实验 5：存储器及数据通路设计

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

Logisim 提供 RAM 和 ROM 两种存储器组件，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。

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。

将鼠标移到 RAM 或 ROM 组件处，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”。

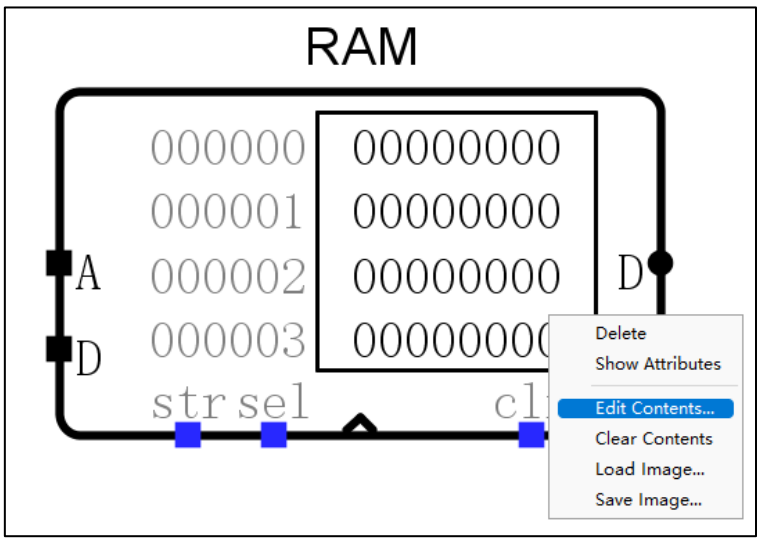


图 5.1 点击鼠标右键弹出组件菜单框

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。

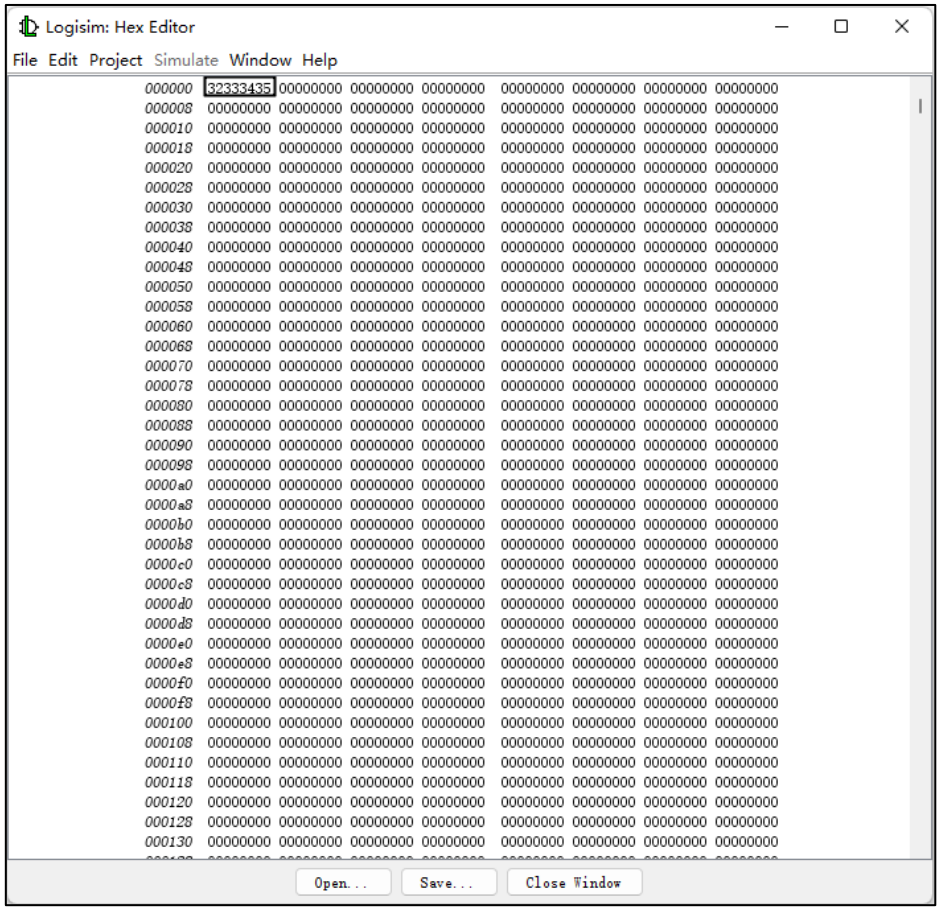


图 5.2 Logisim 十六进制编辑器

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。

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。考虑到需要支持按字节进行存取操作，使用 4 片数据位宽为 8 位 RAM，级联成 32 位数据存储器。

