第5章 MIPS处理器设计实验

5.1 MIPS单周期处理器设计实验

5.1.1 实验目的

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单周期处理器。

5.1.2 背景知识

单周期处理器是指所有指令均在一个时钟周期内完成的处理器。尽管不同指令执行时间不同，但对单周期处理器而言，时钟周期必须设计成对所有指令都等长。需要特别说明的是，在单周期处理器中，一条指令执行过程中数据通路的任何资源都不能被重复使用，因此任何需要被多次使用的资源（如加法器）都需要设置多个，否则就会发生资源冲突，这是设计单周期处理器数据通路时必须考虑的重要环节。另外，取指令和执行指令阶段均需要使用存储器，所以单周期处理器只能采用指令存储器和数据存储器相分离的结构（即哈佛结构）。本实验基于MIPS指令系统构建单周期处理器，在工程实现上可以采用两种方法：简单迭代法和工程化方法。

1. 简单迭代法

这种方法相对比较直观简单，适合指令数量比较少的实验。设计者可以先完成支持一条固定R型指令（如加法指令）的基本数据通路，测试运行通过后再在该基础上不断增加新的数据通路，支持新的指令，直至所有指令都能正常运行。要支持一条指令的运行，必须有取指令逻辑和执行指令逻辑，所有设计者首先应该完成取指令的数据通路。

1）设计取指令数据通路

图5.1为取指令部分数据通路，涉及的功能部件包括程序计数器（PC）、指令存储器和加法器等。取指令通路应能取出PC锁存地址对应的机器指令，并能在时钟上跳沿到来时实现PC自动加4，从而进入下一条指令的指令周期。这里PC可以采用寄存器实现，其输出作为指令存储器的地址输入，指令存储器经过一个存储周期后一次性取出32位的机器指令，故其数据宽度应为32。需要注意的是，无论是在Logisim平台实现，还是在FPGA上实现，设计者均需考虑指令存储器地址位宽的问题。在FPGA实现中，指令存储器地址宽度越大，综合速度越慢。另外，PC锁存地址是32位字节地址，指令存储器输入地址是字地址，二者不匹配。为简化电路设计，Logisim中建议采用ROM组件实现。

在顺序执行方式下，每一个时钟周期内，CPU取出指令后将PC的值加4，形成下一条指令的地址。这里加4是因为32位MIPS机中所有指令字长均为4字节，每条指令在存储器中占用4个字节的存储单元，而PC中存放的地址是字节地址。为避免资源冲突，这里加法

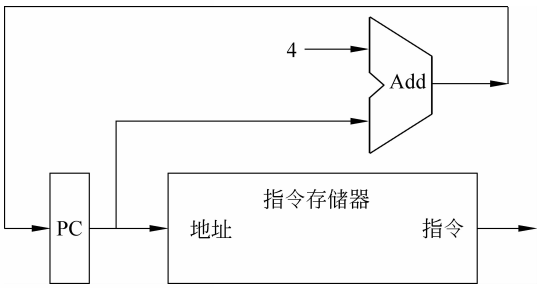

图 5.1　取指令部分数据通路

器应该是独立的器件，与 CPU 中的算术逻辑运算单元分开。

指令取出后即可进入指令的执行周期，接下来就可以设计能支持最为简单的 R 型指令的数据通路，使得执行一条指令的简单 CPU 能够运行起来。

2）构建 R 型指令数据通路

MIPS 中算术逻辑运算指令属于 R 型指令，R 型指令所有的操作数均是寄存器，执行过程中涉及的功能部件包括寄存器文件和 ALU。R 型指令的数据通路如图 5.2 所示，指令执行过程中 ALU 的两个源操作数均来自于寄存器文件输出，其中 RD_1 是寄存器 R1＃的值，RD_2 是寄存器 R2＃的值，R1＃来自于指令字中的$rs 字段，R2＃来自于指令字中的$rt 字段，运算结果写入目的寄存器$rd 中，MIPS 不同类型指令字段分布如表 5.1 所示。

