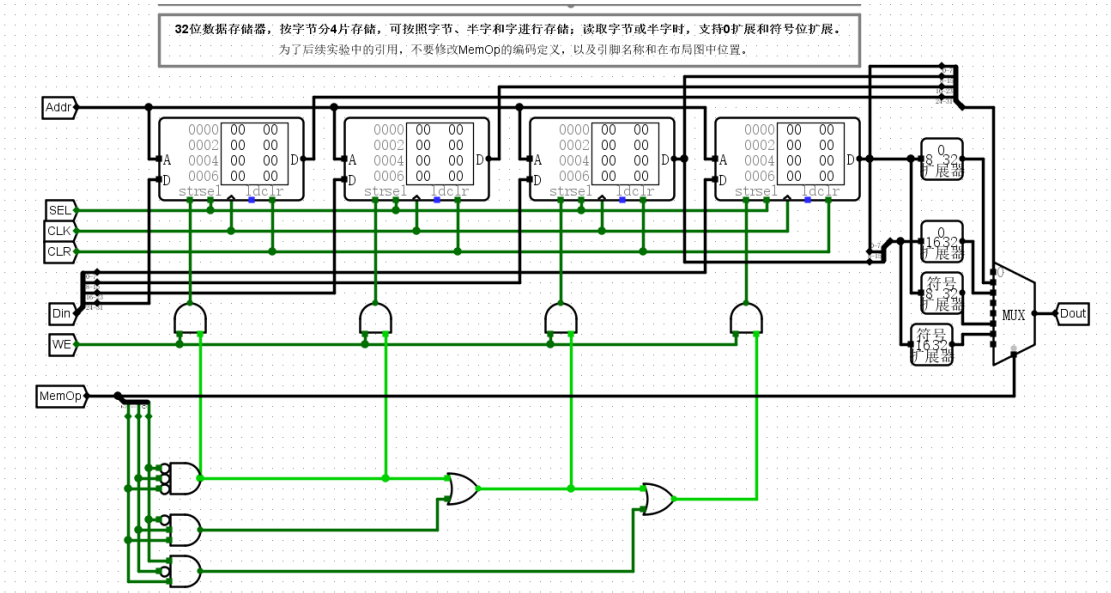
三、 实验内容

1. 存储器读写实验

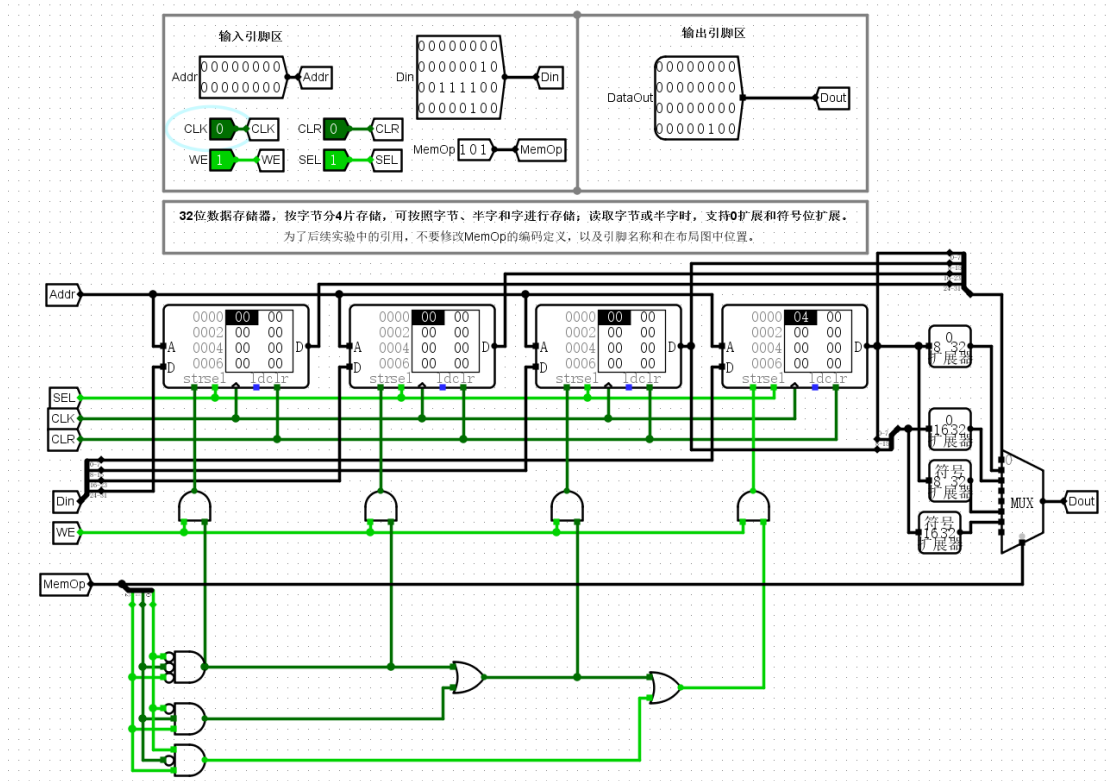
实验步骤如下。

1）数据存储器实验。在 Logisim 中添加 “数据存储器”的子电路，双击该子电路，在工作区中按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 为 高电平有效。设计电路原理图，连接组件，实现功能。