表 5.1 MemOp 控制信号含义

MemOp	指令	含义
000	lw,sw	存取 4 字节
001	lbu	读取最低 1 个字节数据，0 扩展到 4 字节
010	lhu	读取最低 2 个字节数据，0 扩展到 4 字节
101	lb,sb	存取最低 1 个字节，在读取时，按符号位扩展到 4 字节
110	lh,sh	存取最低 2 个字节，在读取时，按符号位扩展到 4 字节

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 5.4 所示。保存电路设计文件 lab4.circ。

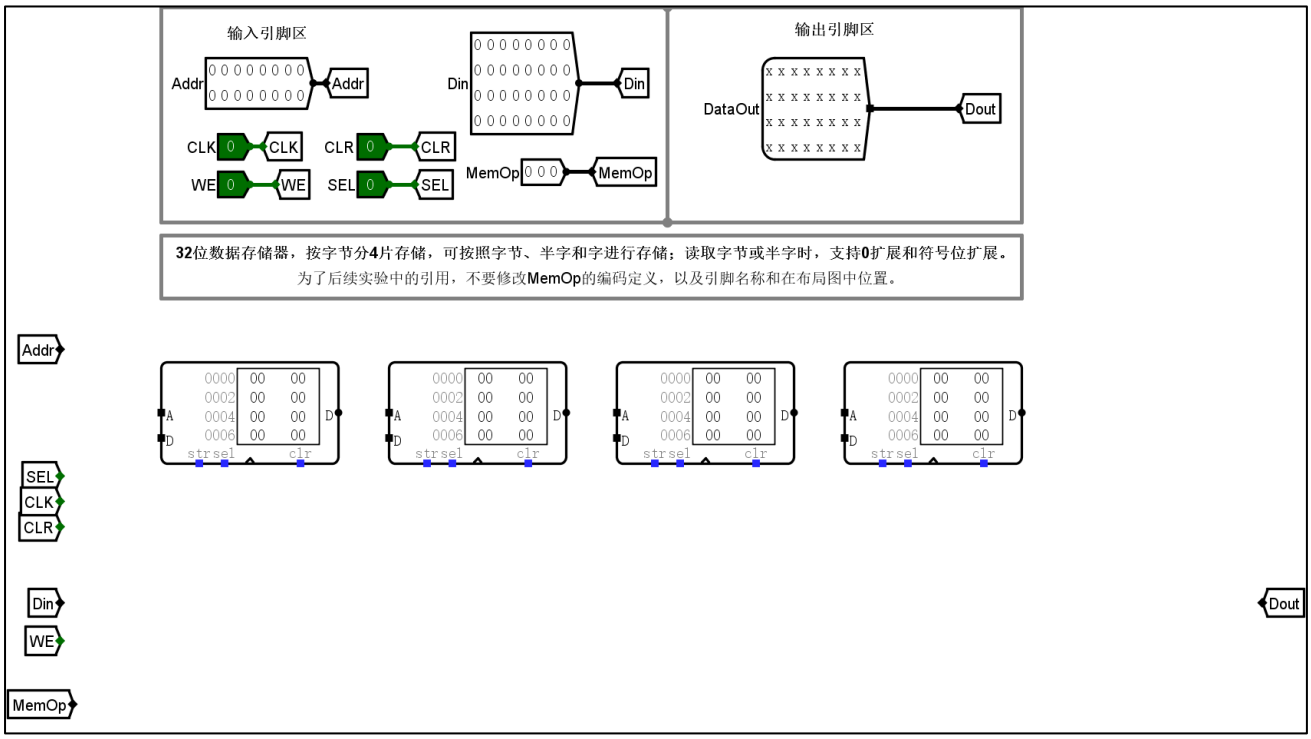


图 5.3　RAM 读写实验布局图

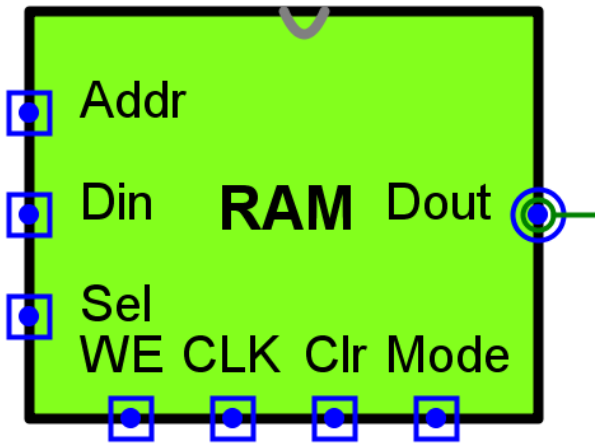


图 5.4　RAM 读写电路封装图

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。要求定义字长为 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，在工作区中按图 5.6 所示的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。