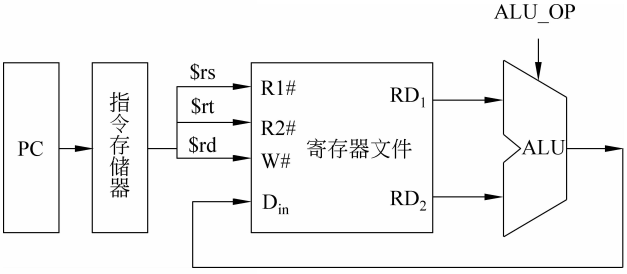

图 5.2　R 型指令的数据通路

表 5.1　MIPS 不同类型指令字段分布

Type	31～26	25～21	20～16	15～11	10～06	05～00
R-Type	opcode	$rs	$rt	$rd	shamt	funct
I-Type	opcode	$rs	$rt	Imm(16 位)		
J-Type	opcode	Address(26 位)				

为测试对应通路能否正常工作，我们将指令 nor $s0，$0，$1 汇编成机器代码 0x00018827，然后将机器码存入指令 ROM 的 0 号地址处，将 ALU_OP 设置为固定值 1010（第 2 章运算器实验中表示或非操作）。如果数据通路正确，运算器输出结果应该为 0xFFFFFFFF，时钟上跳沿到来时数据将写入 16 号寄存器($s0)中。

至此我们设计完成的 CPU 已经可以执行一条或非指令，接下来可以进一步编写一些其

他的 R 型指令，例如 add、sub、and、or 等，当然不同的算术逻辑运算指令对应的 ALU_OP 是不同的，所以可能需要手动修改 ALU_OP 的值，否则所有的指令都会执行或非操作。为了使 CPU 能自动进行不同类型的运算，需要增加相应的控制器组合逻辑自动译码指令，根据指令操作码和扩展码生成对应的 ALU_OP 信号，这就是最简单的硬布线控制器。当设计的 CPU 能自动运行多条 R 型算术逻辑运算指令后，R 型数据指令数据通路部分基本完成。

3）构建访存指令数据通路

MIPS 访存指令属于 I 型指令，访存地址等于变址寄存器$rs 的值加上 16 位立即数，地址运算通过 ALU 完成，所以需要将$rs 的值送入 ALU，同时要将 16 位立即数进行 32 位符号扩展后送入 ALU，二者进行加法运算得到最终的访存地址，图 5.3 是访存指令的部分数据通路。该数据通路包括指令存储器、寄存器文件、符号扩展、ALU、数据存储器等功能部件。如果是加载指令，则数据存储器的结果将会送回到寄存器文件的 D_{in} 端，写入寄存器编号为$rt。如果是存储指令，则$rt 寄存器的值会通过 RD2 # 端口输出到数据存储器的写入端口。

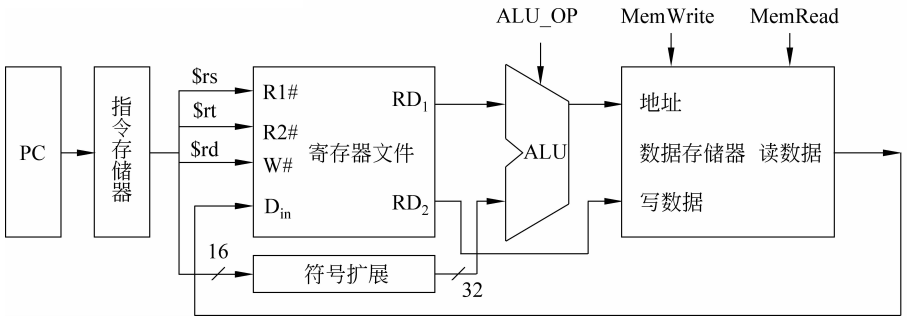

图 5.3　访存指令的部分数据通路

对比 R 型指令的数据通路发现，在访存指令数据通路中，写入寄存器编号 W # 的输入来源、ALU 第二个操作数输入来源、写入数据端口 D_{in} 的输入来源与前者均不同，为了将两个数据通路统一到同一个电路中，我们需要在有多个输入来源的端口增加多路选择器，如图 5.4 所示，每增加一个多路选择器就额外引入一个控制信号，这里分别增加了 RegDst、ALUSrc、MemtoReg 3 个控制信号，另外还包括两个数据存储器读写控制信号 MemWrite（存储器写）、MemRead（存储器读）。这些控制信号都应该由硬布线控制器根据指令译码自动生成，在完成对应控制逻辑之前，可以先手动设置对应的值，验证访存指令数据通路的功

