

**华中科技大学**

人工智能与自动化学院

**计算机组成原理**

**实验报告**

**专业班级： 人工智能2304班**

**学 号： U202315265**

**姓 名： 杜辰宇**

**实验名称： CPU 实 验**

**实验时间： 2025年5月11日**

**目录**

[1、 实验目的 4](#_Toc197947035)

[理解多周期MIPS CPU的基本原理,并进行设计 4](#_Toc197947036)

[（1）设计实现微程序控制器实现的多周期MIPS处理器 4](#_Toc197947037)

[（2）设计实现硬布线控制器实现的多周期MIPS处理器 4](#_Toc197947038)

[（3）加载sort.hex并进行测试 4](#_Toc197947039)

[2、设计思路 5](#_Toc197947040)

[2.1、设计要求 5](#_Toc197947041)

[2.2、设计多周期MIPS CPU数据通路 5](#_Toc197947042)

[2.3、设计多周期MIPS CPU微程序控制器 7](#_Toc197947043)

[2.4、设计多周期MIPS CPU硬布线控制器 10](#_Toc197947044)

[3、实验步骤 13](#_Toc197947045)

[3.1、构建多周期CPU数据通路 13](#_Toc197947046)

[3.2、构建多周期CPU微程序控制器 13](#_Toc197947047)

[3.3 构建多周期 CPU 硬布线控制器 16](#_Toc197947048)

[四、实验结果 20](#_Toc197947049)

[5、实验中的bug和解决方案 21](#_Toc197947050)

[5.1 问题1：仿真软件异常 21](#_Toc197947051)

[5.2 问题2：周期数异常，程序无法停止 21](#_Toc197947052)

[5.3 问题3：状态机逻辑错误 21](#_Toc197947053)

1. 实验目的

理解多周期MIPS CPU的基本原理,并进行设计

（1）设计实现微程序控制器实现的多周期MIPS处理器

• 掌握微程序控制器设计的基本原理，能利用微程序控制器的设计原理设计实现多周期MIPS处理器。

（2）设计实现硬布线控制器实现的多周期MIPS处理器

• 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。

（3）加载sort.hex并进行测试

2、设计思路

2.1、设计要求

在Logisim平台上，利用提供的组件构建一个支持32位MIPS多周期CPU。该CPU需支持图1.1中列出的8条核心指令。任务包括：

1. 绘制多周期MIPS CPU的数据通路。

2. 以两种方式实现控制器：微程序控制器和硬布线控制器。

3. 在完成的CPU上运行冒泡排序测试程序sort.hex，并确保能够获得正确的输出结果。

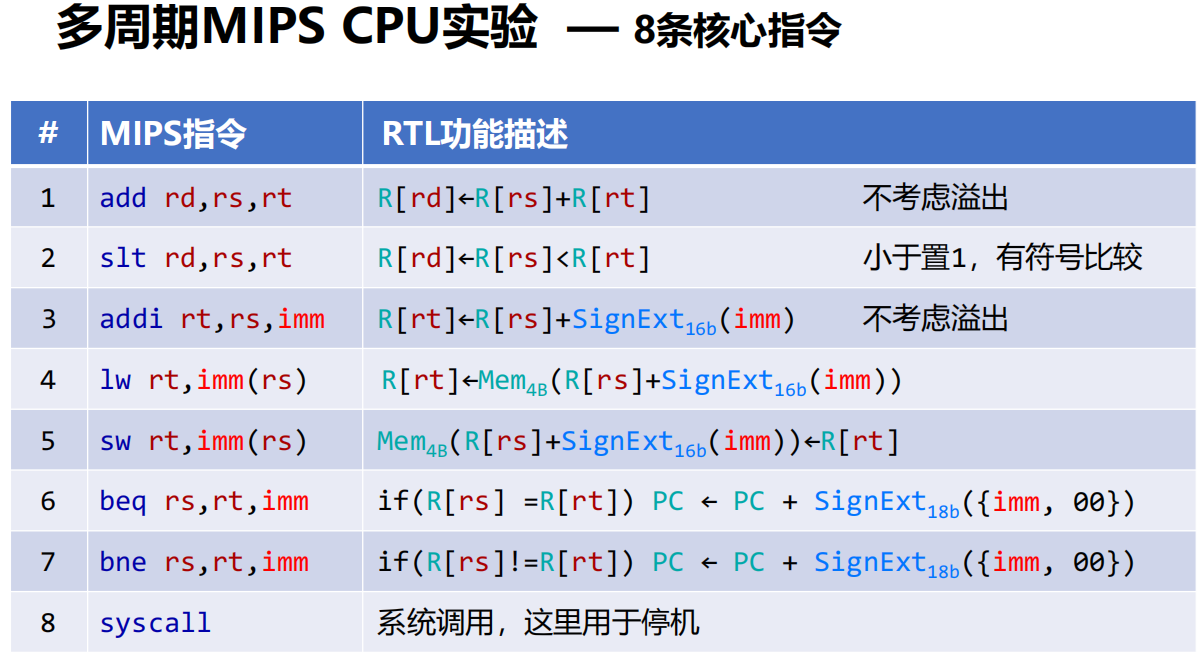


图1.1 8条核心指令

2.2、设计多周期MIPS CPU数据通路

在多周期MIPS CPU设计中，与单周期CPU相比，其核心思想是将指令执行过程细分为多个阶段，通过分时复用各功能部件来实现指令的执行。这种设计方式不仅提高了资源利用率，还降低了硬件成本。以下是多周期MIPS CPU数据通路的设计思路：

1. 功能部件复用与存储器整合

* **存储器整合**：不再区分指令存储器与数据存储器，指令和数据统一保存在同一个存储器中。取指令与存数据操作通过分时复用的方式实现，即在不同的时钟周期内分别进行指令读取和数据存储操作。
* **功能部件复用**：部分功能部件（如存储器、ALU等）在一条指令执行过程的不同时钟周期内可以多次分时使用。例如，ALU在执行阶段用于完成算术或逻辑运算，而在指令地址计算时用于生成目标地址。