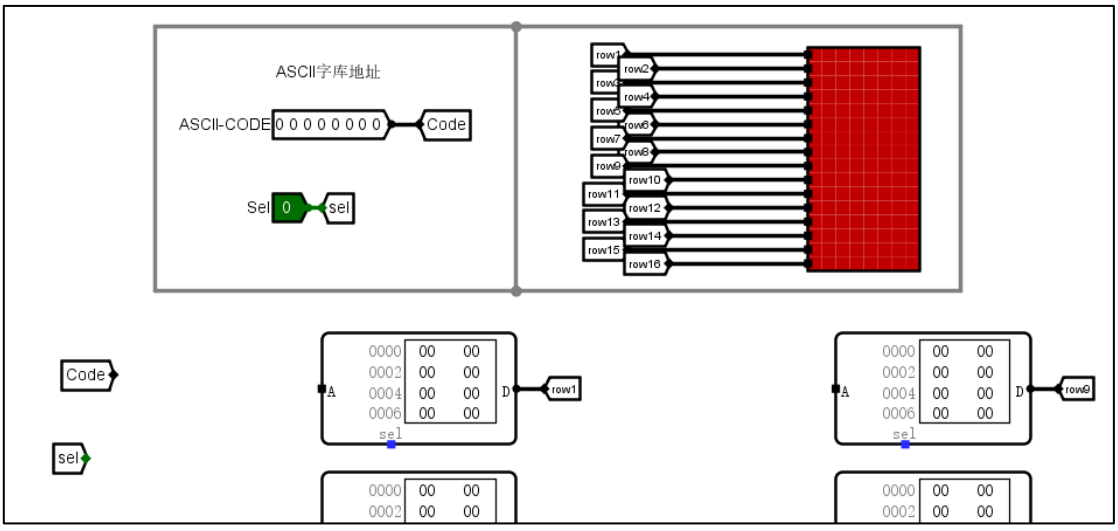


图 5.6　ROM 读写实验布局图

2. 取指令部件 IFU 实验

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。如图 5.7 所示，给出了对应的取指令部件示意图图。

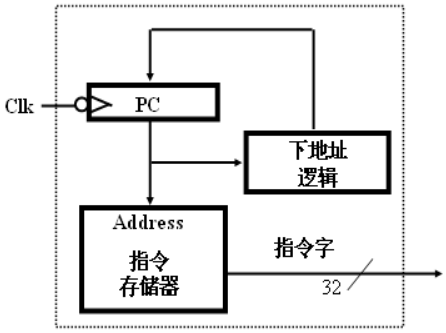


图 5.7 取指令部件示意图

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，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，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，编码定义如表 5.2 所示。提示：为了后续实验中的子电路引用，不要修改编码定义。

表 5.2 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

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。

这里需要注意，由于 Logisim RAM 和 ROM 组件的地址编码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 中的一个存储单位相当于按字节编址的 RISC-V 架构中的 4 个存储单位。而在 RISC-V 程序中指令和数据的地址都以字节为单位计算，因而，将 RISC-V 中指令和数据的地址转换为 Logisim 中的地址时，需要将把指令和数据地址中的最低两位舍弃。同时需保证所有指令和数据都按 4 字节对齐，也即其地址值是 4 的倍数（最低两位总是 0）。因此在读取和写入数据存储器时，需要确保存取的地址是 4 的倍数。因为每条指令占 4 个字节，因此指令的读取地址能够保证总是 4 的倍数。

当定义指令存储器的地址位宽为 16 位时，32 位指令地址 PC[31:0]中高位[31:18]和最低两位[1:0]可舍弃，把 PC[17:2]赋值到指令存储器的地址输入端口 A[15:0]即可。

要求输出当前指令的地址 PC 和指令的内容 IR，以便后续指令的执行处理。

实验步骤：

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，连接组件，实现功能。提示：ROM 存储器的按照 32 位字长编址，寻址地址的低 2 位要舍弃。

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、NxtASrc 和 NxtBSrc 等输入值，观察输出结果，验证电路的正确性，记录测试数据。

