***2024***



**逻辑与计算机系统设计 ·实验报告·**

j0242087[1]

|  |  |
| --- | --- |
| 专 业： | 数据科学与大数据技术 |
| 班 级： | BD2201 |
| 学 号： |  |
| 姓 名： |  |
| 电 话： |  |
| 邮 件： |  |
| 完成日期： | 2024-7-7 |



目 录

[1 CPU实验 2](#_Toc499846042)

[1.1 设计要求 2](#_Toc499846043)

[1.2 方案设计 2](#_Toc499846044)

[1.3 实验步骤 5](#_Toc499846045)

[1.4 故障与调试 16](#_Toc499846046)

[1.5 测试与分析 18](#_Toc499846047)

[2 总结与心得 20](#_Toc499846048)

[2.1 实验总结 20](#_Toc499846049)

[2.2 实验心得 20](#_Toc499846050)

[参考文献 22](#_Toc499846051)

# CPU设计实验

## 设计要求

1. 实验要求构建一个32位MIPS CPU处理器，包括单周期硬布线CPU、多周期微程序CPU以及多周期硬布线CPU，该处理器应支持核心指令集中列出的所有指令，见表1.1。具体指令功能参见MIPS标准文档。最终设计完成的CPU应能运行标准测试程序。

表 1.1 核心指令集

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **指令** | **格式** | **备注** |
| 1 | Add | add $rd, $rs, $rt | 指令功能及指令格式  参考MIPS32指令集 |
| 2 | Add Immediate | addi $rt, $rs, immediate |
| 3 | Load Word | lw $rt, offset($rs) |
| 4 | Store Word | sw $rt, offset($rs) |
| 5 | Branch on Equal | beq $rs, $rt, label |
| 6 | Branch on Not Equal | bne $rs, $rt, label |
| 7 | Set Less Than | slt $rd, $rs, $rt |
| 8 | syscall（display or exit） | syscall | 系统调用，用于停机 |

## 方案设计

### 单周期CPU功能部件

1. 程序计数器PC，用于存放当前指令执行位置的地址；
2. 指令存储器IM，存储CPU要执行的指令；
3. 数据存储器DM，根据设计需求，所有指令都必须在一个时钟周期之内完成，只采用一个存储器不可能在一个时钟周期内同时完成对指令和数据的操作；
4. 立即数扩展器，用于将I型指令中的16位立即数扩展为32位；
5. 控制器，产生控制信号，控制指令执行的数据通路；
6. 寄存器，提供32个MIPS通用寄存器；
7. 算术逻辑单元ALU，产生运算结果。

以上所提到的功能部件在logisim中的外观如下表1-2所示

|  |  |  |  |
| --- | --- | --- | --- |
| 单周期CPU功能部件 | 部件外观（logisim） | 单周期CPU功能部件 | 部件外观（logisim） |
| 指令计数器PC |  | 寄存器 |  |
| 指令存储器IM |  | 算术逻辑单元ALU |  |
| 数据存储器DM |  |  |  |
| 立即数扩展 |  |  |  |
| 控制器 |  |  |  |

表1-2.单周期CPU功能部件

### 多周期CPU功能部件

1. 指令计数器PC，用于存放当前指令执行位置的地址；
2. 存储器Mem，包括数据存储器以及指令存储器；
3. 指令寄存器IR，存放当前指令；
4. 数据存储器DR，存放要操作的数据；
5. 寄存器堆RegiFile，提供32个MIPS通用寄存器；
6. 立即数扩展器S-EXT，用于用于将I型指令中的16位立即数扩展为32位；
7. 控制器，产生控制信号，控制指令执行的数据通路；
8. 算术逻辑单元ALU，产生运算结果；
9. 另外，增加三个寄存器A、B、C暂存RegiFile和ALU的数据输出。