2. 寄存器引入与时序控制

* **寄存器锁存数据**：为实现时序控制，主要功能部件的输出端都增加了寄存器，用于锁存数据。这些寄存器确保在后续时钟周期中需要使用的数据能够被正确保存和传递。具体增加的寄存器包括：
* **数据寄存器（DR）**：用于存放从存储器读取的数据。
* **指令寄存器（IR）**：用于存放从存储器取出的指令。
* **三个中间寄存器（A、B、C）**：分别用于保存寄存器文件（RegiFile）和ALU的输出。
* **时序控制**：通过控制信号精确控制寄存器的写入操作，确保数据在不同阶段的正确传递和处理。

3. 指令计数器（PC）的改进

* **PC控制方式**：由于不同指令的时钟周期数不同，PC不再仅由时钟信号控制，而是增加了专门的写操作控制信号。这使得PC可以根据指令的执行情况动态更新，以支持多周期指令的执行。
* **地址计算**：取消了单周期CPU中独立的加法器，PC+4的操作以及地址转移指令中PC地址加上立即数的操作均由ALU完成。为此，ALU的B输入端增加了两个输入，共4个输入，并采用4路选择器选择输出。同时，A端也增加了一个PC值的输入，通过2路选择器选择输出。

4. 存储器地址输入的改进

* **存储器地址输入**：由于存储器的复用，除了原有的PC值作为地址输入外，还需要将经过ALU计算得到的地址送入存储器地址输入端。这使得存储器能够在不同的时钟周期内接收不同的地址信号，支持指令和数据的分时访问。

5. 控制信号的引入

* **控制信号设计**：随着新寄存器的引入和功能部件的复用，需要设计相应的控制信号来协调各部件的操作。这些控制信号根据当前指令的执行阶段和类型动态生成，确保数据通路的正确运行。

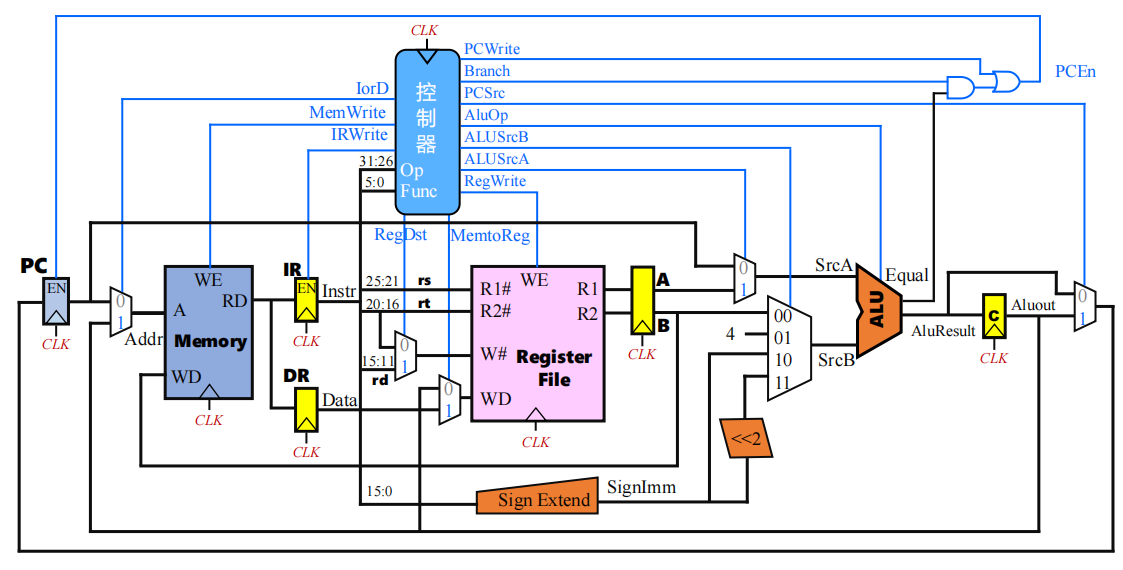
6. 停机功能的实现

* **停机机制**：多周期CPU的停机功能不再依赖单独的halt信号，而是通过状态变迁来实现。当CPU进入特定的停机状态后，会一直循环在该状态，直到外部干预。

7. 数据通路设计步骤

1. **指令分析**：逐条分析每条指令的执行过程，明确各阶段的功能部件使用情况和数据流动路径。
2. **单条指令数据通路设计**：根据指令的执行过程，绘制单条指令的数据通路，确保各阶段的数据流动和控制信号的生成。
3. **综合数据通路设计**：将所有指令的数据通路进行综合，形成完整的多周期CPU数据通路。在设计过程中，确保各功能部件的复用和时序控制的正确性，避免数据冲突和时序错误。

最终，根据上述设计思路，绘制出多周期MIPS CPU数据通路示意图，如图1.2所示。



1.2 多周期CPU数据通路示意图

2.3、设计多周期MIPS CPU微程序控制器

在多周期MIPS CPU中，每条指令的执行过程被细分为多个阶段，主要包括“取指->译码->执行”三个阶段。由于不同指令的执行过程和所需时间不同，每条指令的执行阶段可能包含多个时钟周期。微程序控制器通过将每条指令分解为一系列微指令来实现对CPU的控制，每个微指令对应一个时钟周期，并由一组控制信号驱动完成特定的微操作。以下是微程序控制器的设计思路：

1. 指令执行阶段划分

* **取指和译码阶段**：所有指令的取指和译码阶段具有相同的数据通路和控制信号。取指阶段从存储器中读取指令并更新PC值，译码阶段则解析指令并获取操作数。
* **执行阶段**：执行阶段根据指令类型不同，可能包含多个时钟周期。每条指令的执行过程被分解为一系列微操作，每个微操作对应一条微指令。