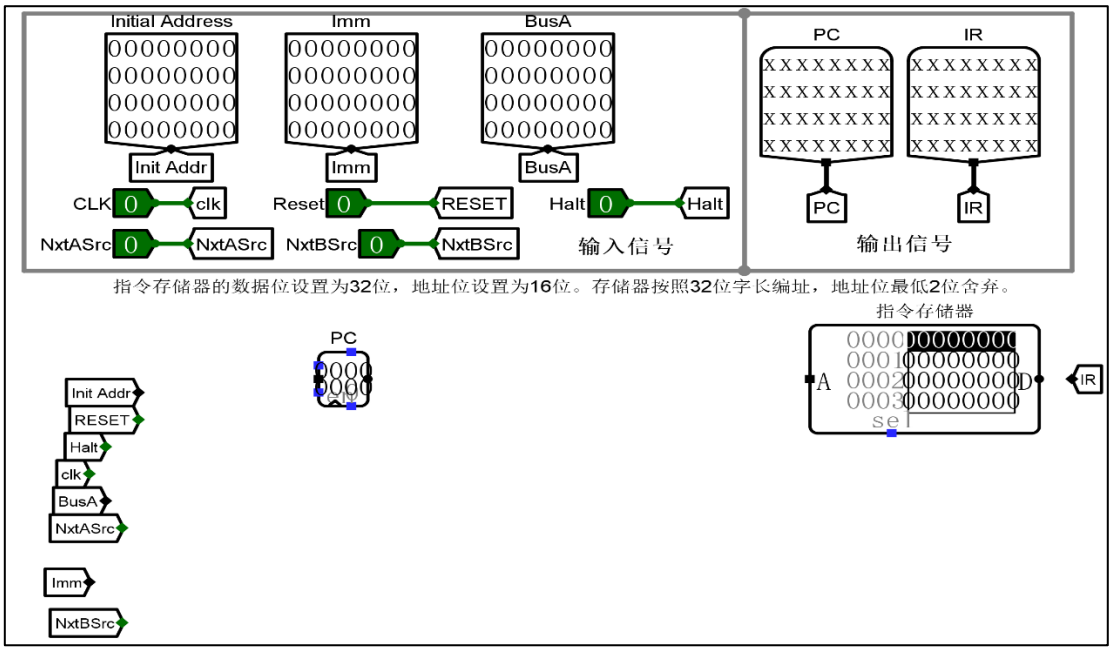


图 5.8　IFU 取指令部件引脚布局图

封装子电路时，需把时钟信号和复位信号定义成输入端，通常 IFU 部件中不包括指令存储器，输出信号只有 PC，封装图如图 5.9 所示。保存电路设计文件。

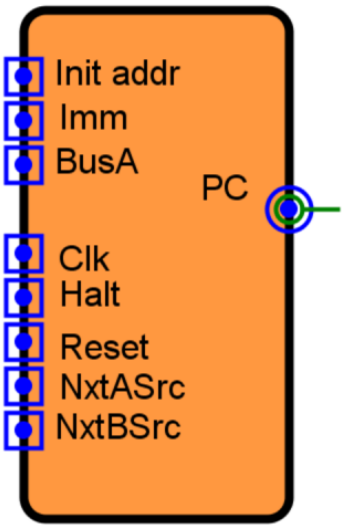


图 5.9　IFU 取指令部件封装图

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，可得到如图 5.10 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。