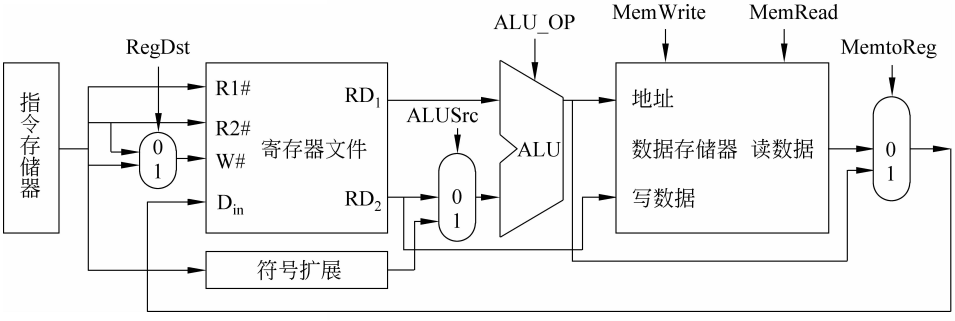

图 5.4　支持算术逻辑运算指令和访存指令的混合数据通路

能是否正常。当访存数据通路能正常实现数据存储器的读出和写入时，就可以在控制器中增加相应的组合逻辑自动生成新增加的5个控制信号。至此，我们设计的CPU已经可以支持R型运算指令和I型访存指令。

在现有基础上，只须修改硬布线控制器逻辑部分——立即数扩展模块，就可以支持I型指令中的立即数运算了。另外，通过增加其他的数据通路、功能模块、多路选择器、控制信号、修改硬布线控制器逻辑，就可以逐步扩展CPU的功能，使得最终的CPU能够支持实验要求的所有指令集。将所有类型指令的数据通路综合后，再增加必要的控制逻辑，就可以得到支持MIPS机器指令系统子集的完整数据通路，图5.5是基于单周期方案的MIPS处理器数据通路顶层视图。

