

华中科技大学

2024

计算机组成原理

·实验报告·

专    业：            网络空间安全  
\_\_\_\_\_  
班    级：            \_\_\_\_\_  
学    号：            \_\_\_\_\_  
姓    名：            \_\_\_\_\_  
电    话：            \_\_\_\_\_

# 华中科技大学课程实验报告

---

## 评 分 表

评 分 项			总分	得分
实 验	单周期 MIPS CPU	设计思路	15	
		电路图	5	
		测试图	5	
		故障分析	10	
	多周期 MIPS CPU	设计思路	15	
		电路图	5	
		测试图	5	
		故障分析	10	
设计 总结	实验总结		10	
	实验心得		10	
格 式	段落缩进是否规范		2	
	图表及编号是否规范		2	
	字号字体是否统一		2	
	是否签名		2	
	其他		2	
合 计				
教 师 签 名				

## 目 录

<b>1</b>	<b>实验概述 .....</b>	<b>3</b>
1.1	实验名称 .....	3
1.2	实验所需软件及设备 .....	3
1.3	实验目的 .....	3
1.4	实验基础 .....	3
1.5	实验内容及要求 .....	3
<b>2</b>	<b>CPU 设计实验 .....</b>	<b>5</b>
2.1	单周期 MIPS CPU 设计 .....	5
2.2	多周期 MIPS 微程序 CPU 设计 .....	14
<b>3</b>	<b>总结与心得 .....</b>	<b>26</b>
3.1	实验总结 .....	26
3.2	实验心得 .....	26

## 1 实验概述

### 1.1 实验名称

CPU 设计---单周期和多周期 MIPS CPU(8 条指令)。

### 1.2 实验所需软件及设备

- (1) Logisim2.7.1 软件 1 套
- (2) 微型计算机 1 台；

### 1.3 实验目的

理解 MIPS 单周期和多周期处理器的基本原理,能分别利用硬布线控制器和微程序控制器的设计原理在 Logisim 平台中设计实现 MIPS 单周期和多周期处理器。

### 1.4 实验基础

硬布线和微程序控制器基本原理、MIPS 指令执行流程、寄存器相关知识、存储器访问相关基础知识等。

### 1.5 实验内容及要求

完成单周期硬布线控制器和多周期微程序控制器电路的设计。

了解单周期硬布线控制器和多周期微程序控制器电路的基本概念及使用原理,支持表 1 中的 8 条核心指令,实现 MIPS 处理器能运行实验包中 sort 排序程序,利用冒泡排序对数据进行升序排序。

# 华中科技大学课程实验报告

表 1 MIPS 指令及功能描述

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

## 2 CPU 设计实验

### 2.1 单周期 MIPS CPU 设计

#### (1) 设计思路

##### (1.1) 总体设计思想

单周期 MIPS CPU 的核心特征是所有指令在单一时钟周期内完成执行。这种设计采用硬布线控制方式，确保各功能部件协调工作并保持时钟同步。为避免取指令和访问数据时的结构冲突，采用了指令存储器和数据存储器分离的架构。控制信号通过组合逻辑电路直接生成，整个电路设计在 Logisim 软件中实现。本实验主要关注 MIPS 指令集中的 R 型和 I 型指令，不包括 J 型指令。

CPU 的基本构成包括控制器、运算器和存储器。其工作过程可划分为五个主要阶段：指令获取、指令解码、指令执行、内存访问和结果写回。其中，内存访问阶段负责根据指令中的地址信息从主存中读取操作数。其他阶段的具体流程和相互关系可通过一个详细的流程图来清晰展示。

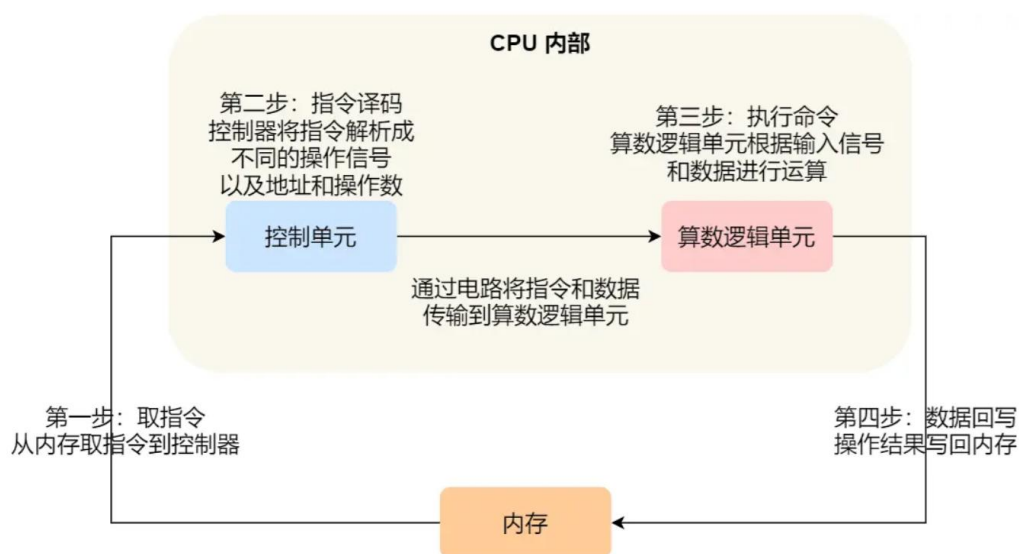


图 1 取指令阶段流程图

为实现所有指令在单一时钟周期内完成，需要对 CPU 的整体执行效率进行优化。主要策略包括：使用专用数据通路设计各功能模块，确保指令执行过程中

# 华中科技大学课程实验报告

数据传输无冲突，从而提高传输效率；采用分离的指令和数据存储器，提高存储访问效率，因为单周期设计要求在同一周期内完成指令获取和执行；使用硬布线控制器生成控制信号，以加快处理速度。

单周期 MIPS CPU 的设计理念主要着眼于简化控制逻辑，使整个 CPU 的设计和实现更加直观。通过采用固定的执行周期，它避免了流水线等复杂执行方式中可能出现的数据冒险和相关性问题，从而使控制逻辑更加清晰和易于理解。这种设计虽然在某些方面可能牺牲了性能，但在学习和基础研究中具有重要价值，为理解计算机体系结构的基本原理提供了理想的平台。

## (1.2) 各种指令数据通路

指令数据通路包括取指令数据通路、R 型指令数据通路、I 型指令数据通路三类，取指令数据通路是根据 PC 的指令地址取出指令存储器中对应指令，然后 PC 累加器作用，PC 指向下一指令地址；R 型指令数据通路根据硬布线控制器控制 ALU 功能；I 型指令数据通路则是据指令功能对数据存储器 and 寄存器组进行不同操作。程序计数器，保存当前运行的指令的地址的寄存器；数据通路合并的目标，把各种功能的数据路径合并，使得每个时钟周期执行一条指令；基本方法，使用多路选择器与控制信号。

之后经常使用到的控制信号如下表所示，下个实验电路设计同理，报告中将不再赘述。

表 1 控制信号条件表

#	控制信号	信号说明	产生条件
1	MemToReg	写入寄存器的数据来自存储器	lw指令
2	MemWrite	写内存控制信号	sw指令 未单独设置MemRead信号
3	Beq	Beq指令译码信号	Beq指令
4	Bne	Bne指令译码信号	Bne指令
5	AluOP	运算器操作控制符	加法，比较两种运算
6	AluSrcB	运算器第二输入选择	Lw指令，sw指令，addi
7	RegWrite	寄存器写使能控制信号	寄存器写回信号
8	RegDst	写入寄存器选择控制信号	R型指令
9	Halt	停机信号，取反后控制PC使能端	syscall指令

### 1) 取指令数据通路 (mov PC, Mem[PC++])

# 华中科技大学课程实验报告

运算器与 PC 累加器分离，程序和数据分开存放：原本为  $\text{Mem}[\text{PC}++] \rightarrow \text{IR}$ ,  $\text{IR} \rightarrow \text{PC}$ ，由于没有 IR 寄存器，直接将  $\text{PC}+1$  放到 PC 计数器中，在电路中通过计数器来具体实现。

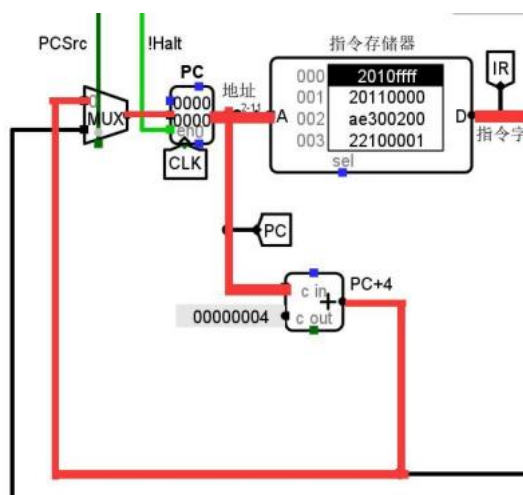


图 2 取指令数据通路

## 2) R 型指令 (ADD s0,s1,s2)

取  $s_0+s_1$ ，将结果存入  $s_2$  中。采用 MIPS 寄存器文件和一个 ALU 加法运算器实现一个周期内完成两个数的运算并将其返回寄存器中，并且在中途将数据存入数据存储器，在电路中通过寄存器写入信号  $\text{RegWrite}$  和相加信号  $\text{AluOp}$  来控制。