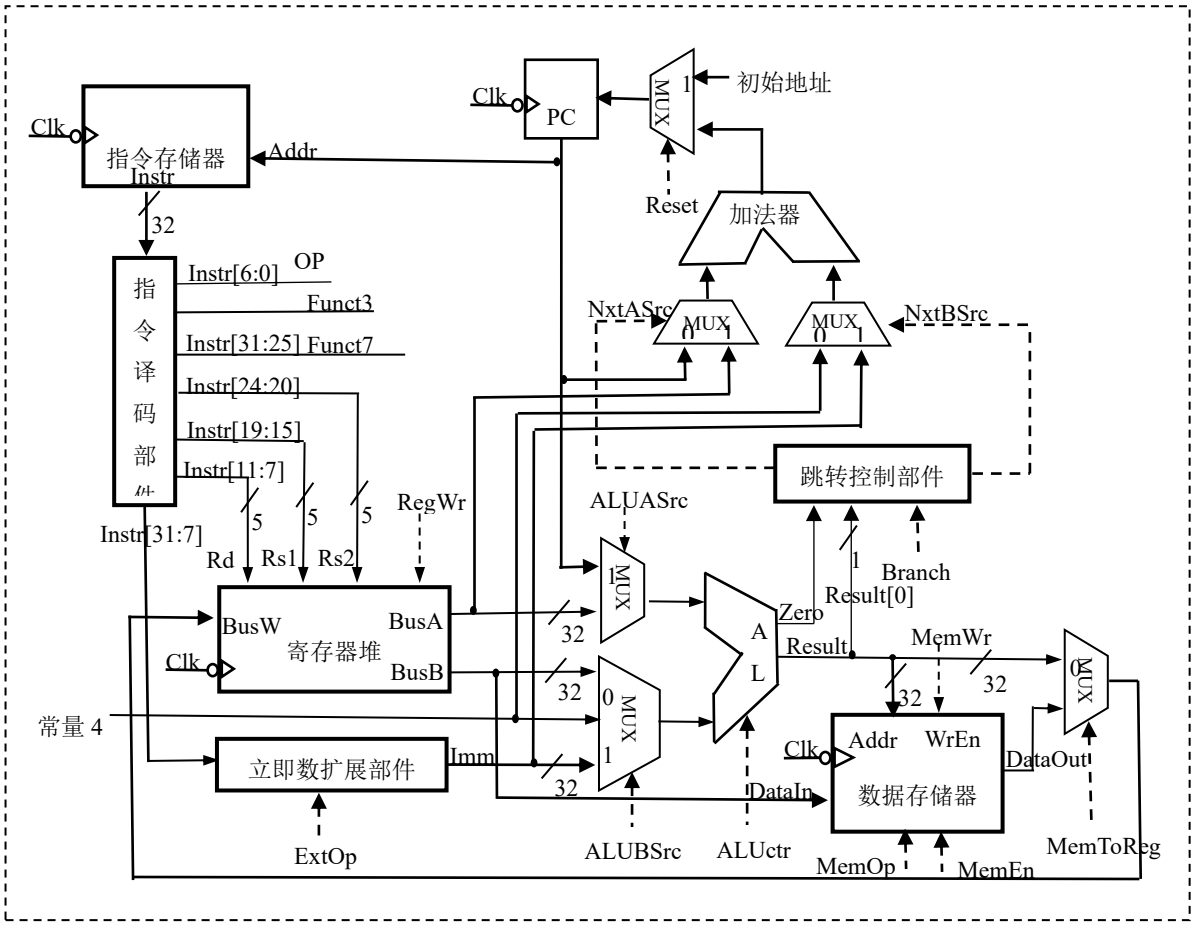


图 5.10　单周期 CPU 数据通路原理图

RISC-V 指令格式如图 5.11 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 5.11　RISC-V 指令格式

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};

其设计示意图如图 5.12 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。

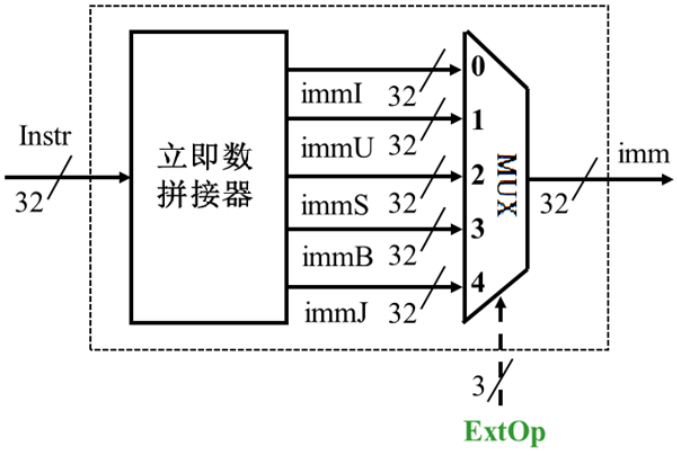


图 5.12　立即数扩展器示意图

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。

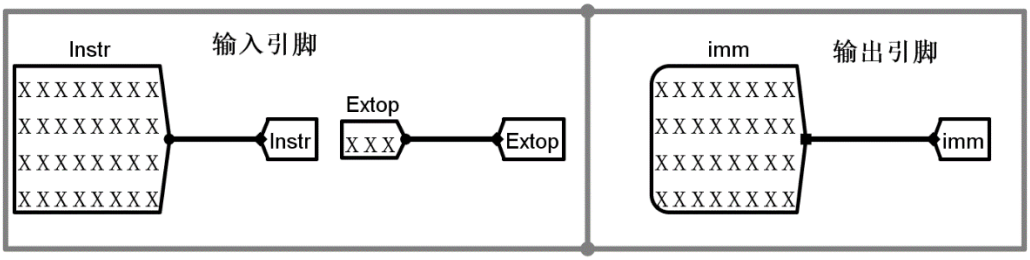


图 5.13 立即数扩展器引脚图

图 5.14 立即数扩展器封装图

2） 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。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。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，保存电路设计文件。