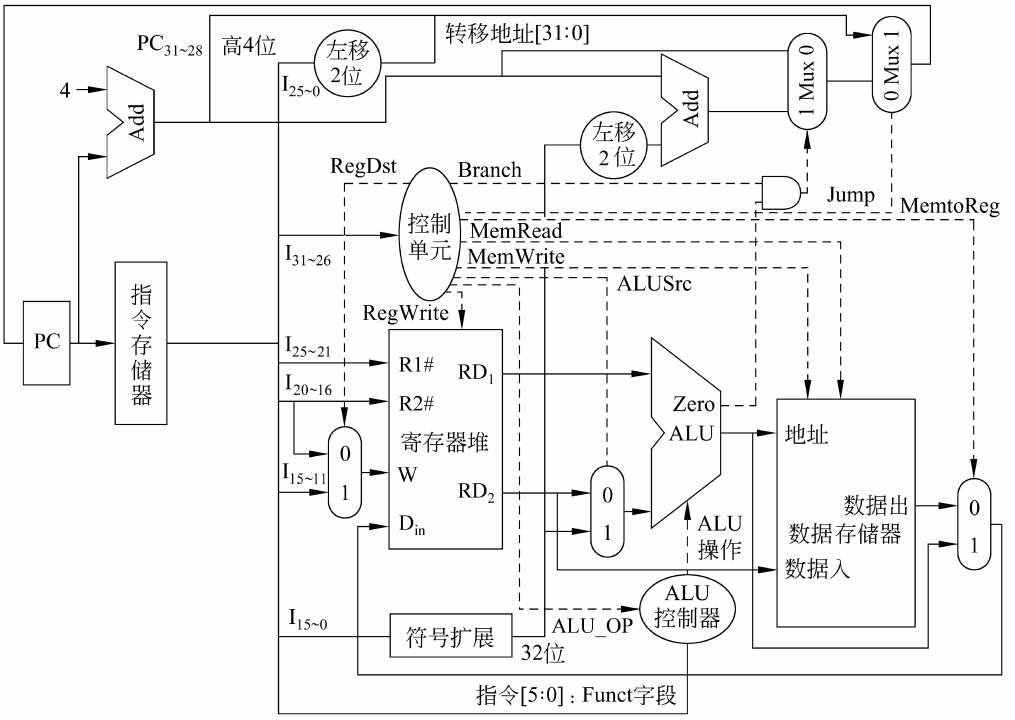

图5.5　基于单周期方案的MIPS处理器数据通路顶层视图

不同MIPS指令的数据通路不同，具体细节可以参考MIPS仿真器MARS中数据通路透视插件功能，该插件可以动态演示当前指令对应的数据通路以及对应的控制信号生成。如图5.6所示，实验时可以以MARS作为黄金参考模型，对照功能进行实现。

2. 工程化方法（推荐）

简单迭代法的原理简单，但在设计过程中会不断修改数据通路、扩充电路、增加新部件、控点，会造成大量重复的工作。如果采用Logisim平台进行设计，对电路布局也不利。当需要实现的指令数目较多时，简单迭代法效率过于低下。下面介绍另外一种工程化方法，该方法最早由北京航空航天大学计算机学院引入，其主要设计步骤介绍如下。

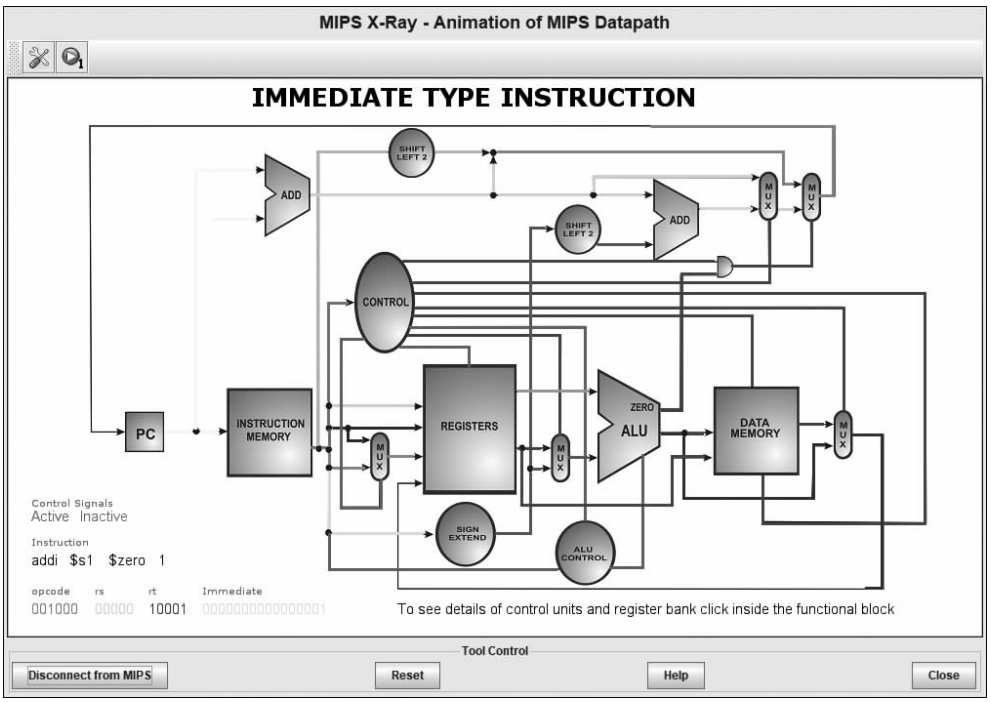

图 5.6　MIPS 数据通路动画展示

1）构建主要功能部件

实现构建 CPU 所需的主要功能部件包括：程序计数器 PC、指令存储器 IM、数据存储器 DM、立即数扩展器、地址转移逻辑 NPC、运算器 ALU。第 2 章中运算器实验已经实现了 ALU，寄存器文件可以在第三方 JAR 库 CS3410.jar 中获得。

