第1章　存储系统实验

1.1 RAM 组件实验

1.1.1 实验目的

掌握 Logisim 平台中 RAM 组件的基本使用方法，进一步熟悉流水传输控制机制。

1.1.2 背景知识

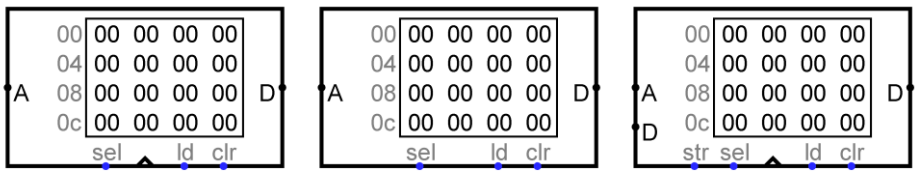

图1.1 RAM组件三种不同接口

RAM组件是Logisim内建库中最复杂的一个组件，它最多能存储2^{24}个存储单元（地址线宽度最大24位），每个存储单元位宽最大32位。电路可以加载和存储值在RAM中。

RAM 组件会用黑底白字显示当前存储单元的值，显示区域左侧是地址列表（灰色显示），所有数据均采用十六进制显示。用戳工具点击 RAM 组件上的地址或存储内容后可以直接利用键盘修改显示区域地址和对应的存储内容，也可以通过菜单工具弹出十六进制编辑器直接编辑修改的 RAM 存储内容。

RAM组件支持三种不同的接口模式，如图1.1所示，具体可以设置属性中的数据接口项，详细使用方式请参考**错误!未找到引用源。**Logisim库参考手册中第**错误!未找到引用源。**节的内容。

1.1.3 实验内容

在数据表示实验的工程文件 data.circ 中新增子电路，复制流水传输测试电路，改造该电路中编码流水传输第五阶段---显示阶段的功能，使得该阶段能将发送方发送过来的数据按原始地址顺序存放在一个 RAM 存储器中，RAM 组件地址线、数据线位宽与发送端 ROM 组件一致。为保证数据存储顺序，需要尝试改造各段流水接口部件，在传输汉字编码的同时增加传输汉字编码地址的逻辑。另需要考虑 RAM 部件何时写入数据，写入控制信号如何控制，时序信号如何连接等问题。请尝试利用三种不同的 RAM 接口形式的实现上述任务。

1.1.4 实验思考

（1） RAM 组件采用与流水接口相反的时钟触发是否可以，从同步时序的角度上思考这样做会带来什么问题？

1.2 存储器扩展实验

1.2.1 实验目的

理解存储系统进行位扩展、字扩展的基本原理，能利用相关原理解决实验中汉字字库的存储扩展问题，并能够使用正确的字库数据填充。

1.2.2 实验内容

在**错误!未找到引用源。**汉字编码实验中实现了汉字字符的 32*32 点阵显示，一个汉字编码的显示需要 32*32=1024 比特的点阵信息，为实现汉字字型码在组合逻辑电路中的直接显示，本实验在 Logisim 平台中利用 32 个 32 位 ROM 组件按位扩展的方式构造了位宽为 1024 的存储系统用于存储汉字字库，将所有汉字字形码存储在该存储系统中，并利用汉字区位码进行索引，给出一个区位码，可一次性取出 1024 位字型码进行显示。

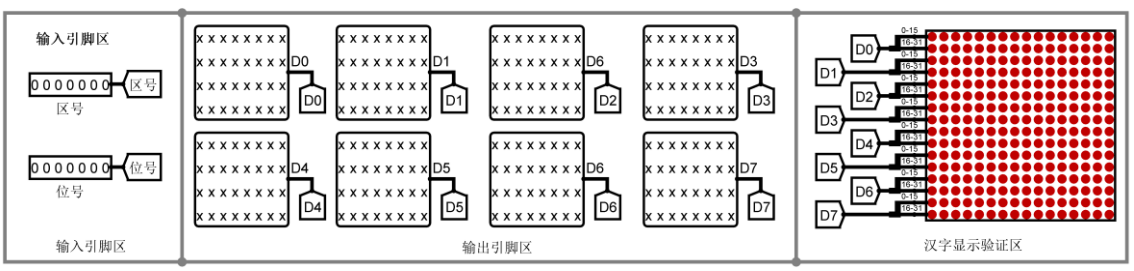

图 1.2 16*16 点阵字库电路输入输出引脚

现有如下 ROM 组件，4 片 4K*32 位 ROM，7 片 16K*32 位 ROM，请在 Logisim 平台构建 GB2312 汉字编码的 16*16 点阵汉字字库，电路输入为汉字区号和位号，电路输出为 8*32 位（16*16=256 位点阵信息），待完成的字库电路输入输出引脚图 1.2 所示，具体参见工程文件中的 storage.circ 文件，图中左侧是输入引脚，分别对应汉字区位码的区号和位号，中间区域为 8 个 32 位的输出引脚，可一次性提供一个汉字的 256 位点阵显示信息，右侧是实际显示区域，用于观测汉字显示是否正常。待完成字库子电路封装已经完成，请勿修改以免影响后续自动测试功能。