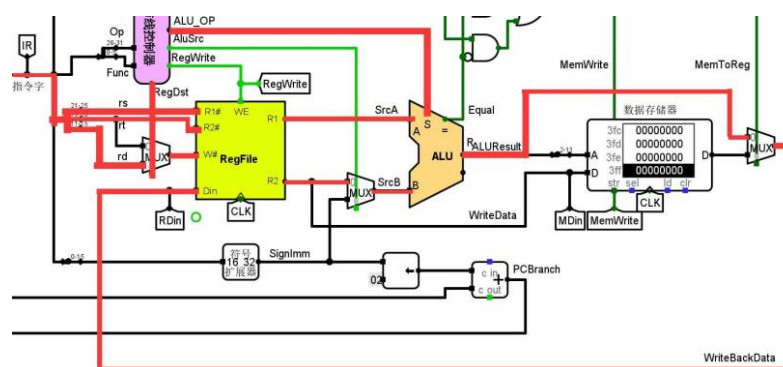


图 3 R 型运算数据通路

## 3) LW 指令 (LW s0,32(s1))

从数据存储器读出数据：由于采用 I 型指令，低十六位为立即数  $\text{SignImm}$ ，第 16-20 位为目的操作数  $R_t$ ，第 21-25 位为源操作数  $R_s$ ，第 26-31 位为指令字，此处采用相对寻址方式，具体实现原理为  $\text{Mem}[\text{Rs}+\text{SignImm} \ll 2] \rightarrow R_t$ 。因此在电路中，采用位扩展器将低十六位的立即数部分扩展成 32 位后左移 2 位，



# 华中科技大学课程实验报告

通过与源操作数  $R_s$  使用 ALU 运算器相加后,形成地址到数据存储器中寻找数据并写入  $R_t$  中。

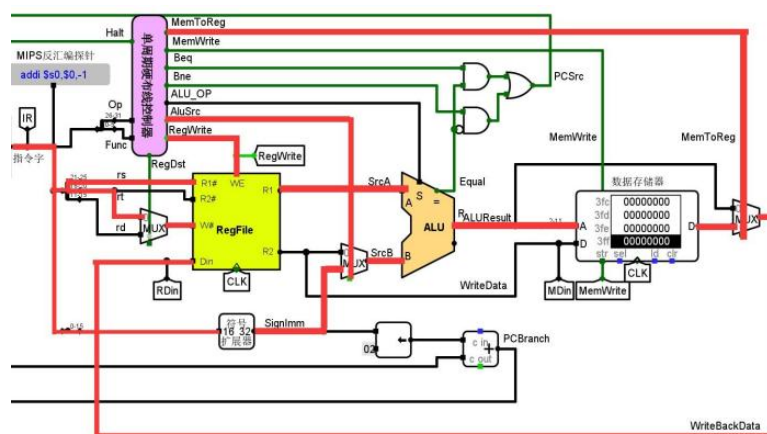


图 4 LW 指令数据通路

## 4) SW 指令 (SW s0,32(s1))

将数据存入数据存储器：采用的指令和寻址方式同 LW，具体实现原理为  $R_t \rightarrow \text{Mem}[R_s + \text{SignImm} \ll 2]$ 。与 LW 地址形成方式相比有一个 MemWrite 内存写入信号，将  $R_t$  的数据写入由 SignImm 和  $R_s$  形成的地址中。

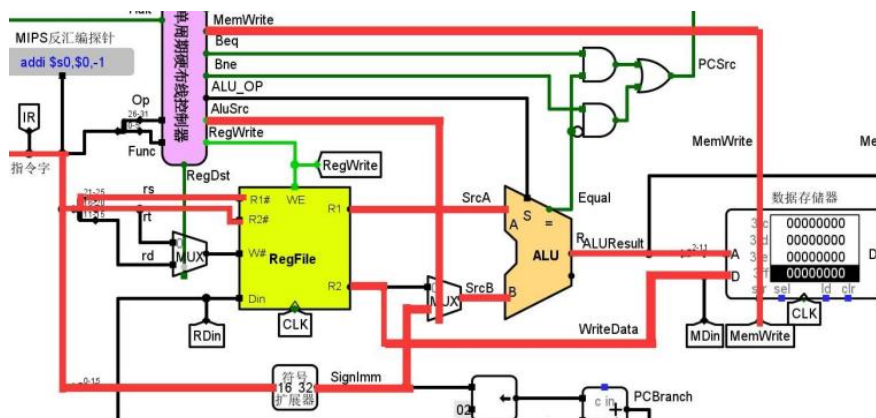


图 5 SW 指令数据通路

## 5) ADDI 指令 (ADDI rt,rs,imm)

该指令中的  $rs$  字段被传输到寄存器堆的  $R1\#$  读端口,并通过控制信号  $\text{RegDst}=0$  来进行控制;而  $rt$  字段则被传输到寄存器堆的  $W\#$  写端口,并将寄存器堆从  $R1$  端口读取的值输出到 ALU 运算器。为了实现这一过程,控制信号  $\text{ALUSrc}=1$ ,使得 ALU 的第二个运算数为指令中经过拓展的立即数。再通过控制信号  $\text{ALU\_OP}=0$ ,控制 ALU 进行加法运算,并将运算结果作为输入数据传输到寄存器堆的

# 华中科技大学课程实验报告

Din 端口。同时通过控制信号 RegWrite=1，在时钟信号到来时，将 Din 中的运算结果写入对应 rt 字段所对应的寄存器中。

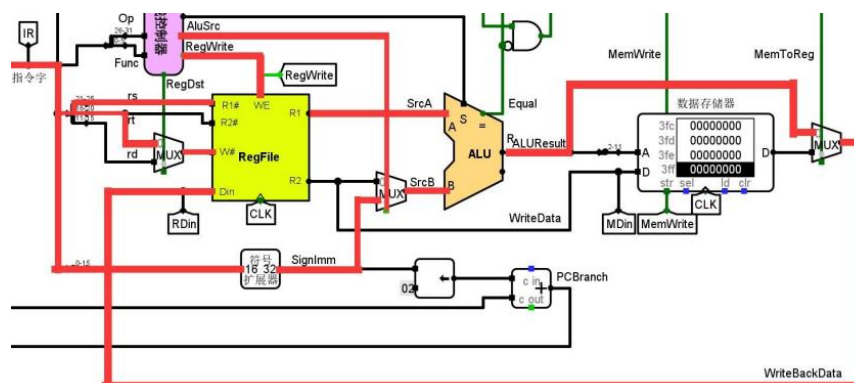


图 6 ADDI 指令数据通路

## 6) BEQ/BNE 指令 (BEQ/BNE rs,rt,imm)

指令中的 rs 和 rt 字段通过调用传输到寄存器堆的 R1#和 R2#读端口，将从寄存器堆中读取的 R1 和 R2 端口的值输出到 ALU 运算器。当这两个值相等/不等时，设置 Equal=1/0，并且设置相应控制信号的值。在时钟信号到达时，通过控制 PC 存储器实现跳转，跳转地址为 PC+SignExt18b。为了实现地址的移位和加法运算，还专门设计了独立的运算器。

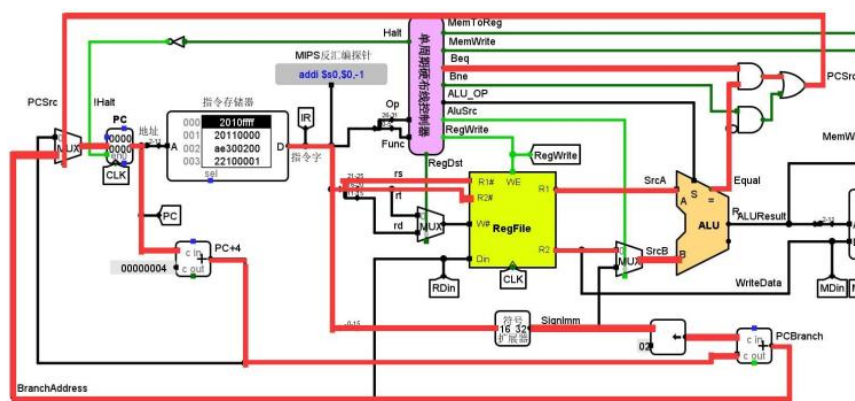


图 7 BEQ 指令数据通路

## 7) SysCall 指令

该指令使 Halt 控制信号值为 1，Halt 控制信号取非后连接 PC 寄存器使能端，此时 PC 寄存器使能端为 0，停止工作。

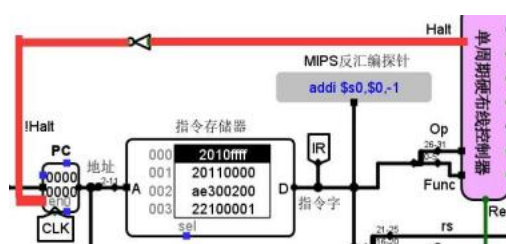


图 8 SysCall 指令数据通路

## (1.3) 单周期 MIPS 控制器设计过程

### 1) 指令译码逻辑

#	指令	OpCode (十进制)	FUNCT (十进制)	ALU_OP
1	ADD	0	32	5
2	SLT	0	42	b
4	SYSCALL	0	12	5
5	BEQ	4	X	5
6	BNE	5	X	5
7	ADDI	8	X	5
9	LW	35	X	5
10	SW	43	X	5