以上所提到的功能部件在logisim中的外观如下表1-3所示

|  |  |  |  |
| --- | --- | --- | --- |
| 多周期CPU功能部件 | 部件外观（logisim） | 多周期CPU功能部件 | 部件外观（logisim） |
| 指令计数器PC |  | 立即数扩展器 |  |
| 指令存储器Mem |  | 控制器 |  |
| 指令寄存器IR |  | 算术逻辑单元ALU |  |
| 数据存储器DR |  |  |  |
| 寄存器堆RegiFile |  |  |  |

表1-3.多周期CPU功能部件

### 地址转移逻辑NPC

执行完一条指令后，下一步该执行哪条指令，根据实验要求的八条核心指令分析，有以下几种情况：

1. 顺序执行，继续执行相邻的下一条语句；
2. 遇到有条件分支，跳转到当前地址加上偏移地址。

两种可能的跳转，一次只能选择一个值，需要1个控制信号，由操作控制器生成。

### 立即数扩展器

立即数扩展有两种，都是扩展成32位。

1. 条件跳转指令中16位偏移地址的扩展，偏移量可正可负，需要用符号扩展；
2. 在I型指令中，操作数为16位的立即数，操作数可正可负，需要用符号扩展。

### 操作控制器

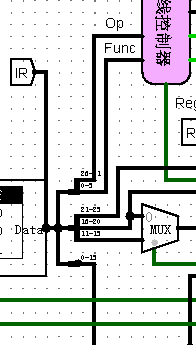
操作控制器根据指令的操作码和func产生控制信号。使用工程化方法生成。

1. 列出所有指令下，各个数据通路的值；
2. 将每一项数据都进行合并，当一项数据有多种情况，表明该数据项需要用控制信号进行选择，否则，不需要选择，可以直连；
3. 控制信号综合，列出各指令下，每个控制信号的值为1或0，从而可以得出控制信号的逻辑表达式，生成电路；
4. 多周期控制器可分为微程序控制器和硬布线控制器，具体功能实现在实验步骤中进行详细的描述。

## 实验步骤

### 指令解析

将32位输入操作码用分线器接出，最高6位为操作码op，21-25位为rs寄存器编号，16-20位为rt寄存器编号、11-15位为rd寄存器编号， 0-5位为功能码func；取低16位，作为I型指令中的立即数操作数。单周期和多周期指令译码方式相同，图1-1和图1-2所示：



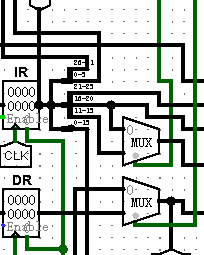


图1-1单周期指令解析 图1-2多周期指令解析

### 实现地址转移逻辑

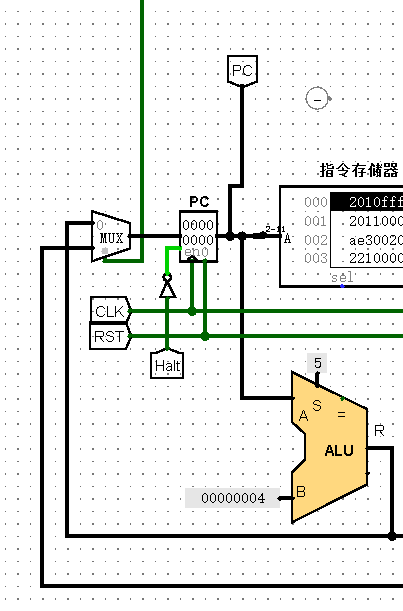
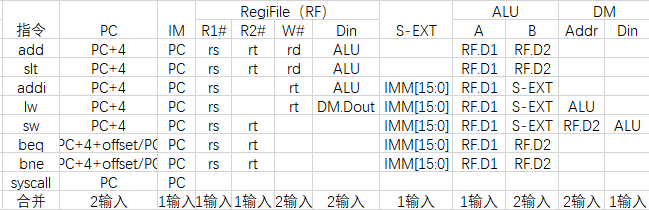
PC+4接到MUX的0输入端，16位立即数符号扩展的结果左移2位，加上PC+4，接到MUX的1输入端，选择信号为branch。实验要求的的分支转移指令为beq和bne，两条指令均为I型指令，由于两条指令的特殊性，可利用ALU的equal信号，最终branch=beq\*equal+bne\*/equal形成分支信号。具体的实现如图1-3所示:

图1-3.地址转移逻辑部分logisim电路图

### 单周期硬布线控制器设计

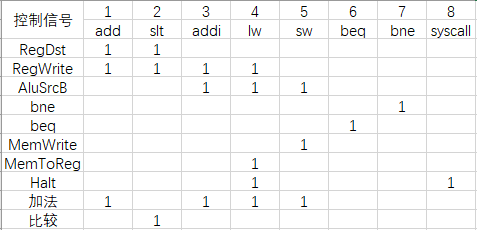
1. 数据通路综合
2. 构建数据通路。绘制主要功能部件输入来源表，该表主要用于描述控制类信号，仅保留数据类信号，具体如表1.4，最左侧为指令助记符，第一行为数据项。
3. 输入源合并。计算表1.4中每一个数据项的输入种类，若有多个输入来源，则需要控制信号配合多路选择器进行选择。

表 1.4单周期功能部件输入来源表



1. 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1.4所示，横坐标给出的是不同指令的译码信号，表中有1的位置表示当前指令会产生对应的信号，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逻辑或即可得到控制信号的逻辑表达式。
2. RegDst为1表示RegiFile写回地址由R型指令rd字段给出，否则由I型指令rt字段给出；
3. RegWrite为1打开RegiFile写使能，表示数据写回RegiFile；
4. AluSrcB为1表示ALU的第二个操作数将由立即数扩展器S-EXT给出，否则由RegiFile的第二个输出给出；
5. bne为1表示指令为bne；
6. beq为1表示指令为beq，beq与bne信号与ALU equal信号结合可控制PC数据来源；
7. MemWrite为1打开数据存储器DM写使能，将ALU运算结果写入数据存储器DM；
8. MemToReg为1表示从数据存储器DM中选数据送入RegiFile；
9. Halt为停机信号，为1时系统停机；
10. 加法信号，表示ALU应对两个操作数执行加法操作；
11. 比较信号，表示ALU应对两个操作数执行比较操作。

表 1.5 单周期硬布线控制器信号表



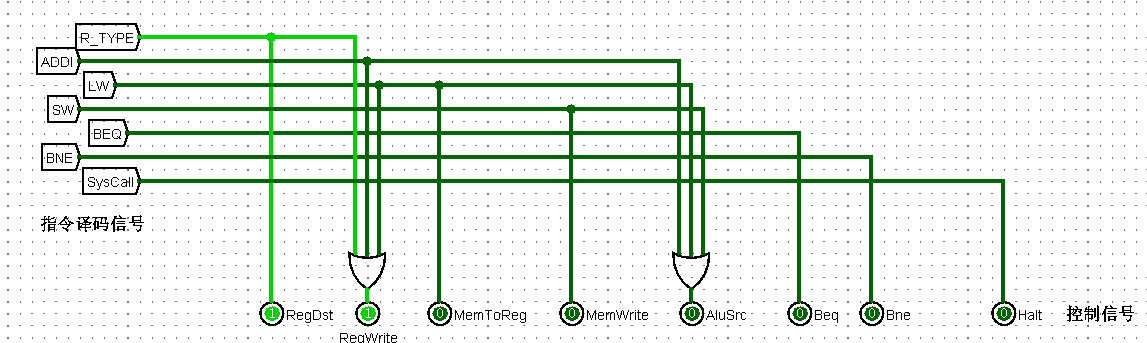
1. 根据以上分析，最终形成的单周期硬布线控制器电路如下：

图 1.4 MIPS单周期CPU控制器电路

### 单周期CPU总体结构图

通过上面的分析，我们已经得到单周期CPU的所有功能部件，我们也知道了所有指令的数据通路，以及如何产生控制信号控制指令数据的传递，因此把这些部件和控制信号连接起来就得到了单周期CPU总体结构图。