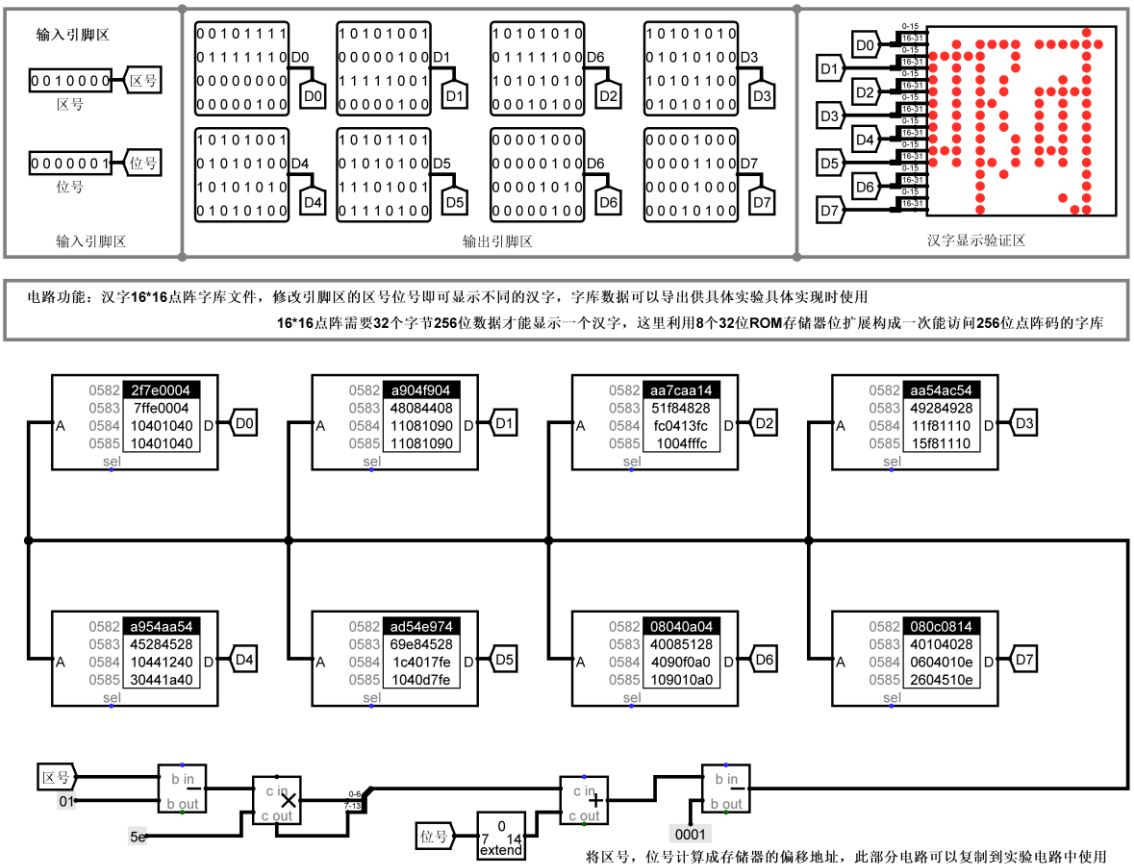

图 1.3 16*16 点阵汉字字库参考实现

本实验的主要目的是进行存储器字扩展（容量扩展，地址总线扩展），实验工程文件 stroage.circ 中已经提供了一个参考实现，如图 1.3 所示。但使用的组件和本实验的要求略有不同，请参考该设计完成本实验。实验所需的点阵信息数据文件已经提供，另外区位码转存储器地址的电路也可一并参考使用。

设计实现对应的汉字字库后，可以在字库测试电路进行功能测试，如图 1.4 所示。测试时按下 CTRL+T(command+T　MAC)键启动时钟自动仿真即可，该电路自动从 ROM 电路中取出不同的汉字 GB2312 编码送待测字库和标准字库进行对比显示，通过对比上下两个显示区内容是否一致即可验证字库功能的正确性。

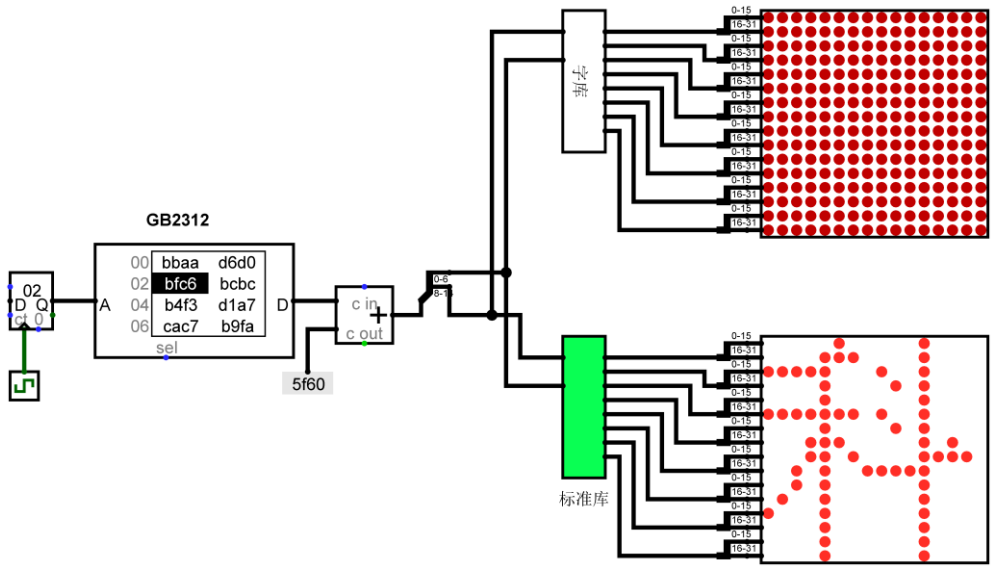

图 1.4 16*16 点阵字库功能测试电路

1.2.3 实验思考

(1) 如果存储芯片的存储容量或数据位宽超过 CPU，是否可以使用吗？如果可以使用，如何使用？

(2) 内存芯片、固态盘上的闪存芯片是如何进行存储扩展的？

1.3 MIPS RAM 设计实验

1.3.1 实验目的

学生理解主存地址基本概念，理解存储位扩展基本思想，并能利用相关原理构建能同时支持字节、半字、字访问的存储子系统。

1.3.2 实验内容

Logisim 中 RAM 组件只能提供固定的地址位宽，数据输出也只能提供固定的数据位宽，访问时无法同时支持字节/半字/字三种访问模式，实验要求利用 4 个 4K*8 位的 RAM 组件进行扩展，设计完成既能按照 8 位、也能按 16 位、也能按照 32 位进行读写访问的 32 位存储器，最终存储器引脚定义如图 1.5 所示，封装外观如图 1.6 所示。