图 9 指令的 OpCode 字段和 FUNCT 字段

根据以上指令的 OpCode 字段和 FUNCT 字段结合 Logisim 自带的比较器，可以设计出指令译码逻辑部分，SysCall 输入特殊的 R 型指令，所以 R\_TYPE 指令形成的信号中 SysCall 要取反再与其他 R 型指令取或的结果做与，而不是直接三个均取或。最终逻辑电路如下图所示。

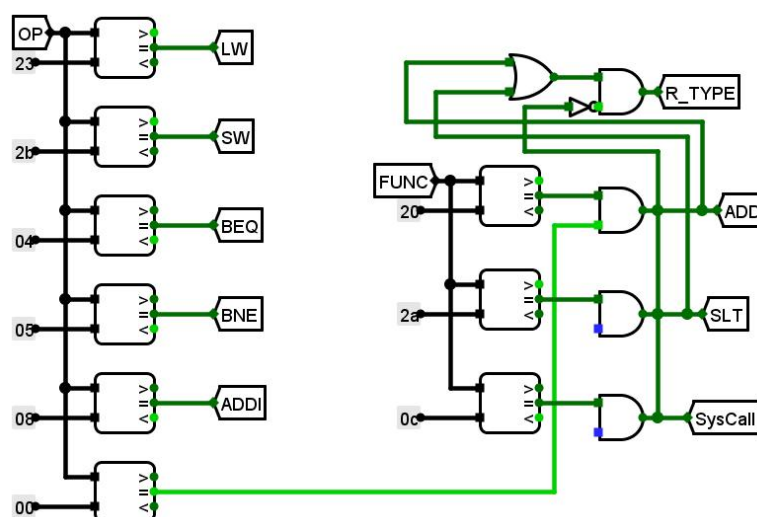


图 10 单周期 MIPS 硬布线控制器-指令译码逻辑电路

### 2) ALU 控制逻辑

由指令控制逻辑，可得只有 SLT 需要用到 ALU 的比较运算，其他均为加

法运算，所以可以直接复制一个 SLT 的隧道加上数据选择器，直接设计 ALU 的控制逻辑，不需要用到给定的 OP 和 FUNC 字段的隧道。最终逻辑电路如下图所示。

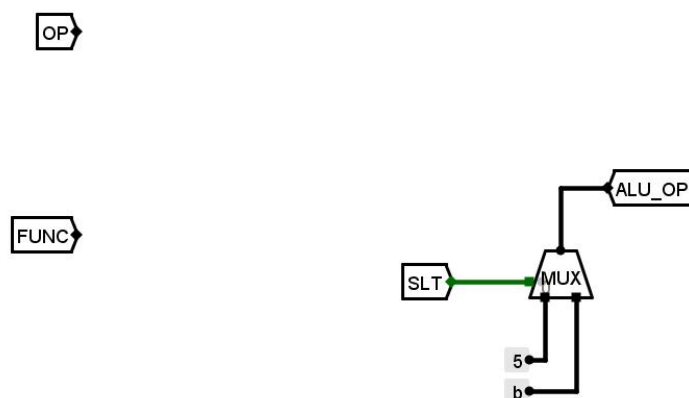


图 11 单周期 MIPS 硬布线控制器 ALU 控制器逻辑电路

### 3) 机器输出信号的控制

根据所给出控制信号的产生条件，结合所学数字逻辑知识，可以设计输出控制信号的逻辑。注意 AluOP 控制器逻辑设计在上一步中已完成，表中 RegWrite 产生条件为“寄存器写回信号”，由表 1 知有 R 型指令、ADDI 和 LW 三种指令会产生此信号，最终逻辑电路如下图所示。