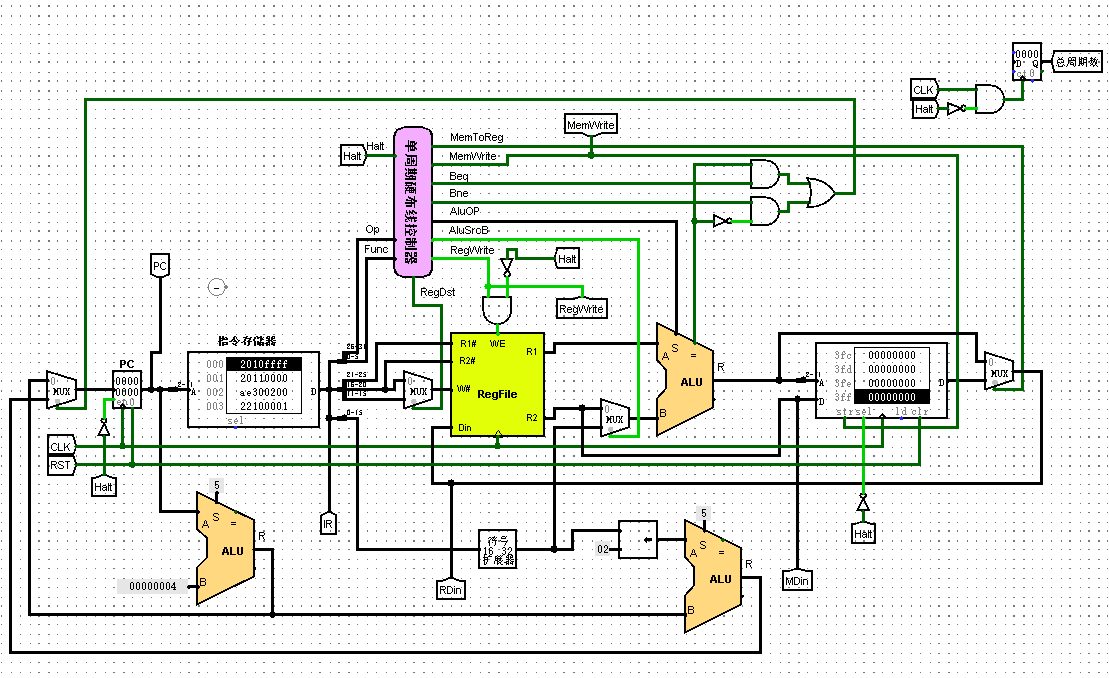


图 1.5 MIPS单周期CPU总体结构图

### 多周期控制信号

多周期CPU控制下，不同指令对应不同的时钟周期数，因此不能像单周期控制器一般一次性给出一条指令对应的所有控制信号，而每一条指令的执行又可以拆解为三部分“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，因此现在的任务退化为分析每条指令执行阶段需要的时钟周期并给出每个时钟周期下所有的控制信号，指令的执行控制应遵循以下几个原则：

1. 有序进行；
2. 不会破坏系统中保存的结果，仅仅对指令本身功能要求对系统状态做出改变；
3. 保证不同周期下指令的每一步执行所用到的数据都来自正确来源，保证每一步的运算结果保存到正确输出。

基于此原则，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 1.6取指阶段操作流程

|  |  |
| --- | --- |
| **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 1.7add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 1.8 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 1.9 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 1.10 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 1.11 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 1.12 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 1.13 addi指令执行操作流程

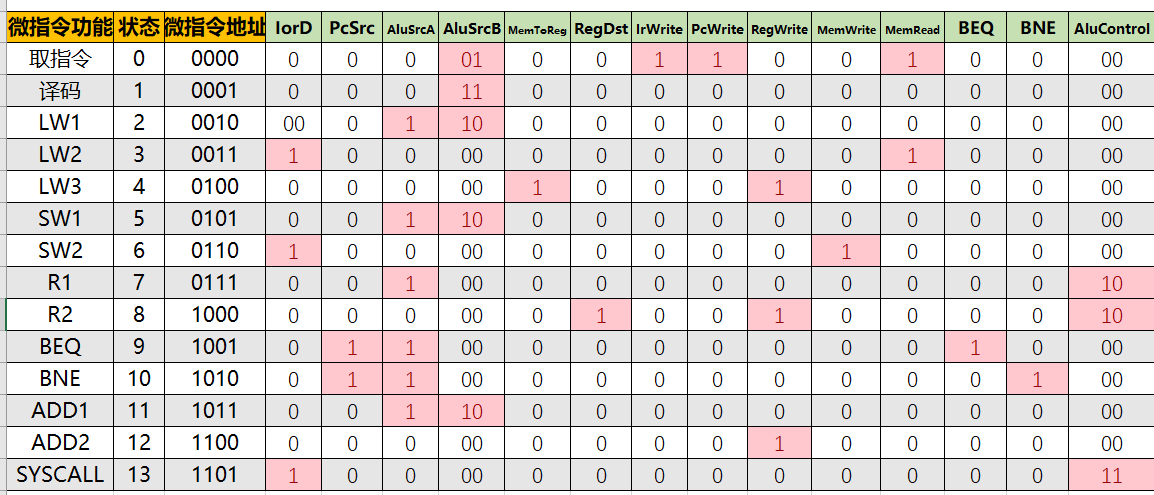
|  |  |
| --- | --- |
| **指令阶段** | **操作流程** |
| 加运算 | C<-(A)+ S-EXT(IR[15:0]) |
| 写回 | R[IR[20:16]]<-(C) |

表 1.13 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 |

综合以上八条操作流程，可以给出所有指令不同阶段的控制信号表，如表1.13所示，控制信号说明如下：

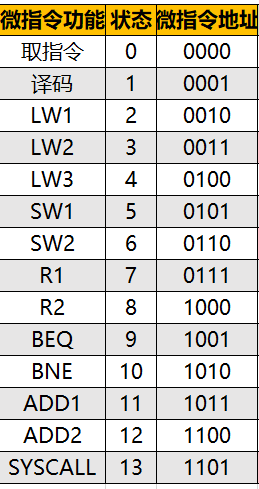
1. PCWrite：PC写使能控制，取指令周期，分支指令执行；
2. IorD：指令还是数据，0表示指令，1表示数据；
3. Irwrite：指令寄存器写使能；
4. MemWrite：写内存控制信号；
5. MemRead：读内存控制信号；
6. Beq：beq指令译码信号；
7. Bne：bne指令译码信号；
8. PcSrc：PC输入源，0表示顺序寻址，1表示跳跃寻址；
9. AluControl：ALU控制信号，即加法或比较运算；
10. AluSrcA：ALU第一输入选择，PC还是寄存器输出；
11. AluSrcB：ALU第二输入选择，R型指令输入为寄存器输出，取指阶段为4，sw、lw、addi指令为立即数，跳转指令bne、beq相应的偏移地址；
12. RegWrite：寄存器RegiFile写使能；
13. RegDst：RegiFile的第二个寄存器编号由R型指令rd给出；
14. MemToReg：lw指令，写入寄存器的数据来自存储器。

表 1.14指令控制信号表

### 多周期微程序控制器

1. 状态编码，8条指令共13个执行阶段，采用4位二进制编码表示微指令地址，如下表1.15：

表 1.15 状态编码表



1. 微指令地址转移逻辑，采用下址字段法，配合判断状态P，取指阶段P为1，表明控制存储器的地址由指令的第一个阶段的微程序地址给出，执行阶段P为0，表明下一条微指令地址由下址字段给出，得出取指阶段指令的地址转移逻辑如下表1.16：