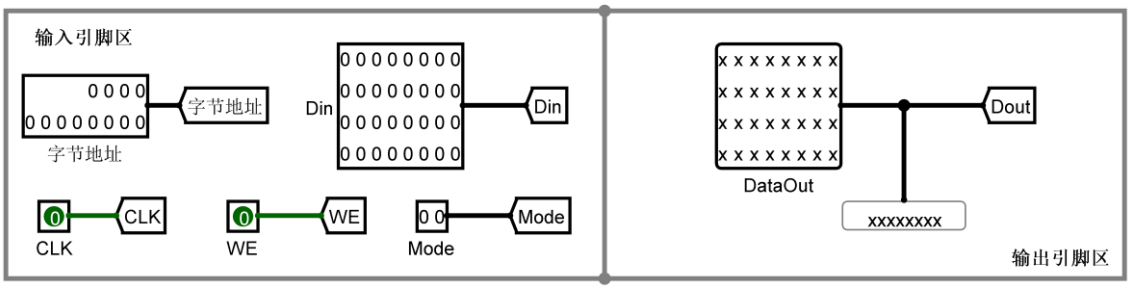

图 1.5 MIPS RAM 输入输出引脚

- Addr，位宽 12 位，字节地址输入（字访问时忽略最低两位，半字访问时忽略最低位，倒数第二位片选，字节访问时，低两位进行片选）；
- Din：位宽 32 位，写入数据 （不同访问模式有效数据均存放在最低位，高位忽略）；
- Dout：位宽 32 位，读出数据 （不同访问模式有效数据均存放在最低位，高位补零）；
- Mode: 位宽 2 位，访问模式控制位（00 表示字访问，01 表示 1 字节访问，10 表示 2 字节访问）；
- WE：位宽 1 位，写使能，1 表示写入，0 表示读出；

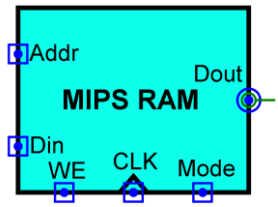

图 1.6 MIPS RAM 封装

对应子电路设计完成后，即可在同一工程文件中的 MIPS RAM 测试子电路中开启时钟自动仿真（Ctrl+K）进行功能自动测试，如图 1.7 所示。测试电路将自动完成对待测部件的写入和读出，并将写入的数据逐一读出并计算校验和，如果校验和和期望一致，说明电路功能正确，否则需要继续调试修改。

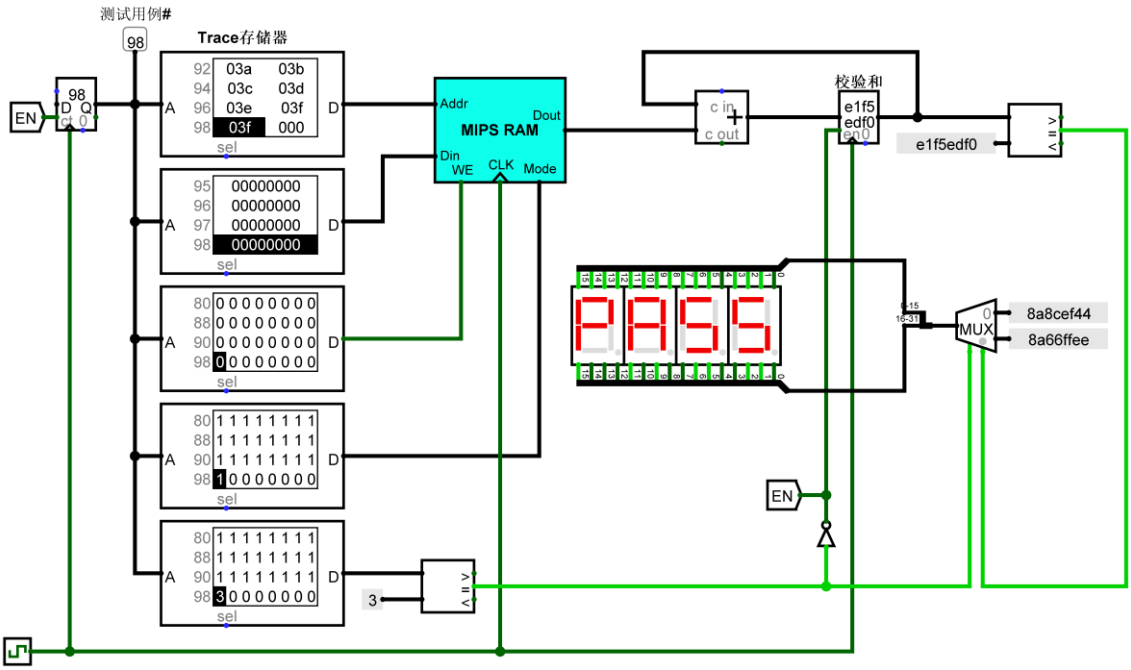

图 1.7 MIPS RAM 自动测试电路

1.3.3 实验思考

(1) 自动测试电路的原理是什么？

(2) 自动测试电路中校验和检测方式是否准确，为什么？

1.4 MIPS 寄存器文件设计实验

1.4.1 实验目的

学生了解寄 MIPS 寄存器文件基本概念，进一步熟悉多路选择器、译码器、解复用器等 Logisim 组件的使用，并利用相关组件构建 MIPS 寄存器文件。

1.4.2 背景知识

寄存器文件（寄存器堆）是 CPU 中通用寄存器的集合，以 MIPS 寄存器文件为例，MIPS 指令集支持 32 个通用寄存器，32 个通用寄存器均包含在寄存器文件中，其中每个寄存器的内容可通过对应的寄存器编号进行访问，类似于一个具有多个地址端口和多个数据端口的高速存储器。图 1.8 为 MIPS 寄存器文件的具体封装形式，其中 R1#和 R2#为读寄存器编号输入，对应编号的寄存器将通过输出引脚 R1，R2 输出；WE 为写使能信号，高电平有效；W#为写寄