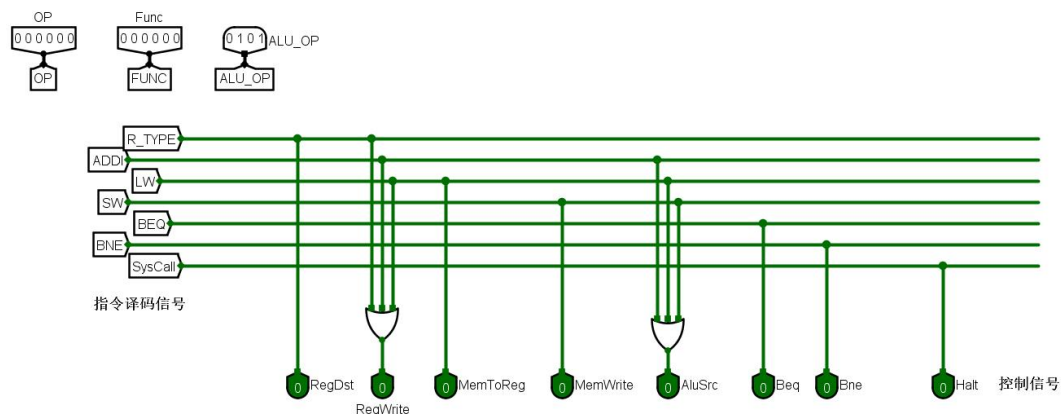


图 12 单周期 MIPS 硬布线控制器-控制信号逻辑电路

## (2) 电路图

### (2.1) 硬布线控制器电路图

# 华中科技大学课程实验报告

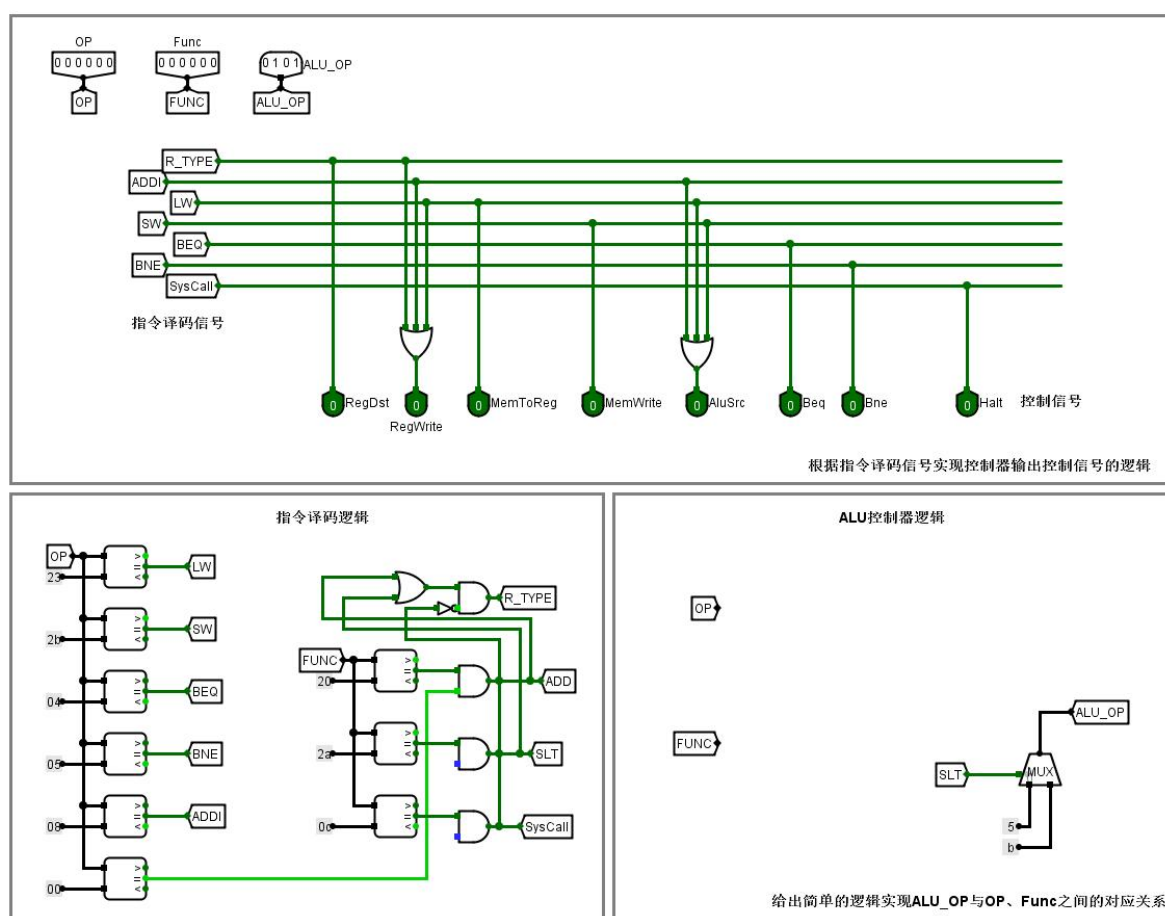


图 13 单周期 MIPS 硬布线控制器电路图

## (2.2) CPU 设计图

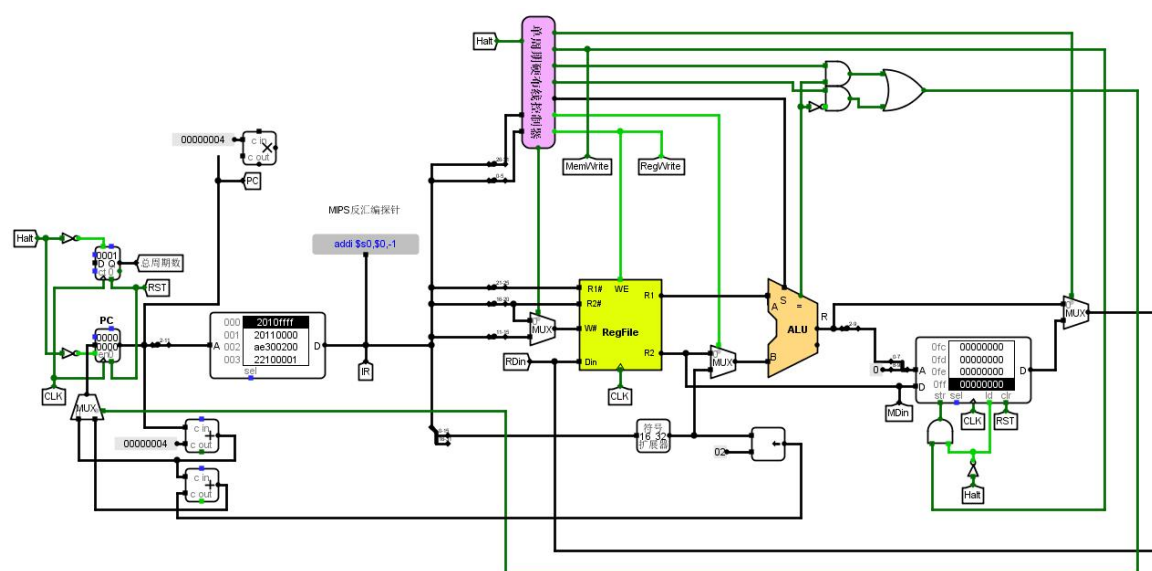


图 14 单周期 MIPS CPU 电路图

## (3) 测试图



# 华中科技大学课程实验报告

如下图所示分别是 Logisim 和头歌的测试图，均通过测试样例表示无错误输出，说明电路输出与实际情况相符，证明了电路设计正确。

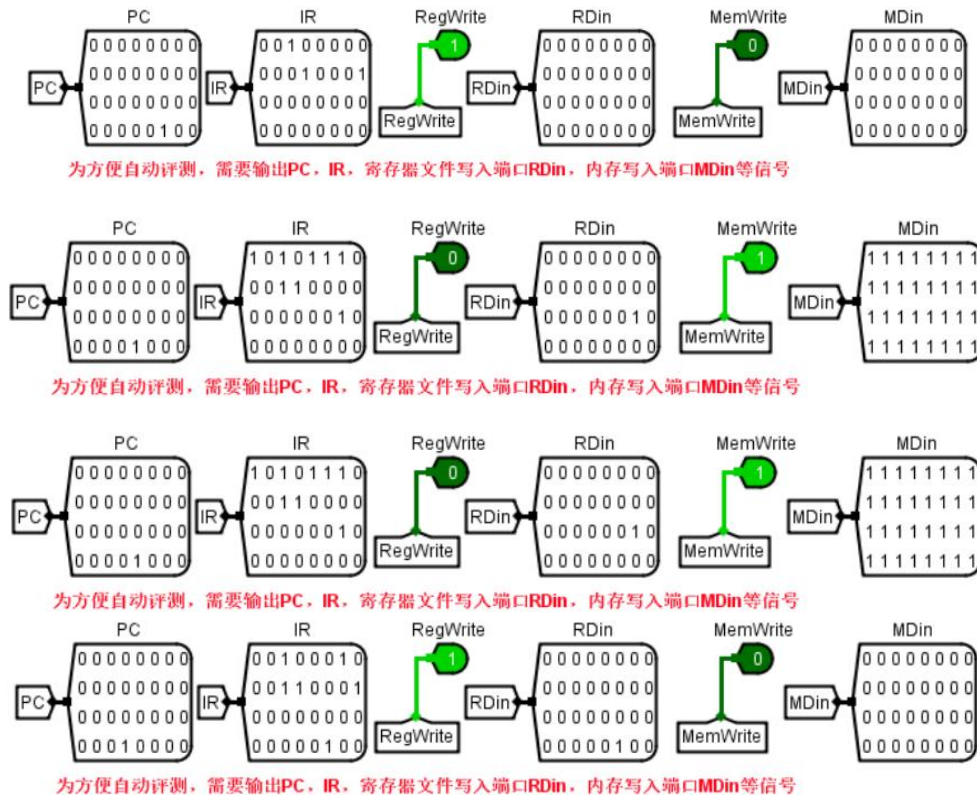


图 15 单周期 MIPS CPU 测试图(Logisim)