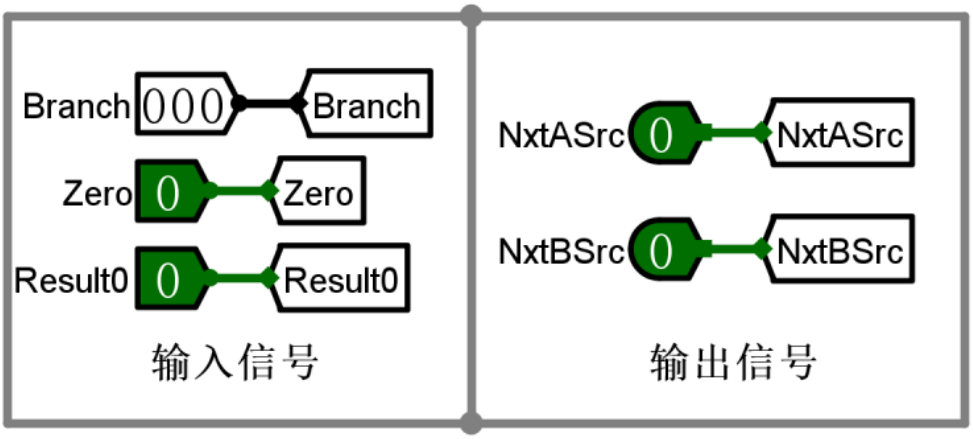


图 5.15 Branch 控制器引脚图

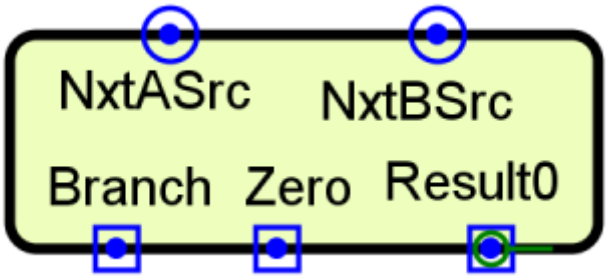


图 5.16　Branch 控制器封装图

3）　设计“IDU”子电路。首先根据图 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。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（用于跳转时计算返回地址 PC+4），为 10 时选择立即数 Imm。此处需要调用实验 3 中实现的寄存器堆子电路，在 Logisim 项目中选择加载 Logisim Library 命令，加载实验 3 的电路设计文件 lab3.circ。由于在 RSIC-V 架构中，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。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。