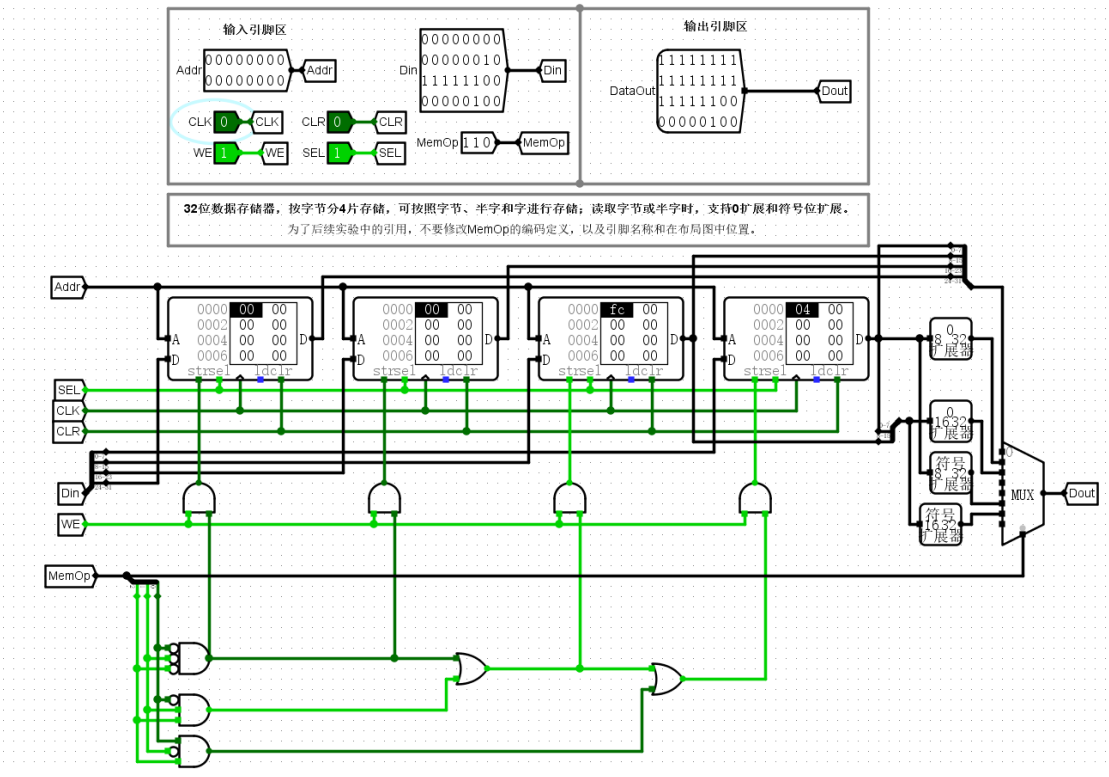
电路图：



运行测试：



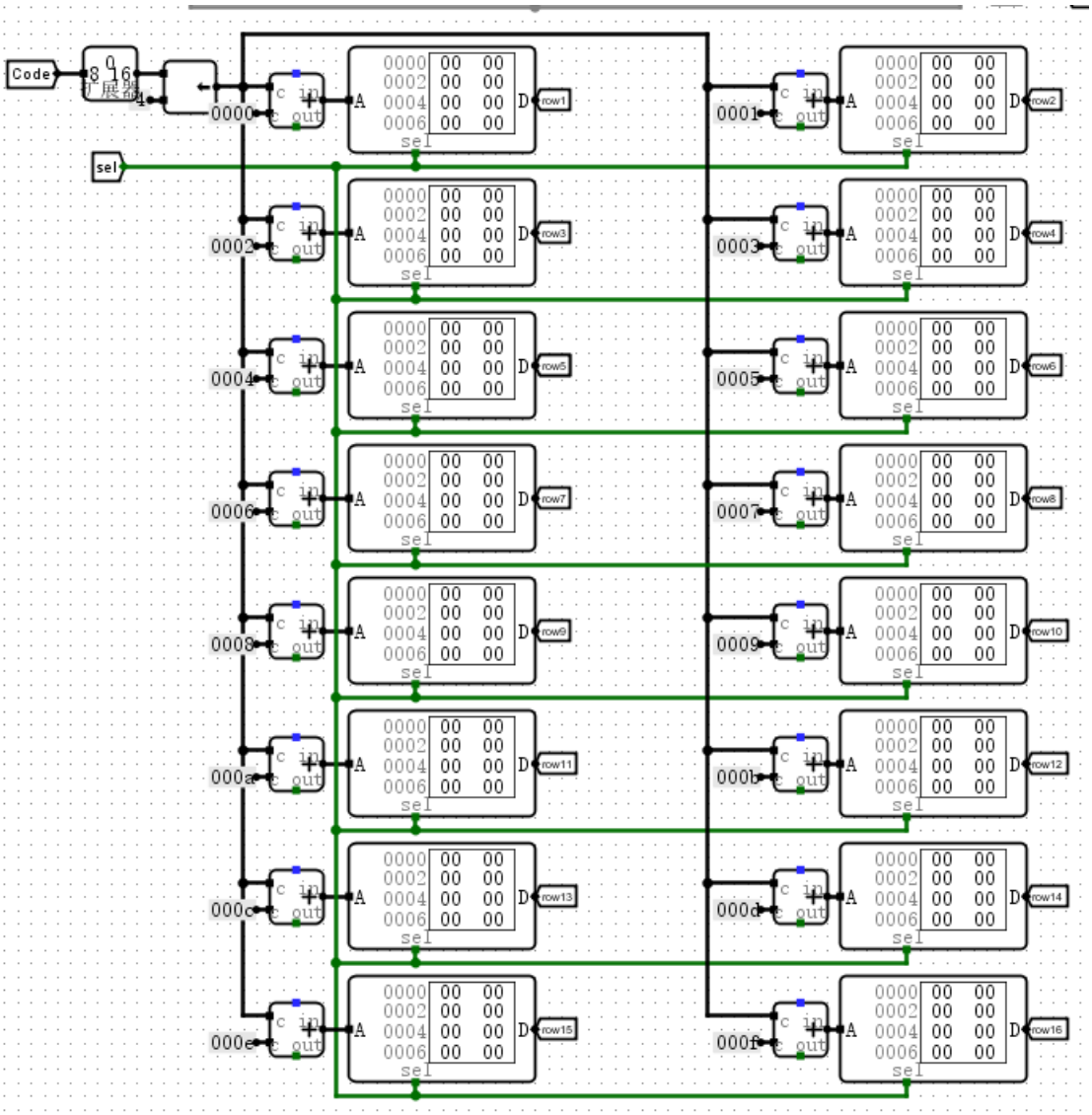
上图为存取最低 1 字节



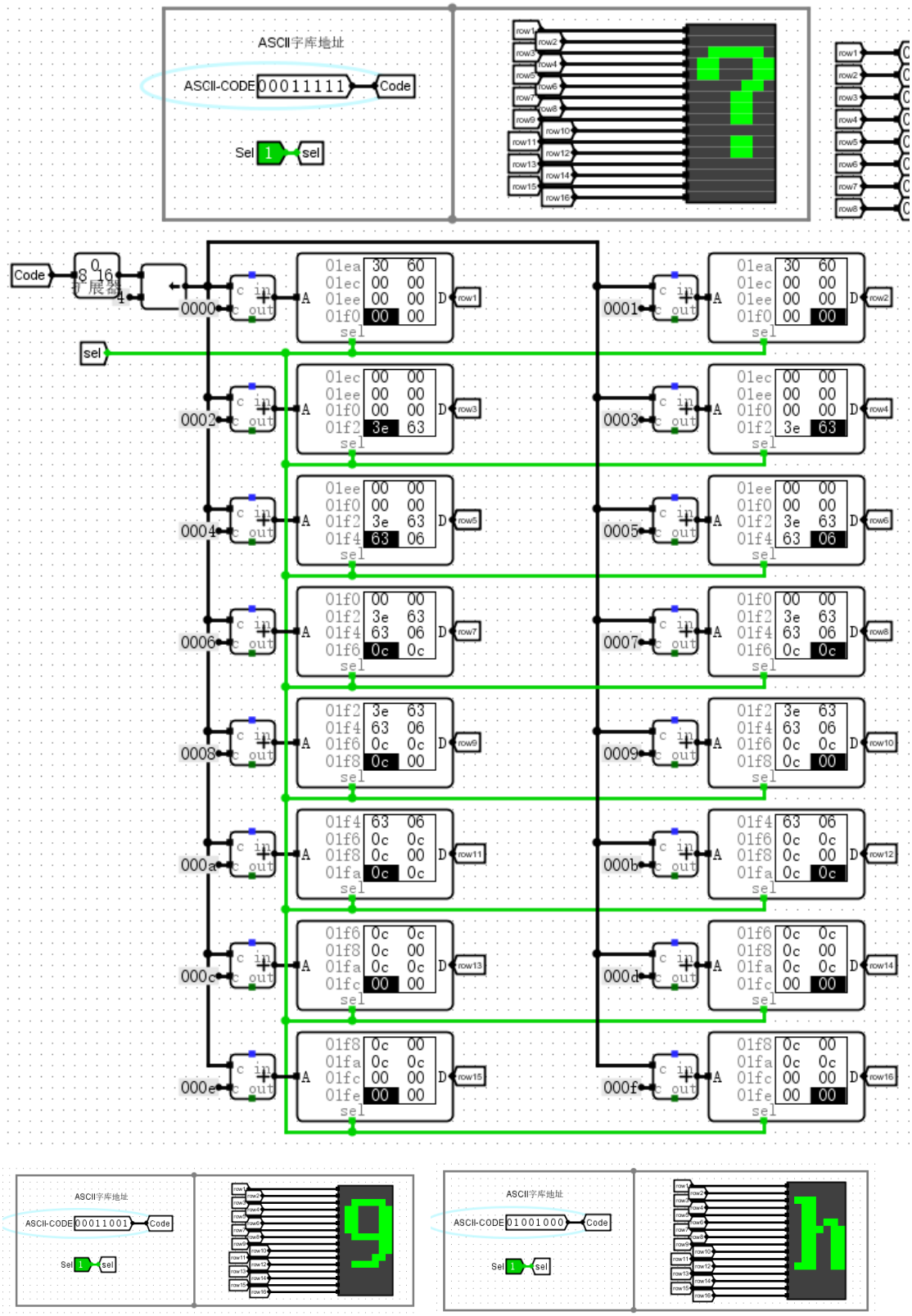
上图为存取最低 2 字节

2）ASCII 码显示实验。在 Logisim 中添加 “ASCII 码显示实验”的子电路，双击该子电路，在工作区中按组件布局图放置输入输出引脚、ROM、隧道等组件。设计电路原理图，连接组件，加载 ASCII 码字库文件。输入不同的 ASCII 编码，观察 LED 点阵显示字符形状，验证电路的正确性，记录测试数据。保存电路设计文件。

电路图：


Code
8
16
扩展器
1
sel
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
000a
000b
000c
000d
000e
000f
c
in
out
A
D
sel
0000
0002
0004
0006
00
row1
row2
row3
row4
row5
row6
row7
row8
row9
row10
row11
row12
row13
row14
row15
row16