存器编号输入；写入数据端口为 Din，在时钟配合下将数据写入对应寄存器。R1#、R2#、W# 的值由 MIPS 指令字中的相关字段的值来确定。

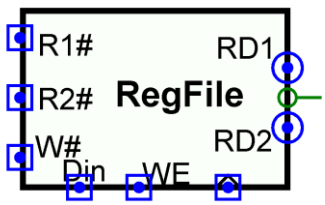

图 1.8 MIPS 寄存器文件封装

1.4.3 实验内容

利用 logisim 平台中构建一个 MIPS 寄存器组，内部包含 32 个 32 位寄存器，其具体引脚与功能描述如表 1.1。

表 1.1 寄存器文件端口与功能描述

引脚	输入/输出	位宽	功能描述
R1#	输入	5	第 1 个读寄存器的编号
R2#	输入	5	第 2 个读寄存器的编号
W#	输入	5	写入寄存器编号
Din	输入	32	写入数据
WE	输入	1	写使能信号，为 1 时在 CLK 上跳沿将 Din 数据写入 W#寄存器
CLK	输入	1	时钟信号，上跳沿有效
RD1	输出	32	R1#寄存器的值，MIPS 寄存器文件中 0 号寄存器的值恒零
RD2	输出	32	R2#寄存器的值，MIPS 寄存器文件中 0 号寄存器的值恒零

相关输入输出引脚以及子电路封装已在工程文件为 storage.circ 的 MIPS regfile 子电路中实现，具体输入输出引脚图 1.9 所示。

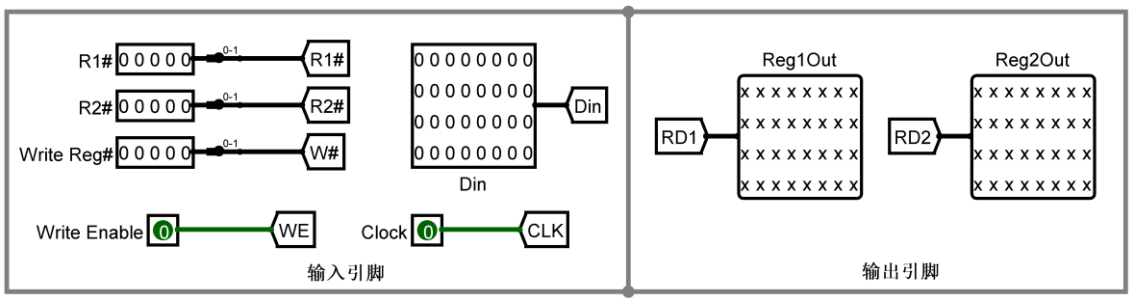

图 1.9 MIPS 寄存器文件输入输出引脚

为减少实验中绘图工作量，实验工程文件中对 5 位寄存器地址进行了简化，具体见左上角的引脚示意图，图中采用分线器将 5 位寄存器编号低两位引出，实际只使用了两位编号，所以最终只需实现 4 个寄存器，其中 0 号寄存器的值仍然是恒零。后续 CPU 实验中如需使用 32 个寄存器的 MIPS 寄存器文件组，将提供标准组件供用户使用。

1.4.4 实验思考

(1) 零号寄存器的值恒零，具体实现是如何实现的，如何实现成本最优，MIPS 指令集中为什么要引入一个恒零的寄存器？

(2) 本实验中利用译码器和解复用器均可实现 MIPS 寄存器文件中的写入控制，尝试用两种不同的方案实现。

1.5 Cache 硬件设计实验

1.5.1 实验目的

学生掌握 cache 实现的三个关键技术：数据查找，地址映射，替换算法，熟悉译码器，多路选择器，寄存器的使用，能根据不同的映射策略在 Logisim 平台中用数字逻辑电路实现 cache 机制。