黄雅婷 33952

第8关：单周期MIPS CPU设计

任务要求 重置记录 评论 49

- 实验目的
- 实验内容
- 电路框架
- 电路引脚
- 电路测试
- 常见故障


- 调试技巧：如果未能通过测试，输出会停帧在出错的时钟节拍，可以尝试单步执行到对应节拍，对比标准输出和实际输出，查找原因！

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。

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。数据通路如下图所示：



说点什么

自己动手画CPU

实验总用时: 00:13:15

资源中心 数据集

代码文件

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project source="2.15.0.2.exe" version="1.0">
  This file is intended to be loaded by Logisim http://logisim.altervista.org
  <lib desc="#Wiring" name="0">
    <tool name="Splitter">
      <a name="facing" val="north"/>
      <a name="fanout" val="1"/>
      <a name="incoming" val="4"/>
      <a name="appear" val="left"/>
    </tool>
  </lib>
</project>
```

测试结果

1/1 全部通过

测试集1

消耗内存 95.19MB 代码执行时长: 1.48秒

预期输出					实际输出						
Cnt	PC	IR	RegW	RDin	MemW	Cnt	PC	IR	RegW	RDin	MemW
0000	00000000	2010ffff	1	ffffff	0	0000	00000000	2010ffff	1	ffffff	0
0001	00000004	20110000	1	00000000	0	0001	00000004	20110000	1	00000000	0
0002	00000008	ae300200	0	00000000	1	0002	00000008	ae300200	0	00000000	1
0003	0000000c	22100001	1	00000000	0	0003	0000000c	22100001	1	00000000	0
0004	00000010	22310004	1	00000004	0	0004	00000010	22310004	1	00000004	0
0005	00000014	ae300200	0	00000000	1	0005	00000014	ae300200	0	00000000	1
0006	00000018	22100001	1	00000001	0	0006	00000018	22100001	1	00000001	0
0007	0000001c	22310004	1	00000008	0	0007	0000001c	22310004	1	00000008	0
0008	00000020	ae300200	0	00000000	1	0008	00000020	ae300200	0	00000000	1
0009	00000024	22100001	1	00000002	0	0009	00000024	22100001	1	00000002	0
000a	00000028	22310004	1	0000000c	0	000a	00000028	22310004	1	0000000c	0
000b	0000002c	ae300200	0	00000000	1	000b	0000002c	ae300200	0	00000000	1
000c	00000030	22100001	1	00000003	0	000c	00000030	22100001	1	00000003	0
000d	00000034	22310004	1	00000010	0	000d	00000034	22310004	1	00000010	0
000e	00000038	ae300200	0	00000000	1	000e	00000038	ae300200	0	00000000	1
000f	0000003c	22100001	1	00000004	0	000f	0000003c	22100001	1	00000004	0

本次最大执行时间: 180秒 本次评测耗时(编译、运行总时间): 1.759 秒

上一步 下一步 自测运行 评测

图 16 单周期 MIPS CPU 测试图(头歌)

## (4) 故障与测试分析

### (4.1)

**故障现象 1:** 电路出现红线，表面无法看出异常。

**原因分析:** 一般有两种情况，一种是逻辑部件输入悬空，输出为错误状态，第二种是短路故障。由于所有逻辑部件都已经有了输入，因此判断为出现了短路。

**解决方案:** 利用手图标点击红线，观察红线连接点，找到短路并排除。

### (4.2)

**故障现象 2:** 多周期 CPU 运行 sort.hex 程序时，能正确得到相应的内存布局，但是遇到停机指令无法停机，PCEn 使能信号产生振荡，PC 循环变化；

**原因分析:** 指令译码错误，将 SysCall 指令也当成了 R 型指令，导致控制器中没有产生正确的 PCWrite 信号，也没有进入 SysCall 的死循环，从而使得 PCEn 信号出现震荡，系统无法停机；

**解决方案:** 区分 SysCall 指令和 R 型指令，使 SysCall 指令与 R 型指令互斥，如图下图所示。

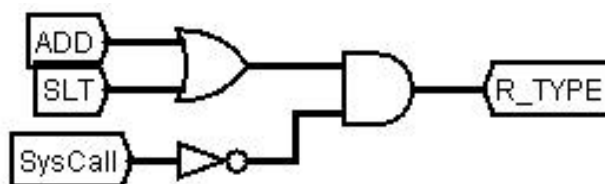


图 17 SysCall 与 R 型互斥

## 2.2 多周期 MIPS 微程序 CPU 设计

### (1) 设计思路

#### (1.1) 总体设计思想

多周期 MIPS CPU 的核心特征在于其采用微程序控制方式。在这种设计中，每条指令被分解为一系列更小的操作步骤，即微指令。每个时钟周期执行一个微指令，使得指令执行过程更加细化和灵活。微指令通常包含三个关键部分：操作控制字段、条件判断字段和下一地址字段。这些微指令被存储在专门的控制存储器中，执行时通过当前微指令的下一地址字段来确定下一条微指令。这

种设计允许功能单元在不同指令执行阶段被多次使用，提高了硬件利用率。

