长沙理工大学

《系统能力综合实训》报告

MIPS 单周期 24 指令处理器设计

学　　院 计算机与通信工程　专　　业 计算机科学与技术

班　　级 计算机 22-05 班　学　　号 202208010508

学生姓名 崔 颢　指导教师 龙际珍

课程成绩 _____　完成日期 2025 年 1 月 3 日

《系统能力综合实训》成绩评定标准

毕业要求	考核与评价方式及成绩比例（%）			成绩比例（%）
	系统演示	项目答辩	实训报告	
实训目标 1：**问题分析**	/	10	10	20
实训目标 2：**设计/开发解决方案**	15	/	10	25
实训目标 3：**研究**	15	10	**10**	35
实训目标 4：**环境和可持续发展**	/	/	10	10
实训目标 5：**个人和团队**	/	/	10	10
合计	30	20	50	100

《系统能力综合实训》成绩评定

毕业要求	考核与评价方式及成绩比例（%）			成绩比例（%）
	系统演示	项目答辩	实训报告	
实训目标 1：**问题分析**	/			
实训目标 2：**设计/开发解决方案**		/		
实训目标 3：**研究**				
实训目标 4：**环境和可持续发展**	/	/		
实训目标 5：**个人和团队**	/	/		
合计				

指导教师对系统能力综合实训的评定意见

综合成绩 _____指导教师签字_____ 2025 年 1 月 6 日

MIPS 单周期 24 指令处理器设计

学生姓名：崔颢　　指导老师：龙际珍

摘　要

本实验报告的主要目的是设计并实现一个 MIPS 单周期 24 指令处理器。通过掌握单周期处理器的数据通路图构成、原理及设计方法，了解指令与 CPU 的关系，并掌握测试单周期处理器的方法。实验采用了 MIPS32 指令集中的 24 条核心指令，设计了相应的数据通路和控制信号生成电路。最终，通过冒泡排序等程序测试了 CPU 的正常运行，验证了设计的正确性和有效性。

关键词：单周期、硬布线控制器、指令集、程序测试

Design of a MIPS Single-Cycle Processor

Student Name: Cui Hao Advisor: Long Jizhen

Abstract

The main objective of this experimental report is to design and implement a MIPS single-cycle 24-instruction processor. By mastering the construction, principles, and design methods of the single-cycle processor's data path diagram, understanding the relationship between instructions and the CPU, and mastering the methods for testing the single-cycle processor. The experiment adopts 24 core instructions from the MIPS32 instruction set, designing the corresponding data path and control signal generation circuits. Finally, the CPU's normal operation was tested through programs such as bubble sort, verifying the correctness and effectiveness of the design.

Keywords: Single-cycle、Hardwired controller、Instruction set、Program testing

目录

一、引言

（一）实训内容

1. 掌握单周期 CPU 数据通路图的构成、原理及其设计方法：通过学习单周期 CPU 的数据通路图，理解其各个组成部分及其工作原理，掌握设计单周期 CPU 的方法和技巧。

2. 掌握单周期 CPU 的实现方法，代码实现方法：通过实际编写代码，掌握如何实现单周期 CPU，包括硬件描述语言 logisim 的使用，确保每条指令在一个时钟周期内完成。

3. 认识和掌握指令与 CPU 的关系：通过学习和实践，理解指令集与 CPU 的关系，掌握指令的执行过程及其对 CPU 各个部件的控制和影响。

4. 掌握测试单周期 CPU 的方法：通过设计和实施测试方案，掌握如何对单周期 CPU 进行测试，验证其功能和性能，确保其能够正确执行指令集中的各条指令。

（二）实训目的

1. 将理论与实践相结合将课堂上学习的内容应用到实际操作中，增加对该知识理解与掌握。

2. 提升问题解决能力，在实训过程中，我们会遇到各种各样的问题，通过解决这些问题，可以提高我的分析与解决能力。

（三）实现思路

1. **指令集设计**：确定 CPU 支持的指令集，包括操作码、操作数和指令格式。

2. **数据路径设计**：设计数据路径，包括寄存器文件、ALU（算术逻辑单元）、内存和总线等组件。确保数据在各个组件之间的传输路径。

3. **控制器设计**：设计硬布线控制器，生成控制信号以协调数据路径中的各个组件。控制器根据当前指令的操作码生成相应的控制信号。

4. **时序设计**：设计时钟信号和时序控制，确保每个指令在一个时钟周期内完成。单周期 CPU 的特点是每条指令在一个时钟周期内完成所有操作。

5. **指令执行流程**：定义每条指令的执行流程，包括取指令、译码、执行、访存和写回等阶段。确保每个阶段在一个时钟周期内完成。

6. **测试与验证**：编写测试程序，对 CPU 进行功能验证和性能测试。确保 CPU 能够正确执行所有指令，并达到预期的性能指标。

二、实验原理分析

（一）MIPS 单周期处理器定义

MIPS 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，但对于单周期处理器而言，时钟周期必须设计成对所有指令都等长。为保证只能在一个时钟周期内完成，一条指令的执行过程中数据通路的任何资源都不能被重复使用，任何需要被多次使用的资源（如加法器、存储器）都需要设置多个，否则就会发生资源冲突[1]。

（二）CPU 指令执行流程

CPU 在处理指令时，一般需要经过以下几个步骤：

(1)取指令(IF)：根据程序计数器 PC 中的指令地址，从存储器中取出一条指令，同时，PC 根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入 PC，当然得到的“地址”需要做些变换才送入 PC。

(2)指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3)指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4)存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得 到数据地址单元中的数据。

(5)结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

（三）数据通路

该实验的数据通路主要包含程序计数器 PC、指令存储器、寄存器文件 RegFile、运算器 ALU、单周期硬布线控制器几个重要模块组成，其基础数据通路顶层视图如图 1.1，本论文将从逻辑到物理的顺序逐一介绍该项目的实现原理与方案。

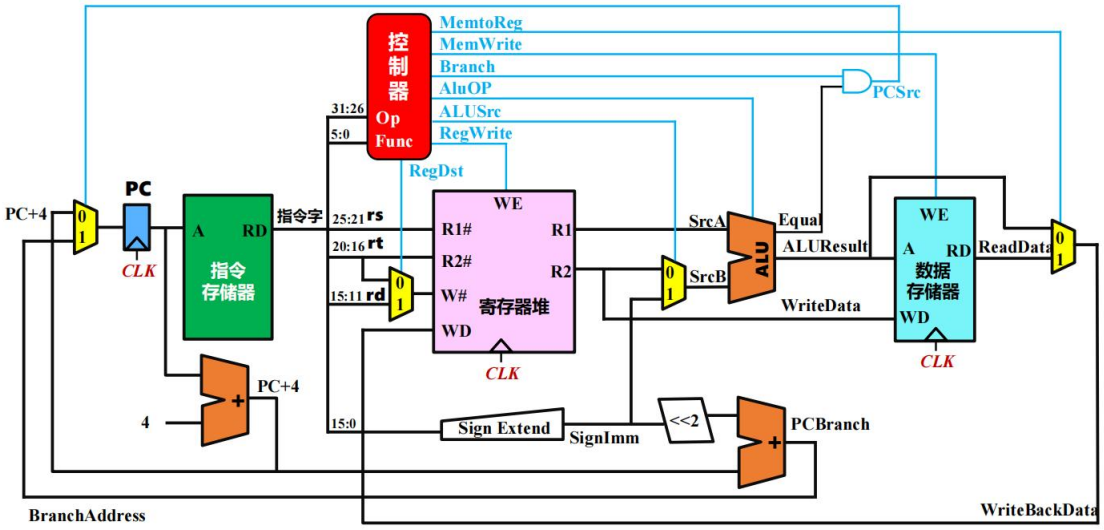

图 1.1 基础数据通路顶视图

三、指令集

本实验参考了 MIPS32 的指令集，并从中挑选了 24 条来实现，接下来将以指令类型（I 型、R 型、J 型）分组，分别介绍。

（一）R 型指令（12 条）

R 型指令的 OP 位为全 0，由 func 位来决定其执行的功能，rs、rt 是源操作数所的寄存器编号，rd 是目的操作数所在的寄存器编号，shamt 是位移量，在执行移位操作的时候需要指明所需移动的位数，指令格式如表 3.1.1 所示，RTL 描述如表 3.1.2 所示

