实验 5：存储器及数据通路设计

一、实验目的

1. 学习存储器的读写方法，掌握指令存储器和数据存储器的设计方法。

2. 学习处理器读取指令的过程，掌握 RV32I 下取指令部件设计方法

3. 分析 RV32I 不同运算指令的执行过程，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. 分析单周期 CPU 的设计要求，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

三、实验内容

Logisim 提供 RAM 和 ROM 两种存储器组件，存储器地址端口的位宽最大可以设置为 24 位，数据端口位宽最大可以设置为 32 位。需注意的是，存储器的存储地址是按照设定数据位宽度作为单位进行编址，而不是按照字节编址。

存储器组件可以设置片选信号 sel 是低电平还高电平有效。RAM 组件中的控制信息包括时钟控制端、清 0 端 clr、片选信号 sel、存数（store）使能端 str、取数（load）使能端 ld。存数使能端 str 和取数使能端 ld 一般不会同时出现，当片选信号 sel 有效，并且存数使能端 str 有效时，则在时钟有效信号到达后，输入数据端信息被写入 RAM 指定地址处。当取数使能端有效 ld 时，可将存储器指定地址中的数据输出到数据输出端。

RAM 组件除了可以通过数据输入端写入数据外，还可以使用 Logisim 十六进制编辑器 Hex Editor 通过键盘输入数据和加载数据镜像文件 Load Image 两种方法来实现数据写入。ROM 组件只能通过十六进制编辑器和加载数据镜像文件的方式写入。ROM 组件通过清除数据(Clear Contents)命令来删除已经写入的数据，否则的话，一直保存在电路中。而 RAM 组件中的数据，每次启动或者仿真复位后都需要重新写入。

在 Logisim 的工作区中放置 RAM 或 ROM 组件，将鼠标移到组件上，点击鼠标右键后弹出对应菜单框，如图 5.1 所示，选中编辑内容(Edit Contents)命令项后，打开“Logisim:十六进制编辑器”。

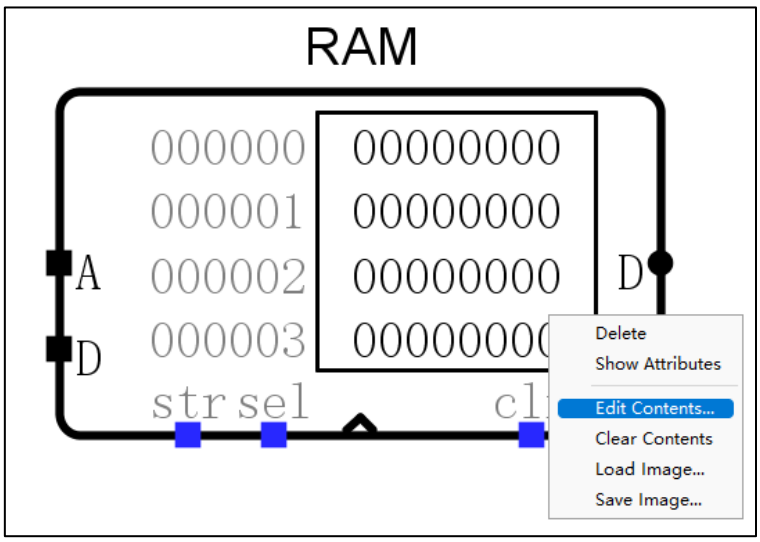

图 5.1 点击鼠标右键弹出组件菜单框

在“Logisim:十六进制编辑器”中，按照所设置的数据位宽和地址位宽，使用键盘在相应的地址处输入数据。输入数据后可点击“保存”按钮把输入数据保存到镜像文件（image）中，如图 5.2 所示。

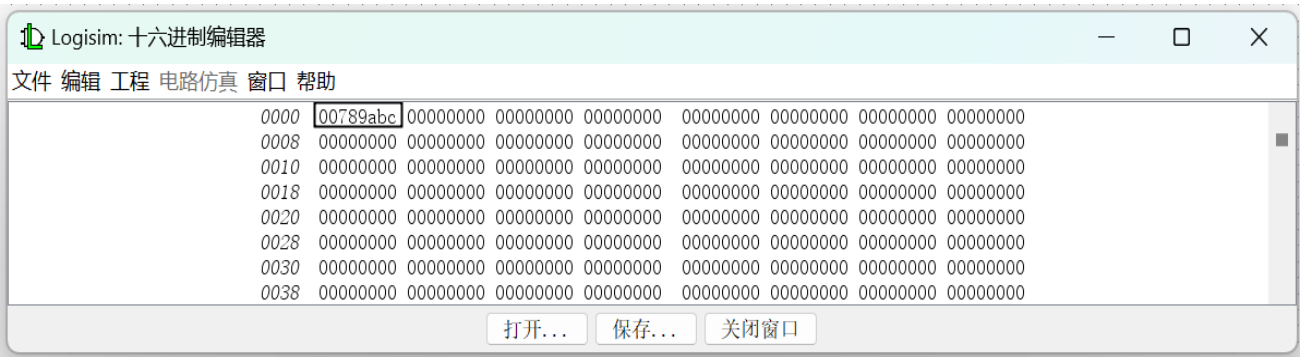

图 5.2 Logisim 十六进制编辑器

保存在镜像文件中的数据，可通过直接加载镜像文件的方式输入到 RAM 或 ROM 中。过程如下：在相应组件菜单中选择加载镜像(Load Image)命令项后，便可以直接读取镜像文件内容到存储器指定地址中。

数据镜像文件可以使用文本编辑器打开，打开该镜像文件后可以发现，第一行为“v2.0 raw”，从第二行开始存放的是存储器的数据。当数据位宽为 32 时，每一项显示 4 字节的数据，以空格或回车隔开。可以利用文本编辑器修改镜像文件中的数据，然后重新加载到 RAM 或 ROM 中。在镜像文件中，如果有 n 个连续重复数据，可以使用“*n**数据”来表示。

1. 只读存储器实验

利用 ASCII 码可显示字符点阵字库，在 Logisim 的 LED 点阵组件上显示 ASCII 码字符形状。ASCII 码点阵字库文件按照可显示字符顺序排列，从字符空格 SP（ASCII 码值 0x20）开始，到删除字符 DEL（ASCII 码值 0x7f）结束共 96 个字符，每个字符使用 8 列 16 行的点阵表示字形。LED 点阵设置为 8 列 16 行，这样在 ASCII 点阵字库中输入某个 ASCII 码的编码时，将输出该 ASCII 码的字形点阵数据 16*8 位（16 个字节），