2. 微指令设计

* **微指令定义**：每条微指令对应一个时钟周期，包含一组控制信号，用于控制数据通路中的数据流动和功能部件的操作。例如，IR <- (Mem[PC]) 表示从存储器读取指令到指令寄存器。
* **微操作分解**：根据指令的执行过程，将每条指令的执行阶段分解为多个微操作。每个微操作的执行需要一组控制信号，这些控制信号在每个时钟周期内并发产生，以完成对应的微操作。

3. 指令状态变换图

* **状态划分**：根据指令的执行阶段和微操作，将每条指令的执行过程划分为多个状态，每个状态对应一个时钟周期。状态的转换由指令的执行流程和控制信号决定。
* **状态图构建**：构建指令状态变换图，明确每个状态的输入条件和输出控制信号。状态图中的每个状态对应一条微指令，状态之间的转换由指令的执行逻辑控制。

4. 控制信号生成

* **控制信号分析**：根据每条指令的执行流程和微操作，分析每个状态所需的控制信号。控制信号用于驱动数据通路中的寄存器、存储器、ALU等部件完成相应的操作。
* **微指令生成**：根据状态图和控制信号分析，生成每条微指令的控制信号集合。微指令地址即为状态编号，通过状态编号可以索引到对应的微指令和控制信号。

5. 微程序控制器实现

* **微程序存储器**：设计一个微程序存储器，用于存储所有微指令及其对应的控制信号。微程序存储器的地址由当前状态编号决定，输出为当前状态的控制信号。
* **状态控制逻辑**：设计状态控制逻辑，根据当前状态和指令执行逻辑，生成下一个状态的编号。状态控制逻辑通过状态变换图实现，确保指令的执行过程按照预定的顺序进行。
* **控制信号输出**：微程序存储器根据当前状态编号输出对应的控制信号，这些控制信号驱动CPU的数据通路完成相应的微操作。

6. 指令执行流程表

根据上述设计思路，首先给出取指、译码阶段以及每条指令的执行流程表，如下表所示：

表1.1 取指、译码阶段操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 取指令 | IR <- (Mem [PC])  PC <- (PC)+4 |
| 译码及取操作数 | A <- (R[IR[25:21]])  B <- (R[IR[20:16]])  C <- (PC) + (S-EXT(IR[15:0])<<2) |

表1.2 add指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 加运算 | C <- (A) + (B) |
| 写回 | R[IR[15:11]] <- (C) |

表1.3 slt指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 比较运算 | C <- ((A)<(B)) |
| 写回 | R[IR[15:11]] <- (C) |

表1.4 lw指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 计算地址 | C <- (A) + S-EXT(IR[15:0]) |
| 访存 | DR <- (Mem[PC]) |
| 写回 | R[IR[20:16]] <- (DR) |

表1.5 sw指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 计算地址 | C <- (A) + S-EXT(IR[15:0]) |
| 访存 | DR <- (Mem[PC]) |

表1.6 beq指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 送目标地址 | if(A==B) PC <- (C) |

表1.7 bne指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 送目标地址 | if(A!=B) PC <- (C) |

表1.8 addi指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 加运算 | C <- (A) + S-EXT(IR[15:0]) |
| 写回 | R[IR[20:16]] <- (C) |

表1.9 syscall指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 空操作，停机 | 锁住PC |

7. 指令状态变换图与微指令设计

根据上述操作流程表，对每条指令划分执行阶段，每个阶段对应一个时钟周期（一个状态），每个时钟周期对应一条微指令。构建指令状态变换图(图1.3)，明确每个状态的控制信号和状态转换条件。微指令及对应控制信号表（图1.4）所示。

通过上述设计思路，可以实现一个多周期MIPS CPU的微程序控制器，确保每条指令能够按照预定的流程正确执行。

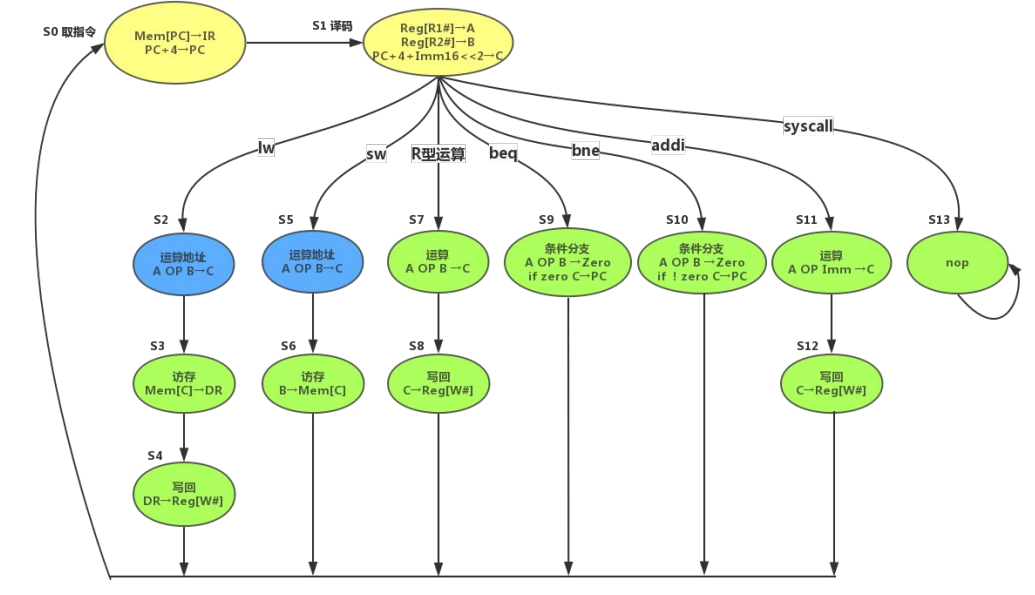


图1.3 指令周期状态转移图

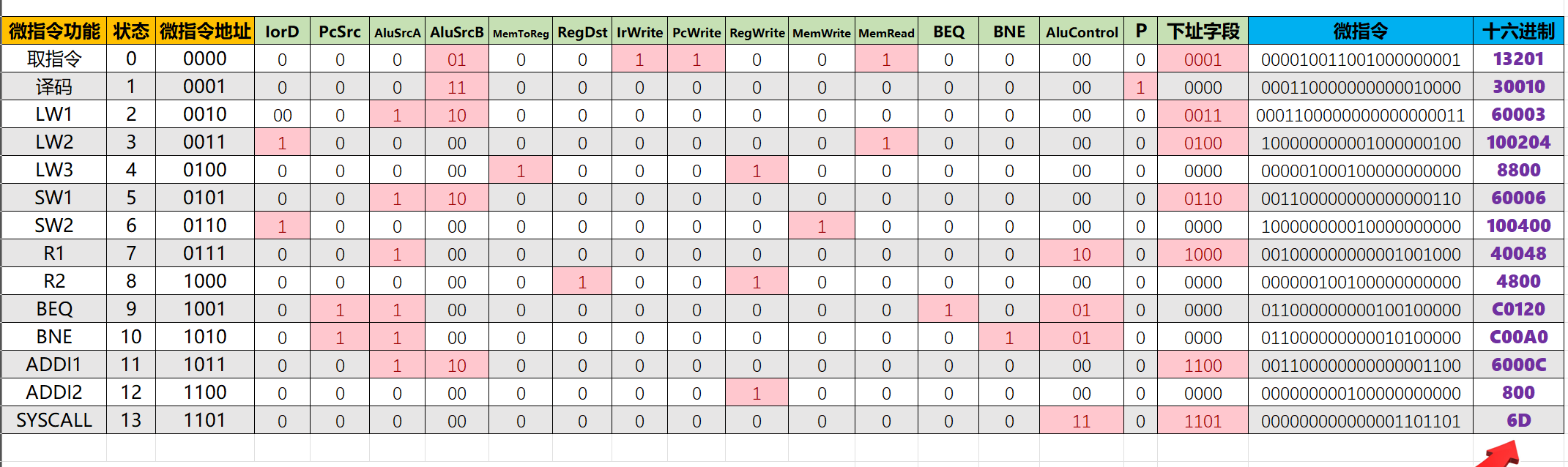


图1.4 微程序控制器微指令

2.4、设计多周期MIPS CPU硬布线控制器

设计多周期MIPS CPU的硬布线控制器主要基于有限状态机（FSM）来实现，其输入为当前状态和指令译码信号，输出为下一个状态（次态），而控制信号仅与当前状态有关，属于Moore型电路。以下是具体的设计思路：

1. 指令周期流程分析

* **指令阶段划分**：分析每条指令的执行过程，包括取指、译码和执行三个主要阶段。每个阶段进一步细分为多个时钟周期，每个时钟周期对应一个状态。
* **控制信号确定**：根据每个状态的操作流程，确定所需的控制信号。这些控制信号用于驱动数据通路中的寄存器、存储器、ALU等部件完成相应的微操作。

2. 指令状态变换图构建

* **状态划分**：根据指令的执行流程，将所有指令的执行过程划分为多个状态。例如，对于取指阶段，对应状态为S0；译码阶段对应状态为S1；执行阶段根据指令类型不同，可能包含多个状态如S2、S3等。
* **状态转换逻辑**：构建指令状态变换图，明确每个状态的输入条件和输出状态。状态转换由指令的执行逻辑和当前状态决定，例如，当取指操作完成后，状态从S0转换到S1。
* **状态编号**：根据状态变换图，对所有状态进行编号，方便后续的状态寄存器设计和状态机实现。假设共13个状态，则可使用4位二进制数进行编号。

3. 状态寄存器设计

* **状态寄存器功能**：设计一个4位状态寄存器，用于记录当前状态的编号。状态寄存器的输出为有限状态机的现态，输入为有限状态机的次态。
* **状态更新机制**：在每个时钟信号到来时，状态寄存器的现态更新为上一个次态。状态寄存器的更新由时钟信号触发，确保状态的正确转换。

4. 有限状态机设计

* **输入信号**：状态机的输入包括指令译码信号、反馈信号和当前状态。指令译码信号用于指示当前指令的类型，反馈信号用于处理指令执行过程中的特殊情况（如分支指令的条件判断）。
* **输出信号**：状态机的输出为下一个状态（次态）。状态机的输出仅与输入信号和当前状态有关，属于纯组合逻辑。
* **状态转换逻辑实现**：利用真值表或状态转换方程，设计状态机的组合逻辑电路。例如，当输入为当前状态S0和取指指令译码信号时，输出为次态S1。根据输入信号和当前状态，生成对应的次态输出。
* **状态机实现**：状态机的组合逻辑可以通过逻辑门电路或查找表（LUT）实现，确保状态转换的正确性和高效性。

5. 控制信号生成

* **控制信号逻辑**：根据状态机的现态，生成相应的控制信号。每个状态对应一组特定的控制信号，用于驱动数据通路完成当前状态的操作。例如，当状态为S0（取指阶段）时，控制信号包括从存储器读取指令到指令寄存器（IR）和更新程序计数器（PC）。
* **组合逻辑设计**：控制信号的生成属于组合逻辑，可以根据状态编号和指令译码信号，通过逻辑门电路或查找表实现。控制信号的输出仅与当前状态有关，符合Moore型电路的特点。

6. 硬布线控制器实现

* **状态寄存器与状态机连接**：状态寄存器的输出连接到状态机的输入，状态机的输出连接到状态寄存器的输入。状态寄存器在时钟信号的驱动下更新状态。
* **控制信号输出**：状态机根据当前状态生成控制信号，控制信号直接输出到数据通路，驱动各功能部件完成相应的操作。
* **指令译码信号反馈**：指令译码信号作为状态机的输入之一，根据当前指令的类型和执行阶段，影响状态机的状态转换和控制信号的生成。