2）构建功能部件输入来源表

功能部件输入来源表主要用于描述 MIPS 处理器各主要功能部件之间的连接关系，记录各功能部件输入端数据来源。注意，这里忽略了控制类信号，仅保留数据类信号，具体如表 5.2 所示。表中给出了主要功能部件程序计数器（PC）、指令存储器（IM）、寄存器文件（RF）、算术逻辑运算单元（ALU）、数据存储器（DM），并分别给出了对应部件的输入端口。逐条分析每一条指令的功能，根据指令功能填写表格，如果发现需要新的功能部件，可以在表格中引入新的部件，并设置其输入来源。

表 5.2　功能部件输入来源表

指令	PC	1M	RF				S-EXT	ALU	
			R1＃	R2＃	W＃	D_{in}		A	B
ori	PC＋4	PC	rs		rt	ALU	IMM[15:0]	RF.RD1	S-EXT
or	PC＋4	PC	rs	rt	rd	ALU		RF.RD1	RF.D2

3）数据通路综合

所有指令都填写完毕后，将主要功能部件输入来源表中功能部件的输入按列进行合并，

如某个输入有多个输入来源，引入多路选择器，同时新增多路选择器选择控制信号，对于空输入如果其输入不影响指令执行，可以直接忽略，否则需要增加额外的控制电路。根据合并后的输入来源，在 Logisim 中连接各主要功能部件，一次性构建 MIPS CPU 的所有数据通路。

4）控制信号综合

列出所有功能部件、多路选择器控制信号、运算操作选择的产生条件，如表 5.3 所示，横坐标给出的是不同指令的译码信号，纵坐标给出的是各功能部件控制信号。其中，ADD、SUB 是指运算器应该执行什么操作的译码信号，表中有黑色圈的位置表示当前指令会产生对应的信号，例如 add、addi、lw、sw 指令都会要求 ALU 进行加法运算。利用译码电路生成各指令译码信号，然后以行为单位将各个产生信号的条件相加（逻辑或）即可得到控制信号的逻辑表达式，这里 ADD＝add＋addi＋lw＋sw。注意，这里的控制信号表仅供参考，不代表最终实现。

表 5.3　控制信号表

控制信号	0	1	2	3	4	5	6	7	1	2	3	4	5	6	7	8	9	10
	or	and	add	sub	sllv	srlv	srav	slt	disp	lui	ori	addi	andl	lw	sw	jump	beq	bue
ADD			●									●		●	●			
SUB				●													●	●
AND		●																
OR	●										●							
sll					●													
srl																		
sra																		
slt								●										
lui										●								
RegWriteBack	●	●	●		●	●		●			●		●					
Alusrc													●	●	●			
RegDst										●	●	●						
disp									●									
JumBranch																●		
beq																		
bne																		●
SingnedIMM												●		●	●			
MemRead																		
Memwrite															●			

5）硬布线控制器封装

根据各控制信号的逻辑表达式，设计最终的控制器并封装，即可得到最终的 MIPS 单周期处理器的控制器。将控制器的控制信号输出与所有控点相连，即可构建最终的 MIPS 单周期处理器系统。

工程化方法通过功能部件输入来源表一次性构建了所有数据通路，有效避免了电路的重复迭代。另外，控制器也是一次性得到所有信号的逻辑表达式，也减少了重复设计的工作量。对于指令系统较多的实验中，采用这种方法效率更高，实际实验时可以酌情进行选择。

5.1.3 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 中其他功能部件构建一个 32 位 MIPS CPU 单周期处理器。

1. 方案一

支持表 5.4 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。

2. 方案二

同时支持表 5.4 核心指令集和表 5.5 基础指令集中的 MIPS 指令，最终实现的 MIPS 处理器能运行标准测试程序 benchmark.asm。要求统计指令条数，并与 MARS 中的指令统计数目进行对比。

表 5.4 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[$rt]←$Mem_{4B}$(R[$rs]+$SignExt_{16b}$(imm))
5	sw$rt,imm($rs)	Mem_{4B}(R[$rs]+$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

3. 方案三

在方案二的基础上增加表 5.6 中的扩展指令，包括 2 条 C 类指令、1 条 M 类指令、1 条 B 类指令，简称 CCMB 系列指令。最终实现的 MIPS 处理器能正确运行标准测试程序 benchmark.asm，要求修改 syscall 系统调用的功能，当系统调用号不等于 34 时，一律暂停当前程序的执行，而不是停机，暂停后等待用户按下电路中的 Go 按钮才能继续运行。实验包