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	功能（func）
ADD	000000	rs	rt	rd	00000	100000	寄存器加
ADDU	000000	rs	rt	rd	00000	100001	无符号寄存器加
SUB	000000	rs	rt	rd	00000	100010	寄存器减
AND	000000	rs	rt	rd	00000	100100	寄存器与
OR	000000	rs	rt	rd	00000	100101	寄存器或
NOR	000000	rs	rt	rd	00000	100111	寄存器或非
SLL	000000	00000	rt	rd	sa	000000	逻辑左移
SRL	000000	00000	rt	rd	sa	000010	逻辑右移
SRA	000000	00000	rt	rd	sa	000011	算术右移
JR	000000	rs	00000	00000	00000	001000	寄存器跳转
SLT	000000	rs	rt	rd	00000	101010	小于置数
SLTU	000000	rs	rt	rd	00000	101011	小于无符号数置数

表 3.1.1 R 型指令格式

指令	操作
ADD	R[rd] <- R[rs] + R[rt]
ADDU	R[rd] <- R[rs] + R[rt] (无符号)
SUB	R[rd] <- R[rs] - R[rt]
AND	R[rd] <- R[rs] & R[rt]
OR	R[rd] <- R[rs]|R[rt]
NOR	R[rd] <- ~(R[rs]|R[rt])
SLL	R[rd] <- R[rt] << shamt
SRL	SRLR[rd] <- R[rt] >> shamt
SRA	R[rd] <- R[rt] >> shamt (算术右移)
JR	PC <- R[rs]
SLT	R[rd] <- (R[rs] < R[rt]) ? 1 : 0
SLTU	R[rd] <- (R[rs] < R[rt]) ? 1 : 0 (无符号)

表 3.1.2 R 型指令 RTL 描述

（二）I 型指令（9 条）

I 型指令的操作功能由 OP 决定，其中 rs 是第一个操作数，immediate 为立即数，是第二个源操作数，rt 是目的操作数所在的寄存器编号，指令格式如表 3.2.1，RTL 描述如图 3.2.2。

指令	[31:26]	[25:21]	[20:16]	[15:0]	功能
ADDI	001000	rs	rt	immediate	立即数加
ADDIU	001001	rs	rt	immediate	无符号立即数加
ANDI	001100	rs	rt	immediate	立即数与
ORI	001101	rs	rt	immediate	立即数或
LW	100011	rs	rt	immediate	取字数据
SW	101011	rs	rt	immediate	存字数据
BEQ	000100	rs	rt	immediate	相等转移
BNE	000101	rs	rt	immediate	不等转移
SLTI	001010	rs	rt	immediate	小于立即数置数

表 3.2.1 I 型指令格式

指令	RTL 描述
ADDI	R[rt] <- R[rs] + SignExt(imm)
ADDIU	R[rt] <- R[rs] + SignExt(imm) (无符号)
ANDI	R[rt] <- R[rs] & ZeroExt(imm)
ORI	R[rt] <- R[rs]|ZeroExt(imm)
LW	R[rt] <- M[R[rs] + SignExt(offset)]
SW	M[R[rs] + SignExt(offset)] <- R[rt]
BEQ	if (R[rs] == R[rt]) then PC <- PC + 4 + (SignExt(offset) << 2)
BNE	if (R[rs] != R[rt]) then PC <- PC + 4 + (SignExt(offset) << 2)
SLTI	R[rt] <- (R[rs] < SignExt(imm)) ? 1 : 0

表 3.2.1 I 型指令 RTL 描述

（三）J 型指令

指令	[31:26]	[25:0]	功能
J	000010	address	跳转
JAL	001100	address	调用

表 3.3.11 J 型指令格式

指令	RTL 描述
J	PC <- (PC & 0xF0000000) \(target << 2)
JAL	R[31] <- PC + 8; PC <- (PC & 0xF0000000) \(target << 2)

还有一条特殊指令 SysCall 不在表中。

四、数字电路的设计与模拟

本项目通过软件 Logisim 来进行数字电路物理部分的设计与模拟，将按照整体到局部的顺序进行介绍，顺序为：单周期 MIPS 数据通路、单周期硬布线控制器、运算控制器、控制信号。

（一）单周期 MIPS 数据通路

数据通路是计算机系统中用于传输和处理数据的硬件组件集合，它的主要作用包括：数据传输、数据处理、指令执行、寄存器操作。数据通路的设计和实现对于 CPU 性能和功能至关重要，它决定了指令的执行速度与效率，以及系统的整体性能。、

在本项目中，数据通路主要包括 PC（程序计数器）、指令存储器、单周期硬布线控制器、RegFile（寄存器文件）、ALU（算术逻辑单元）组成，其实现方案如图 4.1 所示。

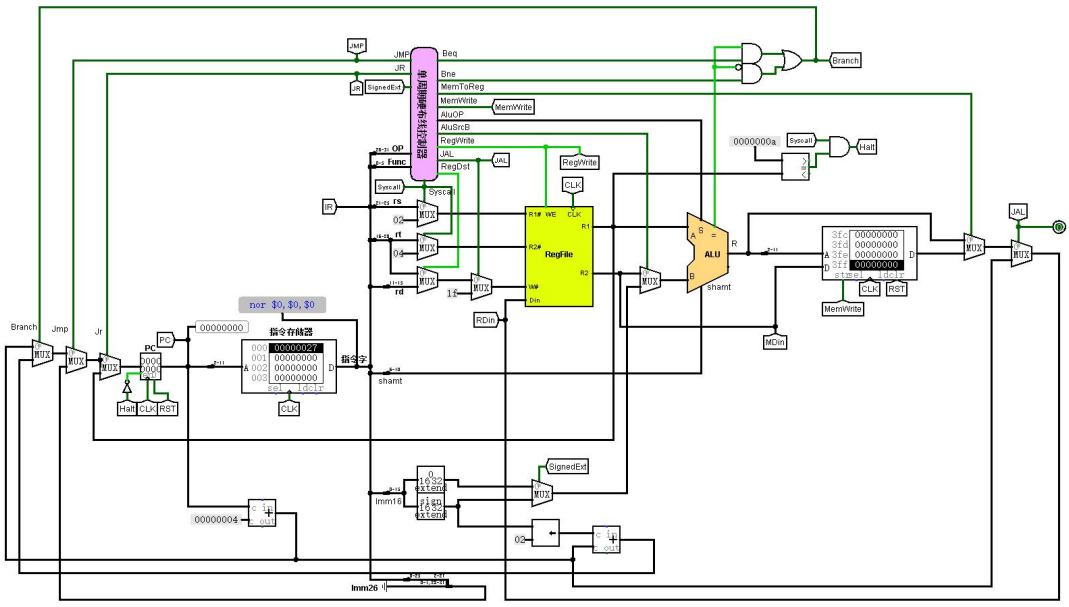

图 4.1 单周期 MIPS 数据通路实现方案

其中 PC 负责指向当前要执行指令的位置，信号传输至指令存储器中，从存储器中取出将要执行的指令，再根据指令类型从指令本身中或者 RegFile 中的寄存器里取出操作数，最后送入 ALU 中进行运算，或者送入其它元件进行处理。

以上指令的执行流程均需要单周期硬布线控制器的参与。

（二）单周期硬布线控制器

单周期硬布线控制器在 CPU 设计中起着至关重要的作用，任何指令的执行都少不了它的支持，它的主要功能包括：指令译码、控制信号的生成、执行指令。

本项目的硬布线控制器内部实现如图 4.2 所示。

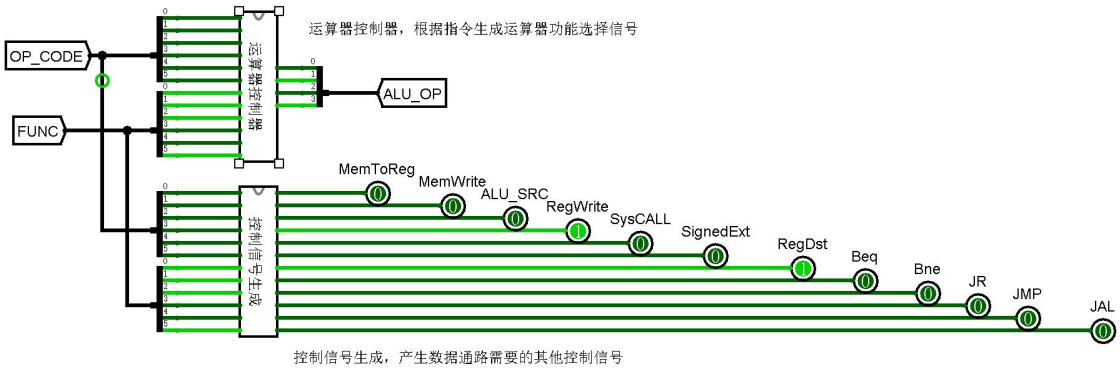

图 4.2 单周期硬布线控制器的内部实现

该控制器左边读入指令的 OP 段与 FUNC 段，根据第三节指令集的规定进行译码，通过运算控制器与控制信号生成器两个元件来生成操控信号，从而实现对整个 CPU 的操控与对指令的实现。

硬布线控制器封装后的样式图如图 4.3 所示。

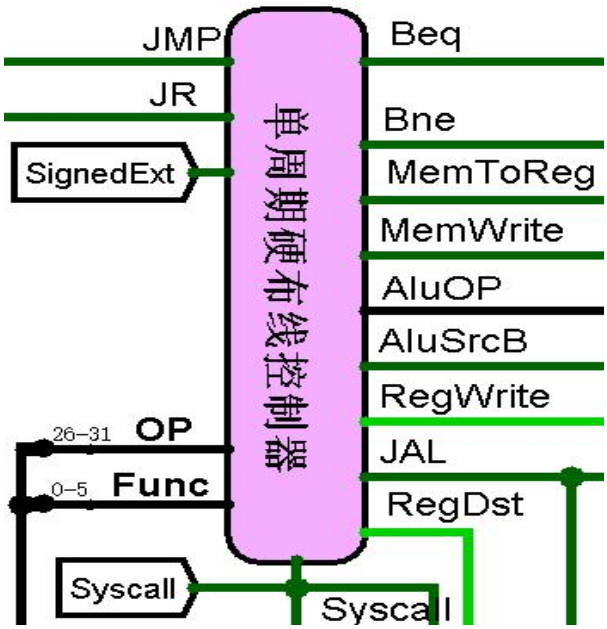

图 4.3 单周期硬布线控制器封装

（三）运算控制器

运算控制器负责根据 OP 字段，按照指令集的规定进行译码，译码后的结果通过单周期硬布线控制器的 ALUOP 送入数据通路中，最后进入 ALU 的 S 接口处。

本实验中，该元件的内部电路由 Logisim 的自动生成电路功能，按照图 4.4 的规定进行译码。

ALU_OP	功能		
0	逻辑左移	12	无符号比较
1	算数右移	11	符号比较
2	逻辑右移	10	按位或非
3	无符号乘	9	按位异或
4	无符号除	8	按位或
5	加		
6	减		
7	按位与		

图 4.4 ALU_OP 规定

将逻辑输入 Logisim 中的自动生成电路中，生成电路，如图 4.5 所示（由于电路过于复杂，仅展示部分）

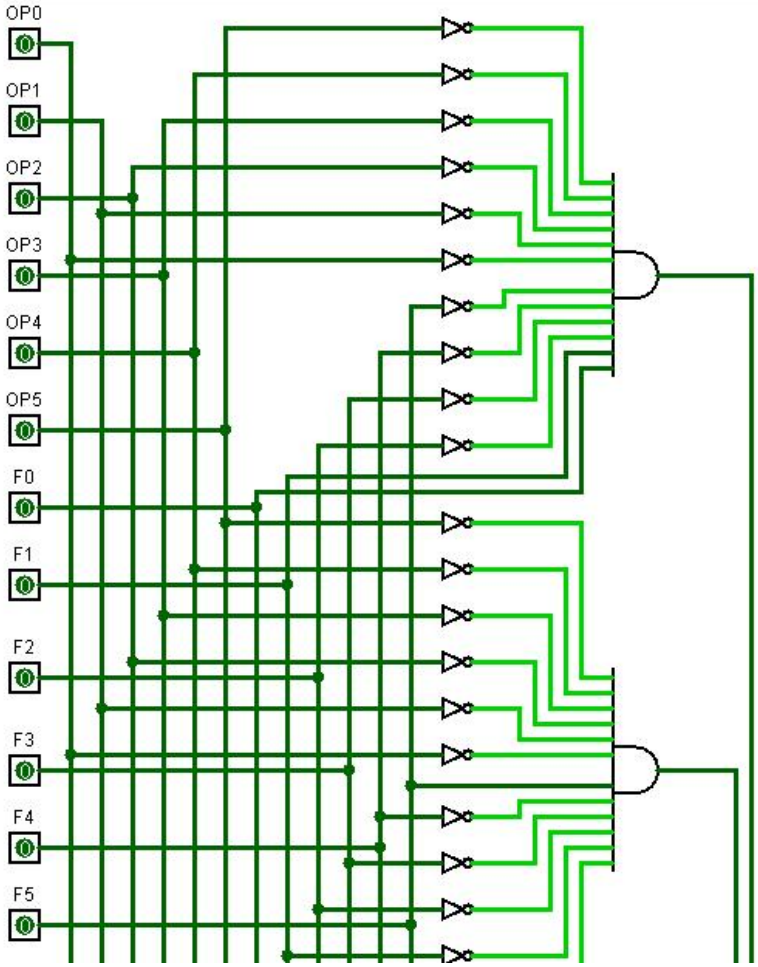

图 4.5 部分运算控制器电路

（四）控制信号

控制信号在计算机系统中起着至关重要的作用。它们用于协调和管理计算机各个部分的操作，确保系统能够正确执行指令。以下是控制信号的一些主要作用：

指令解码：控制信号帮助解码器识别和解释指令，从而确定需要执行的操作。

数据传输：控制信号管理数据在不同组件之间的传输，例如在寄存器、内存和 ALU（算术逻辑单元）之间。

时序控制：控制信号确保各个操作按正确的时间顺序进行，避免数据冲突和错误。

状态监控：控制信号用于监控系统的状态，并根据需要进行调整，例如在发生错误时触发中断处理。

本实验中，控制信号的电路由 Logisim 自动生成，生成所需的控制信号真值表如图 4.6 所示。

#	指令	OpCode (十进制)	FUNCT (十进制)	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL
1	SLL	0	0	0				1			1					
2	SRA	0	3	1				1			1					
3	SRL	0	2	2				1			1					
4	ADD	0	32	5				1			1					
5	ADDU	0	33	5				1			1					
6	SUB	0	34	6				1			1					
7	AND	0	36	7				1			1					
8	OR	0	37	8				1			1					
9	NOR	0	39	10				1			1					
10	SLT	0	42	11				1			1					
11	SLTU	0	43	12				1			1					
12	JR	0	8	X										1	1	
13	SYSCALL	0	12	X					1							
14	J	2	X	X											1	
15	JAL	3	X	X				1							1	1
16	BEQ	4	X	X								1				
17	BNE	5	X	X									1			
18	ADDI	8	X	5			1	1		1						
19	ANDI	12	X	7			1	1								
20	ADDIU	9	X	5			1	1		1						
21	SLTI	10	X	11			1	1		1						
22	ORI	13	X	8			1	1								
23	LW	35	X	5	1		1	1								
24	SW	43	X	5		1	1									

图 4.6 控制信号真值表

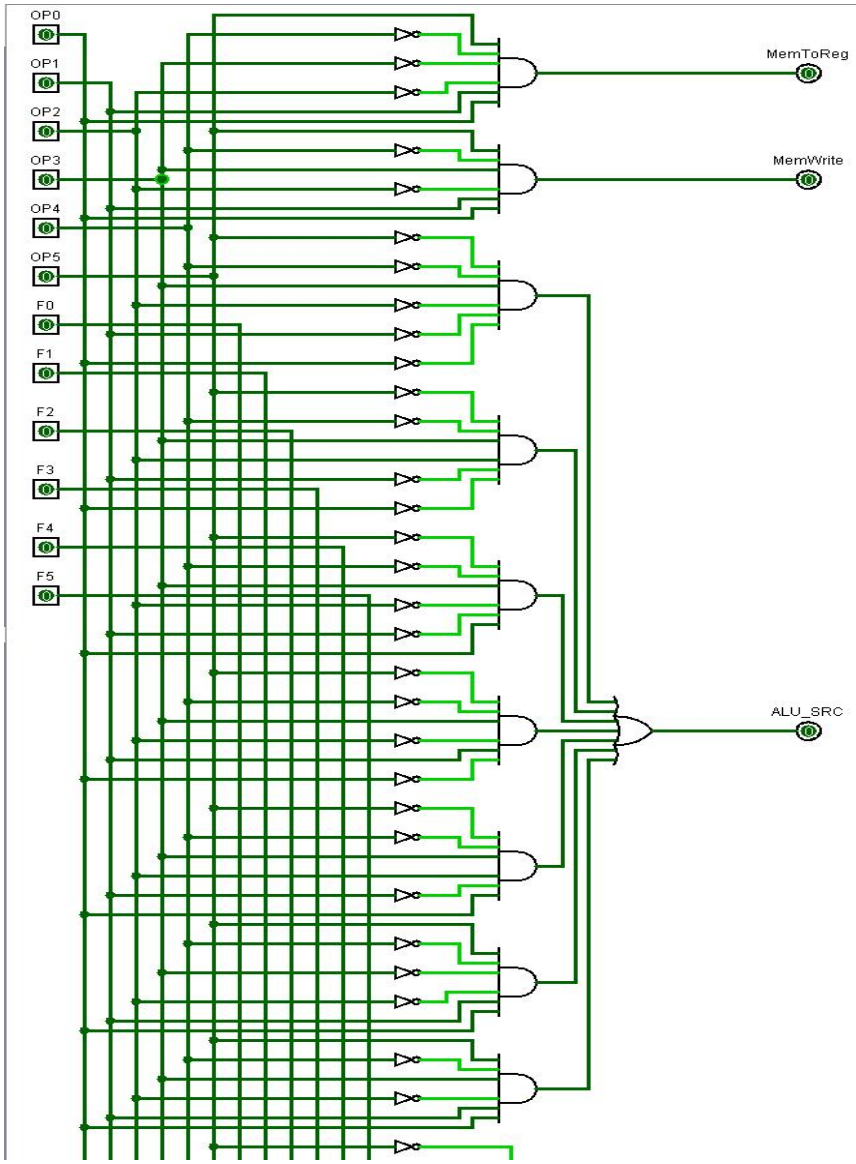

图 4.7 部分控制信号的电路图

五、指令的实现与测试

本章节将测试指令的正确性，将按照 R、I、J 三种类型进行测试，将会每种类型中的一组相似指令中选一个进行测试。

（一）R 型指令

ADD、SUB、ADDU 指令测试：

这一类指令实现了简单的加减运算，其执行流程为从指定寄存器地址中取处数据 rt 与 rs，送入 ALU 运算中放入寄存器 rd 中，其中 SUB 与 ADD 相比，数据通路只是控制器中多发出一个 ALUOP 信号让 ALU 从加法变为减法，ADDU 同理，具体区别如图 5.1.1，5.1.2,5.1.3 所示。

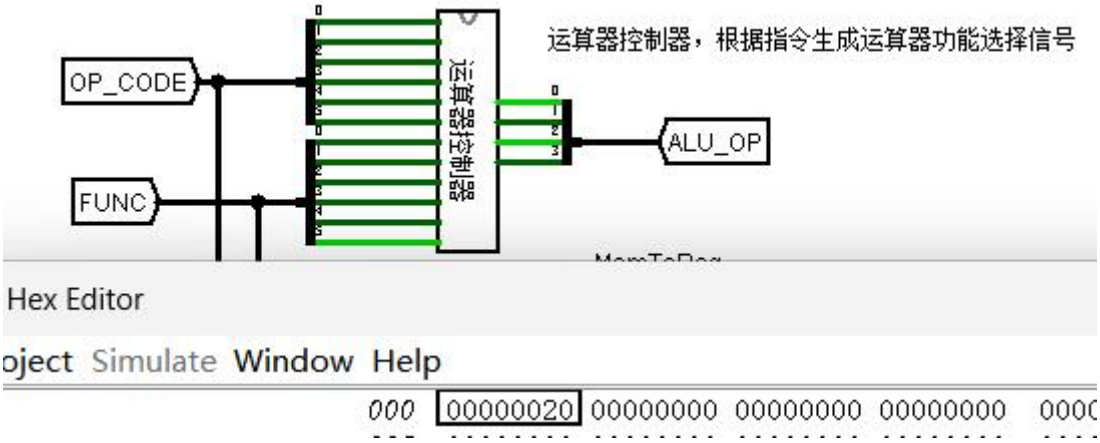

图 5.1.1 ADD 的运算控制器

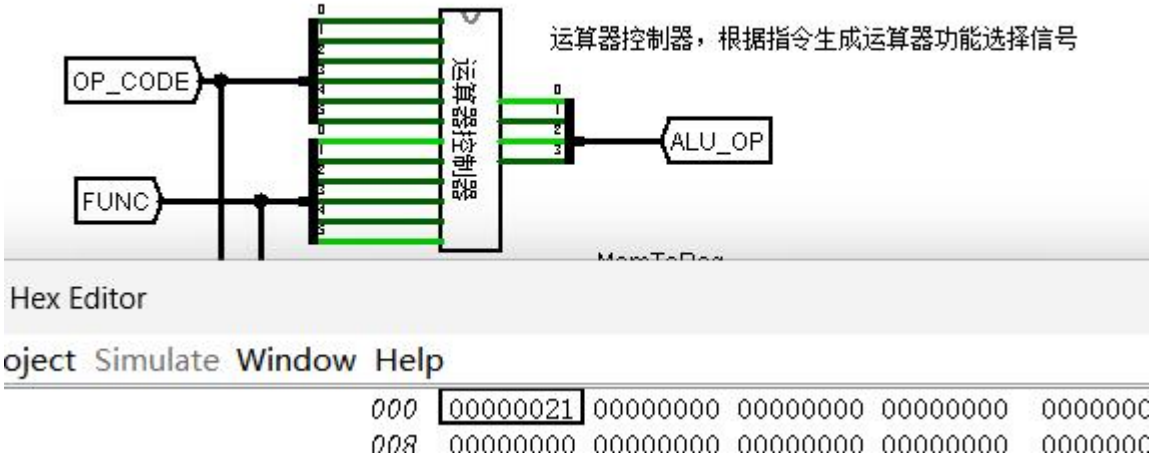

图 5.1.2 ADDU 的运算控制器

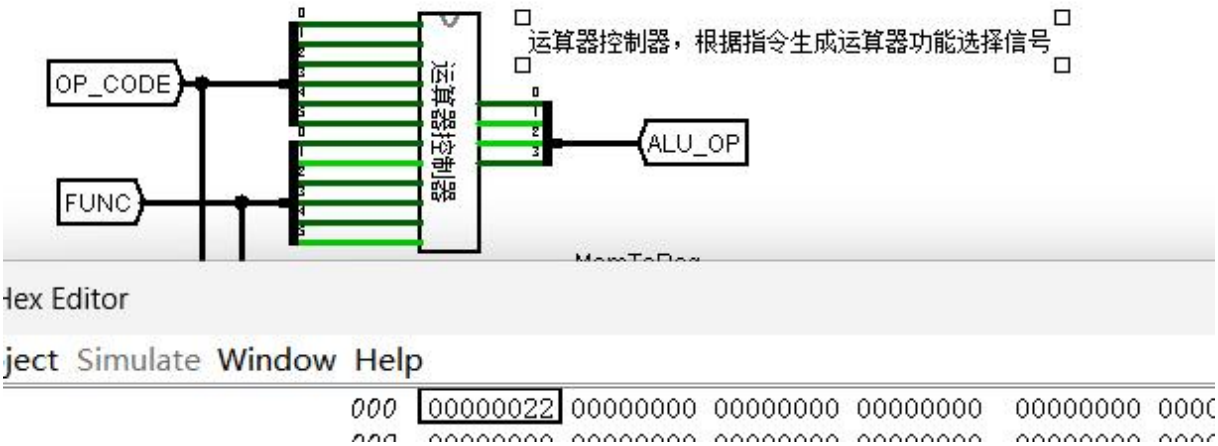

图 5.1.3 SUB 的运算控制器

这里我们只展示 ADD 的具体实现，当代码检测为 ADD 时，硬布线控制器发出 RegWrite、RegDst 两个信号，RegWrite 用于将寄存器设置为写模式，便于 ALU 运算结果写入寄存器中，RegDst 则被输入多路选择器中，将 rd 地址通路打开，让 rd 的地址输入寄存器中寻址，具体数据通路如图 5.1.4 所示。