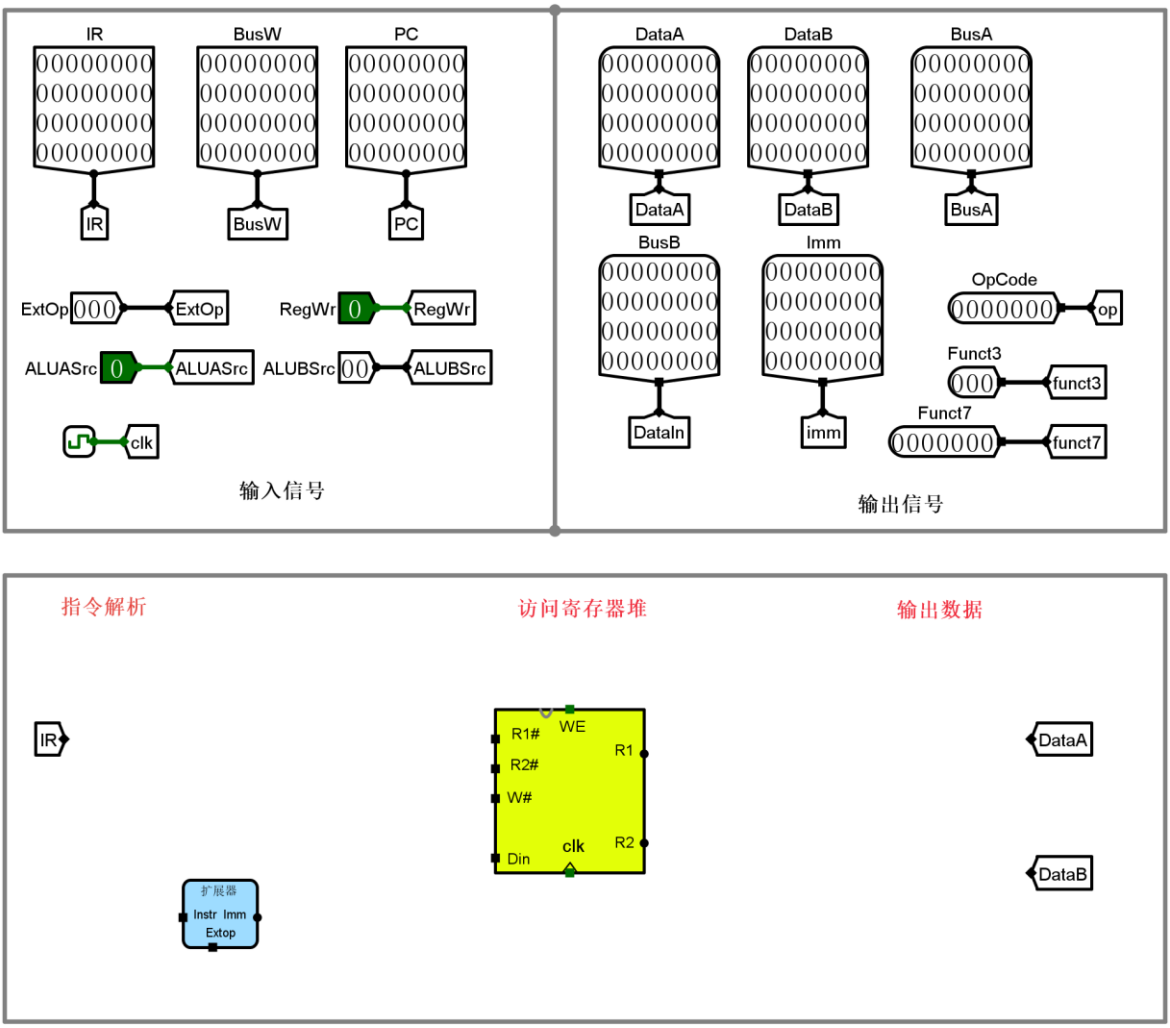


图 5.17　IDU 取操作数部件布局引脚图

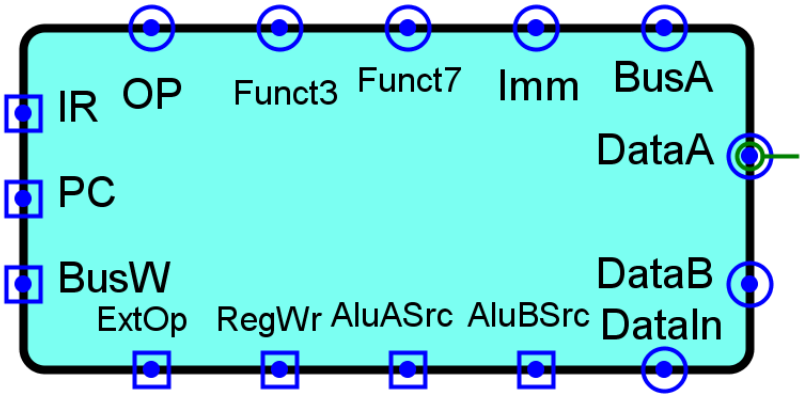
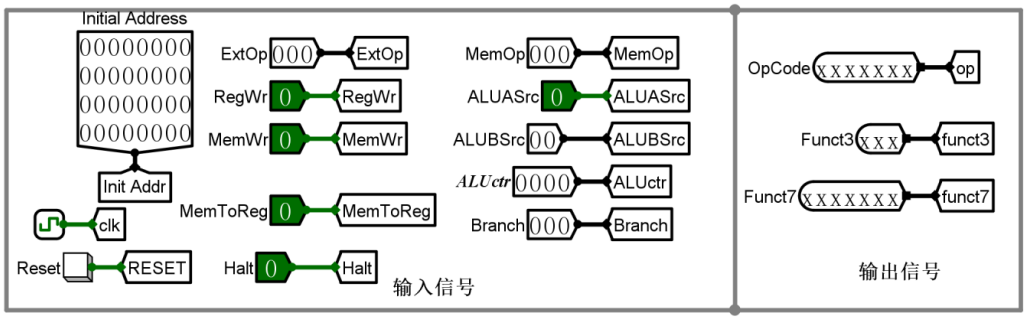