1.5.2 实验内容

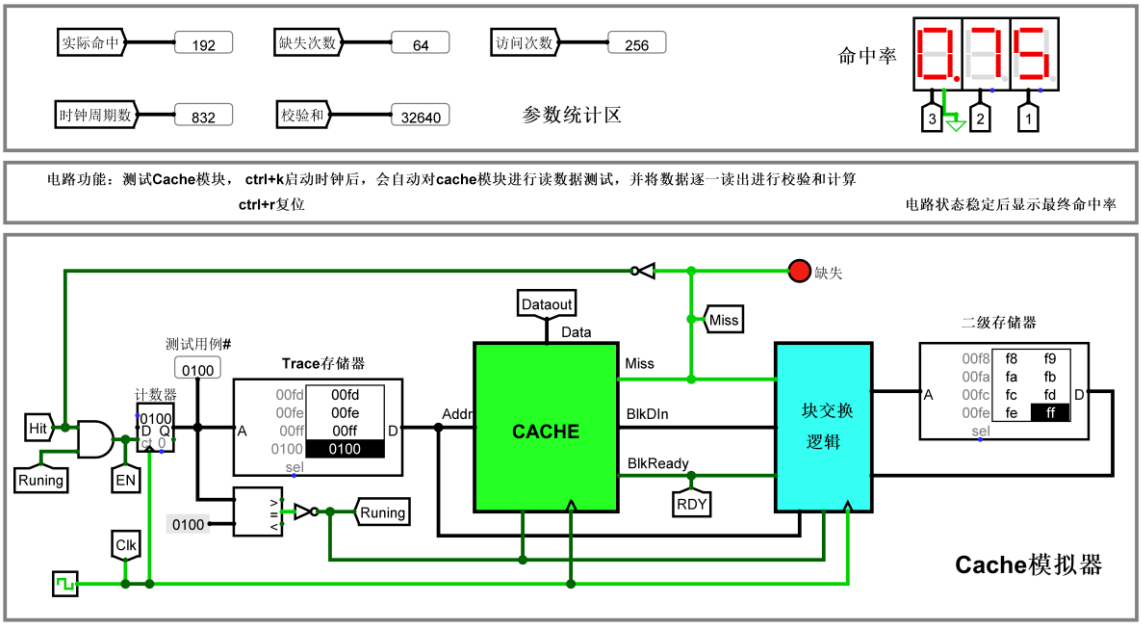

图 1.10 Cache 模块自动测试电路

图 1.10 给出了一个在 Logisim 中设计完成的 cache 系统自动测试电路，为简化实验设计，这里所有 cache 模块均为只读 cache（类似指令 cache），无写入机制。电路左侧计数器与存储器部分会在时钟驱动下逐一生成地址访问序列给 cache 模块。计数器模块的使能端受命中信号

驱动，缺失时使能端无效，计数器不计数，等待系统将待请求数据所在块从二级存储器中调度到 cache 后才能继续计数。cache 与二级存储器之间通过块交换逻辑实现数据块交换，由于二级存储器相比 cache 慢很多，所以一次块交换需要多个时钟周期才能完成，cache 模块判断数据块准备好的逻辑是 blkready 信号有效，该信号有效且时钟到来时，cache 将块数据从 BlkDin 端口一次性载入到对应 cache 行缓冲区中，此时 cache 数据命中，直接输出请求数据，解锁计数器使能端，继续访问下一个地址。

自动测试电路会逐一取出 trace 存储器中的主存地址去访问存储系统，并逐一将数据从 cache 模块取出送校验和计算电路计算校验和，当计数器值为 256 时会停止电路运行，此时所有存储访问的 cache 命中率将会在右上角 LED 数码管显示。本次实验的主要任务就是设计该电路的核心模块 cache 子电路。

实验要求：根据图 1.11 所示的 cache 模块输入输出引脚示意图，结合表 1.2 的引脚功能说明，实现只读 cache 模块，该 cache 模块共包括 8 个 cache 行，每个数据块包含包括 4 个字节共 32 位数据。

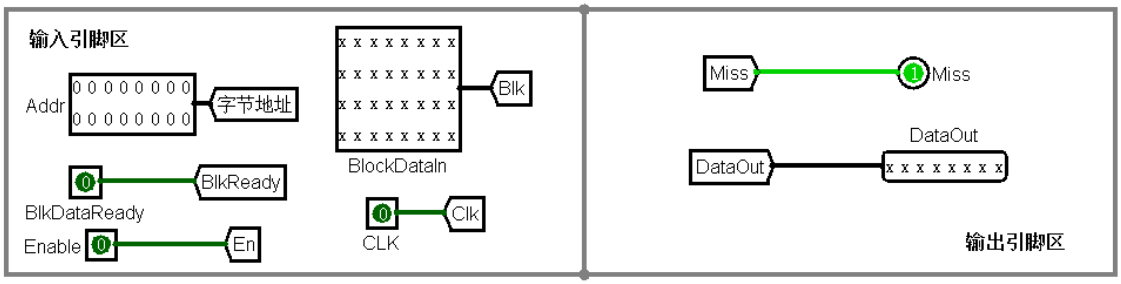

图 1.11 Cache 模块输入输出引脚

Cache 映射策略和调度策略共有如下几种组合：

1、 全相联映射（8 路组相联），LRU 调度；

2、 直接相联映射（1 路组相联）；

3、 组相联映射（2 路组相联，4 路组相联二选一），LRU 调度。

请在以上三个选项中人选一个进行电路实现，并在最终的测试电路中进行命中率测试，最终校验和应为 32640？

表 1.2 芯片引脚与功能描述

引脚	输入/输出	位宽	功能描述
Addr	输入	16	主存地址
BlkDataIn	输入	32	块数据输入
BlkDataReady	输入	1	块数据准备就绪
Enable	输入	1	使能信号，1：工作；0：输出高阻态
CLK	输入	1	时钟输入
Miss	输出	1	1：数据缺失；0：数据命中
DataOut	输出	8	数据输出

实验提示：为简化电路绘图工作量，可以将 cache 行模块化后进行复制，具体可以参考图

1.12 的 cache 行参考设计，图中存储器件全部采用寄存器实现，valid 数据位通过三态门输出到 V0，标记位通过三态门输出到 Tag0，淘汰标志位通过三态门输出到 C0，三个三态门全部由组索引信号 Set0 进行控制，也就是说只有当前第 0 组被选中时对应数据才能输出，同一组中所有标志位标记位均同时输出到多路并发比较电路；数据块副本通过三态门输出到 SlotData，输出控制由组索引信号和行选中信号 L0 逻辑与后得到，也就是说只有当前组第 0 组被选中且第 0 行命中时才将数据副本输出。图中 R0 信号表示当前组被替换的信号，如果此信号为 1，且组索引信号 Set0 有效则所有寄存器使能端为 1，时钟到来新的有效位、标记位、淘汰标志位、数据块副本均载入新数据，实现新数据块的载入。