通过以上设计思路，可以实现一个多周期MIPS CPU的硬布线控制器，确保每条指令能够按照预定的流程正确执行。

3、实验步骤

3.1、构建多周期CPU数据通路

在 Logisim 中构建多周期 CPU 数据通路布线图时，应严格按照多周期 CPU 数据通路示意图进行。以下是具体注意事项：

首先，Sign Extend 部分要利用16位至32位扩展器实现有符号数扩展。接着，通过 ALU 的 equal 信号来生成分支信号 branch，其计算方式为 branch = beq \* equal + bne \* (~equal)，生成的 branch 信号应连接至 PC 寄存器输入端的数据选择器使能端。由于 PC 信号为32位，而存储器地址输入端为10位，且 PC 是字节地址，存储器输入为字地址，因此需要使用分离器提取 PC 的第2-11位，并将其送入存储器地址输入端。最后，确保将复位信号连接到各个寄存器，以便在需要时能够正确复位整个数据通路。

按照上述要求完成布线后，可得到如图2.1所示的多周期 CPU 数据通路布线图。

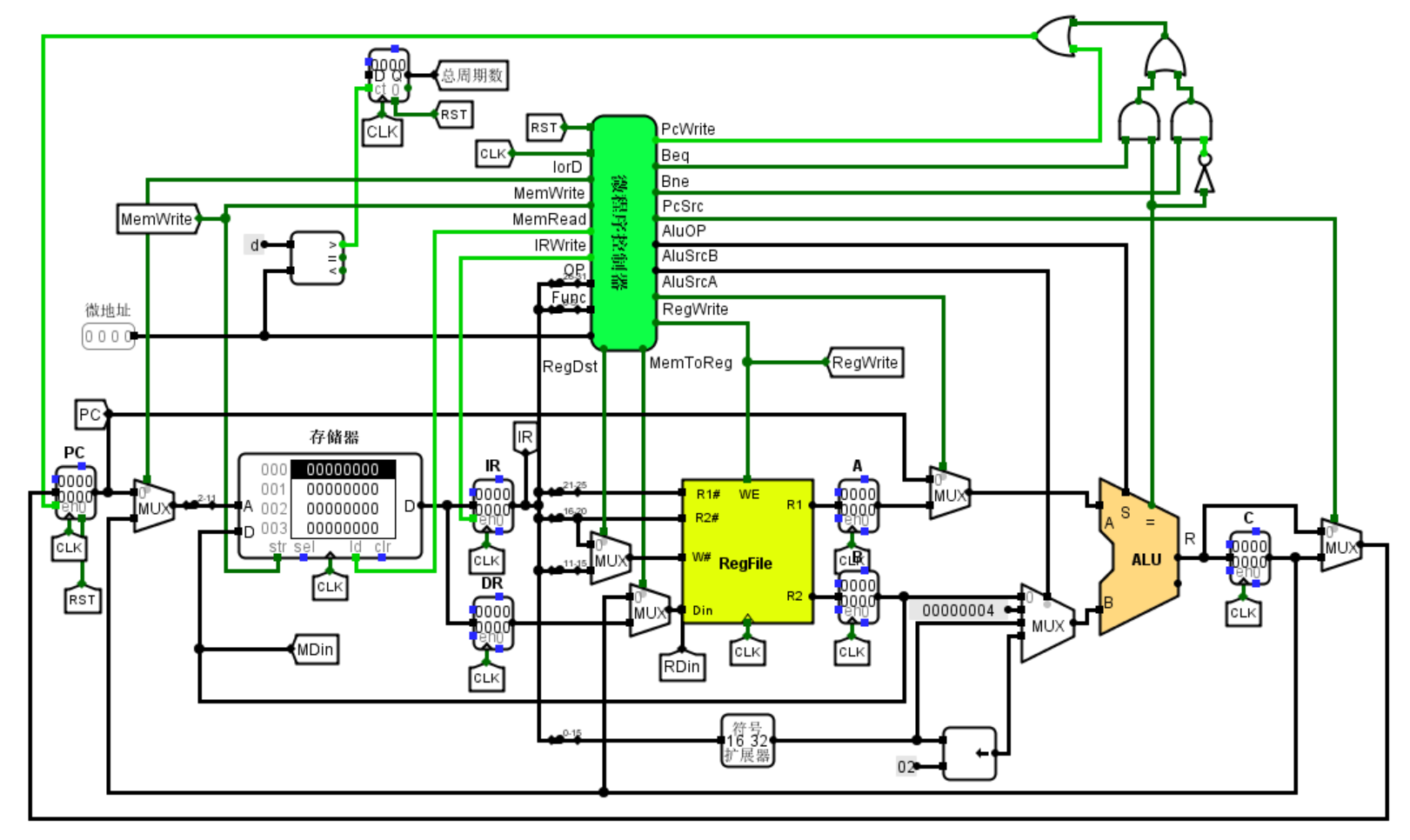


图2.1 多周期CPU数据通路布线图

3.2、构建多周期CPU微程序控制器

（1）设计地址转移逻辑

地址转移逻辑，输入为7个指令译码信号，输出为微程序入口地址。该地址转移逻辑会根据不同的指令译码信号生成相应的微程序入口地址。

根据图1.3状态转移图填写excel表格微程序地址转移逻辑自动生成表，自动生成微程序地址逻辑表达式，如图2.2所示。





图2.2 微程序地址转移逻辑自动生成表达式

将生成的逻辑表达式填入Logisim中，利用Logisim自动生成电路功能得到微程序地址转移逻辑，如图2.3所示。

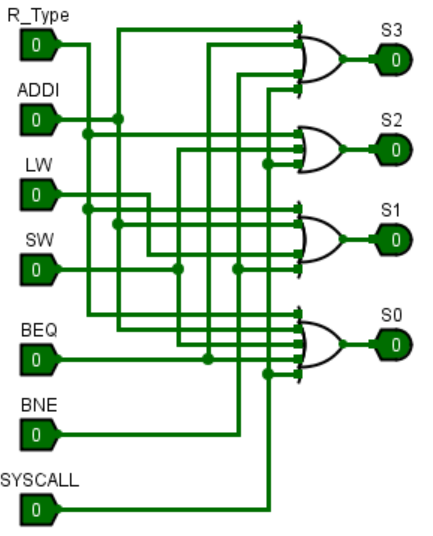


图2.3 微程序地址转移电路图

（2）设计指令译码信号与ALU控制信号

在Logisim中设计多周期MIPS CPU的指令译码信号和ALU控制信号，需注意以下要点：

对于地址转移逻辑，其输入为7个指令译码信号，生成方式与单周期MIPS CPU相同。而ALU控制信号则需重新设计，其由ALU\_Control决定：ALU\_Control为00时ALU执行加法，为01时执行减法，为10时则依据指令的Func字段确定操作。