将该点阵数据输入到 LED 点阵中，则在 LED 点阵组件上显示该 ASCII 码的字符形状。

ASCII 码可见字符点阵字库 ascii8-16.zk 的文件内容如下：

1. v2.0 raw
2. 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3. 00 00 18 3C 3C 3C 18 18 18 00 18 18 00 00 00 00
4. 00 63 63 63 22 00 00 00 00 00 00 00 00 00 00 00
5. 00 00 00 36 36 7F 36 36 36 7F 36 36 00 00 00 00
6. …

首行是 Logisim 数据文件的标识，不存储到存储器中；第二行表示字符空格的字形点阵，第三行表示字符“!”的字形点阵，第四行表示字符“””的字形点阵，以此类推；载入存储器中时从第二行开始。每个字符的字形点阵都是 16 个字节，表示 16 行 8 列矩阵，如果某 1 位为 1，则表示对应矩阵点是可显示的。

ASCII 码可见字符共有 96 个，每个字符用 16 个字节的数据来表示，点阵字库共有 1536 个字节。

实验设计的关键一是在于如何根据输入的 ASCII 码数值，查找到字符点阵字库中的初始位置。在点阵字库中 ASCII 码可见字符点阵从地址 0 开始存储，因此在根据 ASCII 码来读取点阵字库时，需要先把 ASCII 码值减去十六进制 0x20，然后根据每个字符的点阵数据是 16 个字节，把得到的差值左移 4 位在低位补 4 个 0，则得到该字符点阵在字库中的起始位置，字符点阵数据为连续存放的 16 个字节。

实验设计的关键二是如何同步输出某个字符 16 个字节的点阵数据。首先需要将点阵字库文件加载的只读存储器，由于提供的点阵字库文件是以字节为单位，为了能够同时输出 16 个字节的点阵数据，则需要复制 16 个加载相同点阵字库的只读存储器，读取点阵数据时每个只读存储器的地址数据按点阵行的次序加 1 递增，则可以同时读取第 1 行、第 2 行、…、第 16 行的点阵数据，然后再同步输出到 LED 点阵组件的输入端中。

根据点阵字库中的数据宽度为 8 位，则设置只读存储器的数据位宽度为 8 位；点阵字库文件大小为 1536 个字节，则只读存储器的地址位宽度至少需设置为 11 位。

如果输入的 ASCII 码不在可见字符范围内，则输出错误标志位 CodeErr。

实验要求根据输入的 ASCII 码值，在 LED 点阵中显示该 ASCII 码的字形；如果 ASCII 值不在可显示字符范围内，则输出 ASCII 码错误标志位；设计电路原理图。

在 Logisim 中的工作区中按图 5.3 所示的组件布局图放置输入输出引脚、加减法器、16 个只读存储器 ROM、LED 点阵等组件。设置只读存储器 ROM 的属性：数据位宽度以及地址位宽度，设置片选信号（Sel Acitve On）为高电平有效，然后加载 ASCII 码字库文件 ascii8-16.zk。连接组件，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。保存电路设计文件为 lab5.1.circ。

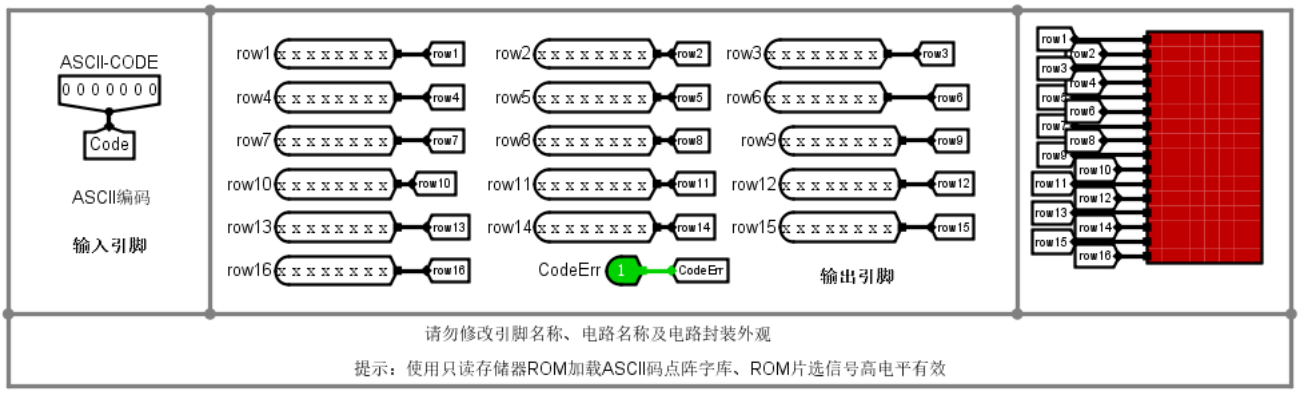

图 5.3　ROM 读写实验布局图

2.　数据存储器实验

Logisim 中的 RAM 组件的数据接口 Data Interface 中有三种不同的工作模式，若设置为分离加载和存储端口模式 Separate load and store ports，则分别显示输入数据端口和输出数据端口；其它两种模式则只使用同一个数据端口进行同步或异步读写操作，区别在于是否有时钟输入端口。

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。提示：为了能够在后续实验作为子电路引用，不要修改编码定义。

表 5.1 MemOp 控制信号含义

MemOp	指令	含　义
000	lw,sw	按字存取，4 字节
001	lbu	按字节读取，1 个字节，0 扩展到 4 字节
010	lhu	按半字读取，2 字节，0 扩展到 4 字节
101	lb,sb	按字节存取，1 个字节，在读取时，按符号位扩展到 4 字节
110	lh,sh	按半字存取，2 个字节，在读取时，按符号位扩展到 4 字节

lw 指令从存储器中读取（加载）4 个字节数据到寄存器 rd 中。lh 指令从存储器中加载 2 个字节数据，根据符号位扩展到 32 位，然后再传送到寄存器 rd 中。lhu 指令从存储器中加载 2 个字节数据，零扩展到 32 位，再传送到寄存器 rd 中。lb 和 lbu 指令的功能类似，只是从存储器中加载 1 个自己数据，然后再根据符号位或零扩展到 32 位后传送到寄存器 rd 中。

sw、sh 和 sb 指令的功能是将寄存器 rs2 中从低位开始的 4 个字节、2 个字节和 1 个字节的数据存储到存储器中，此时存储器写使能信号有效。

在本次实验中，为了获得较好的性能，假设对于存取多字节数据的有效地址都是自然对齐，如对于 4 个

字节数据访问，存储器地址在四字节边界上（有效地址的最低 2 位为 0），对于 2 个字节数据访问，存储器地址在两字节边界上，对于 1 个字节数据的访问，则可在任意地址边界上。