基于这种架构，多周期 MIPS CPU 的数据通路呈现出一些独特特点：指令和数据存储不再分离，而是共用同一存储器；时钟周期变短，数据传输路径相应缩短；功能单元的输出端增加了寄存器以保持中间结果。尽管指令长度可变，但整体处理流程与单周期 CPU 相似。在实际实现中，需要将各指令的控制信号编码为微指令并存入控制器。可以利用提供的微程序自动生成表来辅助这一过程。指令获取后，相应的微程序序列就会被执行。

设计过程包括编写微指令、构建地址转移逻辑电路、完成微程序控制器，最后设计并优化数据通路。这种设计方法的主要目标是增强 CPU 的灵活性和可扩展性。通过自定义微指令序列，可以根据不同指令的复杂度和需求，调整每个执行阶段的时间和操作顺序。此外，多周期设计还考虑了执行效率，通过将指令执行分为多个阶段，提高了硬件资源利用率和指令并发度。虽然这种设计相对复杂，但在处理复杂指令和实现高级功能时具有显著优势，为未来的扩展和优化提供了良好的基础。

## (1.2) 各种指令数据通路

### 1) 取指周期

取指令数据通路：

以原 PC 计数器的值作为地址从内存中读取指令，进行 PC 计数器累加： $Mem[PC] \rightarrow IR, PC+4 \rightarrow PC$ 。其中控制信号  $IRWrite=MemRead=1$ ，将 PC 寄存器的内容作为存储器地址访问对应存储空间，并从存储器中取出指令存入 IR 寄存器  $ALU\_OP=0, AluSrcB=0, AluSrcB=1, PCWrite=1$  实现将  $PC+4$  寄存到 PC 中。

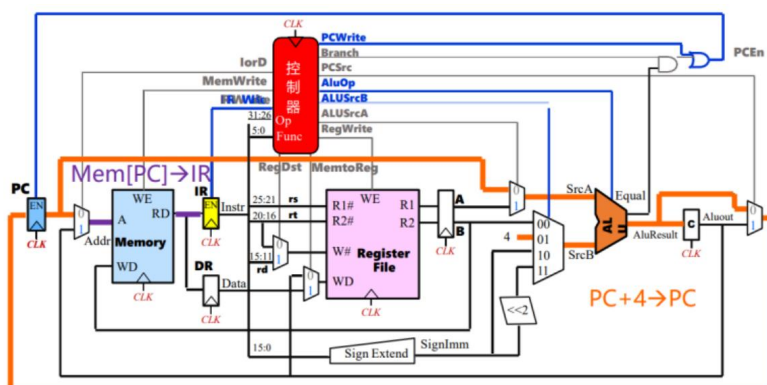


图 18 多周期 MIPS 取指令数据通路



# 华中科技大学课程实验报告

取操作数/译码数据通路：

取指令并译码，通过寄存器组读出存放在 A 和 B 寄存器，并将低十六位的立即数 Imm 扩展成 32 位并左移两位，再与 PC 计数器加字节数后的值相加写入 C 寄存器中： $Reg \rightarrow A, B, PC+4+Imm16 \ll 2 \rightarrow C$ 。控制信号  $AluSrcB=0$ ,  $AluSrcA=3$ ,  $ALU\_OP=0$ 。

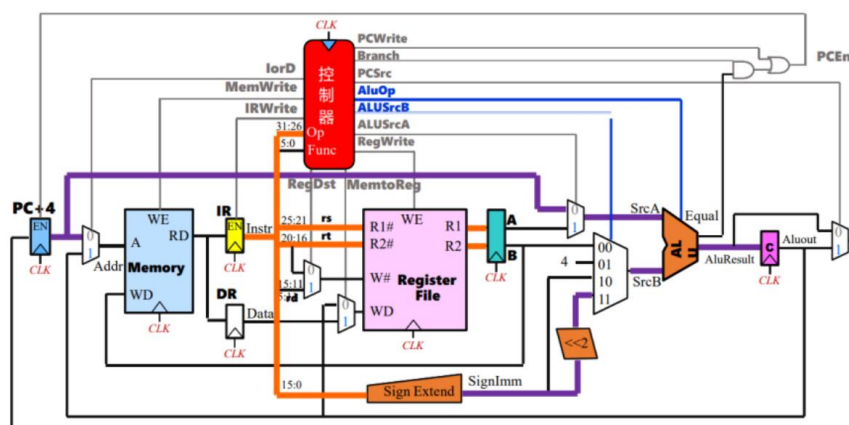


图 19 多周期 MIPS 译码数据通路

2) ADD、SLT 指令数据通路 ( $A \text{ op } B \rightarrow C$ ,  $C \rightarrow R[rd]$ )

将两个数经过运算后后存入一个寄存器中，分为运算 T3 和写回 T4 两个时钟周期。T3 周期中，从 MIPS 寄存器取指令导出到 A,B 寄存器相加，并将其存放在 C 寄存器中。控制信号  $AluSrcA=1$ 、 $AluSrcB=0$ ，ALU 的运算数 A 为取指令阶段赋值的  $R[rs]$ ，ALU 的运算数 B 为在取指令阶段赋值的  $R[rt]$ 。又  $AluControl=10$ ，ALU 采用的运算符由指令字的 Func 字段决定。

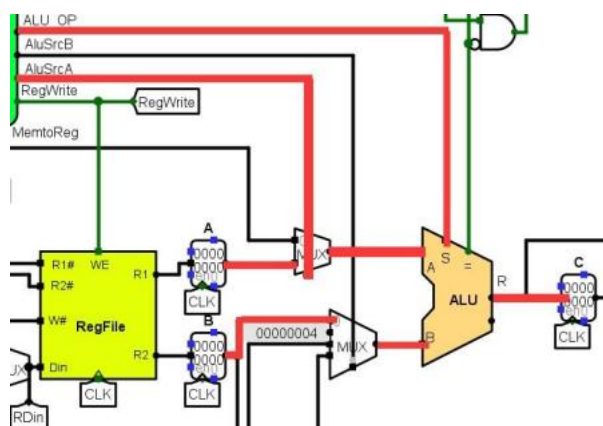


图 20 ADD、SLT 指令执行周期中 T3 操作的数据通路

T4 周期中，负责将运算结果写入适当的寄存器。控制信号  $RegDst$  被置为

# 华中科技大学课程实验报告

1, 指示系统使用指令中的 rd 字段作为目标寄存器的选择信号。同时, RegWrite 信号被激活, 表示即将进行寄存器写入操作。在这个阶段, 寄存器 C 中存储的运算结果被传送并写入到寄存器文件中由 rd 指定的位置。

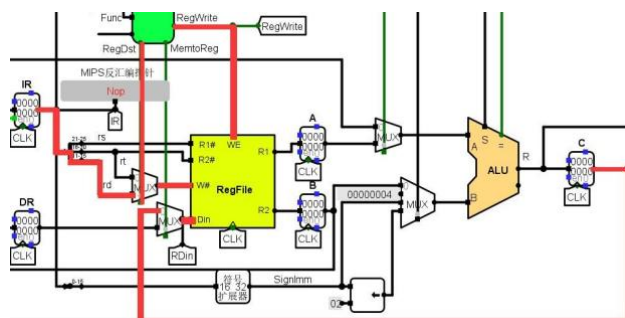


图 21 ADD、SLT 指令执行周期中 T4 操作的数据通路

## 3) ADDI 型指令数据通路 ( $A + \text{imm} \rightarrow C$ , $C \rightarrow R[\text{rt}]$ )

将寄存器中的值与立即数相加并将结果存回寄存器的操作需要两个执行周期: 运算阶段 (T3) 和写回阶段 (T4)。

在运算阶段 (T3) 中, 控制单元设置以下信号:  $\text{AluSrcA}=1$  和  $\text{AluSrcB}=10$ 。这些设置使 ALU 的第一个操作数选择了之前从寄存器文件读出的  $R[\text{rs}]$  值, 第二个操作数则是经过符号扩展的立即数。ALU 被配置为执行加法运算 ( $\text{AluControl}=10$ )。计算得到的结果被暂存在一个中间寄存器, 我们称之为寄存器 C。

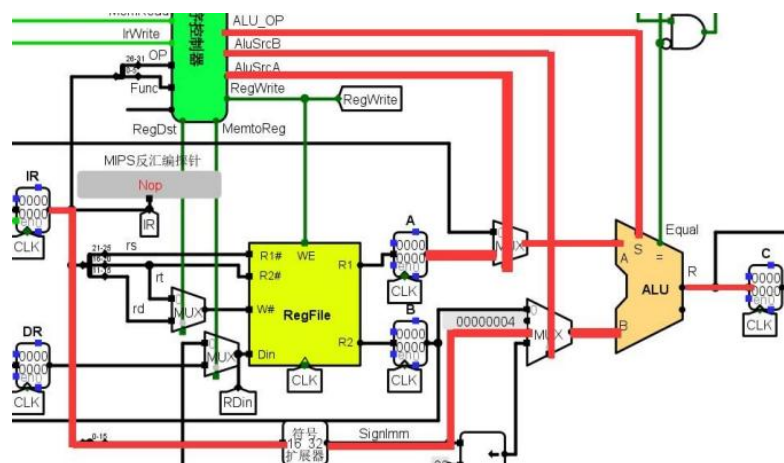


图 22 ADDI 指令执行周期中 T3 操作的数据通路

写回阶段 (T4) 主要负责将计算结果存入适当的寄存器。在这个阶段, 控制信号 RegWrite 被激活 (置为 1), 表示即将进行寄存器写入操作。同时,  $\text{RegDst}=0$  的设置指示系统使用指令中的 rt 字段作为目标寄存器的选择信号。此时,

# 华中科技大学课程实验报告

寄存器 C 中存储的运算结果被传送并写入到寄存器文件中由 rt 指定的寄存器位置。

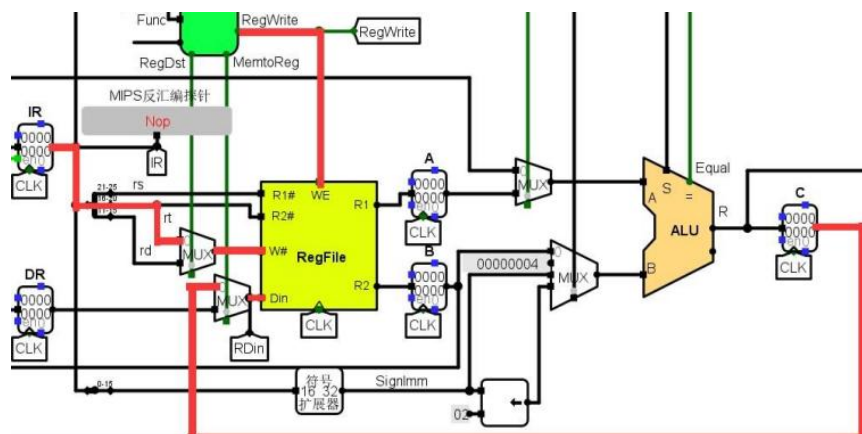


图 23 ADDI 指令执行周期中 T4 操作的数据通路

4) LW 指令的数据通路 ( $A + \text{SignExt}(\text{imm}) \rightarrow C$ ,  $M[C] \rightarrow \text{DR}$ ,  $\text{DR} \rightarrow R[\text{rt}]$ )

从主存中读取数据的过程可以分为三个时钟周期：地址计算 (T3)、内存访问 (T4) 和数据写回 (T5)。

在地址计算阶段 (T3)，控制单元激活以下信号：AluSrcA=1 和 AluSrcB=10。这使得 ALU 的输入 A 选择了先前从寄存器文件中读出的 R[rs]值，而输入 B 则是经过符号扩展的立即数。ALU 被设置为执行加法操作 (AluControl=10)。计算得到的地址被存储在一个临时寄存器 C 中。

内存访问阶段 (T4) 中，控制信号 IorD 被置为 1，同时 MemRead 信号被激活。这导致系统使用寄存器 C 中的值作为内存地址，并从该地址读取数据。读取的数据被暂存在系统的数据寄存器中。

在数据写回阶段 (T5)，控制单元设置 MemToReg=1 和 RegWrite=1。这个配置允许系统将从内存读取的数据写回到寄存器文件中。具体而言，指令中的 Rt 字段被用作目标寄存器的选择信号，而读取的数据则通过 AR 和 DR 这两个功能寄存器的配合，最终被写入到指定的寄存器中。