为实现此逻辑，采用4路选择器。首路输入为ALU加法操作码，次路为减法操作码。第三路输入需结合Func字段，故使用比较器比较Func字段与SLT指令的Func字段值（0x2A）。若相等，经二路选择器将减法操作码（用于比较大小）输入第三路；否则，输入加法操作码。第四路输入置为加法操作码，以防悬空。

依此设计，可准确生成ALU控制信号，确保ALU操作精准无误，其布线如图2.4所示。

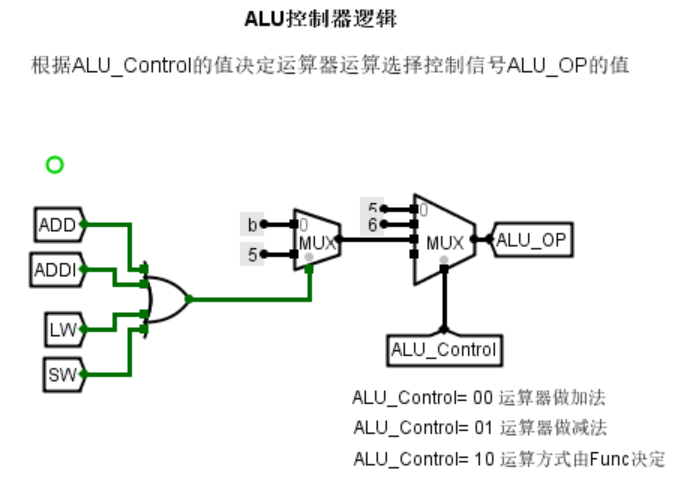


图2.4 ALU控制信号布线图

（3）构建微程序

依据指令状态变换图及微指令控制信号表，填写Excel格式的微指令自动生成表格，如图1.4所示。每条指令执行阶段的末条微指令下地址字段填0，非末条微指令该字段则填为下条微指令地址。取指微指令的下地址字段对应译码微指令地址；译码微指令的下地址字段为0，表示需依当前状态与控制信号选定下一状态。syscall微指令的下地址字段设为自身地址，使其在该状态持续循环，从而实现停机操作。