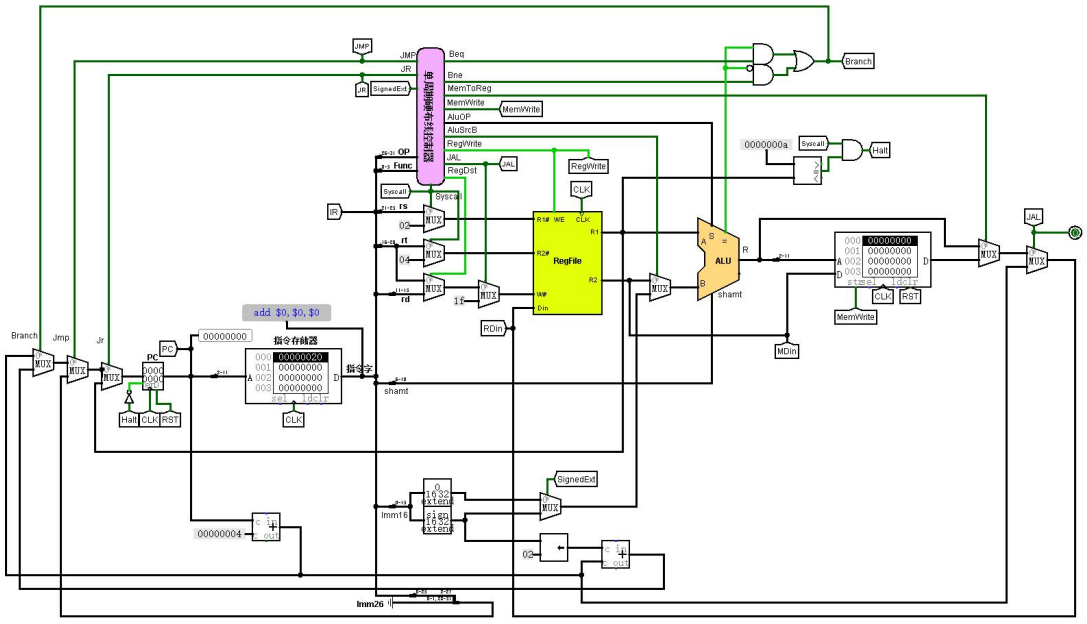

图 5.1.4 ADD 指令的数据通路

我们写一个简易程序进行测试，首先我们写入 20210001，意思是将寄存器 1 号加上数字一，如图 5.1.5，再写入 20420001，意思是将寄存器 2 号加上数字 1，如图 5.1.6，然后执行 00221820，意思是将 0 号寄存器与 1 号寄存器的内容相加存入 3 号寄存器中，若程序正确，我们将在 3 号寄存器中看到数字 2，如图 5.1.7。

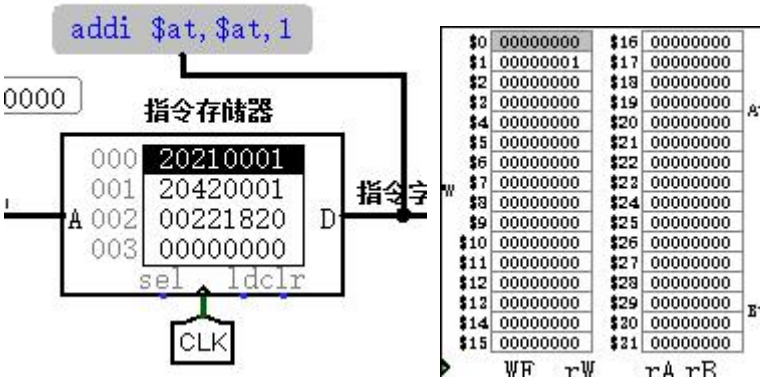

图 5.1.5

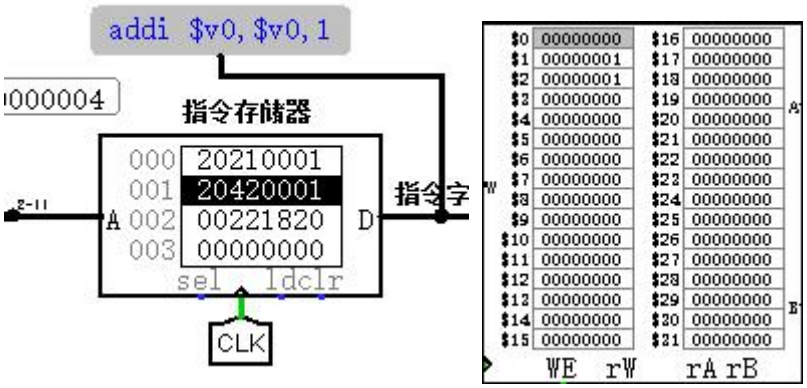

图 5.1.6

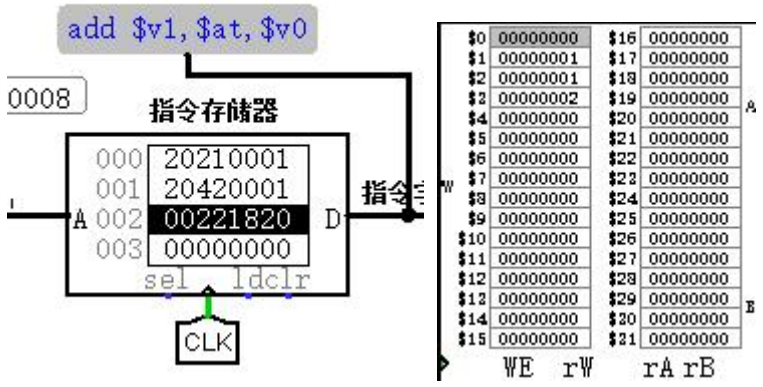

图 5.1.7

结果表明程序正确。

AND、OR、NOR 指令测试：

这一类指令实现了简单的逻辑运算，其操作数和结果均通过寄存器进行存取与上一类指令类似，这三条指令之间的区别也只是改变了 ALU 的操作信号，且相对于上一类指令，逻辑通路没有任何区别，也只是改变了 ALUOP 信号从而改变了 ALU 的运算规则，因此不做测试。

SLL、SRL、SRA 指令测试：

这一类指令实现分别实现了逻辑左移，算数左移与算数右移，其数据通路依然如图 5.4 所示，仅改变 ALUOP 的信号，我们这里仅对逻辑左移 SLL 进行测试。

我们写程序 20210001、00011140，这段指令将寄存器 1 号加上数字 1，如图 5.1.5，再对 1 号执行逻辑左移 5 位的指令，结果存入寄存器 2 中如图 5.1.8 所示。