运行测试：



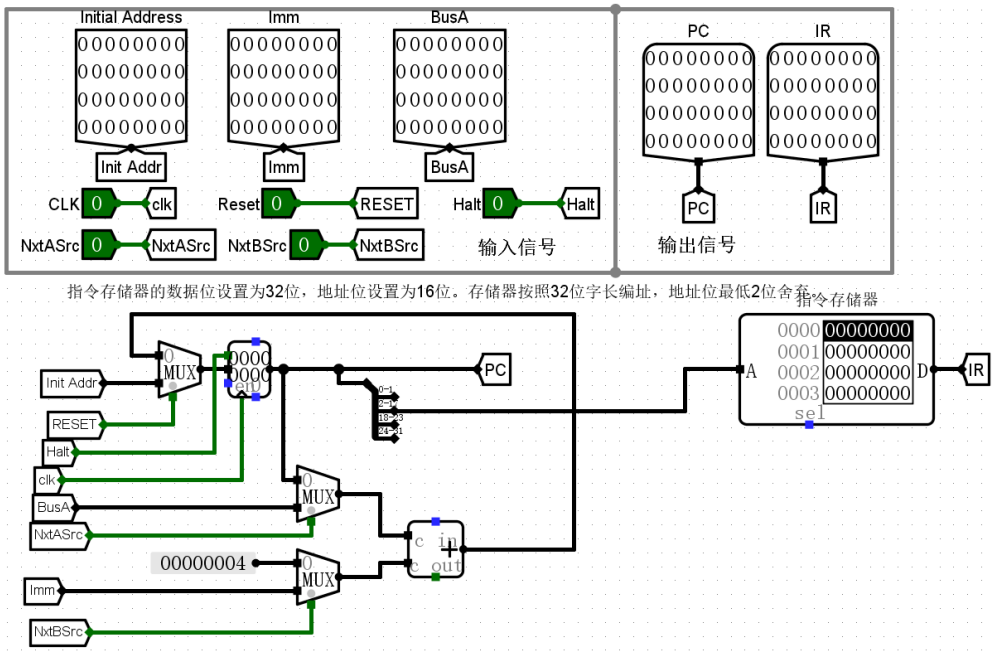
2. 取指令部件 IFU 实验

实验步骤如下。

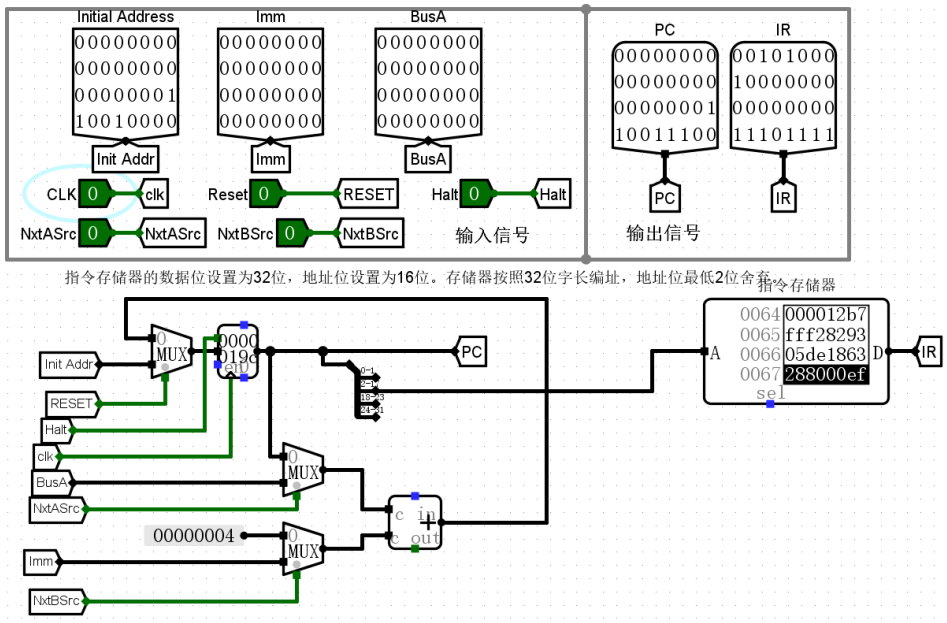
在 Logisim 中添加 “IFU 实验”的子电路，双击该子电路，在工作区中按组件引脚布局图放置输入输出引脚、指令存储器 ROM、隧道等组件。修改 ROM 属性，设置数据位宽为 32 位，地 址宽度为 16 位，片选信号 Sel 为高电平有效。按照上述要求设计电路