表 5.5　基础指令集

#	MIPS 指令	RTL 功能描述
1	addu $rd,$rs,$rt	R[$rd]←R[$rs]+R[$rt],无符号加
2	addiu$rt,$rs,imm	R[$rt]←R[$rs]+$SignExt_{16b}$(imm)
3	and $rd,$rs,$rt	R[$rd]←R[$rs]&R[$rt],逻辑与
4	andi$rt,$rs,imm	R[$rt]←R[$rs]&{0×16,imm}
5	sll$rd,$rt,shamt	R[$rd]←R[$rt]<<shamt,逻辑左移
6	srl$rd,$rt,shamt	R[$rd]←R[$rt]>>shamt,逻辑右移
7	sra$rd,$rt,shamt	R[$rd]←R[$rt]>>shamt,算术右移
8	sub $rd,$rs,$rt	R[$rd]←R[$rs]−R[$rt],溢出时产生异常,且不修改 R[$rd]
9	or $rd,$rs,$rt	R[$rd]←R[$rs]|R[$rt],逻辑或
10	ori$rt,$rs,imm	R[$rt]←R[$rs]|{0×16,imm}
11	nor $rd,$rs,$rt	R[$rd]←!(R[$rs]|R[$rt])
12	sltu $rd,$rs,$rt	R[$rd]←R[$rs]<R[$rt],小于置 1,无符号比较
13	slti$rt,$rs,imm	R[$rt]←R[$rs]<$SignExt_{16b}$(imm),小于置 1,有符号比较
14	j address	PC←{(PC+4)[31:28],address,00}
15	jal address	R[31]←PC+4　PC←{(PC+4)[31:28],address,00}
16	jr$rs	R[$rs],值应是 4 的倍数
17	syscall	If $v0==34,数码管显示$a0 值;else 停机。注意,数码管输入数据应该用寄存器锁存

中已经包括了所有 CCMB 系列指令的测试程序,请将对应的测试程序放置在 benchmark.asm 尾部,以方便教师检查。

表 5.6　扩展指令集

#	MIPS 指令	RTL 功能描述
C1	sllv$rd,$rt,$rs	R[$rd]←R[$rt]<<R[$rs],可变左移
C2	srlv$rd,$rt,$rs	R[$rd]←R[$rt]>>R[$rs],逻辑可变右移
C3	srav$rd,$rt,$rs	R[$rd]←R[$rt]>>R[$rs],算术可变右移
C4	subu $rd,$rs,$rt	R[$rd]←R[$rs]−R[$rt],无符号减
C5	xor $rd,$rs,$rt	R[$rd]←R[$rs]^R[$rt],异或
C6	xori$rt,$rs,imm	R[$rt]←R[$rs]^{0×16,imm}
C7	lui$rt,imm	R[$rt]←{(imm)[15:0],0×16}
C8	sltiu$rt,$rs,imm	R[$rt]←R[$rs]<$SignExt_{16b}$(imm),无符号比较

续表

#	MIPS 指令	RTL 功能描述
C9	multu $rs,$rt	{HI,LO}←R[$rs] * R[$rt],无符号乘
CA	divu $rs,$rt	LO←R[$rs]/R[$rt] HI←R[$rs]%R[$rt],无符号除法
CB	mflo $rd	R[$rd]←LO,取 LO 寄存器的值
M1	lb$rt,imm($rs)	R[$rt]←$SignExt_{8b}$($Mem_{1B}$(R[$rs]+$SignExt_{16b}$(imm)))
M2	lh$rt,imm($rs)	R[$rt]←$SignExt_{16b}$($Mem_{2B}$(R[$rs]+$SignExt_{16b}$(imm)))
M3	lbu$rt,imm($rs)	R[$rt]←{0×24,$Mem_{1B}$(R[$rs]+$SignExt_{16b}$(imm))}
M4	lhu$rt,imm($rs)	R[$rt]←{0×16,$Mem_{2B}$(R[$rs]+$SignExt_{16b}$(imm))}
M5	sb$rt,imm($rs)	Mem_{1B}(R[$rs]+$SignExt_{16b}$(imm))←(R[$rt])[7:0]
M6	sh$rt,imm($rs)	Mem_{2B}(R[$rs]+$SignExt_{16b}$(imm))←(R[$rt])[15:0]
B1	blez $rs,imm	if(R[$rs]<=0) PC←PC+$SignExt_{18b}$({imm,00}),有符号比较
B2	bgtz$rs,imm	if(R[$rs]>0) PC←PC+$SignExt_{18b}$({imm,00}),有符号比较
B3	bltz $rs,imm	if(R[$rs]<0) PC←PC+$SignExt_{18b}$({imm,00}),有符号比较
B4	bgez$rs,imm	if(R[$rs]>=0)PC←PC+$SignExt_{18b}$({imm,00}),有符号比较

5.1.4 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结构。访问数据时应该访问数据存储器,数据存储器的起始地址是 0,实验包中的测试程序均直接使用 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 中的 Memory Configuration 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,Data at Address 0 模式,这样数据段起始位置就为 0 开始的位置,如图 5.7 所示。注意,如果采用 Default 模式在 MARS 中运行 sort.asm 排序程序,访存指令会越界访问代码段或内核段,引起保护错误。

2. MIPS 机器指令导出

在 MARS 中可以利用 File 菜单中的 Dump Memory To File 功能将汇编程序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,这样导出的文件就可直接加载到 Logisim 平台的 ROM 和 RAM 组件中,如图 5.8 所示。

3. 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,这里提供了一个包含 32 个寄存器的 MIPS 寄存器文件的库 CS3410.

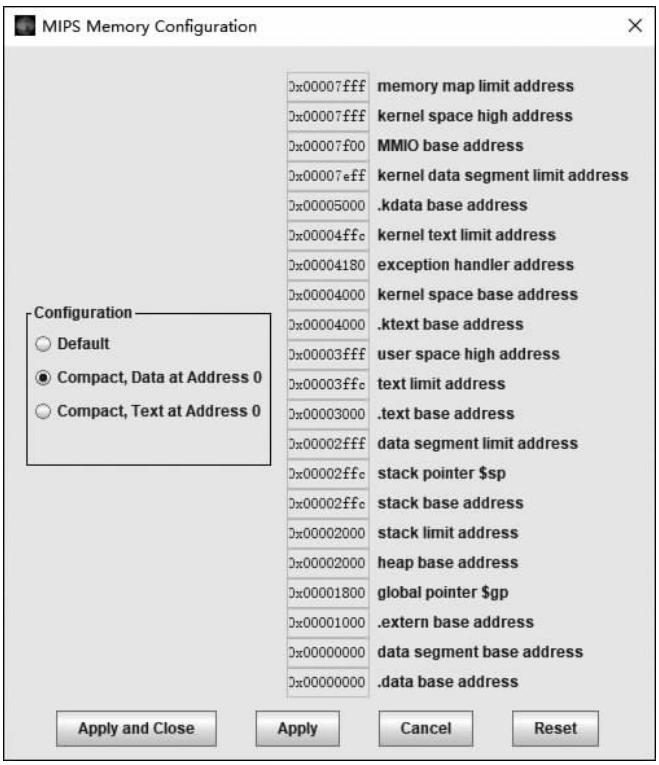

图 5.7　MIPS 虚存模式配置

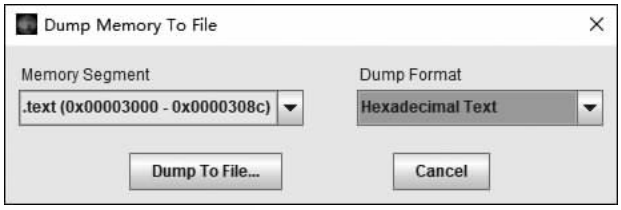

图 5.8　MIPS 机器指令代码导出

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 5.9 所示。

选择实验包中的 CS3410.jar 库，加载成功后在 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 5.10 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戳工具直接修改寄存器的值。需要注意的是，该组件缩小显示时，组件上数字的显示可能不正常。

4. MIPS RAM 库

在存储系统实验中，我们设计过 MIPS RAM 电路，CS3410 库中也提供了一个标准的