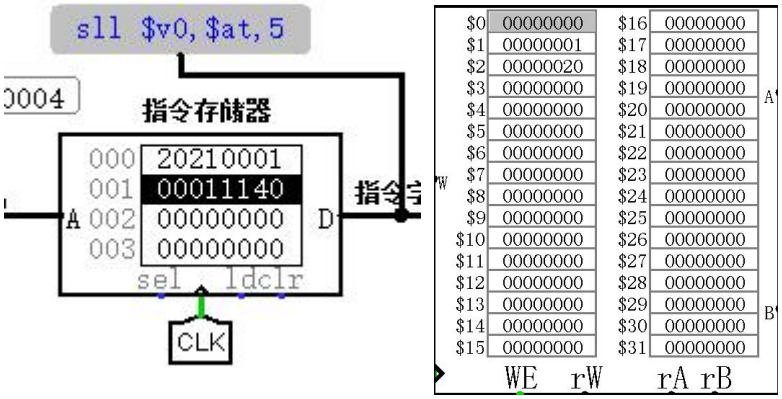

图 5.1.8 SLL 运行结果

可以看到 1 左移 5 位变成 32，此时 2 中存放的数据为 0010 0000，十进制为 32，证明程序正确。

JR 指令测试：

JR 指令为跳转指令，具体执行流程为取出 rs 寄存器中的操作数，直接修改 PC 的值，使得程序跳转至相应位置，程序数据通路如图 5.1.9。

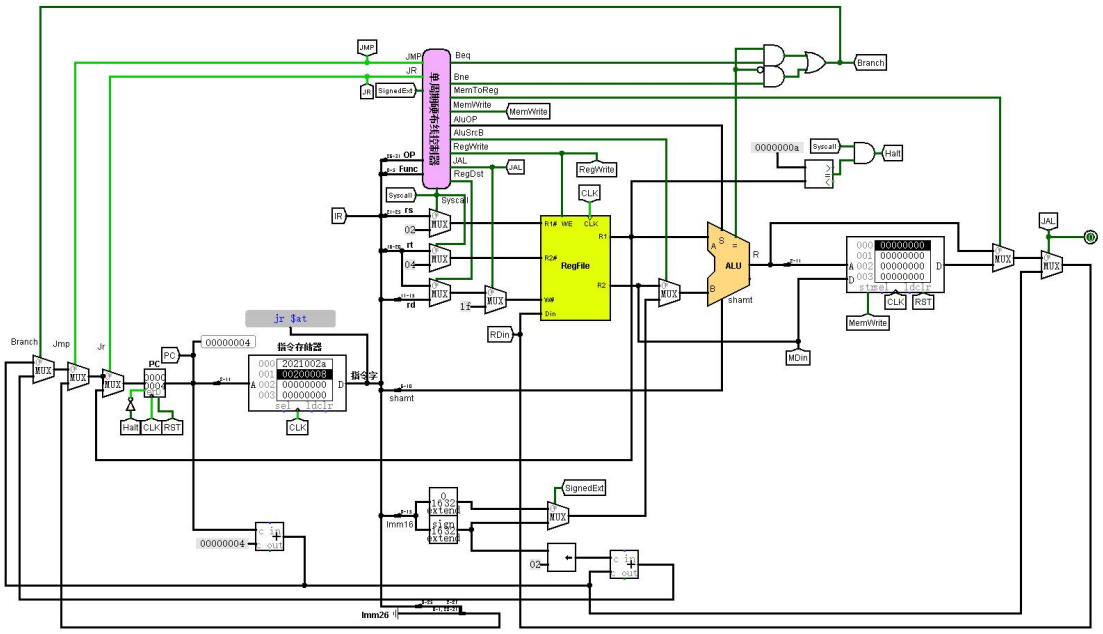

图 5.1.9 JR 指令的数据通路

根据上图分析可知，JMP 与 JR 的数据通路被打开，此时将从 RegFile 中取数据，送入多路选择器中，多路选择器将不再选用 PC+4 作为下一个 PC 值，而是 RegFile 中的数字。

我们写程序 2021 002a，意思时将寄存器 1 处加上数字 42，结果如图 5.1.10 我们再执行指令 0020 0008，意思是跳转至寄存器 1 所指位置，结果如图 5.1.11。

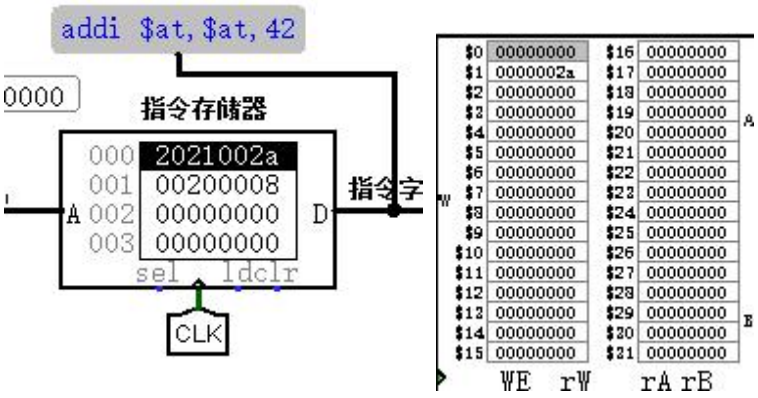

图 5.1.10 JR 结果一

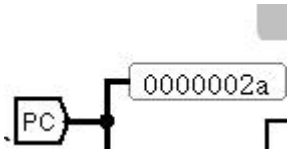

图 5.1.11 JR 结果二

如图可知，PC 值被设为 2a，十进制为 42，指令正确。

SLT、SLTU 指令测试：

这一类指令实现了比较运算，将 rs 与 rt 中的值进行比较，若 rs 小于 rt，则将 rd 的内容置为 1，否则置 0，SLTU 是 SLT 的无符号版本，它们之间的差距仅在 ALUOP 信号中体现，我们这里只对 SLT 指令进行测试。

测试程序：

```
ADDI  $1,$1,23
ADDI  $2,$2,34
ADDI  $3,$3,28
SLT  $1,$2,$4
SLT  $2,$3,$5
```

十六进制代码为：

```
2021  0017
2042  0034
2063  0028
0022  202a
0043  282a
```

程序将分别测试小于和大于的情况，测试结果如图 5.1.12。

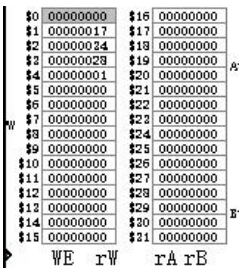

图 5.1.12 SLT 结果

（二）I 型指令

ADDI、ADDIU、ANDI、ORI 指令测试：

本组指令与 R 型指令 的 ADD、ADDU、AND、OR 指一一对应，为上述指令的立即数版本，它们的区别在 I 型指令会将 rs 寄存器的数据与一个立即数相加，而不是另一个寄存器的数据，这里我们只对 ADDI 进行测试，数据通路如图 5.2.1，测试结果如图 5.2.2。

测试指令：

ADDI $0,$0,23

十六进制代码:

2021 0017

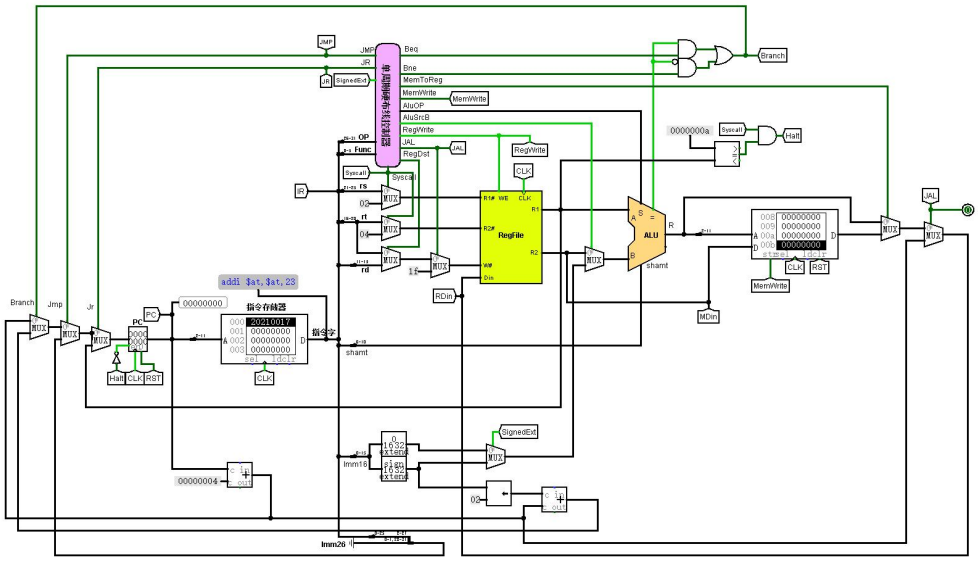

图 5.2.1 数据通路

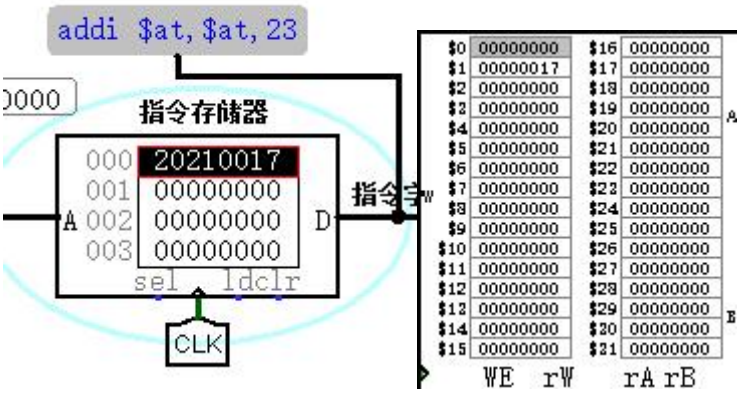

图 5.2.2 测试结果

LW、SW 指令测试：

SW 指令作用是存一个寄存器中的数据到内存中，而 LW 的作用是取内存中一个数据到寄存器中，我们将用一个程序对两个指令进行测试，数据通路如图 5.2.3 和图 5.2.4，测试结果如图 5.2.5。