将得到的微指令复制生成一个txt文本（如图2.5所示），将其加载到微程序控制器的控制存储器组件。即完成微程序控制器的设计。如图2.6所示。

图形用户界面, 应用程序, Word

描述已自动生成

图2.5 导出的微指令

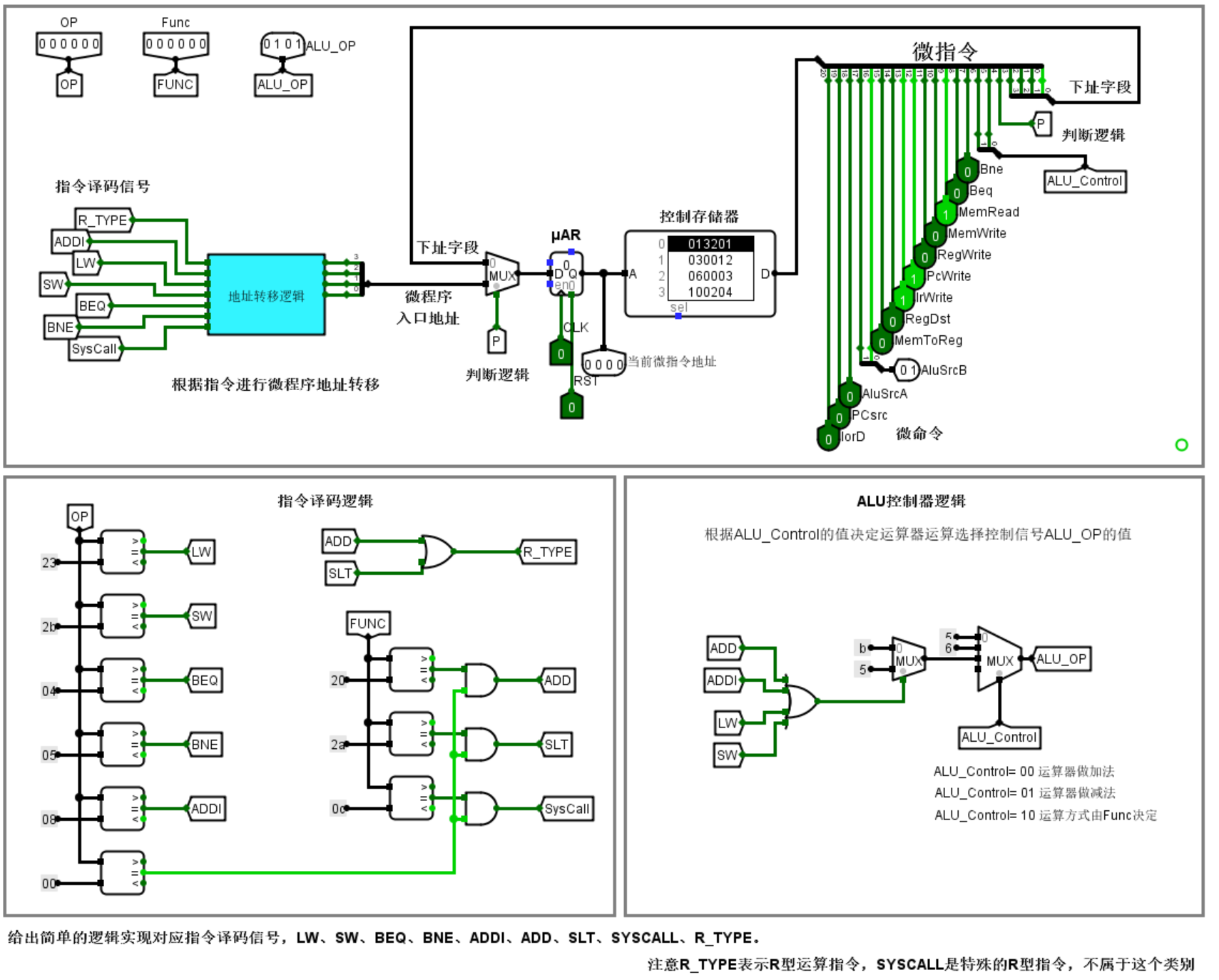


图2.6 微程序控制器布线图

3.3 构建多周期 CPU 硬布线控制器

(1) 构建指令译码信号、ALU 控制信号

指令译码信号与 ALU 控制信号的构建方式与微程序控制器中的设计相同。

(2) 构建状态机逻辑

状态机的输入为指令译码信号和当前状态，输出为下一个状态。根据指令状态变换图和微指令控制信号表，填写 Excel 格式的硬布线控制器状态转换逻辑自动生成表，如图2.7所示。



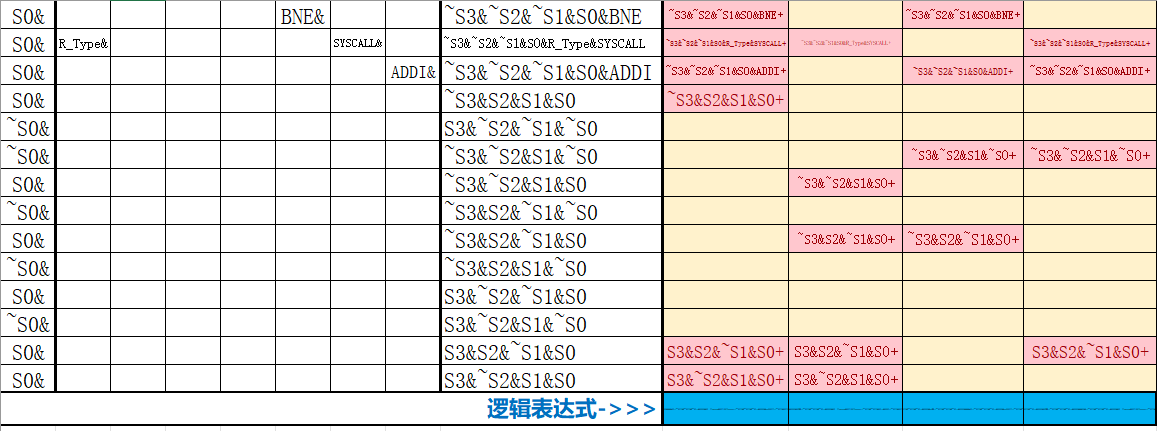


图2.7 硬布线控制器状态转换逻辑自动生成表达式

依据自动生成的逻辑表达式，将其填入 Logisim 中，利用其自动生成电路功能构建状态机 FSM 电路，如图 2.8 所示。

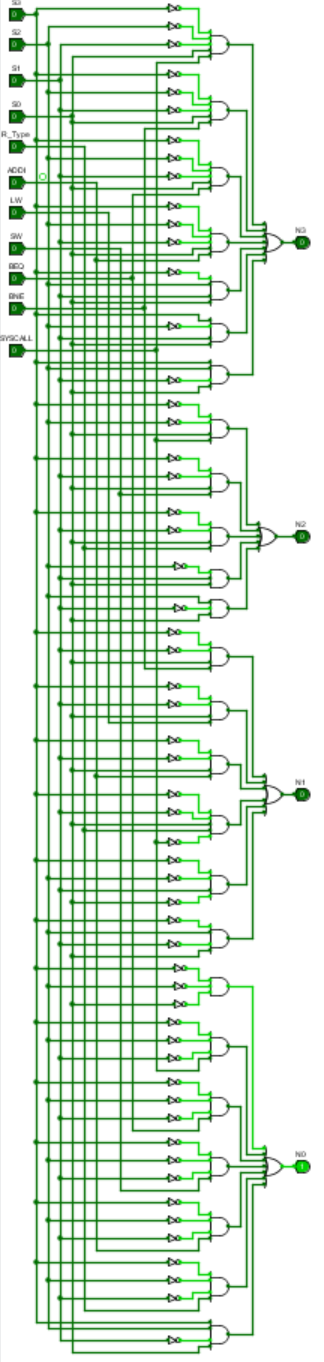


图2.8 状态机FSM电路

(3) 硬布线控制器组合逻辑生成

还需实现一个从当前状态到控制信号的组合逻辑。由于硬布线控制器的状态寄存器相当于微程序控制器中的微地址寄存器，状态字的作用与微地址相似，因此可复用微程序控制器中的控制存储器。将状态寄存器的输出作为控制存储器的输入，即可利用控制存储器生成各控制信号。

至此，硬布线控制器的构建完成，如图 2.9 所示。

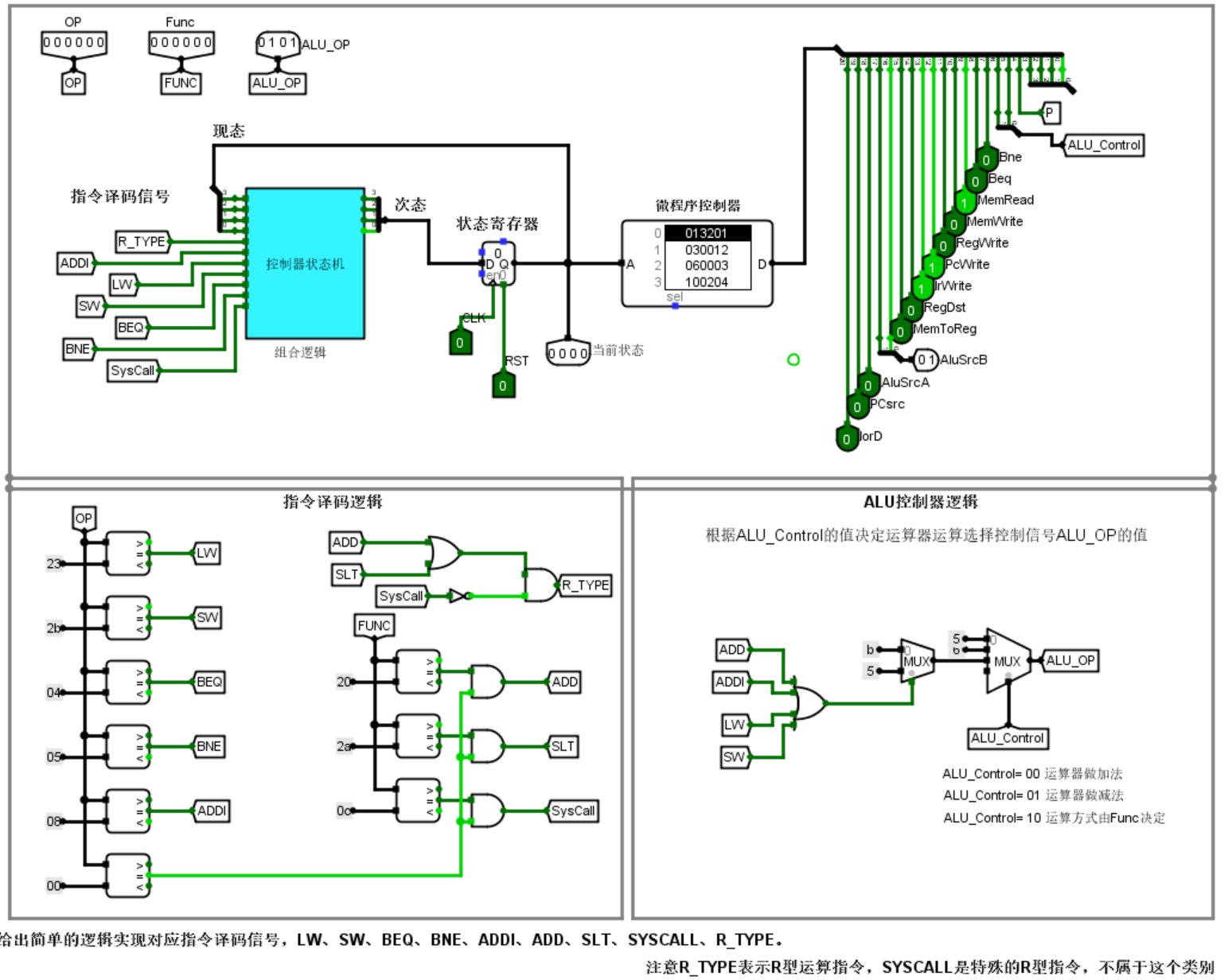
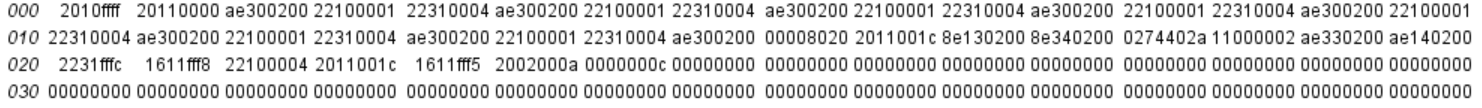


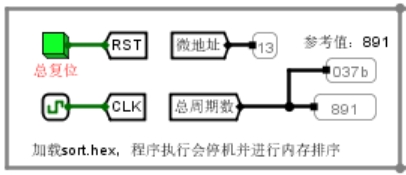
图2.9硬布线控制器布线图

四、实验结果

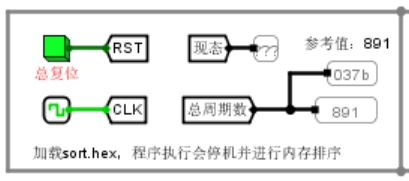
sort.hex程序：



多周期微程序结果：



多周期硬布线结果：



实验结果均在891处停下，结果正确。

5、实验中的bug和解决方案

5.1 问题1：仿真软件异常

Logisim仿真软件存在一些问题。试验过程中会出现明明电路正确，出现整块报错的问题。重启软件后才恢复正常。

5.2 问题2：周期数异常，程序无法停止

**故障现象**：在设计微程序 CPU 并进行测试时，发现周期数显著超出预期的正确范围。尽管数据写入功能正常，但周期数不断攀升。

**原因分析**：

**地址转移逻辑问题**：初步怀疑地址转移逻辑存在错误。经检查，地址转移逻辑的相关表格未发现明显问题。

**微指令设计缺陷**：进一步检查微指令设计，发现某条微指令的下址字段未能正确指向后续微指令的地址，导致指令执行流程陷入循环。

**解决方案**：纠正微指令的下址字段，使其正确指向后续微指令地址后，成功解决了指令循环问题，周期数回归正常范围。

5.3 问题3：状态机逻辑错误

**故障现象**：在构建硬布线控制器的状态机逻辑时，根据指令状态变换图和微指令控制信号表填写了 Excel 格式的硬布线控制器状态转换逻辑自动生成表。然而，由于表格填写错误，利用 Logisim 自动生成电路功能构建的状态机 FSM 电路未能正常工作。

**原因分析**：填写的状态转换逻辑表存在错误，导致生成的逻辑表达式不正确。这使得状态机的输出次态与预期不符，进而影响了整个控制器的正常运行。

**解决方案**：经过仔细检查和核对状态转换逻辑表，纠正了其中的错误条目。随后，依据修正后的表格重新生成逻辑表达式，并在 Logisim 中更新状态机 FSM 电路。最终，状态机能够按照正确的逻辑进行状态转换，确保了硬布线控制器的正常运行。