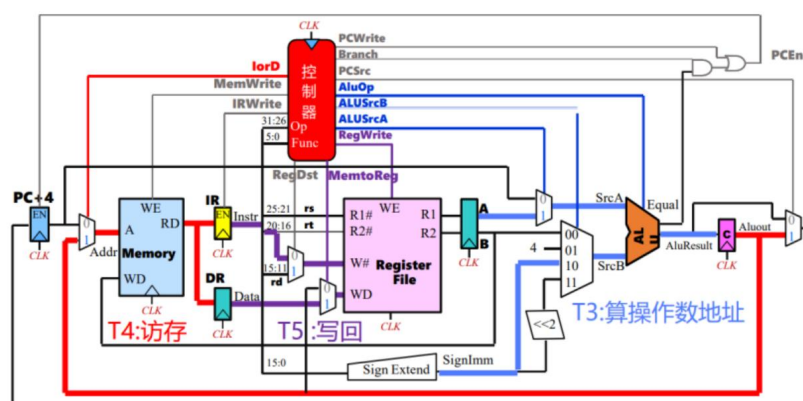


图 24 LW 指令执行周期操作的数据通路

## 5) SW 指令的数据通路 ( $A+imm \rightarrow C, B \rightarrow M[C]$ )

将数据写入主存的指定位置需要经过两个执行周期，我们将其标记为 T3 和 T4。在 T3 周期中，我们主要进行地址计算。控制单元设置以下信号：  
 $AluSrcA=1$ ，选择之前从寄存器文件读出的  $R[rs]$  值； $AluSrcB=10$ ，选择经过符号扩展的立即数。ALU 的操作由  $AluControl=10$  指定，执行加法运算。这个阶段的计算结果被暂存在一个中间寄存器，我们称之为寄存器 C。

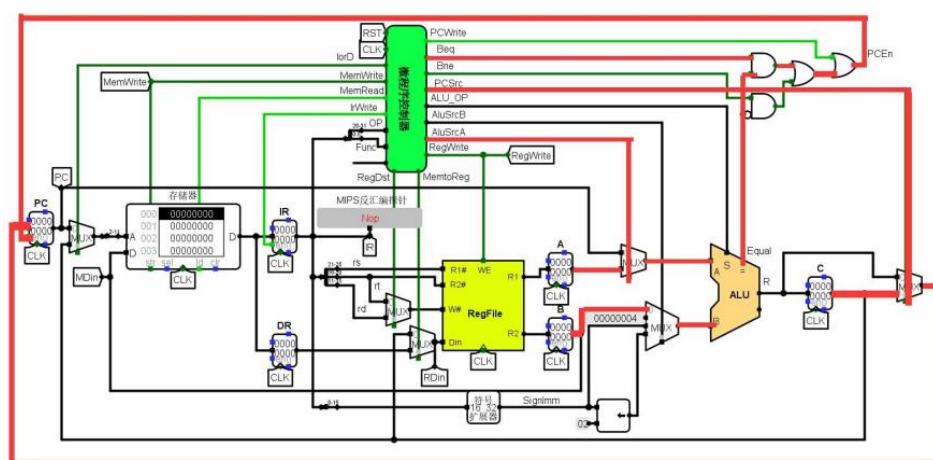


图 25 SW 指令执行周期中 T3 操作的数据通路

T4 周期负责实际的数据写入操作。此时，控制信号  $IorD$  被置为 1，这使得寄存器 C 中的值被用作存储器的地址输入。同时， $MemWrite$  信号被激活，表示即将进行写入操作。在取指阶段已经准备好的  $R[rt]$  寄存器的内容此时被传送到存储器的数据输入端口（通常称为 MDin）。最后，这些数据被写入到由寄存器 C 指定的内存地址中。

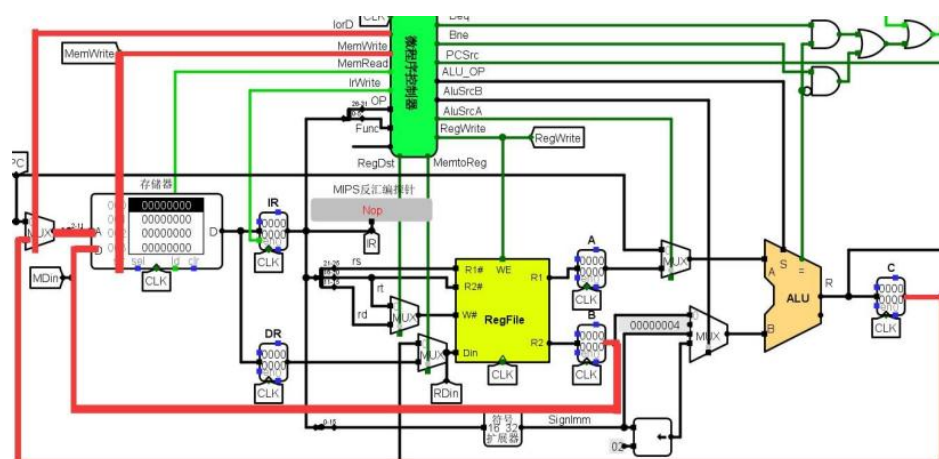


图 26 SW 指令执行周期中 T4 操作的数据通路

## 6) BEQ 指令的数据通路 (if (A=B) C→PC)

判断取指令时是否采用分支地址。控制信号  $AluSrcA=1$ 、 $AluSrcB=0$ ，将 A 和 B 寄存器中数据进行比较，得出结果与 PCWrite 和 Branch 信号相与后赋给 PC 计数器，并同时取指令中写入 C 寄存器的值作为分支地址写入 PC 寄存器。

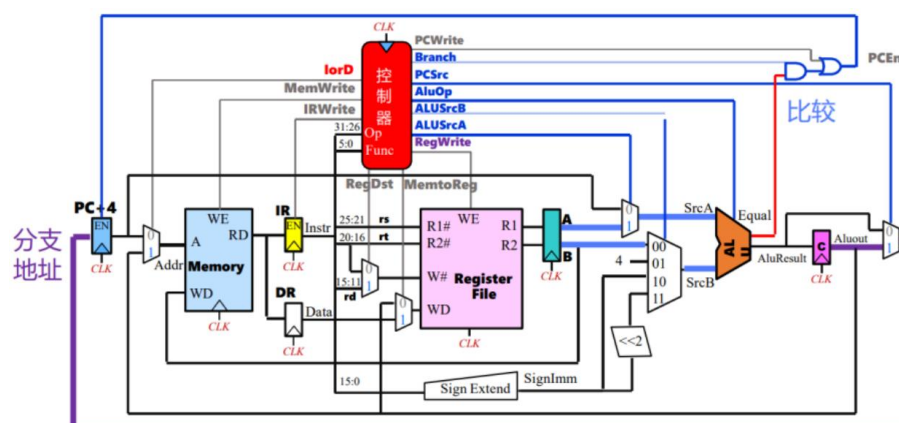


图 27 BEQ 指令执行周期中 T3 操作的数据通路

BNE 指令的数据通路 (if (A!=B) C→PC)，实现方式同 BEQ 可得。

## 7) SysCall 指令的数据通路

控制信号全部置 0，从而实现停机，系统停止工作。

### (1.3) 多周期 MIPS 微程序控制器设计过程

在我们的微程序设计中，共有八条微指令，涵盖了 13 个执行阶段。指令执行始于取指令阶段 (T1)。在这个阶段，我们观察到几个关键的控制信号：IRWrite 被置为 1，表示当前指令被写入指令寄存器，PCWrite 同样为 1，意味



着程序计数器进行自增操作（PC+4），MemRead 信号激活，指示从内存读取数据，AluSrcB 设为 01，选择立即数 4 作为 ALU 的 B 输入，P 被设为 0，指示下一阶段进入译码。

译码阶段的微地址被设定为 0001。值得注意的是，在取指令的 T2 阶段，P 被设为 1，这意味着下一条微指令的选择将由微程序入口逻辑决定。对于 R 型指令，执行分为 T3 和 T4 两个阶段：在 R1 阶段（T3），AluControl 被设为 10，具体的运算方式由 Func 字段决定。接下来的 R2 阶段（T4）微地址为 1000。在这个阶段，我们看到 RegDst 为 1，选择 RD 作为写回目标，同时 RegWrite 为 1，激活寄存器写入操作。执行完 R 型指令后，控制流程返回到取指令阶段，对应的微地址为 0000。对于 Load Word (LW) 和 Branch if Equal (BEQ) 指令，它们的执行阶段（T3 到 T5）遵循类似的模式，但具体的控制信号设置会有所不同，以适应各自的操作需求。

在实现过程中，我们将这些微指令的控制字转换为十六进制格式，并存储在控制存储器中，选择相应微程序的入口地址由微程序地址转移逻辑自动生成表生成，如下所示。

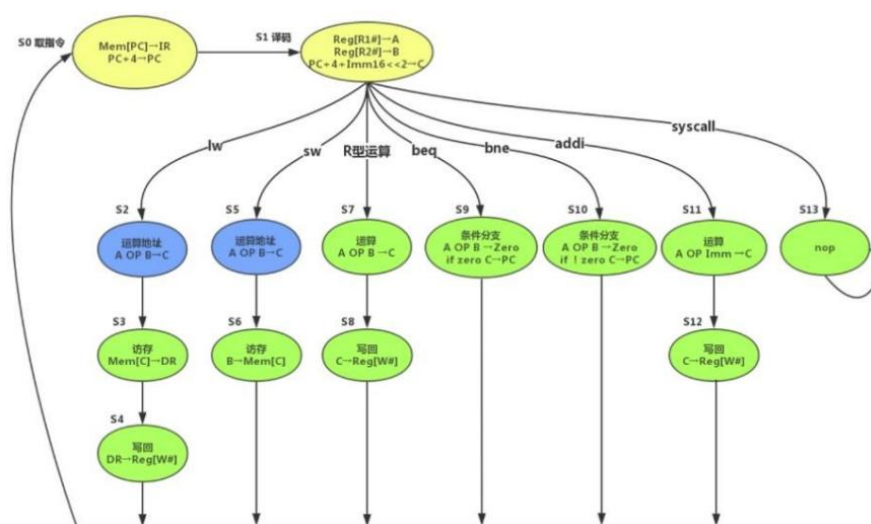


图 28 多周期 MIPS CPU 指令状态图

# 华中科技大学课程实验报告

表 2 多周期 MIPS 微程序 CPU 的微指令二进制序列

微指令功能	状态	微指令地址	lorD	PcSrc	AluSrcA	AluSrcB	MemToReg	RegDst	IrWrite	PcWrite	RegWrite	MemWrite	MemRead	BEQ	BNE	AluControl	P	下址字段	微指令	十六进制
取指令	0	0000	0	0	0	01	0	0	1	1	0	0	1	0	0	00	0	0001	0000100110010000000001	13201
译码	1	0001	0	0	0	11	0	0	0	0	0	0	0	0	0	00	1	0000	0001100000000000010000	30010
LW1	2	0010	00	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0011	0001100000000000000011	60003
LW2	3	0011	1	0	0	00	0	0	0	0	0	0	1	0	0	00	0	0100	100000000001000000100	100204
LW3	4	0100	0	0	0	00	1	0	0	0	1	0	0	0	0	00	0	0000	0000010001000000000000	8800
SW1	5	0101	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0110	001100000000000000110	60006
SW2	6	0110	1	0	0	00	0	0	0	0	0	1	0	0	0	00	0	0000	100000000010000000000	100400
R1	7	0111	0	0	1	00	0	0	0	0	0	0	0	0	0	10	0	1000	001000000000001001000	40048
R2	8	1000	0	0	0	00	0	1	0	0	1	0	0	0	0	00	0	0000	000000100100000000000	4800
BEQ	9	1001	0	1	1	00	0	0	0	0	0	0	0	1	0	00	0	0000	011000000000100000000	C0100
BNE	10	1010	0	1	1	00	0	0	0	0	0	0	0	0	1	00	0	0000	011000000000100000000	C0080