实验要求设计一个 256KB 的数据存储器，可按照字节进行存取操作，数据字长为 32 位，则地址位为 18 位。提示：为了方便按照字节进行存取操作设计，可使用 4 片数据位宽为 8 位 RAM，分别存储 32 位数据的不同字节段数据。

本次实验的关键在于如何根据 MemOp 控制信号和地址的低 2 位选中合适的 RAM 组件进行数据的存储以及如何组织输出数据。

假设 4 片 RAM3~RAM0 的片选使能端分别定义为 SEL0~SEL3，SEL0 对应最低字节存储器的使能端。可根据 MemOp 控制信号和存储器地址 Addr 最低 2 位来确定片选信号的数值，当存取多字节数据的有效地址不是自然对齐时，显示错误信号有效 AlignErr=1。出现未定义的 MemOp 状态时，错误信号也有效 AlignErr=1，如表 5.2 所示。

表 5.2 存储器控制信号、地址低 2 位和片选信号、地址对齐的对应关系

MemOp[2][1][0]	Addr[1][0]	SEL3	SEL2	SEL1	SEL0	AlignErr
000	00	1	1	1	1	0
*00	01	0	0	0	0	1
*00	10	0	0	0	0	1
*00	11	0	0	0	0	1
*01	00	0	0	0	1	0
*01	01	0	0	1	0	0
*01	10	0	1	0	0	0
*01	11	1	0	0	0	0
*10	00	0	0	1	1	0
*10	01	0	0	0	0	1
*10	10	1	1	0	0	0
*10	11	0	0	0	0	1
其它	**	0	0	0	0	1

根据片选信号确定每一片 RAM 在存储时的写入字节，例如当片选信号 1111 全部有效时，存储高字节的 RAM3 写入输入数据 Din 的最高字节；当片选信号为 1000 时，写入输入数据 Din 的最低字节；当片选信号为 1100 时，写入输入数据 Din 的次低字节。

同时，可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；当片选信号为 0010 时，最低字节的数据来自 RAM1 的数据输出；当片选信号为 0100、1100 时，最低字节的数据来自 RAM2 的数据输出；当片选信号为 1000 时，最低字节的数据来自 RAM3 的数据输出。次低字节数据可根据片选信号判断选取 RAM1 或 RAM3 的输出数据。

写入数据也需要根据写入数据的字节进行选择，并传输到对应的存储器的数据输入口。例如写入 1 个

字节时，将 32 位输入数据的低 1 个字节写入到任意地址；当写入 2 个字节时，将 32 位数据的低 2 个字节写入到从偶数地址开始的存储器中。

实验步骤如下。

1）数据存储器实验。在 Logisim 中主电路的工作区中按图 5.4 所示的组件布局图放置输入输出引脚、4 个 RAM 组件、多路选择器、位扩展器、逻辑门电路、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。设计电路原理图，连接组件，实现功能。通过设置不同的输入值，观察输出结果，验证电路的正确性，记录测试数据。封装子电路如图 5.5 所示。保存电路设计文件为 lab5.2.circ。

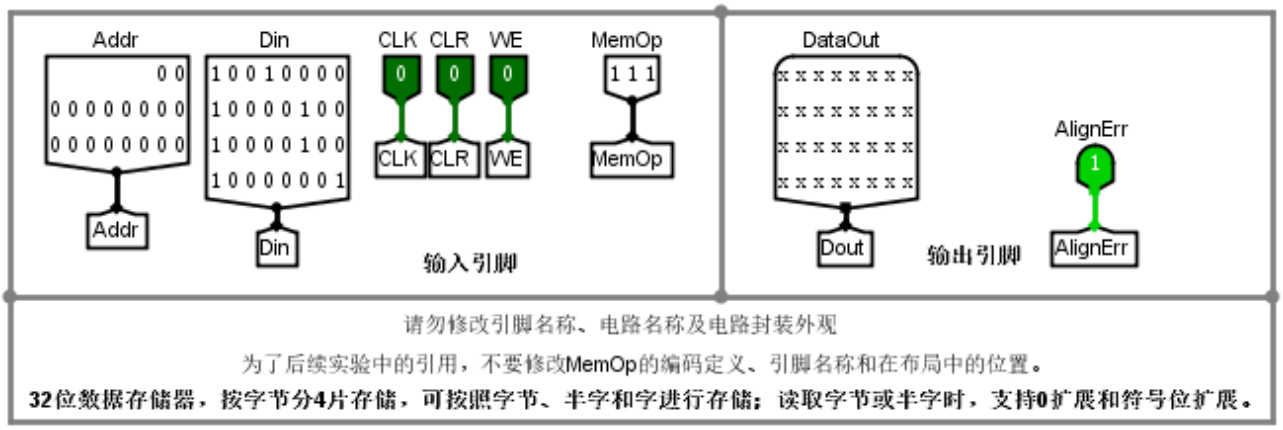

图 5.4　RAM 读写实验布局图

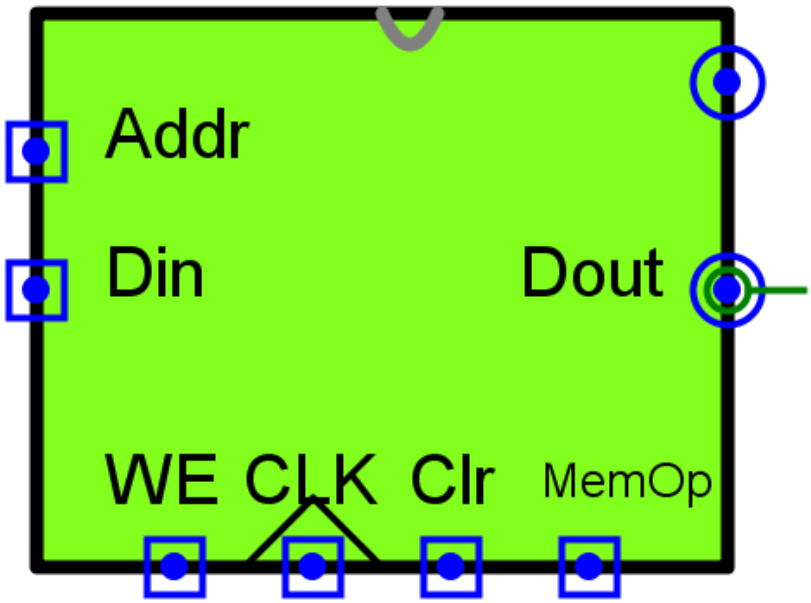

图 5.5　RAM 读写电路封装图

3.　取指令部件实验

RISC-V 指令格式如图 5.6 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。

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
B	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
J	imm[20|10:1|11|19:12]					rd	opcode

图 5.6　RISC-V 指令格式

综合分析 RISC-V37 条整数运算指令数据通路的结构，可得到如图 5.7 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。

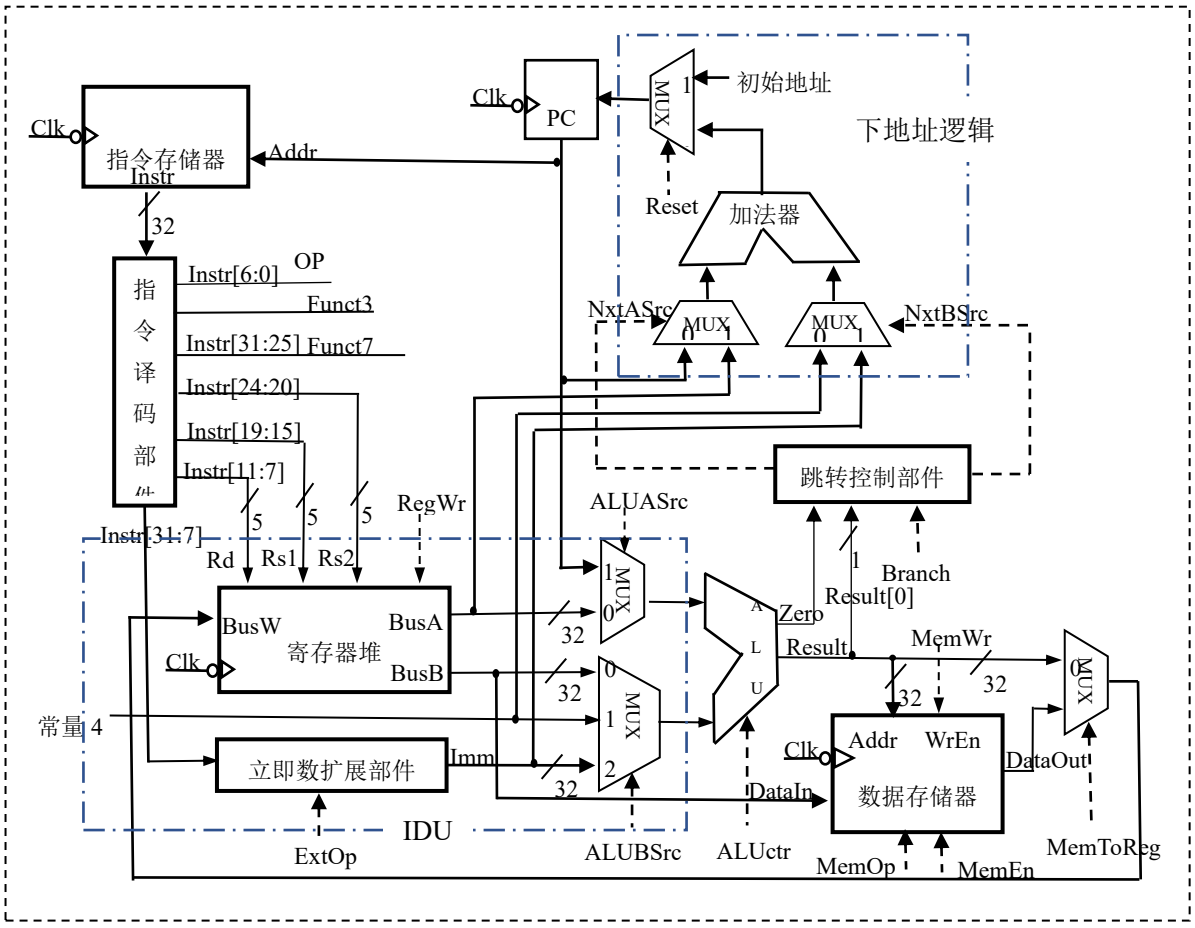

图 5.7　单周期 CPU 数据通路原理图

程序运行的第一步就是取指令，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。初始时（系统复位或刚启动）32 位的 PC 寄存器中保存着当前程序在指令存储器中起始指令的物理地址。开始执行程序后，一方面把地址送到指令存储器的地址端输出指令

内容，另一方面通过下地址逻辑来计算机下条指令的地址，然后送回 PC 寄存器。在单周期处理器中，每个时钟周期执行一条指令，所以每来一个时钟信号 Clk，PC 寄存器的值都会被更新一次，因而，PC 寄存器无须“写使能”信号控制。注意 PC 寄存器是下降沿触发。

在程序执行过程中，下一条指令地址的计算有多种情形：1、顺序执行指令，则 PC=PC+4。2、无条件跳转指令，jal 指令，PC = PC + 立即数 imm；jalr 指令，PC = R[rs1] + 立即数 imm。3、分支转移指令，根据比较运算的结果和 Zero 标志位来判断，如果条件成立则 PC = PC +立即数 imm，否则 PC=PC+4。

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，具体赋值定义如下表 5.3 所示。

表 5.3 NxtASrc 和 NxtBSrc 赋值定义表

指令类型	NxtASrc	NxtBSrc	下条指令地址
顺序指令	0	0	PC=PC+4
无条件跳转 jal	0	1	PC=PC+Imm
无条件跳转 jalr	1	1	PC= R[rs1]+Imm
分支跳转指令	0	1	条件成立 PC=PC+Imm，否则 PC=PC+4

当 NxtASrc=0 时，选择 PC 寄存器的值，否则选择 Rs1 寄存器值 BusA。当 NxtBSrc =0 时选择常量 4，否则选择立即数 Imm。

这里需要提示的是：由于 Logisim 存储器组件的存储单元是按照数据位宽为单位，当定义数据位宽为 32 位时，每个地址中包含 4 个字节内容，因而 Logisim 中的一个存储单元相当于 RISC-V 程序中的 4 个字节。因而，将 RISC-V 中指令地址转换为 Logisim 中指令存储器的地址时，需要将把指令地址中的最低两位舍弃。

取指令部件中包括初始可执行程序地址 Initial Address，当 Reset 信号为高电平有效时，在下一个时钟信号有效后，将把初始地址 Initial Address 加载到程序计数器 PC 中。

实验要求根据初始地址 Initial Address、立即数寄存器 Imm 和 rs1 寄存器的数据 BusA，以及控制信号 Reset、NxtASrc 和 NxtBSrc 的赋值输出当前指令的地址寄存器 PC 和指令存储器的输出内容 IR。按照上述要求设计电路原理图，连接组件，实现功能。

实验步骤：

在 Logisim 中主电路的工作区中按图 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、程序计数器 PC、加法器、多路选择器、隧道等组件。修改指令存储器 ROM 属性，设置数据位宽为 32 位，地址宽度为 16 位，片选信号为高电平有效。所有寄存器的数据宽度都是 32 位；程序计数器 PC 的片选信号为高电平有效，使能信号始终有效，上升沿触发。输入引脚初始地址 Initial Address、立即数 Imm 和 BusA

的数据位宽度设置为 32 位。

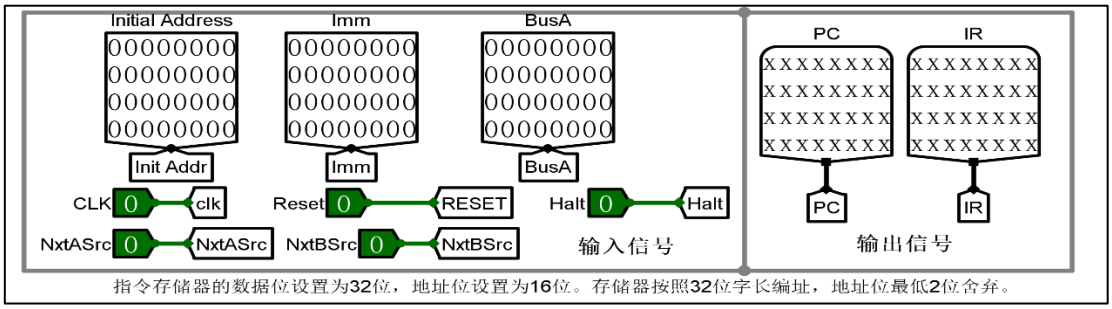

图 5.8　IFU 取指令部件引脚布局图

设置指令存储器 ROM 的容量为 256KB，定义数据字长为 32 位，则地址位宽为 16 位。在 RISC-V 程序代码中，指令地址的位宽是 32 位，因此可把指令地址中高位[31:18]和最低两位[1:0]可舍弃，只把 PC[17:2]赋值到指令存储器的地址输入端口 A[15:0]即可。在指令存储器 ROM 中加载数据文件：lab5.3.hex，改变输入数据，观察输出结果，填写 IFU 测试数据表。验证电路的正确性，记录测试数据。

表 5.4 IFU 测试数据表

	输入						输出（边沿信号有效前）	
序号	Reset	Initial Address	Imm	BusA	NxtASrc	NxtBSrc	PC	IR
1	1	0x400	0	0	0	0		
2	0	0	0	0	0	0		
3	0	0	0	0	0	0		
4	0	0	0x7f8	0	0	1		
5	0	0	0x20c	0	0	1		
6	1	0x1500	0	0	0	0		
7	0	0	0	0	0	0		
8	0	0	0	0	0	0		
9	0	0	0	0x3fbc0	1	1		
10	0	0	0	0	0	0		
11	0	0	0x1fc	0	0	1		
12	0	0	0	0x150c	1	1		
13	0	0	0x0c00	0	1	1		
14	0	0	0	0	0	0		
15	0	0	0	0	0	0		

16	0	0	0xffffff9c8	0	0	1		
17								

封装子电路时，需把时钟信号和复位信号定义成输入引脚。为了方便观察程序代码执行情况，需要把指令存储器移出 IFU 部件单独放置。可修改主电路名称为 “IFU”，保存电路设计文件为 lab5.3.circ。

4. 取操作数部件 IDU 实验

不同类型的指令取操作数的过程大致相同，根据图 5.7 所示，取操作数的部件主要包括指令译码、立即数扩展、寄存器堆读取，以及 ALU 操作数选择等部件。由于 RV32I 是 32 位定长指令，指令译码通过分线器生成，寄存器堆读写电路已经在实验 3 中完成，在本次实验中需要完成立即数扩展器部件。

根据图 5.7 所示，ALUAsrc 和 ALUBsrc 用来控制两个多路选择器的输出数据， ALUASsrc 控制 1 个两路选择器，当 ALUASsrc=0 时，选择 BusA 输出到 ALU 的操作数 A 口，当 ALUASsrc=1 时输出 PC。ALUBSrc 控制 1 个四路选择器，当 ALUBSsrc=00 时选择 BusB 输出到 ALU 的操作数 B 口，当 ALUBSsrc=01 时选择输出常数 4，当 ALUBSsrc=10 时选择输出 32 位立即数 Imm。

立即数是 ALU 一个数据输入源。在 RV32I 指令集中除了 R 型指令外，其它 5 种指令都带有立即数，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[31], Instr[7], Instr[30:25], Instr[11:8], 1'b0};
immJ = {11{Instr[31]}, Instr[31], Instr[19:12], Instr[20], Instr[30:21], 1'b0};
```

其设计示意图如图 5.9 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。

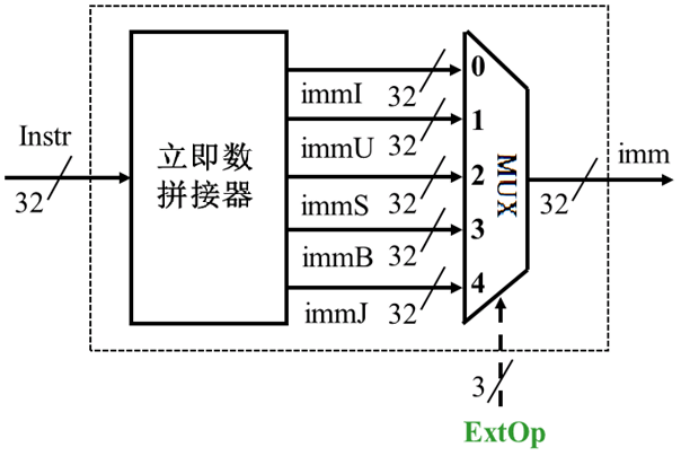

图 5.9　立即数扩展器示意图

实验步骤：

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。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。添加标识符和电路功能描述文字。改变 ExtOp 输入数据，观察输出信号值，记录测试数据，验证电路的正确性。如图 5.14 所示封装子电路，保存电路设计文件。

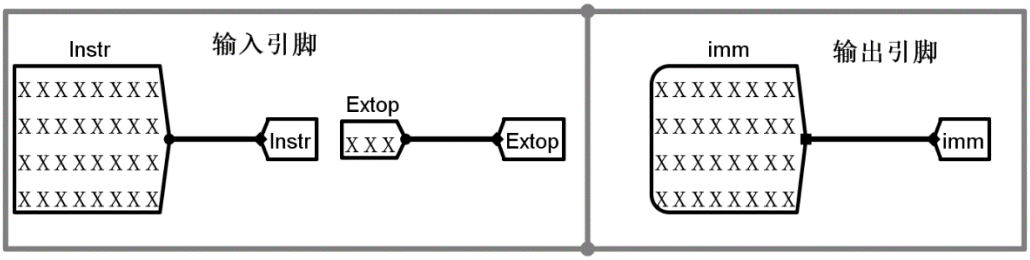

图 5.13 立即数扩展器引脚图

图 5.14 立即数扩展器封装图

2） 修改寄存器堆子电路。在 Logisim 中添加一个名为“RegFile”的子电路，在 Logisim 项目中选择加载 lab3.4.circ 库文件，将电路复制到 RegFile 子电路中，并复制相同的子电路外观，然后卸载 lab3.4.circ 库文件。在 RSIC-V 体系架构中，要求 0 号寄存器硬设计为零，因此需要修改寄存器堆子电路，使得 0 号寄存器始终为 0，设置寄存器堆中的寄存器为上升沿触发。

3） 设计“IDU”子电路。在 Logisim 中主电路的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 RegFile 子电路、多路选择器、输出引脚和隧道等组件。首先设计指令译码器件，根据图 5.6 所示的指令字段的位置分布，将输入指令寄存器 IR 利用分线器分解出 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。修改组件属性，进行线路连接，添加标识符和电路功能描述文字。改变输入指令及控制信号的输入数据，观察输出信号值，记录测试数据，验证电路的正确性。封装子电路如图 5.18 所示，保存电路设计文件为 lab5.4.circ。

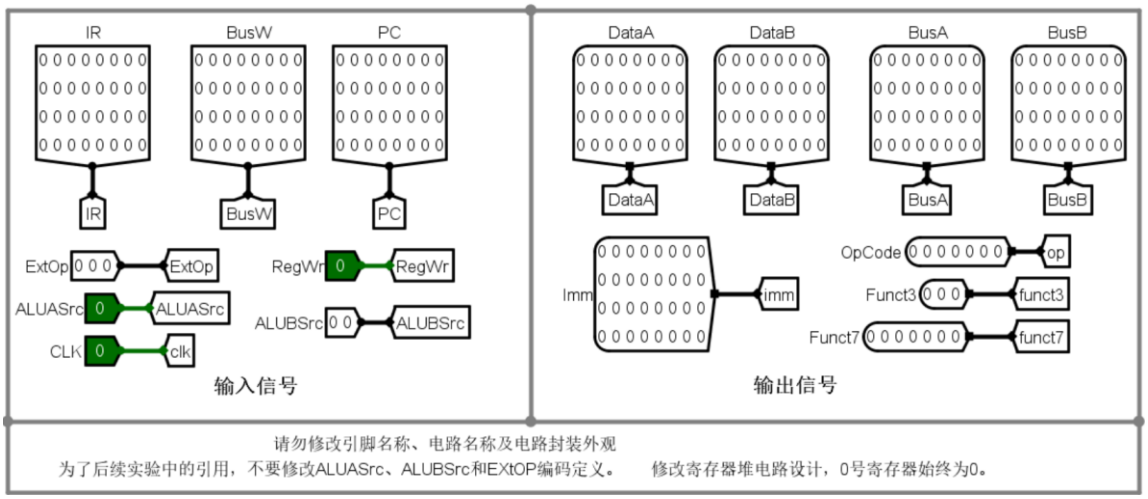

图 5.17　IDU 取操作数部件布局引脚图

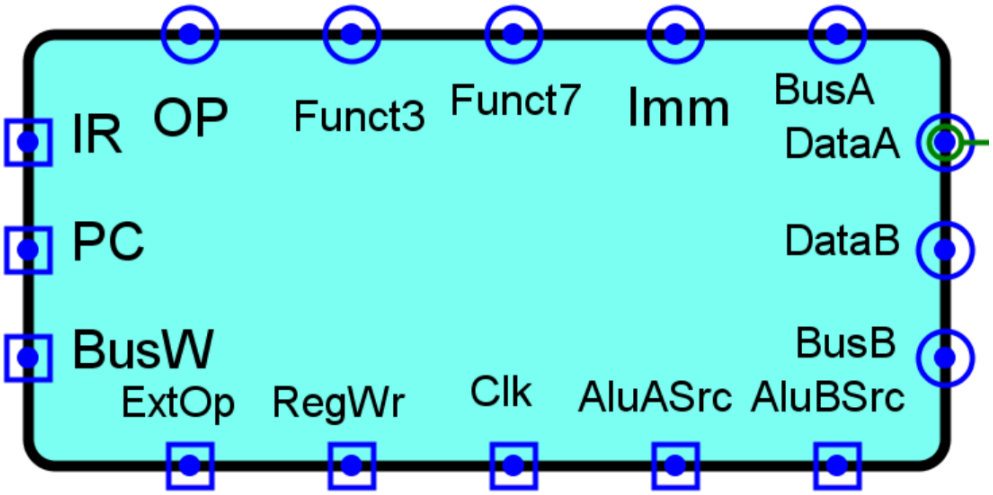

图 5.18　IDU 取操作数部件封装图

在指令寄存器 IR 依次输入下列数据和控制信号，单步执行后，根据 RV32I 指令集手册，按照指令码、功能码写出指令名称和功能，写出 DataA、DataB 的输出值，填写表 5.5。

表 5.5　取操作数部件输入输出数据列表

PC	IR	BusW	控制信号	指令功能	DataA、DataB 输出值
0	fedca2b7	fedca000	ExtOp=1 RegWr=1 ALUASrc=0 ALUBSrc=2		
4	f9c28293	fedc9f9c	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=2		
8	01006013	10	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=2		
c	06502223	64	ExtOp=2 RegWr=0 ALUASrc=0 ALUBSrc=2		
10	06400303	ffffff9c	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=2		
14	06601423	68	ExtOp=2 RegWr=0 ALUASrc=0 ALUBSrc=2		
18	06405383	9f9c	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=2		
1c	06701623	6c	ExtOp=2 RegWr=0 ALUASrc=0 ALUBSrc=2		
20	4042d413	ffedc9f9	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=2		
24	006444b3	123665	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=0		

28	00649533	50000000	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=0		
2c	008505b3	4fedc9f9	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=0		
30	00b2a633	1	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=0		
34	00b2b6b3	0	ExtOp=0 RegWr=1 ALUASrc=0 ALUBSrc=0		
38	40b287b3	aeeed5a3	ExtOp=2 RegWr=0 ALUASrc=0 ALUBSrc=0		
3c	06f02823	70	ExtOp=3 RegWr=0 ALUASrc=0 ALUBSrc=2		
40	0067c263	1	ExtOp=3 RegWr=0 ALUASrc=0 ALUBSrc=0		
44	0067d263	1	ExtOp=4 RegWr=1 ALUASrc=1 ALUBSrc=0		
48	0040086f	4c	ExtOp=0 RegWr=1 ALUASrc=1 ALUBSrc=1		
4c	004808e7	50	ExtOp=1 RegWr=1 ALUASrc=1 ALUBSrc=1		
50	00001917	1050	ExtOp=0 RegWr=0 ALUASrc=0 ALUBSrc=3		

5. 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。大致可分为取指令 IFU、取操作数 IDU、执行指令 EX、访问存储器 M 和写回寄存器堆 WB 等阶段。支持 RV32I 中不同类型指令的单周期数据通路电路原理图如图 5.7 所示，主要部件在前述实验中已基本完成，如在 lab4.5 中完成了 32 位 ALU 的设计，在 lab5.2 中完成数据存储器的设计，在 lab5.3 中完成了取指令部件的设计，在 lab5.4 中完成了取操作数部件 IDU 的设计。在本次实验中需要先完成跳转控制部件的设计。

1） 设计跳转控制器子电路

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，编码定义如表 5.4 所示。提示：为了后续实验中的子电路直接引用，不要修改编码定义。

表 5.4 Branch 控制信号含义

Branch	NxtASrc	NxtBSrc	指令跳转类型
000	0	0	非跳转指令
001	0	1	jal：无条件跳转 PC 目标
010	1	1	jalr：无条件跳转寄存器目标
100	0	Zero	beq：条件分支，等于
101	0	! Zero	bne：条件分支，不等于
110	0	Result[0]	blt,bltu：条件分支，小于
111	0	Zero | (! Result[0])	bge,bgeu：条件分支，大于等于

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 5.4 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。改变 Branch、Zero、Result0 输入数据，观察输出信号值，记录测试数据，验证电路的正确性。如图 5.16 所示封装子电路。

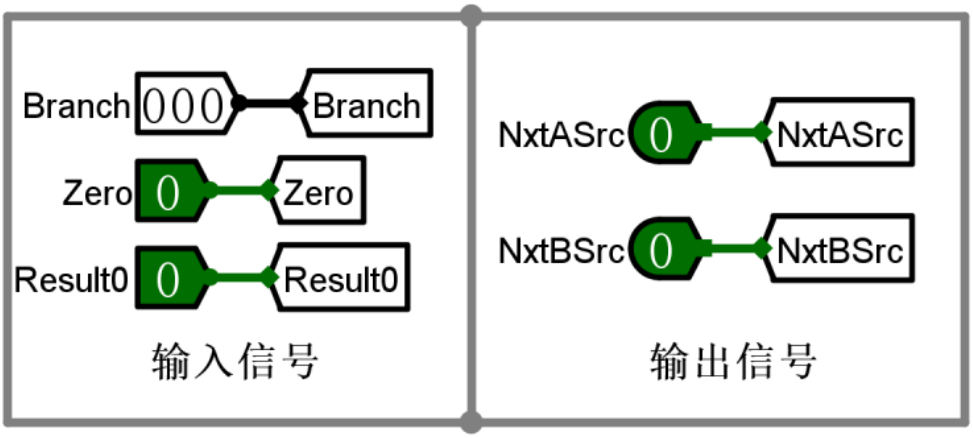

图 5.15 Branch 控制器引脚图

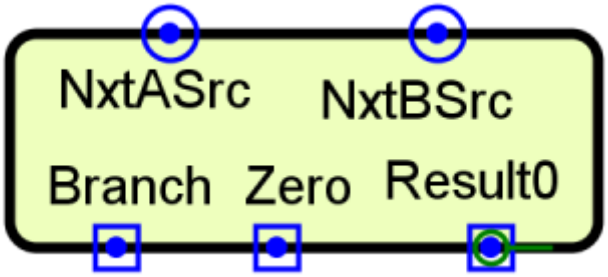

图 5.16　Branch 控制器封装图

2） 修改取指令子电路

在 Logisim 中添加一个名为“IFU”的子电路，在 Logisim 项目中选择加载 lab5.3.circ 库文件，将库文件中的电路复制到 IFU 子电路中，并复制相同的子电路外观，然后卸载 lab5.3.circ 库文件。为了便于观察指令的执行状态，将指令存储器移出 IFU 子电路，单独部署在数据通路中，移出指令存储器后，在 IFU 电路组件中删除 IR 输出端口，并修正 IFU 子电路外观。

3） 数据通路实验