原理图，连接组件，实现功能。

电路图：



运行测试：



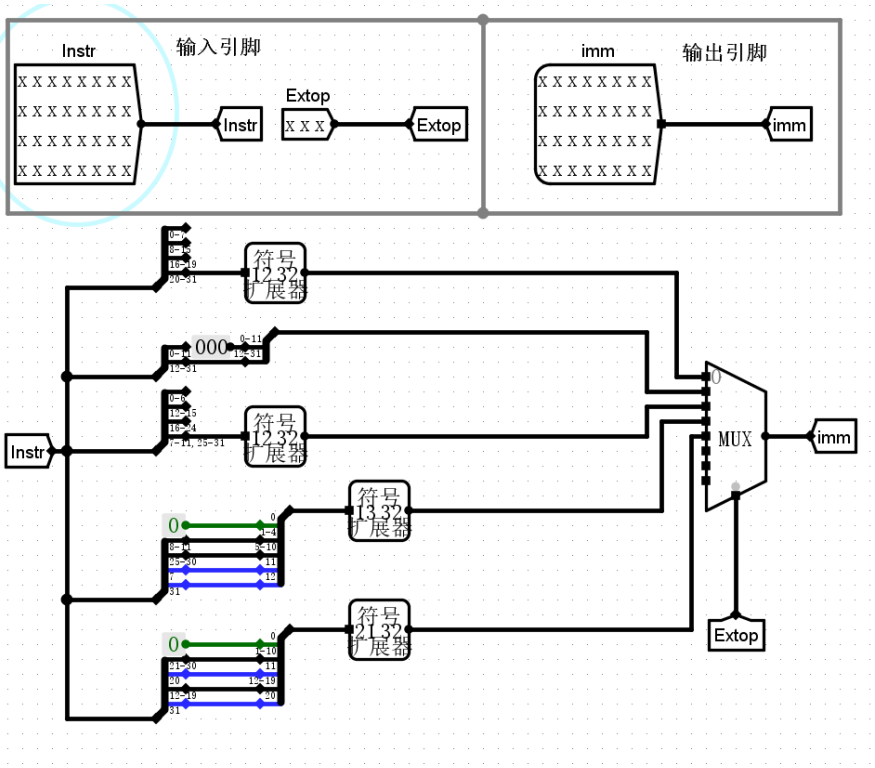
3. 数据通路实验

实验步骤如下。

1）“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。参考图 5.12，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。

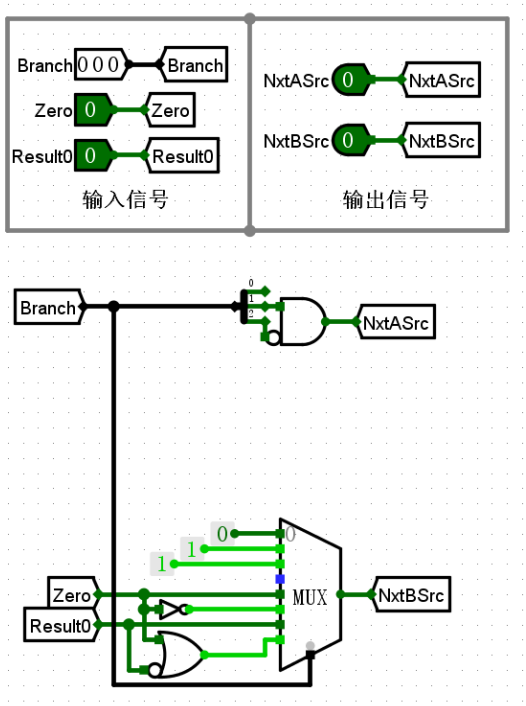
电路图：



2）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。引脚参考如图 5.15 所示，根据表 5.2 所示，列出 NxtASrc 和 NxtBSrc 两个信 号的逻辑表达式，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。添加标识符和电路功能描述文字。

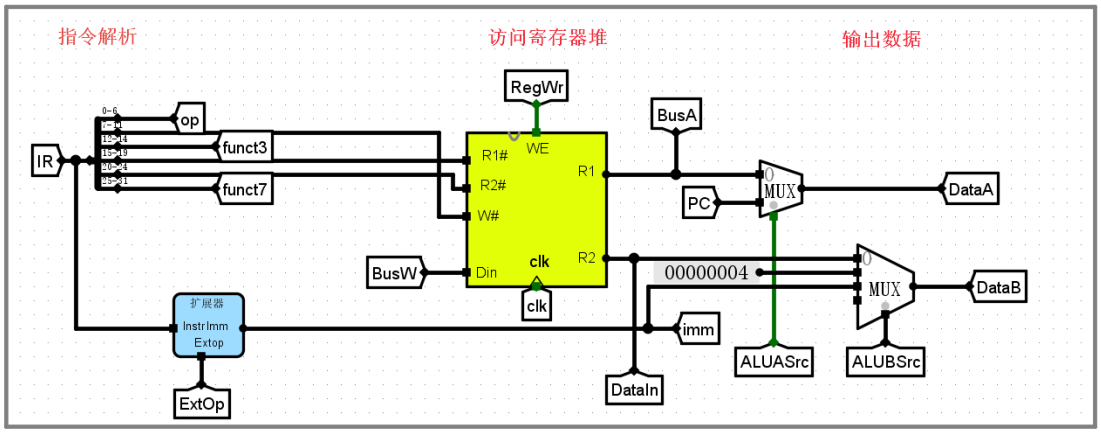
电路图：



3）“IDU”子电路。

根据图 5.11 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。调用实验 3 中实现的寄存器堆子电路，修改寄存器堆子电路，使得 0 号寄存器始终为 0。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。引脚布局如图 5.17 所示，在工作区中添加指令输入引脚、立即数扩展器子电路、寄存器堆子电路、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，添加标识符。