表 1.16 微指令地址转移逻辑

1. 生成的地址转移逻辑表达式如下所示：

S3 = ADDI + BEQ + BNE + SYSCALL；

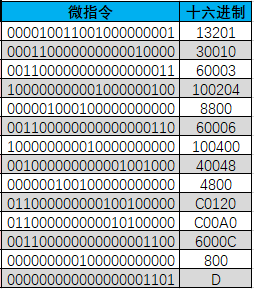
S2 = R\_Type + SW + SYSCALL；

S1 = R\_Type + ADDI + LW + BNE；

S0 = R\_Type + ADDI + SW + BEQ + SYSCALL。

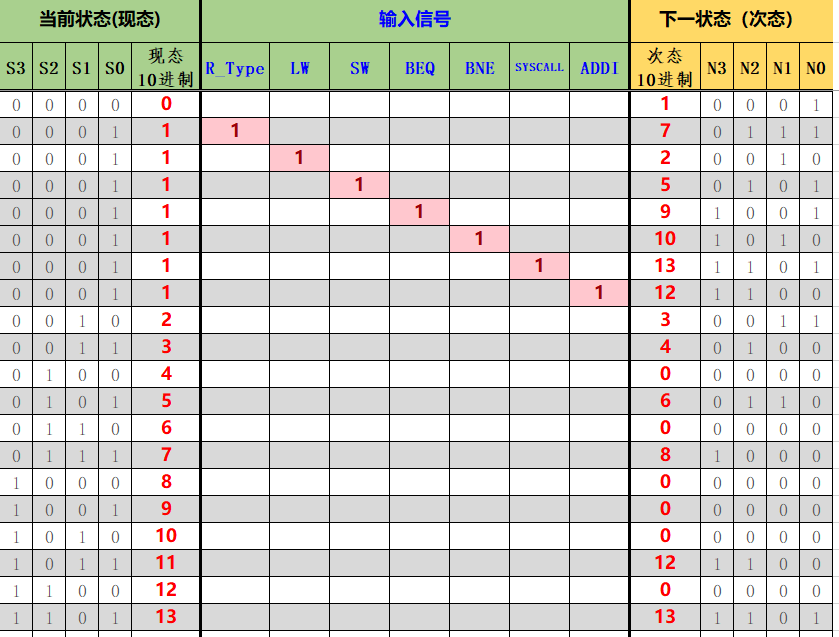
1. 控制存储器16进制微指令生成，根据1.3.6的分析，8条指令执行需要的控制信号需要16位编码，而采用下址字段法，下址所需编码位数和微指令地址位数相同，即4位，一位判断标志位P，P为0表示取指阶段，控制存储器地址由指令的微指令转移逻辑产生，P为1表示控制存储器地址由下址字段产生，即表示指令的不同执行阶段（即每条指令对应多条微指令），因此微指令编码21位，地址位编码4位，按1.3.6的控制信号编码，产生的微指令如下表1.18：