测试指令：

ADDI $1,$1,7//寄存器 1 号存储数据 1

ADDI $2,$2,4//寄存器 2 号存储数据 4，后续访问 4 号内存块需要借助这个寄存器

SW $2,$1,2//将寄存器 2 号数据存入寄存器 1 号所指向的位置，并偏移 2 位

LW $3,$2,2//将寄存器 2 指向位置偏移 2 位的数据存入寄存器 3 号中

十六进制代码：

2021 0007

2042 0004

Ac22 0002

8c23 0002

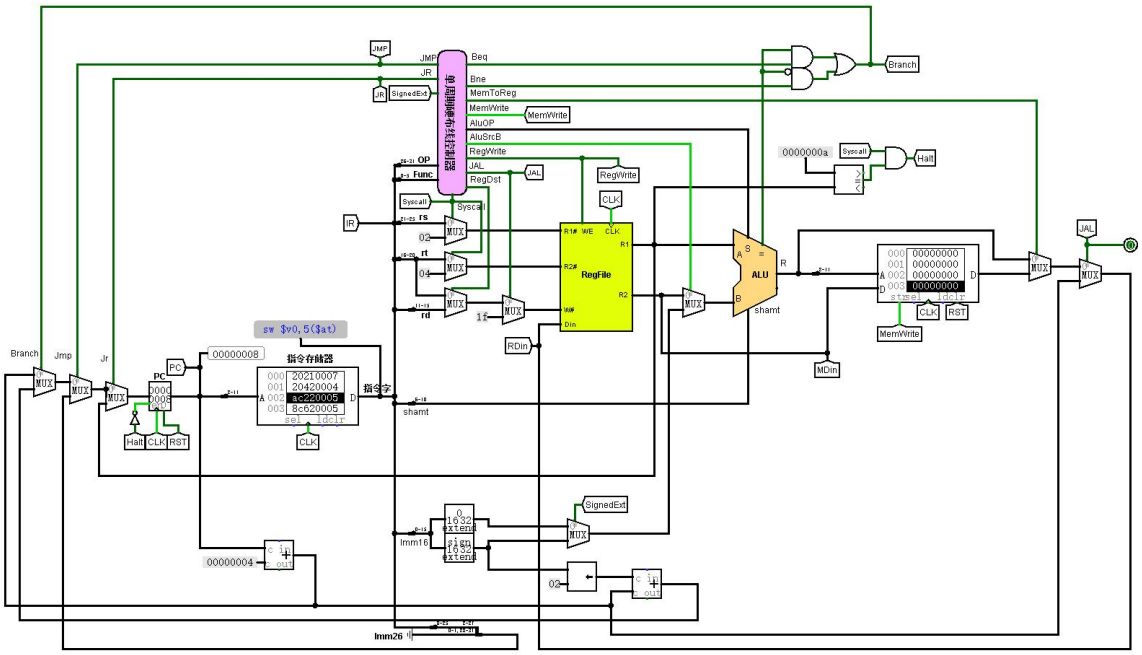

图 5.2.3 SW 数据通路

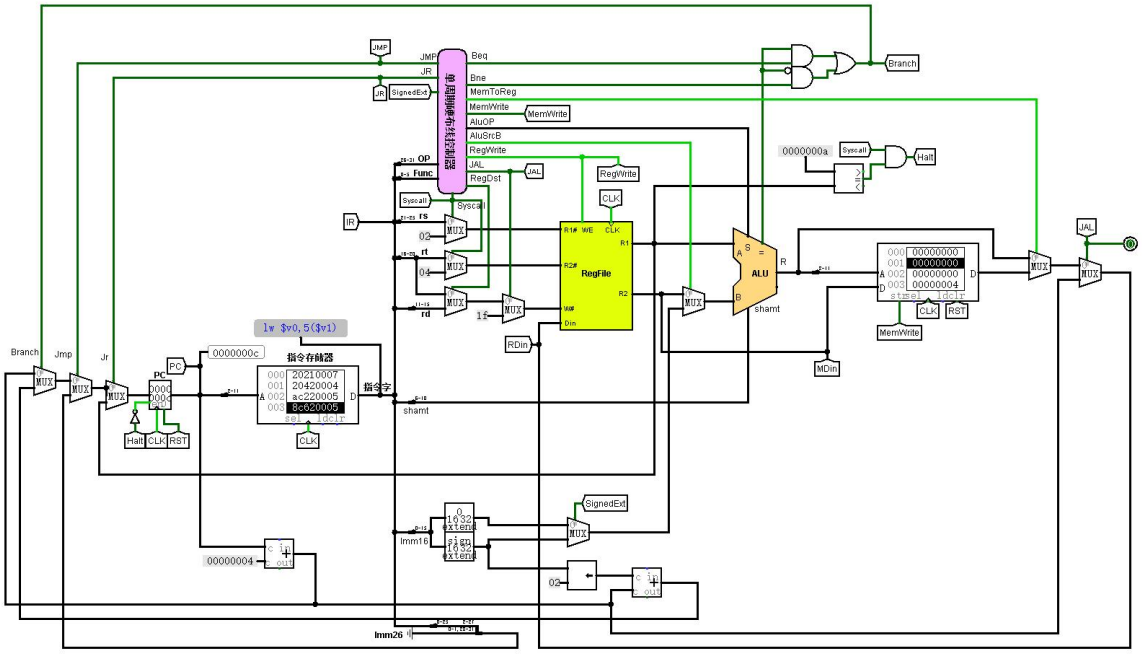

图 5.2.4 LW 数据通路

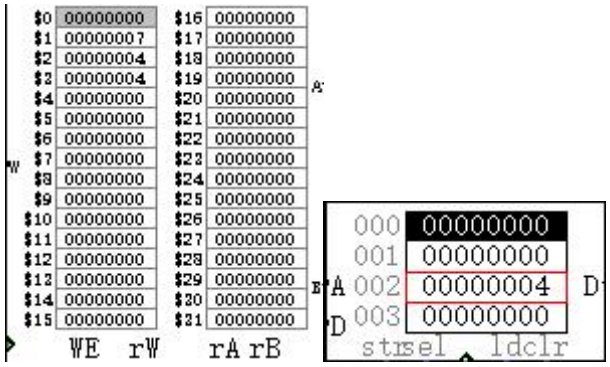

图 5.2.5 结果

我们可以看到寄存器$3 位置获取到了数据 4，而内存中相应位置被存入数据 4，可知指令执行正确。

BEQ、BNE 指令测试：

这两条指令含义分别是相等则转移，第二条是不相等则转移，指令会取 rs、rt 寄存器的内容进行比较，按照结果执行下一步，我们这里仅对第一条进行测试,数据通路如图 5.2.6，结果如图 5.2.7。

测试代码：

ADDI $1,$1,5

ADDI $2,$2,5

BEQ $1,$2,0

十六进制代码：

2021 0007

2042 0004

1022 029A

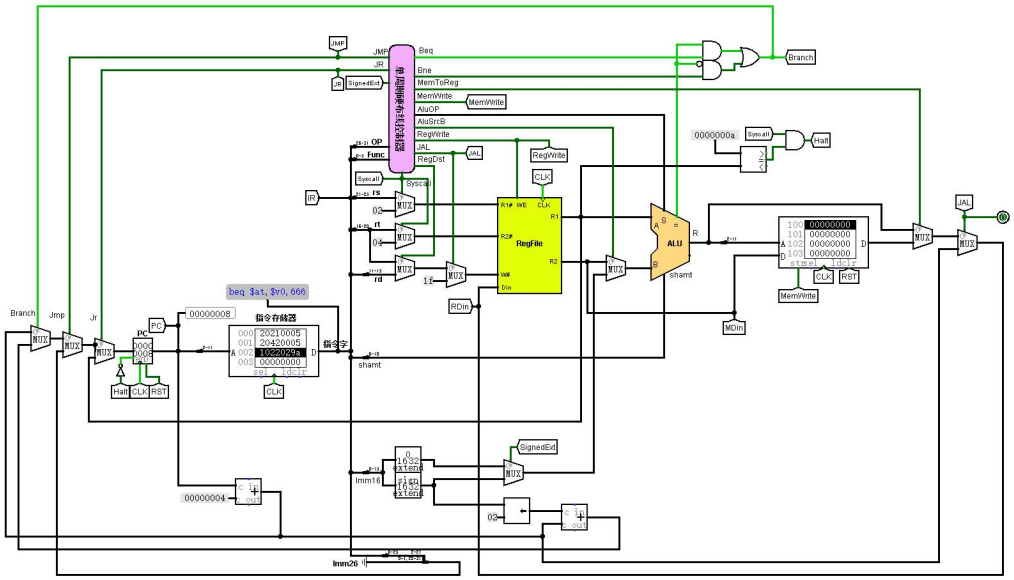

图 5.2.6 数据通路

可以看到当所指定的寄存器内部的值相等时，控制位 BEQ 连通，此时将选择立即数 Imm26 通过多路选择器输入 PC 中，改变 PC 值实现跳转。

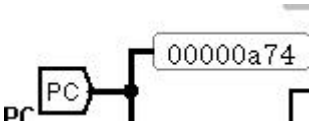

图 5.2.7 执行结果

（三）J 型指令

J、JAL 指令测试：

这两条指令的作用都是无条件跳转，但是 JAL 是函数调用时使用的跳转，跳转后会把返回数据存在指定位置，因此会保存部分信息，我们这里仅对 J 指令进行测试,数据通路