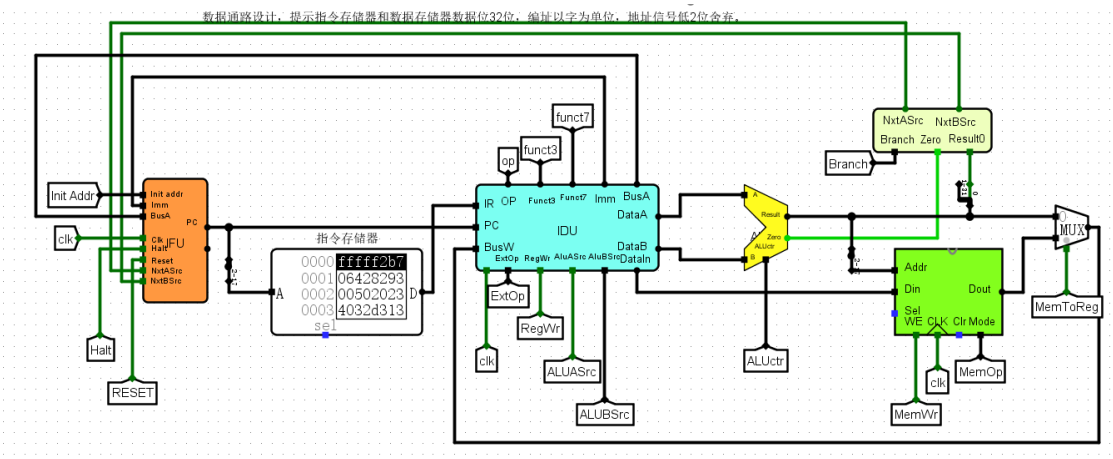
电路图：



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。根据引脚布局和电路原理图，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，在 PC、Imm 和 BusW 等输出信号上添加探针，添加标识符和电路功能描述文字。保存电路设计文件。

电路图：



5）数据通路测试。

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，表 5.3 中列出了每条指令下，控制信号不为 0 的赋值。时钟单步执行，观察输出信号值以及 PC、 BusW、Imm 等中间数据，填写表 5.4，记录测试数据，

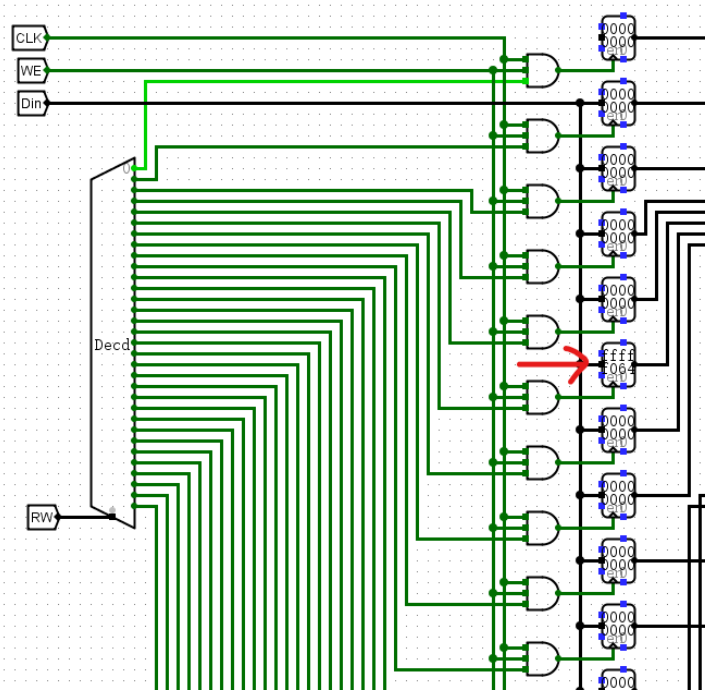
验证电路的正确性。

运行结果：

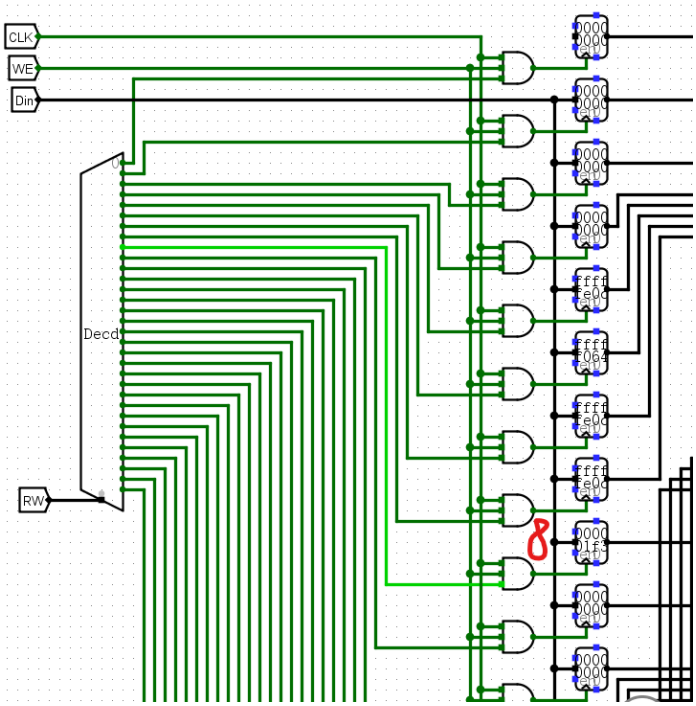
序号	汇编指令	PC	BusW	立即数	寄存器堆	数据存储器
1	lui x5,-1	0	0xfffff000	0x fffff000	X5=0xfffff000	
2	addi x5,x5,100	4	0xffff064	0x0000064	X5=0xfffff064	
3	sw x5,0(x0)	8	0x0	0x0	X5=0xfffff064	0: 0xfffff064
4	srai x6,x5,3	0c	0xfffffe0c	0x00000403	X6=0xfffffe0c	
5	sw x6,4(x0)	10	0x00000004	0x00000004	X6=0xfffffe0c	1: 0xfffffe0c
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	0xfffffb8		
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	

（更正：第十一条指令得到的立即数为 0xfffffd8）

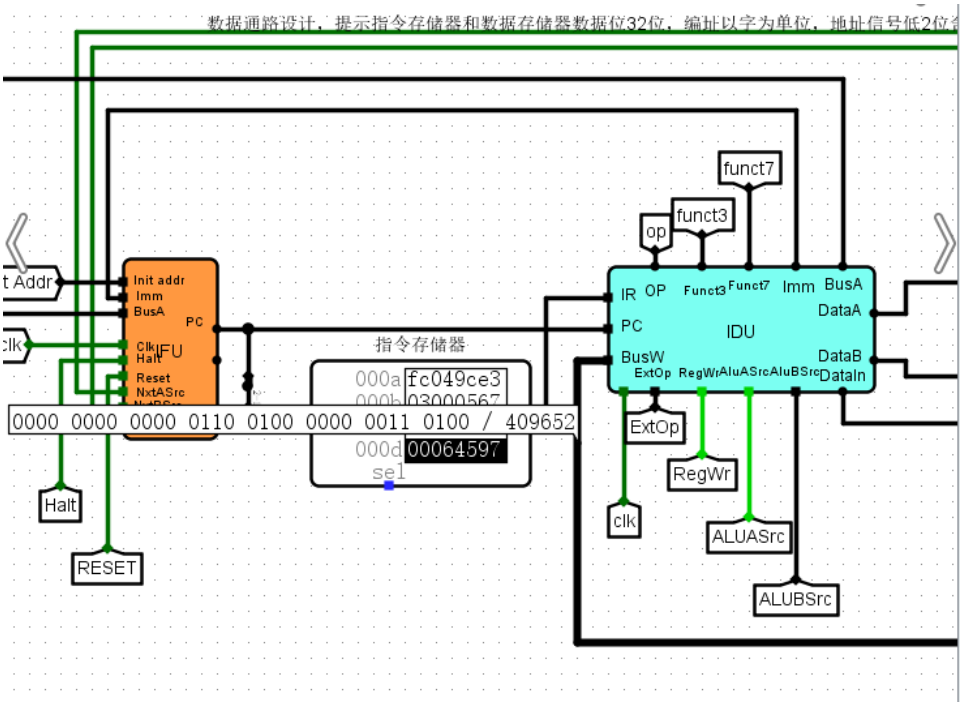
总体结果



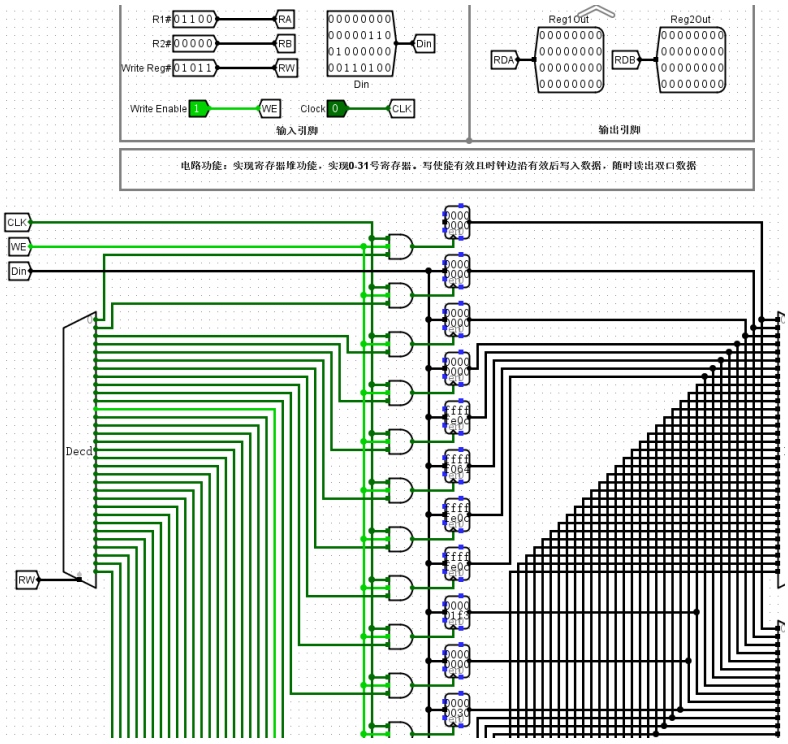
第三条指令，五号寄存器的值↑（fffff064）



第 8 条指令，8 号寄存器的值↑（000001f3）



第 14 条指令，busW 的值（0x00064034）

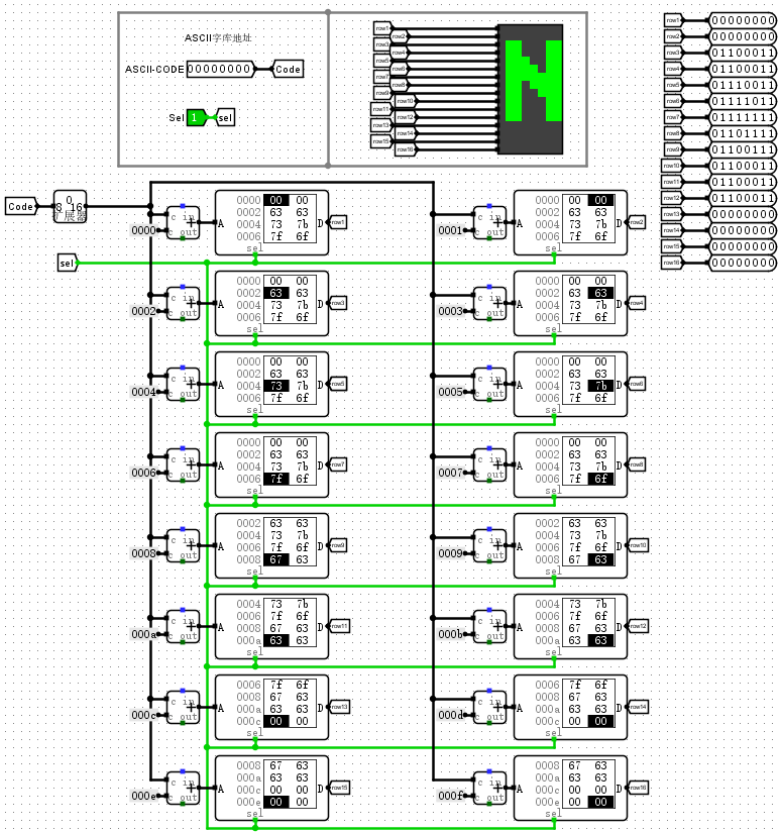


第 14 条指令，各寄存器的值（Din 写入 x11）

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”。

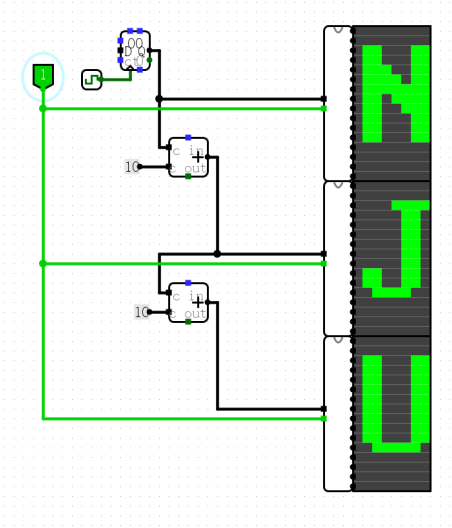
1）对 ASCII 码原实验进行修改，得到如下电路图：



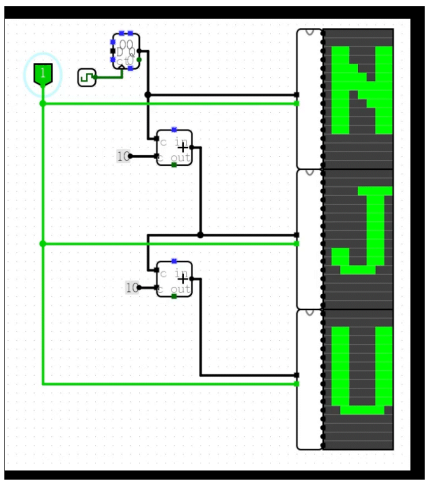
2）修改寄存器内容，按 NJUCSNJU 的顺序：

```
0000  00 00 63 63  73 7b 7f 6f
0008  67 63 63 63  00 00 00 00
0010  00 00 0f 06  06 06 06 06
0018  06 66 66 3c  00 00 00 00
0020  00 00 63 63  63 63 63 63
0028  63 63 63 3e  00 00 00 00
0030  00 00 1e 33  61 60 60 60
0038  60 61 33 1e  00 00 00 00
0040  00 00 3e 63  63 30 1c 06
0048  03 63 63 3e  00 00 00 00
0050  00 00 63 63  73 7b 7f 6f
0058  67 63 63 63  00 00 00 00
0060  00 00 0f 06  06 06 06 06
0068  06 66 66 3c  00 00 00 00
0070  00 00 63 63  63 63 63 63
0078  63 63 63 3e  00 00 00 00
0080  00 00 00 00  00 00 00 00
0088  00 00 00 00  00 00 00 00
0090  00 00 00 00  00 00 00 00
0098  00 00 00 00  00 00 00 00
00a0  00 00 00 00  00 00 00 00
00a8  00 00 00 00  00 00 00 00
00b0  00 00 00 00  00 00 00 00
00b8  00 00 00 00  00 00 00 00
00c0  00 00 00 00  00 00 00 00
```

封装成子电路，得到最终电路：



效果：



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）？

指令: 0xfc049ce3 -> 二进制: 1111110 00000 01001 001 11001 1100011

-> imm[12:1] = 1111 1110 1100 -> SEXT[imm[12:1]<<1] = FFFF FFD8(16)

Label2 = PC + SEXT[imm[12:1]<<1] = 28(10) + FFFFFFD8(16) = FFFF FFF4(16)

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，伪指令是一种在汇编语言级别上提供便利的指令，但在实际的硬件中并没有对应的指令操作码。伪指令在汇编语言中被解释器或汇编器识别，并转换为一系列实际的指令来实现所需的功能。伪指令一般会被汇编器翻译成一条或者多条等价的实际指令。

举例：mul rd, rs1, rs2（将寄存器 rs1 和 rs2 的值相乘，并将结果存储到寄存器 rd 中。）

伪指令的实现取决于特定的编译器或汇编器。不同的工具链可能会采用不同的方法来将伪指令转换为实际指令序列。这些转换通常在编译或汇编阶段进行，以生成适用于特定 RISC-V 架构的可执行代码。

5. 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?

可以通过 PC 控制程序的终止，当程序执行到最后一条指令时，可以将 PC 设置为一个特殊的值，例如指向一个非有效地址或指向程序结束标记的地址。这样一来，当处理器尝试执行下一条指令时，由于程序计数器指向的地址无效，执行流程会终止。