ADDI1	11	1011	0	0	1	10	0	0	0	0	0	0	0	0	0	10	0	1100	001100000000001001100	6004C
ADDI2	12	1100	0	0	0	00	0	0	0	1	0	0	0	0	0	00	0	0000	000000000100000000000	800
SYSCALL	13	1101	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	1101	00000000000000001101	D

根据图填写微程序地址转移逻辑表，如表 3，得到表达式后自动生成微程序地址转移逻辑电路，如图 29。

表 3 多周期 MIPS 指令的微程序入口地址

机器指令译码信号							微程序入口地址				
R_Type	ADDI	LW	SW	BEQ	BNE	SYSCALL	入口地址 10进制	S3	S2	S1	S0
1							7	0	1	1	1
	1						11	1	0	1	1
		1					2	0	0	1	0
			1				5	0	1	0	1
				1			9	1	0	0	1
					1		10	1	0	1	0
1						1	13	1	1	0	1

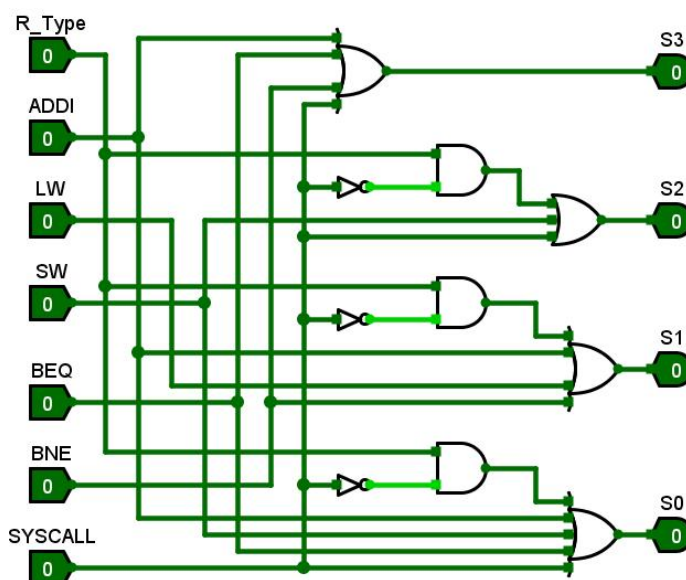


图 29 微程序地址转移逻辑电路

## (2) 电路图

### (2.1) 微程序控制器电路图

---



# 华中科技大学课程实验报告

## (3) 测试图

如下图所示分别是 Logisim 和头歌的测试图，均通过测试样例表示无错误输出，说明电路输出与实际情况相符，证明了电路设计正确。

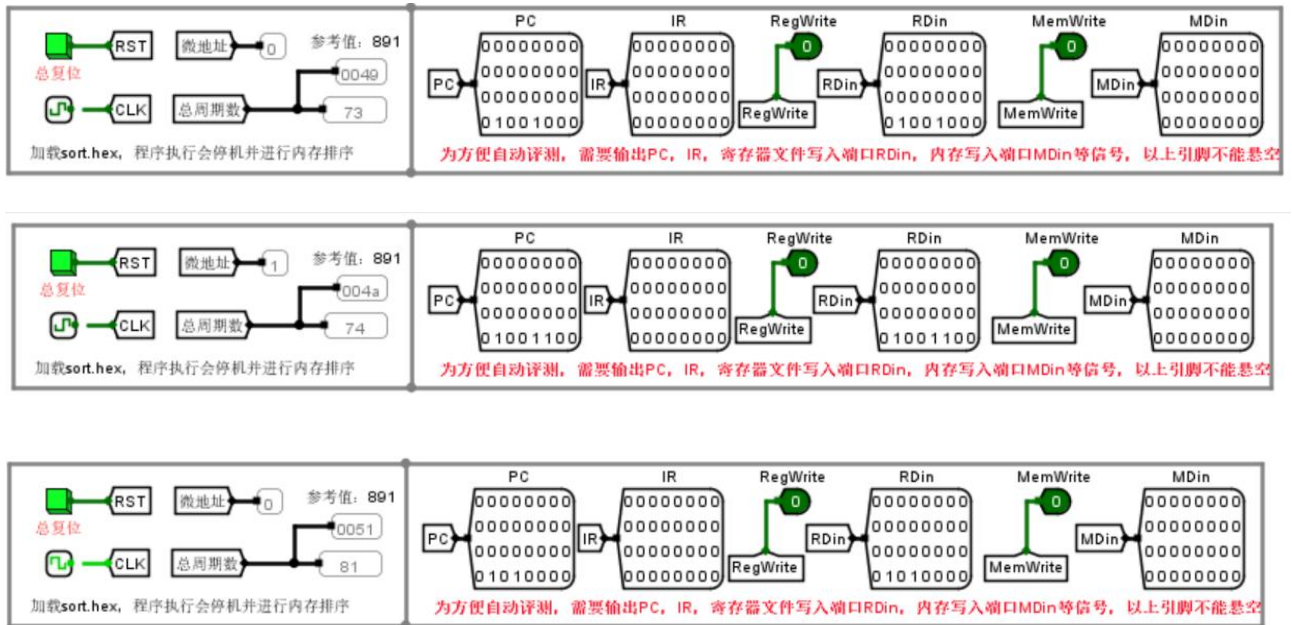


图 32 多周期 MIPS CPU 测试图(Logisim)

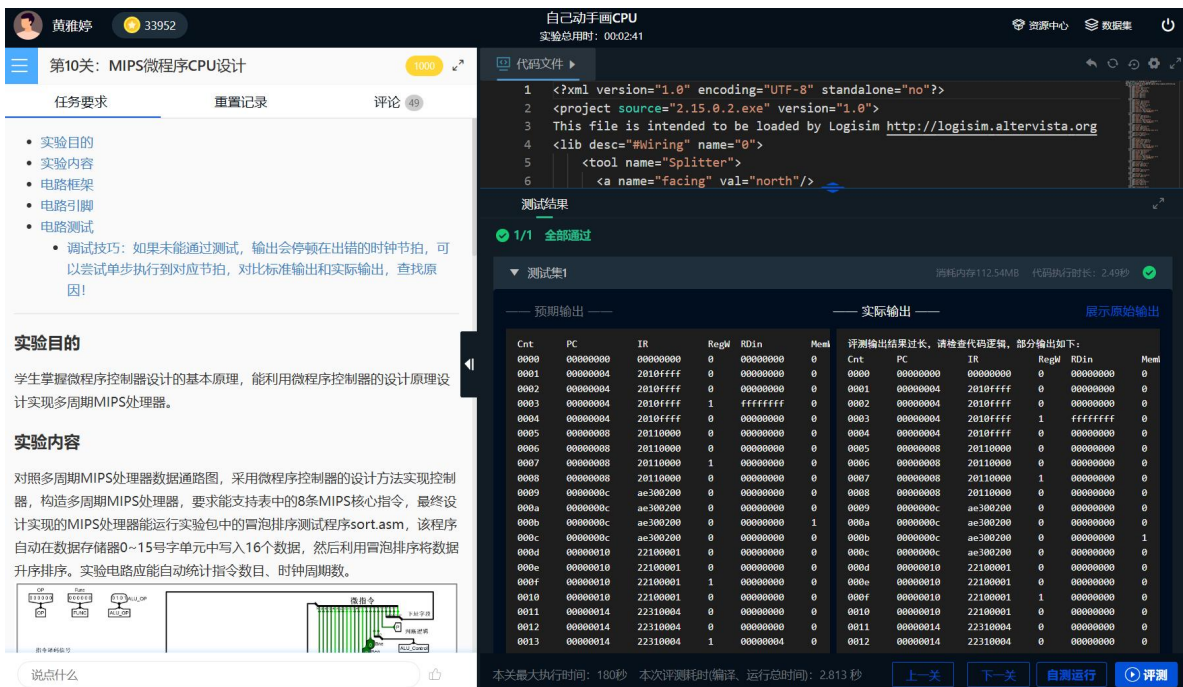


图 33 多周期 MIPS CPU 测试图(头歌)

## (4) 故障与测试分析

(4.1)

**故障现象 1:** 程序执行时每执行完一条命令后跳过 3 条命令执行。

**原因分析:** 本次实验中存储器中都是按字寻址而不是按字节。由于指令存储器使用的是 32 位数据位宽，需要使用 PC 寄存器的 0~9 比特位作为指令存储器的地址输入端。根据 32 位 MIPS CPU 的设计，每条指令占据 4 字节，因此每次 PC+4 实际上增加了 4 个 4 字节。

**解决方案:** 将 PC 寄存器的 2~11 比特位作为指令存储器的地址输入端。通过这样的改变，每次 PC+4 对应的指令存储器地址将增加 1，即相当于增加了 4 字节的偏移量。同样的方法也可以应用于数据存储器，使用 2~11 比特作为地址输入端。同时立即数扩展成 32 位后也不用在左移两位，修改如下图所示。

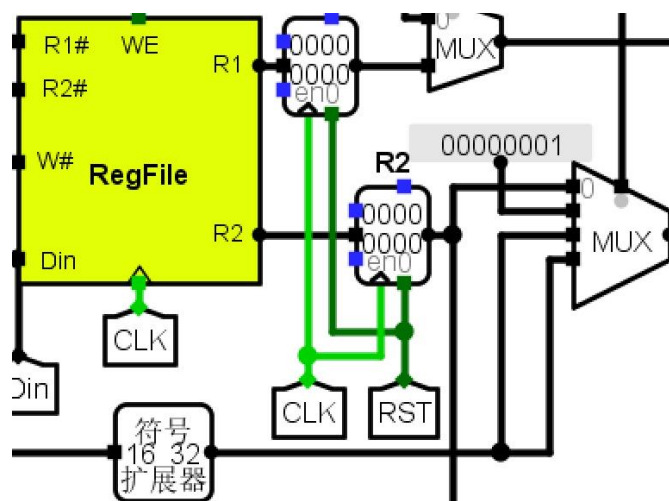


图 34 多周期 MIPS ALU 输入数据通路设计图

## 3 总结与心得

### 3.1 实验总结

本门课实验主要完成了如下几点工作：

#### 1) 完成方案总结

本次实验学习围绕 MIPS 架构展开，涉及了从基础组件到完整处理器的设计与实现。实验内容包括：算术逻辑单元（ALU）的自主设计，随机存取存储器（RAM）的构建，单周期 MIPS CPU 的实现，采用硬布线控制方式，多周期 MIPS CPU 的开发，基于微程序控制，以及相应控制器的设计与优化等。这些实验项目循序渐进，逐步深入，使我们能够全面掌握计算机体系结构的核心知识。

#### 2) 实现的功能总结

通过一系列实验，我们成功完成了以下关键功能：设计并实现了功能完备的 ALU；构建了能够有效存储和访问数据的 RAM 系统；开发了两种不同类型的 MIPS CPU，分别是采用硬布线控制的单周期处理器和基于微程序；这两种 CPU 均能执行一组核心 MIPS 指令，包括 ADD、SLT、LW、SW、BEQ、BNE、ADDI 和 SysCall。

#### 3) 其他需要总结的内容

参与本次实验使我获益良多，不仅加深了对计算机硬件架构的理解，同时将理论知识转化为实际应用。作为网络安全专业的学生，这些实验帮助我建立了坚实的计算机基础，这对未来的安全工作至关重要，提升了解决问题的能力和动手实践的技能，并且认识到持续学习和积累的重要性，这是成为优秀网络安全人才的必经之路。这些实验不仅巩固了课堂所学，还培养了我的实践能力和创新思维，为今后的专业发展奠定了良好基础。

### 3.2 实验心得

通过本学期的实验课程，我对计算机组成原理有了更深入的认识和实践体

# 华中科技大学课程实验报告

---

验。这些实验不仅巩固了课堂所学，还大大拓展了我的视野和技能。实验内容的深度与广度给我留下了深刻印象。从基础的运算器和存储器设计，到复杂的单周期和多周期 MIPS CPU 实现，每一个环节都充满挑战和学习机会。通过亲自动手搭建这些系统，我对 CPU 的结构、功能以及指令处理流程有了更加清晰和直观的理解。

在实验过程中，我遇到了不少困难和疑惑。例如，理解寄存器间的数据传输、处理符号扩展和位扩展等问题初期让我感到困惑。然而，正是这些挑战促使我回顾基础知识，查阅额外资料，最终达到了更深层次的理解。这种"困惑-探索-领悟"的过程不仅增强了我的问题解决能力，也培养了我的耐心和毅力。使用 Logisim 软件进行实验设计是一次宝贵的经历。通过反复调试和优化电路，我不仅提高了软件使用技能，还锻炼了逻辑思维和调试能力。找出短路、解决时序问题等实践经验对我今后的学习和工作都将大有裨益。

这些实验让我深刻认识到计算机科学的严谨性和复杂性。每一个细节都至关重要，一个微小的错误可能导致整个系统失效。这种认识不仅适用于 CPU 设计，也适用于其他工程领域，让我对专业学习有了更加严谨的态度。

实验的难度安排循序渐进，这种设计很好地平衡了挑战性和可行性。对于较为复杂的实验，我建议可以增加一些预备性的讲解和指导。例如，在实验前进行原理回顾和内容分析，不仅可以帮助学生更好地理解实验目标，还能提高实验效率。此外，适当增加一些拓展性内容，如探讨不同 CPU 架构的优缺点，或介绍当前 CPU 设计的前沿技术，都将大大增加学生的收获。

总的来说，这次实验课程不仅加深了我对计算机硬件的理解，还培养了我的实践能力和创新思维。它让我意识到理论知识和实际应用之间的紧密联系，激发了我对计算机科学更深入学习的兴趣。这种结合理论与实践的学习方式，无疑为我今后的学术和职业发展奠定了坚实的基础。

---

## 原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字: