**计算机组成原理**

**--实验报告**

**专业班级： 人工智能2204**

**学 号： U202215150**

**姓 名： 王子旭**

**实验名称： CPU实验**

**报告日期： 2024年5月30日**

目录

[一、实验目的 2](#_Toc11983)

[1.1、MIPS多周期微程序实验 2](#_Toc17410)

[1.2、MIPS多周期硬布线实验 2](#_Toc27895)

[二、设计要求 2](#_Toc30111)

[三、实验步骤 8](#_Toc22913)

[3.1、构建多周期CPU数据通路 8](#_Toc23293)

[3.2、构建多周期CPU微程序控制器 9](#_Toc6930)

[3.3、构建多周期CPU硬布线控制器 12](#_Toc26983)

[四、 实验结果 15](#_Toc32283)

[五、实验中的问题与解决方法 16](#_Toc32403)

[5.1、问题1 16](#_Toc18524)

[5.2、问题2 16](#_Toc13471)

## **一、实验目的**

1.1、MIPS多周期微程序实验

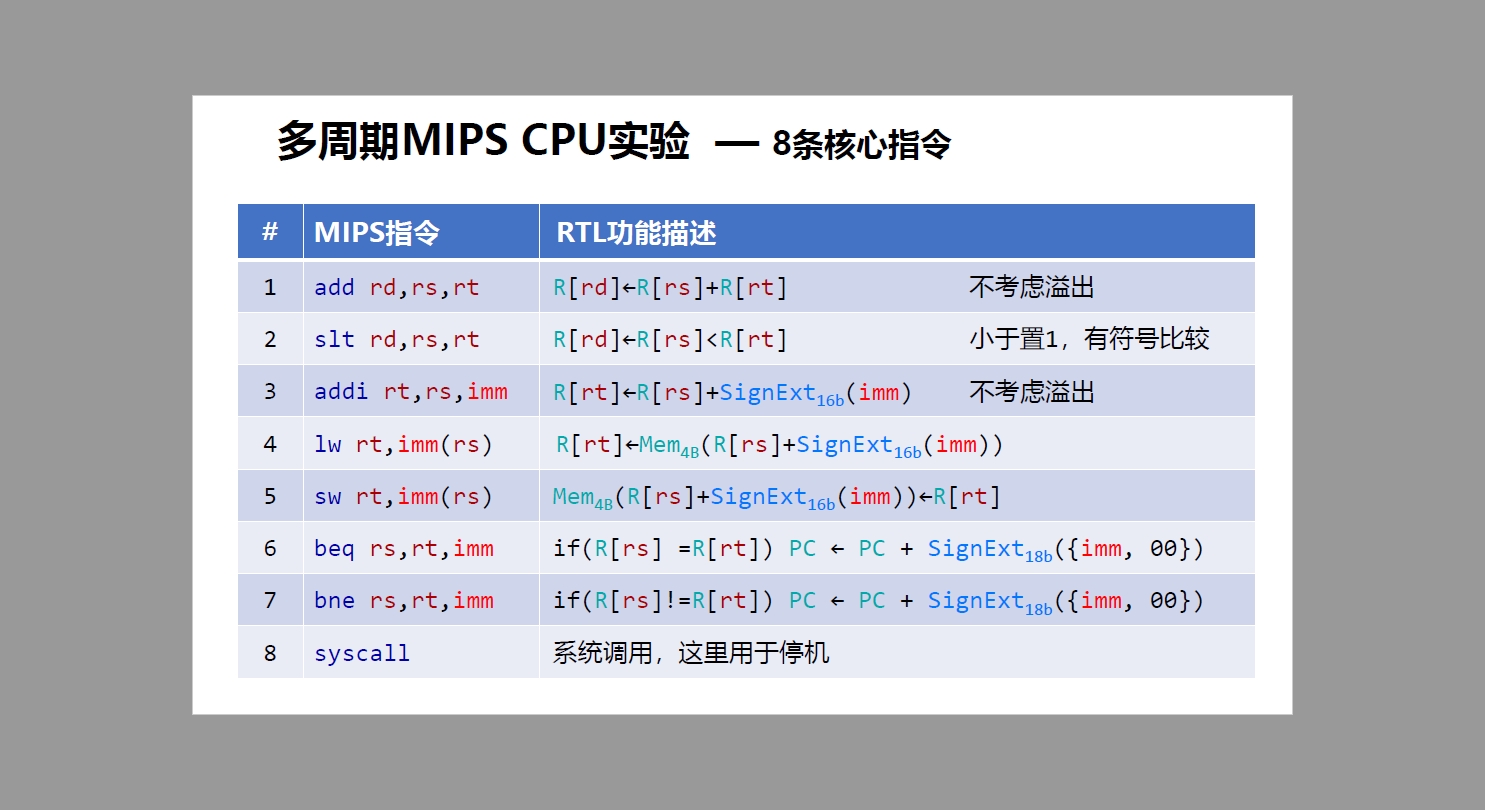
完成cpu.circ中多周期MIPS（微程序）和多周期微程序控制器电路，加载sort.hex并进行测试。

1.2、MIPS多周期硬布线实验

完成cpu.circ中多周期MIPS（硬布线）和多周期硬布线控制器电路，加载sort.hex并进行测试。

## **二、设计要求**

在logisim平台利用已给出的组件构建一个32位MIPS 多周期CPU，该CPU支持如表1. 1列出的核心指令集中的8条指令。要求绘制多周期MIPS CPU数据通路，以两种方式实现控制器，即分别实现微程序控制器和硬布线控制器。并最终能在完成的CPU上运行冒泡排序的测试程序sort.hex且获得正确的输出结果。



### **2.1、方案设计**

#### 2.1.1、设计多周期MIPS CPU数据通路

在多周期MIPS CPU中，不再区分指令存储器与数据存储器，各功能部件也只保留一个（如存储器、ALU），分时使用各功能部件，在时间上错开。且在功能部件的输出端增加相应寄存器锁存数据方便时序控制。将指令执行的过程细分成若干段，每条指令的执行分为“取指->译码->执行”三个阶段，取指和译码阶段是所有指令的必经阶段，执行阶段具体不同的指令占用的时钟周期不尽相同。不同的时钟周期下也会产生相应的控制信号。多周期CPU的整体架构与单周期CPU的不同之处具体如下：

1. 不再区分指令存储器与数据存储器，指令和数据保存在同一个存储器中，取指令与存数据操作分时使用存储器；
2. 部分功能部件，如存储器、ALU等可以在一条指令执行过程的不同时钟周期内多次分时使用；
3. 主要功能部件输出端都增加了寄存器并引入时序控制，在后续时钟周期中要用到的所有数据必须锁存在相应的寄存器中。具体增加了数据寄存器DR，用于存放从存储器读取的数据；增加了指令存储器IR，用于存放从存储器取出的指令，增加了三个存储器A、B、C分别用于保存RegiFile和ALU的输出；
4. PC作为指令计数器，由于不同指令时钟周期数不同，因此PC不再仅由时钟周期控制，而是增加了专门的写操作控制信号；
5. 由于取消了单周期CPU中加法器的使用，PC+4的操作和地址转移指令中PC地址加上立即数的操作都由ALU来完成，所以ALU的B输入端又增加了两个输入，共4个输入，应采用4路选择器选择输出。A端也多了一个PC值的输入，增加一个2路选择器选择输出。
6. 由于存储器复用，除了原有的PC值外，存储器的输入端口还需将经过ALU计算得到的地址送入存储器地址输入端。
7. 由于新引入了一些寄存器，需引入相应的控制信号。
8. 多周期CPU的停机功能无需halt信号，借由状态的变迁来实现（一直循环在某个状态达到停机的目的）。

根据上述设计思想，逐条指令分析其执行过程，绘制单条指令数据通路，最后对数据通路进行综合，绘制多周期CPU数据通路示意图，如图1. 1所示。

图示, 示意图

描述已自动生成

图1. 1 多周期CPU数据通路示意图

#### 2.1.2、设计多周期MIPS CPU微程序控制器

多周期CPU控制下，不同指令对应不同的时钟周期数，每一条指令的执行分为“取指->译码->执行”三个阶段，其中取指和译码两个阶段所有指令对应的数据通路相同控制信号也相同，执行阶段由于不同指令所进行的微操作不同所需时间也不同。

对于微程序控制器，每条指令由若干条微指令组成，每条微指令对应一个微操作（如IR<-(MEM[PC])），一个微操作的顺利执行需要一系列可以并发产生的控制信号进行控制。且每条微指令对应一个时钟周期，在每一个时钟周期内这些并发的控制信号控制完成此条微指令。一系列微指令共同完成一条指令的全部操作。故接下来分析取指、译码阶段以及每条指令执行阶段的具体执行过程，每条微指令执行过程（即每个时钟周期）进行何种微操作。并根据每一微指令划分时钟周期（每一时钟周期对应状态图中的每一个状态），根据该微指令执行过程的数据通路给出所需要的控制信号，进而通过控制信号生成微指令。

（1）首先给出取指、译码阶段以及每条指令的执行流程表，如下各表所示：

表1.2 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1.3 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1.4 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1.5 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1.6 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1.7 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1.8 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1.9 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1.10 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 |

（2）根据以上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，如图1. 2所示。

并根据状态图的操作和指令执行时的数据通路分析给出不同状态所需的控制信号。其中微指令地址即为状态编号。微指令及对应控制信号表如表1. 11所示。

图示

描述已自动生成

图1.2 指令状态变换图

表1. 11 微指令及对应控制信号表

表格, 日历

描述已自动生成

#### 2.1.3、设计多周期MIPS CPU硬布线控制器

设计硬布线控制器的关键是实现一输入为现态和指令译码信号、输出为次态有限状态机。而控制信号则只与该有限状态机的次态有关。首先需分析各指令画出所有指令的指令周期流程图，明确各节拍控制信号，构建指令状态变换图。不同的状态需要不同的控制信号，形成相应的数据通路，而不同的控制信号仅与状态有关（输出仅与状态有关，为Moore型电路）。

根据指令状态变换图，共13个状态，首先需要一4位的状态寄存器记录状态编号。状态寄存器的输出为FSM的现态，输入为FSM的次态。当时钟到来时，现态更新为上一个次态。状态的转换由状态机组合逻辑实现。

状态机的输入为指令译码信号、反馈信号和现态，输出为次态。状态机为纯组合逻辑，可利用真值表自动生成。

状态机生成后可根据状态机的现态生成相应的控制信号（硬布线控制器组合逻辑）。正如前所述，不同的状态对应相应的控制信号。这一部分也为组合逻辑。

## **三、实验步骤**

### **3.1、构建多周期CPU数据通路**

根据多周期CPU数据通路示意图，利用如所示组件，在logisim中完成多周期CPU数据通路布线图。

需额外注意之处：

1. 示意图中Sign Extend部分使用16位to 32位扩展器进行有符号数扩展。
2. 利用ALU的equal信号，生成branch=beq\*equal + bne\*~equal分支信号，此branch信号即对应示意图中的Branch信号，将其连接至PC寄存器输入端的数据选择器使能端。
3. PC信号为32位，存储器地址输入端为10位，又因为PC为字节地址，而存储器输入为字地址，故需用分离器取PC的第2-11位送入存储器地址输入端。
4. 将复位信号连接至各寄存器。

完成布线图如图1. 4所示。

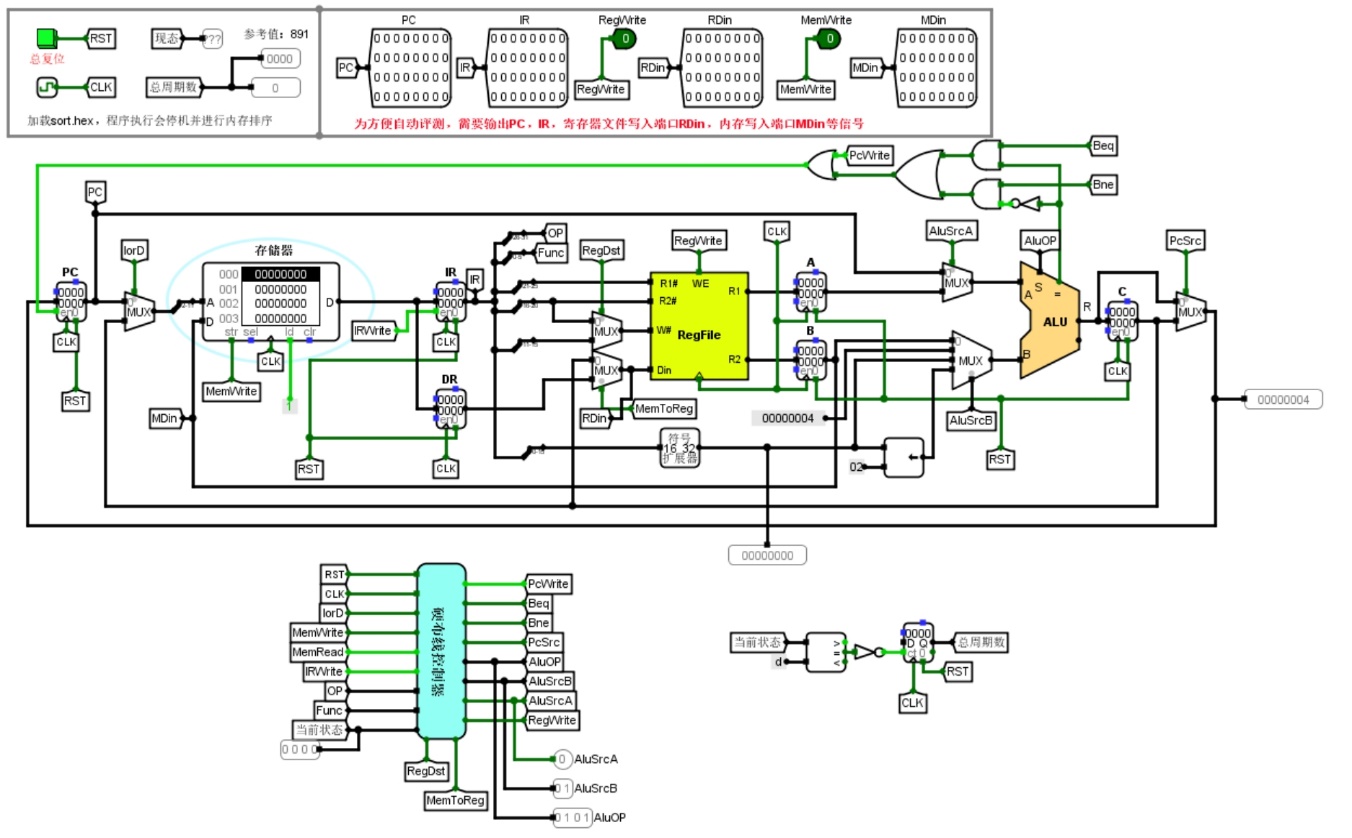


图1.4 多周期CPU数据通路布线图

### **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1.2.4中的微指令控制信号表填写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，如下所示。

表1. 12 微程序地址转移逻辑自动生成表



表1. 13 微程序地址转移逻辑自动生成表达式

图示

中度可信度描述已自动生成

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1. 5所示。

图示, 示意图

描述已自动生成

图1. 5 微程序地址转移逻辑

（2）设计指令译码信号与ALU控制信号

地址转移逻辑的输入为7个指令译码信号，指令译码信号的得到与单周期MIPS CPU中一致。但ALU控制信号发生了一些变化，需重新设计。根据ALU\_Control的值决定运算器运算选择控制信号ALU\_OP的值，ALU\_Control为00时做加法，为01时做减法，为10时由Func决定。故采用4路选择器。第一个输入端为ALU的加法操作码，第二个输入端为ALU的减法操作码。为了不使有输入端存在悬空，第四个输入置为加法的操作码5（默认为加法操作）。第三个输入取决于FUNC字段，故将FUNC字段与SLT的FUNC字段值2a用比较器进行比较后控制一二路选择器输出5（作加法运算）和b（作比较大小运算）。当比较器输出相等时，二路选择器输出b给第三个输入，表示此时应作比较大小运算。其他时候默认作加法运算。布线如图1. 6所示。

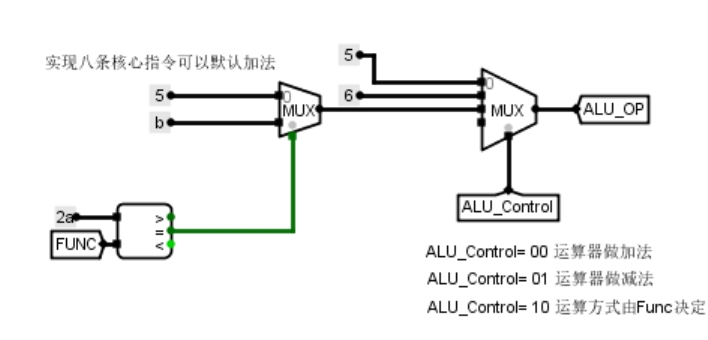


图1.6 ALU控制信号布线图

（3）构建微程序

根据指令状态变换图和微指令控制信号表填写Excel表格微指令自动生成表。如表1. 14所示。

每条指令执行阶段的最后一条微指令的下地址字段为0，非最后一条微指令则为下一条微指令的地址。取指微指令的下地址字段为译码微指令的地址，译码微指令的则为0（表示需要根据当前状态和控制信号选择进行跳转的下一状态）。syscall微指令的下地址字段为自身，一直在该状态循环，即实现停机操作。

表1. 14 微指令自动生成表



将得到的微指令复制生成一个txt文本（如图1. 7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1. 8所示。

图形用户界面, 应用程序, Word

描述已自动生成

图1.7 导出的微指令

图示, 示意图

描述已自动生成

图1.8 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控制信号的构建与微程序控制器中的完全相同。

（2）构建状态机逻辑

状态机的输入为指令译码逻辑和现态，输出为次态。根据指令状态变换图和微指令控制信号表填写Excel表格硬布线控制器状态转换逻辑自动生成表。如表1. 15所示。

表1.15 硬布线控制器状态转换逻辑自动生成表

电脑萤幕画面

低可信度描述已自动生成

表1. 16 硬布线控制器状态转换逻辑自动生成表达式

表格

描述已自动生成

根据自动生成的表达式填入logisim中用自动生成电路功能生成状态机FSM电路。如图1. 9所示。

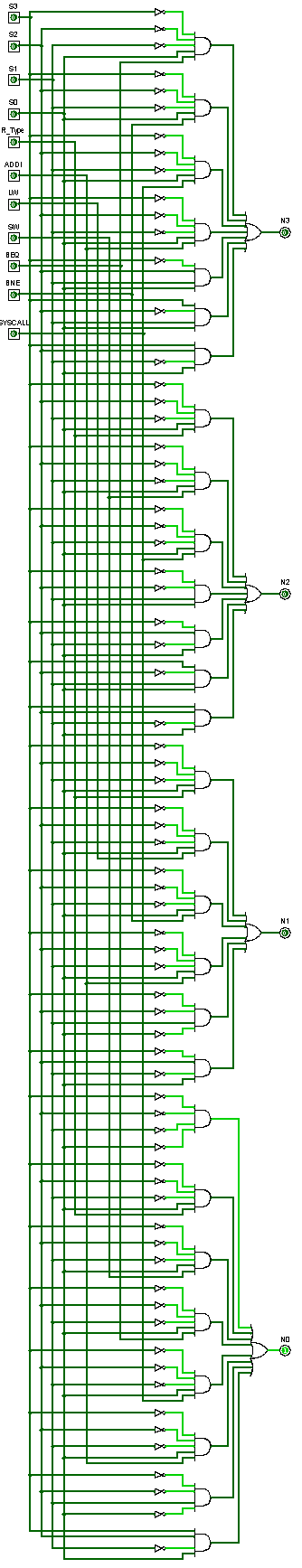


图1.9 状态机FSM电路

（3）硬布线控制器组合逻辑生成

还须实现一个由现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本质上等同于微程序控制器中的微地址寄存器，状态字等价于微地址，故在此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，便可利用控制存储器生成各控制信号。

自此，便完成了硬布线控制器的构建（如图1. 10所示）。

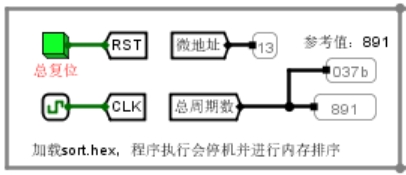
图示, 示意图

描述已自动生成

图1.10 硬布线控制器布线图

## **实验结果**

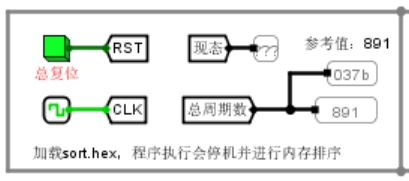
多周期微程序：



图片包含 日历

描述已自动生成

多周期硬布线：



实验结果均在891处停下，结果正确。

## **五、实验中的问题与解决方法**

### 5.1、问题1

**故障现象：**设计微程序CPU时测试的时候发现周期数远远超过测试的正确结果

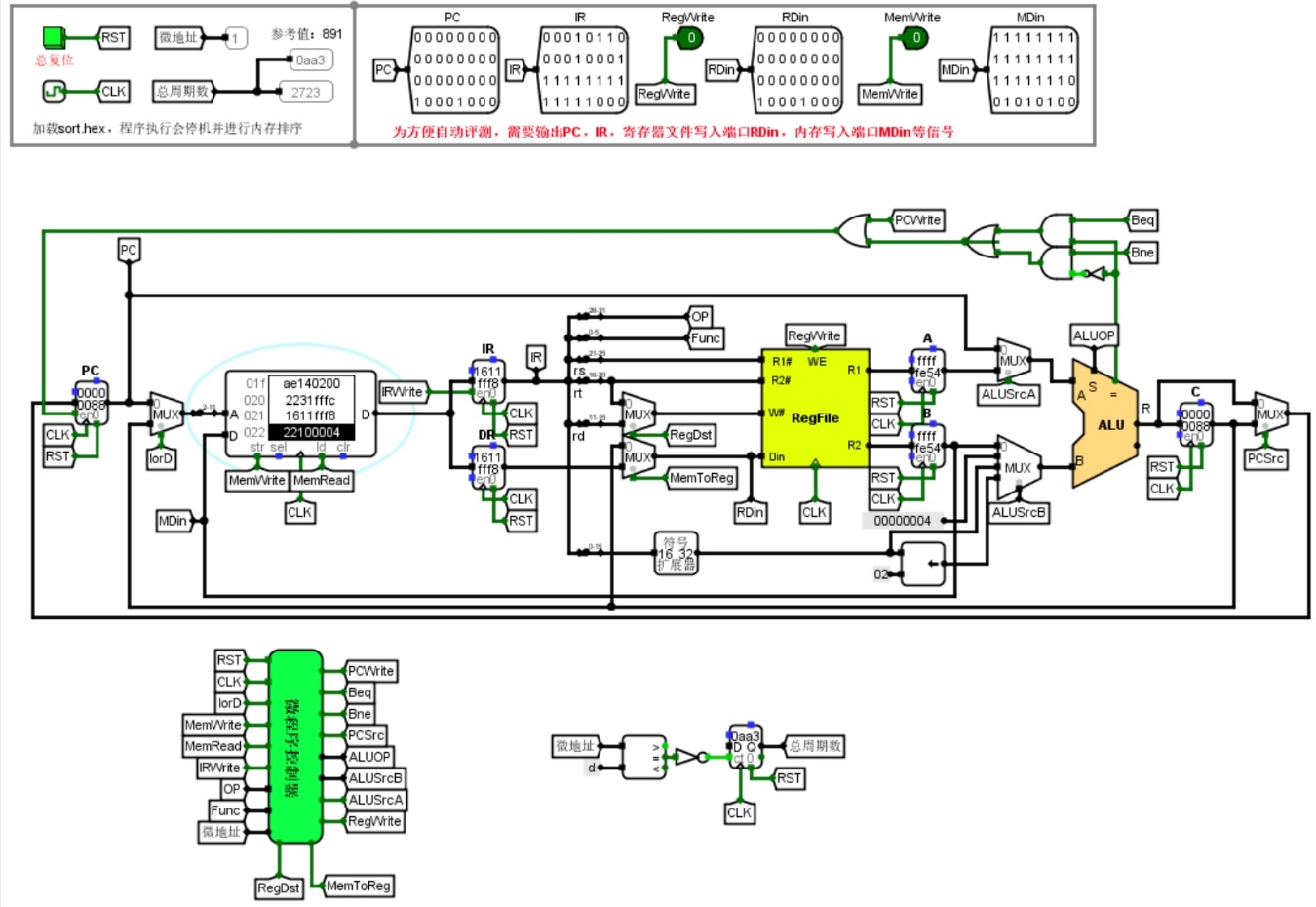


图1. 11 故障1图

**原因分析：**如图1. 11所示，运行过程中周期数已经远远超过参考值，数据能正常写入，但sort.hex中的指令一直在01a到022周围无限循环，首先怀疑是地址转移逻辑出现了问题，检查地址转移逻辑的表格发现没有问题，还有可能是微指令设计有误，检查微指令表格发现R1的下址字段并没有转移到R2，显然出现了问题



**解决方案：**修改下址字段为1000即可

### 5.2、问题2

**故障现象：**进行sort.hex测试后结果异常，有数据写入但是数据顺序存在问题

**原因分析：**猜测是MemWrite控制信号出了问题。

**解决方案：**后来重新检查了下数据通路，发现MemWrite信号和MemToReg信号连反了，将其调换即可。