图 5.18　IDU 取操作数部件封装图

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，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。



数据通路设计，提示指令存储器和数据存储器数据位32位，编址以字为单位，地址信号低2位舍弃。

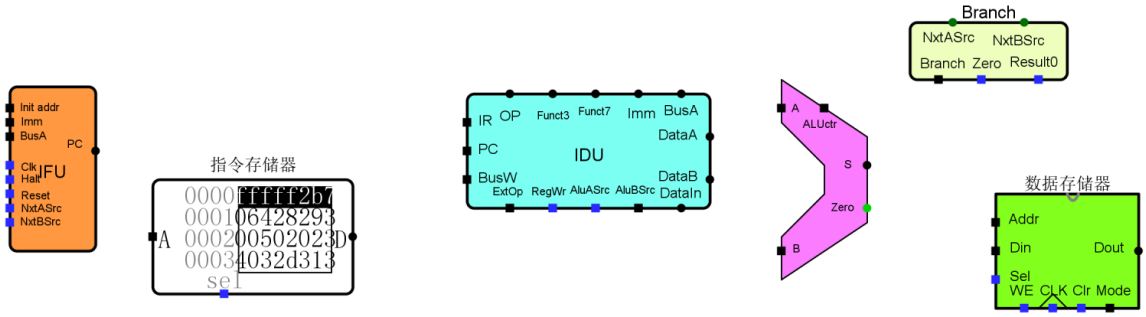


图 5.19　数据通路布局引脚图

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，其内容如下：

v2.0 raw

ffffff2b7

06428293

00502023

4032d313

00602223

00401383

fff3c413

00802423

007424b3

00902623

fc049ce3

03000567

00a02823

00064597

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。

表 5.3　14 条 RV32I 指令代码及控制信号赋值

序号	汇编指令	机器代码	控制信号赋值，未列出的控制信号值为 0	类型
1	lui x5,-1	0xfffff2b7	ExtOp=001，RegWr=1，ALUBSrc=10，ALUctr=1111	U
2	addi x5,x5,100	0x06428293	RegWr=1，ALUBSrc=10	I
3	sw x5,0(x0)	0x00502023	ExtOp=010，MemWr=1，ALUBSrc=10	S
4	srai x6,x5,3	0x4032d313	RegWr=1，ALUBSrc=10，ALUctr=1101	I
5	sw x6,4(x0)	0x00602223	ExtOp=010，MemWr=1，ALUBSrc=10	S
6	lh x7,4(x0)	0x00401383	RegWr=1，MemtoReg =1，MemOp =110，ALUBSrc=10	I
7	xori x8,x7,-1	0xfff3c413	RegWr=1，ALUBSrc=10，ALUctr=0100	I
8	sw x8,8(x0)	0x00802423	ExtOp=010，MemWr=1，ALUBSrc=10	S
9	slt x9,x8,x7	0x007424b3	RegWr=1，ALUctr=0010	R
10	sw x9,12(x0)	0x00902623	ExtOp=010，MemWr=1，ALUBSrc=10	S
11	bne x9,x0,label2	0xfc049ce3	ExtOp=011，ALUctr=0010，Branch=1010	B
12	jalr x10,x0,48	0x03000567	RegWr=1，ALUASrc =1，ALUBSrc=01，Branch=010	I
13	sw x10,16(x0)	0x00a02823	ExtOp=010，MemWr=1，ALUBSrc=10	S
14	auipc x11,100	0x00064597	ExtOp=001，RegWr=1，ALUASrc=1, ALUBSrc=10	U

表 5.4　指令执行时信号赋值

序号	汇编指令	PC	BusW	立即数	寄存器堆	数据存储器
1	lui x5,-1				X5：	
2	addi x5,x5,100					
3	sw x5,0(x0)					0:
4	srai x6,x5,3				X6：	
5	sw x6,4(x0)					1:
6	lh x7,4(x0)				X7：	
7	xori x8,x7,-1				X8：	
8	sw x8,8(x0)					2:
9	slt x9,x8,x7				X9：	
10	sw x9,12(x0)					3:
11	bne x9,x0,label2					
12	jalr x10,x0,48				X10：	

13	sw x10,16(x0)					4:
14	auipc x11,100				X11：	

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”。
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）？
3. 在 Risc-V 架构中，举例说明什么是伪指令？伪指令如何实现？
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？