在 Logisim 主电路的工作区中构建相应电路。引脚布局如图 5.19 所示，在 Project 中添加 lab4.5、lab5.2、lab5.4 等 Logisim 库，在主电路工作区中放置 IFU 取指令部件子电路、指令存储器、Branch 子电路、lab5.4 库文件中的 IDU 子电路、lab4.5 库文件中的 ALU 子电路和 lab5.2 库文件中的数据存储器 DataRAM 子电路。添加输入输出引脚、隧道和集线器等组件；修改指令存储器的片选信号 sel 为高电平有效，且连接到常量 1 始终有效。设置数据存储器的清零信号连接到复位信号 ReSet， 复位信号有效时清除数据。根据图 5.7 中的

数据通路原理图来设计电路图，并进行连接。定义指令存储器 ROM 的容量为 256KB，定义数据字长为 32 位，则地址位宽为 16 位，按照字长编址；将 PC 连接到指令存储器的地址端口时，只需选择 PC[17:2]，最低 2 位舍弃。定义数据存储器的容量是 256KB，按照字节编制，地址位宽是 18 位。将 ALU 的输出连接到数据存储器的地址端时，需要连接其低 18 位数据。添加标识符和电路功能描述文字。

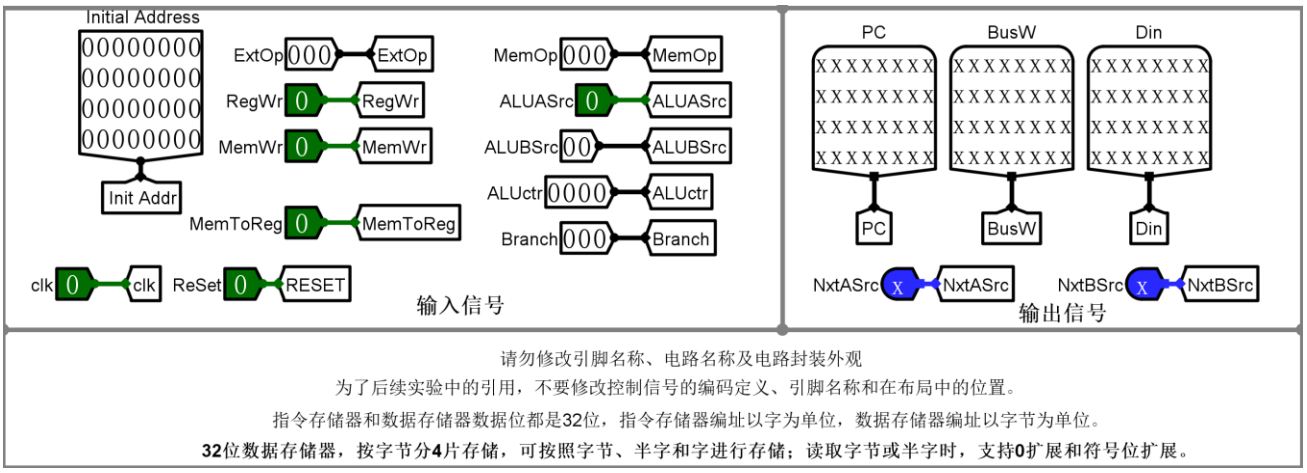

图 5.19　数据通路布局引脚图

4）　数据通路测试

在取指令部件 IFU 的指令存储器第 0x100 单元开始，顺序写入表 5.6 中的 RV32I 指令的机器代码，或者加载程序代码镜像文件 lab5.5.hex，其内容如下：

v2.0 raw

256*0

fedca2b7 f9c28293 01006013 06502223 06400303 06601423 06405383

06701623 4042d413 006444b3 00649533 008505b3 00b2a633 00b2b6b3

40b287b3 06f02823 0067c263 0067d263 0040086f 004808e7 00001917

加载指令代码后，设置初始地址为 0x400，设置复位信号 ReSet 高电平有效，单击时钟信号，读取指令初始地址到 PC 寄存器中，然后取消复位信号。依次读出指令存储器中的指令，读出指令后，根据表 5.6 中给出的控制信号值，单步执行，填写 PC、BusW、Din、NxtASrc、NxtBSrc 等信号的输出值，观测寄存器堆中寄存器和数据存储器中存储的数据，验证电路的正确性。保存电路设计文件为 lab5.5.circ。

表 5.6　14 条 RV32I 指令代码及控制信号赋值

序号	指令	控制信号赋值，未列出的控制信号值为 0。	输出信号值 PC、BusW、Din、NxrASrc、NxtBSrc
1	fedca2b7	ExtOp=1，RegWr=1，ALUBSrc=2，ALUctr=f	
2	f9c28293	RegWr=1，ALUBSrc=2	

3	01006013	RegWr=1，ALUBSrc=2，ALUctr=6	
4	06502223	ExtOp=2，MemWr=1，ALUBSrc=2	
5	06400303	MemOp=5，RegWr=1，ALUBSrc=2，MemToReg=1	
6	06601423	ExtOp=2，MemWr=1，MemOp=6，ALUBSrc=2	
7	06405383	MemOp=2，RegWr=1，ALUBSrc=2，MemToReg=1	
8	06701623	ExtOp=2，MemWr=1，MemOp=6，ALUBSrc=2	
9	4042d413	RegWr=1，ALUBSrc=2，ALUctr=d	
10	006444b3	RegWr=1，ALUctr=4	
11	00649533	RegWr=1，ALUctr=1	
12	008505b3	RegWr=1	
13	00b2a633	RegWr=1，ALUctr=2	
14	00b2b6b3	RegWr=1，ALUctr=3	
15	40b287b3	RegWr=1，ALUctr=3	
16	06f02823	ExtOp=2，MemWr=1，ALUBSrc=2	
17	0067c263	ExtOp=3，ALUctr=2，Branch=6	
18	0067d263	ExtOp=3，ALUctr=2，Branch=7	
19	0040086f	ExtOp=4，RegWr=1，ALUABSrc=1, ALUBSrc=1，Branch=1	
20	004808e7	RegWr=1，ALUABSrc=1, ALUBSrc=1，Branch=2	
21	00001917	ExtOp=1，RegWr=1，ALUABSrc=1, ALUBSrc=2	

四、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”。

2. 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？

3. 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？