完成一个 cache 槽设计后，可直接复制该电路 8 份，分别修改其中信号的隧道标签使其构成有效的电路，然后利用外围电路实现整个 cache 的功能。

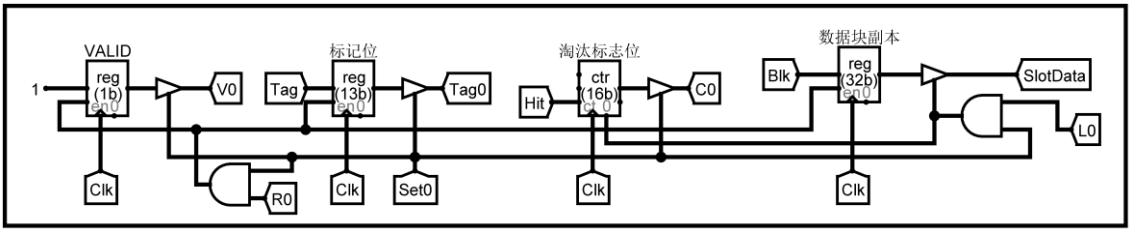

图 1.12 cache 行参考设计

1.5.3 实验思考

1) 不同地址映射方式在具体实现时硬件开销有哪些区别？

2) 如果采用软件实现 cache，那种映射方法更有效？LRU 算法采用什么数据结构实现更方便？软件实现中不可能实现全相联查找的并发查找机制，如何提升查找速度？

第2章　MIPS 处理器设计实验

2.1 MIPS 单周期处理器设计实验

2.1.1 实验目的

学生掌握控制器设计的基本原理，能利用硬布线控制器的设计原理在 Logisim 平台中设计实现 MIPS 单周期 CPU。

2.1.2 实验内容

利用运算器实验，存储系统实验中构建的运算器、寄存器文件、存储系统等部件以及 Logisim 中其它功能部件构建一个 32 位 MIPS CPU 单周期处理器。

方案一：支持表 2.1 中的 8 条 MIPS 核心指令，最终设计实现的 MIPS 处理器能运行实验包中的冒泡排序测试程序 sort.asm，该程序自动在数据存储器 0~15 号字单元中写入 16 个数据，然后利用冒泡排序将数据升序排序，要求统计指令条数并与 MARS 中的指令统计数目进行对比。

表 2.1　核心指令集

#	MIPS 指令	RTL 功能描述
1	add $rd,$rs,$rt	R[$rd]←R[$rs]+R[$rt]　　溢出时产生异常，且不修改 R[$rd]
2	slt $rd,$rs,$rt	R[$rd]←R[$rs]<R[$rt]　　小于置 1，有符号比较
3	addi $rt,$rs,imm	R[$rt]←R[$rs]+SignExt_{16b}(imm)　　溢出产生异常
4	lw $rt,imm($rs)	R[$rt]←$\text{Mem}_{4B}$(R[$rs]+SignExt_{16b}(imm))
5	sw $rt,imm($rs)	Mem_{4B}(R[$rs]+$\text{SignExt}_{16b}$(imm))←R[$rt]
6	beq $rs,$rt,imm	if(R[$rs] = R[$rt]) PC ← PC + SignExt_{18b}({imm, 00})
7	bne $rs,$rt,imm	if(R[$rs] != R[$rt]) PC ← PC + SignExt_{18b}({imm, 00})
8	syscall	系统调用，这里用于停机

2.1.3 注意事项

1）MIPS 虚存模式设置

由于实验中设计的单周期 MIPS 处理器并不包括内存管理单元 MMU 部件，所以程序运行时的访存地址均是物理地址，设计时采用指令存储器和数据存储器分离的哈佛架构。访问数据时应该访问数据存储器，数据存储器的地址起始地址是 0，实验包中的测试程序均直接使用了 0 号地址开始的数据单元，为了使测试程序在 MARS 和实验设计的处理器中的运行结果保持

一致，必须配置 MARS 模拟器中的 MIPS 虚存模式，具体可以选择菜单 Setting 的 Memory Configration 选项，该选项可以设置 MIPS 虚拟地址空间的地址模式，这里应将虚存模式设置为 Compact 模式，这样数据段起始位置 0 开始的位置，如图 2.1 所示。注意，如果采用 Default 模式在 MARS 中运行 sort.asm 排序程序时访存指令会越界访问代码段或内核段，引起保护错。

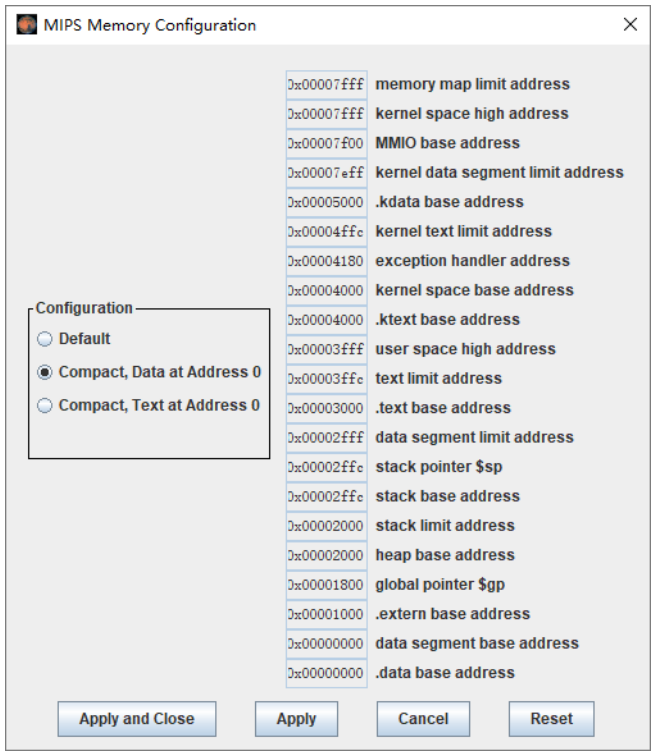

图 2.1 MIPS 虚存模式配置

2）MIPS 机器指令导出

在 MARS 中可以利用 File 菜单中的 Dump Memory 功能将汇编程序的代码段的机器指令和数据段的数据导出，为了能直接在 Logisim 的 RAM 和 ROM 组件中宋使用，推荐采用十六进制文本的方式导出到某个文本文件，然后在文本文件第一行加入“v2.0 raw”，这样导出的文件即可直接加载到 Logisim 平台的 ROM 和 RAM 组件中，如图 2.2 所示。

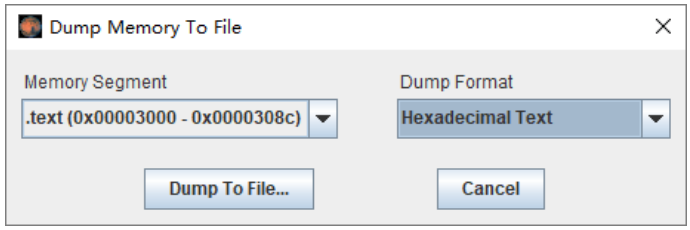

图 2.2 MIPS 代码导出

4、MIPS 寄存器文件库

存储系统实验中我们仅仅设计了包含 4 个寄存器的寄存器文件，为满足 MIPS 单周期处理器设计的需要，这里我们提供了一个包含 32 个寄存器的 MIPS 寄存器文件的库 CS3410.jar，

该库由美国康奈尔大学开发，在 Logisim 平台中可以通过加载 JAR 库的方式加载第三方 JAVA 库，如图 2.3 所示。

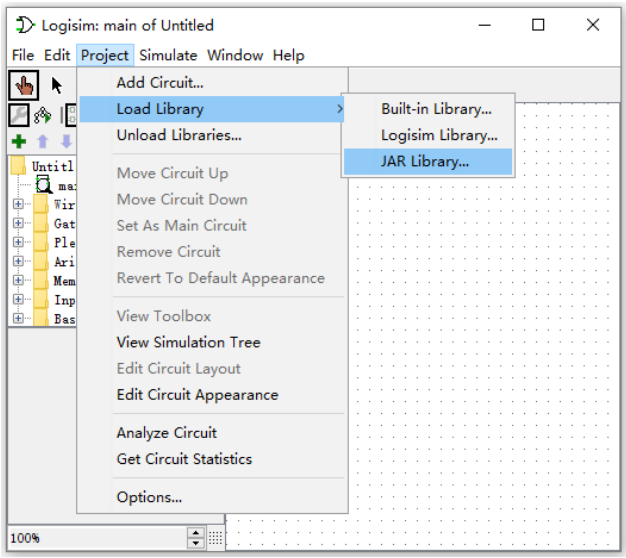

图 2.3 JAR 库加载

选择实验包中的 CS3410.jar 库，加载成功后 logisim 左下角的组件库会增加 CS3410 组件的选项，其中第一个组件就是寄存器文件，添加该组件到画布中，如图 2.4 所示，这个寄存器文件的引脚 rA，rB 为读寄存器编号，rW 为写入寄存器编号，WE 为写使能，W 为写入数据，A、B 为 rA，rB 寄存器的输出值，该组件直接提供了 32 个寄存器观察窗口，非常直观，用户还可以使用手型的戳工具直接修改寄存器的值。需要注意的该组件缩小显示时组建上数字的显示可能不正常。

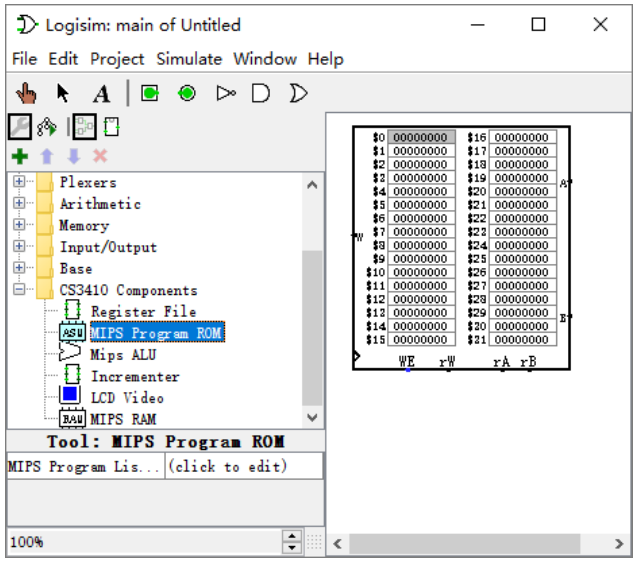

图 2.4 MIPS 寄存器文件库

2.2 MIPS 多周期处理器设计实验

2.2.1 实验目的

学生理解 MIPS 多周期处理器的基本原理，能利用硬布线控制器的设计原理设计实现 MIPS 多周期 CPU。

2.2.2 实验内容

采用硬布线控制模型设计控制器，构造多周期 MIPS 处理器，要求能支持表 2.1 中的 8 条 MIPS 核心指令，最终设计实现的 MIPS 处理器能运行实验包中的冒泡排序测试程序 sort.asm，该程序自动在数据存储器 0~15 号字单元中写入 16 个数据，然后利用冒泡排序将数据升序排序，实验电路应能自动统计指令数目、时钟周期数。

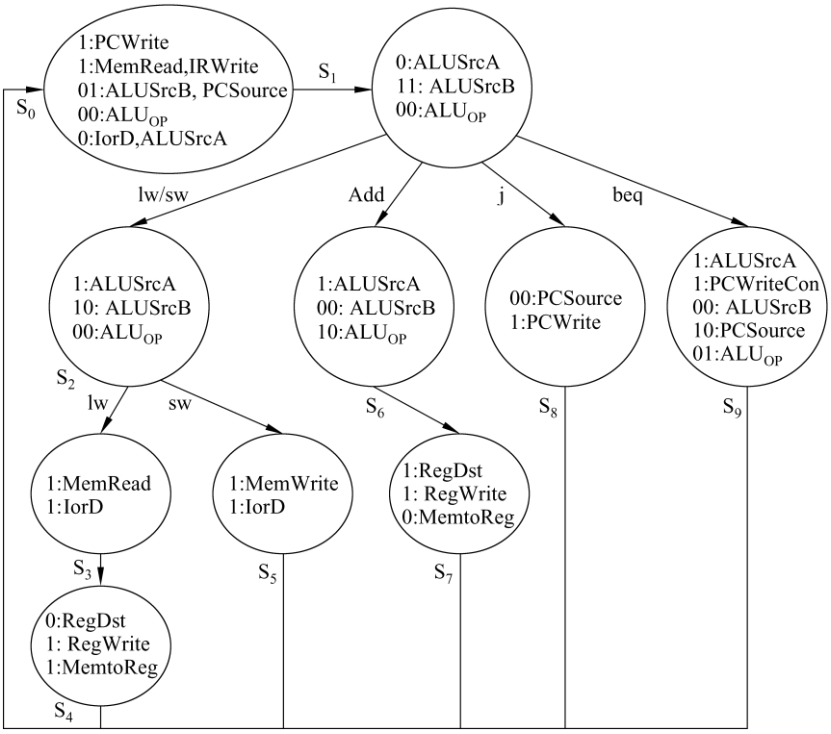

图 2.5 指令执行状态转换图

2.2.3 实验步骤

1）构建主要功能部件和数据通路

在 Logisim 平台中设计 MIPS 多周期处理器所需的主要功能部件，其中寄存器文件可以参

考本章 2.1.3 节的介绍的标准库，其中运算器既可以使用运算器实验中自行设计的运算器，也可以使用标准库中的 ALU 模块，对照图 5.13 所示多周期 MIPS 处理器数据通路，最终将各功能部件连接形成数据通路。

2）构建硬布线控制器

列出上述 8 条指令的 Moore 型状态转换图，如图 2.5 所示。这里仅仅给出了 5 条基本指令的状态转换图，其它 3 条指令请酌情补充完善。图中 10 个状态的意义分别为：$\mathbf{S_0}$ 为取指、$\mathbf{S_1}$ 为译码及取操作数、$\mathbf{S_2}$ 为访存指令计算有效地址、$\mathbf{S_3}$ 为 lw 指令访问存储器、$\mathbf{S_4}$ 为 lw 指令数据写回寄存器、$\mathbf{S_5}$ 为 sw 指令数据写回存储器、$\mathbf{S_6}$ 为 Add 指令两数加运算、$\mathbf{S_7}$ 为 Add 指令数据写回寄存器、$\mathbf{S_8}$ 为 beq 指令目标地址送 PC、$\mathbf{S_9}$ 为 j 指令目标地址送 PC。10 个状态需要四个寄存器来表示，图中 S 的编号用对应的四位二进制编码表示，状态之间的转换关系如表 2.2 所示。

表 2.2 控制器状态状态表

现态	输入(指令操作码)/次态						输出(即操作控制信号)
	XXXX	Add	lw	sw	beq	j	
0000(S0)	0001						MemRead =IRWrite= PCWrite=1 ， IorD=0 ，ALUSrcA =0, ALUSrcB =01, PCsource=01 ，ALUop =00
0001(S1)		0110	0010	0010	1000	1001	ALUSrcA =0 , ALUSrcB=11, ALUop =00
0010(S2)			0011	0101			ALUSrcA=1 ,ALUSrcB=10,ALUop=00
0011(S3)		0100					MemRead=1 ,IorD=1
0100(S4)	0000						RegDst=0, RegWrite=1,MemtoReg=1
0101(S5)	0000						MemWrite=1, IorD=1
0110(S6)		0111					ALUSrcA=1, ALUSrcB=00,ALUop =10
0111(S7)	0000						MemtoReg=0,RegWrite= RegDst=1
1000(S8)	0000						PCSource=00,PCWrite=1
1001(S9)	0000						ALUSrcA=1 ,PCWriteCon=1, ALUSrcB=00,PCSource=10,ALUop=01

利用译码电路生成指令译码信号 add、lw、sw、beq、j 等 8 条指令的译码信号，根据次态和现态之间的逻辑关系，利用逻辑表达式生成状态转换电路，根据控制信号与现态之间的关系，给出对应的逻辑表达式，并给出最终的逻辑电路，将状态寄存器、状态转换电路和控制信号电路封装在一起就是最终的多周期处理器硬布线控制器单元。

3）系统调试

将控制器输出与所有控点相连，构造完整的多周期 MIPS 处理器电路，根据状态转换图逐条指令测试控制器逻辑，状态转换无误后测试 sort.asm 程序直至功能正确。注意多周期处理器中指令和数据存放在同一存储器中，sort.asm 中使用的数据段在主存 0~15 号单元，所以代码段不能放在低地址部分，实验时应注意程序代码如何存放。

2.2.4 实验思考

1）利用 mars 统计 sort.asm 程序运行中执行指令总数（单周期处理器周期数），和实现的多周期处理器周期数进行对比，多周期处理器运行 sort.asm 程序周期数明显增多，是否多周期

处理器性能更差？