如图 5.3.1 所示，结果如图 5.3.2 所示。

测试指令：

J 0x10

十六进制代码：

0800 0010

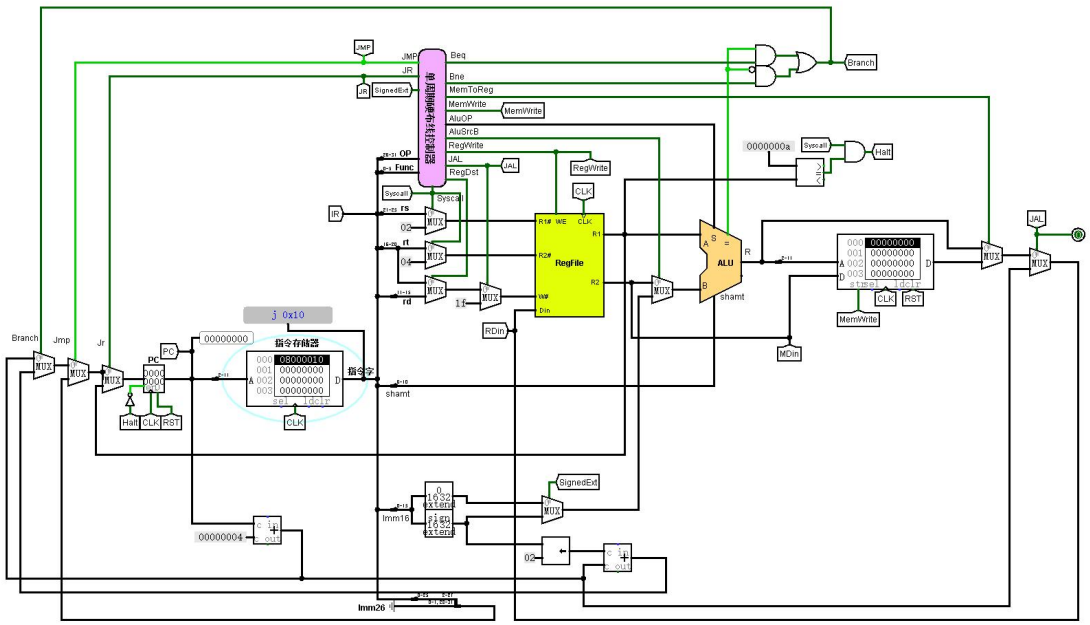

图 5.3.1 数据通路

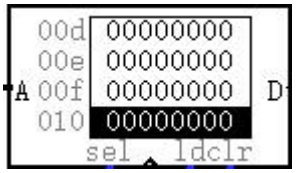

图 5.3.2 结果

可以看到指令寄存器跳转至 10 处，指令正确执行。

（四）综合测试（冒泡排序）

综合测试中我将实现一个简单的冒泡排序程序，综合上述指令，进行系统性的指令正确性检查。

十六进制代码：

v2.0 raw

20110001 8000c05 20110001 20120002 20130003 8000c09 20110001 20120002

20130003 8000c0d 20110001 20120002 20130003 8000c11 20110001 20120002
20130003 c000cb8 20100001 20110001 118fc0 112020 20020022 c
118882 12200001 8000c15 112020 20020022 c 20110001 118880
112020 20020022 c 12200001 8000c1f 20110001 118fc0 112020
20020022 c 1188c3 112020 20020022 c 118903 112020
20020022 c 118903 112020 20020022 c 118903 112020
20020022 c 118903 112020 20020022 c 118903 112020
20020022 c 118903 112020 20020022 c 118903 112020
20020022 c 20100001 109fc0 139fc3 8021 2012000c 24160003
26100001 3210000f 20080008 20090001 139900 2709825 132020 20020022
c 1094022 1500fff9 22100001 2018000f 2188024 108700 20080008
20090001 139902 2709825 132021 20020022 c 1094022 1500fff9
108702 2c9b022 12c00001 8000c50 4020 1084027 84400 3508ffff
82021 20020022 c 2010ffff 20110000 ae300000 22100001 22310004
ae300000 22100001 22310004 ae300000 22100001 22310004 ae300000 22100001
22310004 ae300000 22100001 22310004 ae300000 22100001 22310004 ae300000
22100001 22310004 ae300000 22100001 22310004 ae300000 22100001 22310004
ae300000 22100001 22310004 ae300000 22100001 22310004 ae300000 22100001
22310004 ae300000 22100001 22310004 ae300000 22100001 22310004 ae300000
22100001 22310004 ae300000 22100001 22310004 22100001 8020 2011003c
8e130000 8e340000 274402a 11000002 ae330000 ae140000 2231fffc 1611fff8
102020 20020022 c 22100004 2011003c 1611fff2 20020032 c
20100000 22100001 102020 20020022 c 22100002 102020 20020022
c 22100003 102020 20020022 c 22100004 102020 20020022
c 22100005 102020 20020022 c 22100006 102020 20020022
c 22100007 102020 20020022 c 22100008 102020 20020022
20020022
c
3e00008

结果如图 5.3.3 所示。

图 5.3.3 综合测试结果

程序运行结束后，所有数据被按照从大到小的顺序排列，说明程序运行成功，该处理器在执行多个指令构成的程序时没有问题。

六、实验总结

（一）研究结果与规律

本研究设计并实现了一个 MIPS 单周期 24 指令处理器，解决了单周期处理器设计中的数据通路和控制信号生成问题。通过实验验证，处理器能够正确执行 MIPS32 指令集中的 24 条核心指令，证明了设计的正确性和有效性。

（二）研究的不足之处

尽管本研究取得了一定的成果，但仍存在一些不足之处。首先，实验中仅选取了 MIPS32 指令集中的 24 条指令，未能覆盖所有指令，可能会影响处理器在实际应用中的全面性。

其次，实验环境较为简单，未能模拟复杂的应用场景，可能会影响处理器在实际应用中的性能表现。此外，实验中使用的 Logisim 软件虽然便于设计与模拟，但在实际硬件实现中可能会遇到更多的挑战。

（三）研究时遇到的困难

在指令测试阶段，我参考了很多资料，有些资料用的汇编语言格式时 Intel 的格式，还有一些用的是 AT&T 格式，我最开始并没有特别在意格式问题，导致在某几条指令测试时发生了格式错误，自己写的代码出现了乱码，尤其在 LW 与 SW 指令中，两种格式的逻辑几乎相反，对我的测试造成了非常大的阻碍。

我还认为手动将汇编语言转换成 16 进制代码虽然短时间内很省事，但是当需要测试的代码多起来后会非常低效率，如果下次我还要进行相关实训，我会提前用 c 语言写一个用于转换的程序，提高效率。

（四）后续研究建议

针对上述不足之处，后续研究可以从以下几个方面进行改进和扩展。首先，可以进一步扩展指令集的覆盖范围，设计并实现更多的 MIPS32 指令，以提高处理器的全面性和适用性。其次，可以在更复杂的实验环境中进行测试，模拟实际应用场景，以验证处理器在不同条件下的性能表现。此外，可以尝试在实际硬件平台上实现设计，解决在硬件实现过程中可能遇到的问题，为实际应用提供更为可靠的解决方案。

参考文献

[1] 华中科技大学计算机科学与技术学院组编，谭志虎主编，周军龙、肖亮副主编．计算机组成原理实验指导与习题解析[M]．北京：人民邮电出版社，2022 年 2 月 1 版．

[2] 张俊涛、陈晓莉．数字电路与逻辑设计（第 3 版）[M]．北京：清华大学出版社，2023 年 8 月 1 版．

[3] 臧利林、徐向华、姚福安．数字电子技术基础[M]．北京：清华大学出版社，2022 年 7 月 1 版．

[4] 数字逻辑 第 4 版[M]．北京：机械工业出版社，2024 年．

[5] 李明、王强．计算机组成原理与系统结构[M]．北京：高等教育出版社，2023 年 3 月 1 版．