表 1.18 微指令编码



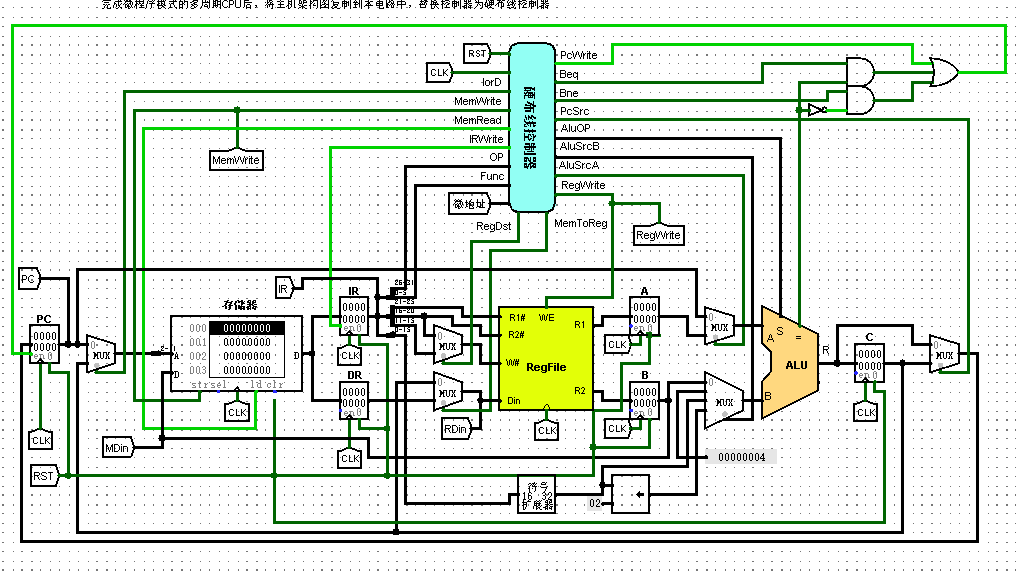
### 多周期硬布线控制器

1. 状态编码，和微程序控制器编码相同；
2. 状态转移，增加有限状态机FSM实现现态到次态的转换，次态即下一条微指令在控制存储器中的地址，而FSM采用纯组合逻辑电路实现，中间是具体的连线， FSM状态转换如表1.19；

表 1.19 FSM状态转换

1. 硬布线控制存储器，地址编码4位，由于状态转移已经由FSM给出，因此不再需要下址字段和判断字段P，可减少为16位二进制编码，当然本实验中为减少工作量，可同样采用21位二进制编码，弃用下址字段和P即可，从而硬布线控存和微程序控存相同。

### 多周期CPU总体结构图

 图1.6 MIPS多周期CPU总体结构图

## 故障与调试

### 单周期数据存储器地址错误

**故障现象：** sort.hex程序执行完毕后，没有在指定的80号存储单元形成降序排列的数据，而是在200号存储单元形成降序排列数据，而且两个数据地址之差为4。

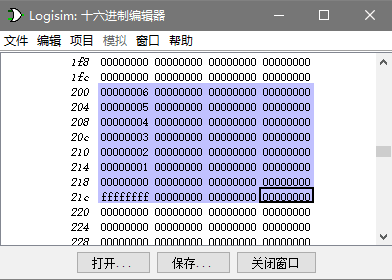


图 1.7存储地址错误

**原因分析：**经查发现Logisim中RAM只支持一种访问模式，一次访问读出32位数据，直接给出字地址使得内存布局错误，如下图：

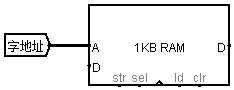


图 1.8直接字地址访问

**解决方案：**字地址除以4即可得到正确的内存布局，即高两位补零作为字节地址送入存储器地址即可，如下图：

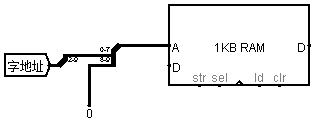


图 1.9字节地址访问

### 指令问题

**故障现象：**现代时序中beq指令计算周期第二个节拍给出的信号有误。；

**原因分析：**开始多给了一个slt信号，原本以为需要再次判断equal，但ALU电路不需要此操作；

**解决方案：**去掉beq指令计算周期第二个节拍给出的信号

### 多周期CPU循环震荡

**故障现象：**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，如下图：

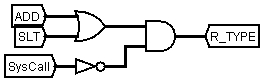


图 1.10syscall与R型互斥

## 测试与分析

### 单周期CPU（硬布线）执行sort.hex

1. 内存布局，在80号单元开始处出现6,5,4,3,2,1,ffff的有符号降序数据，如图：

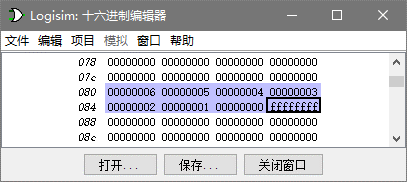


图1.11 单周期执行sort.hex后内存布局

1. 时钟周期，执行完毕后，系统停机，sort.hex的时钟周期数为224，如图：

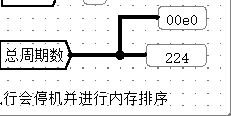


图1.12 单周期执行sort.hex所需时钟周期数

### 多周期CPU（硬布线）执行sort.hex

1. 内存布局，在80号单元开始处出现6,5,4,3,2,1,ffff的有符号降序数据，如图：

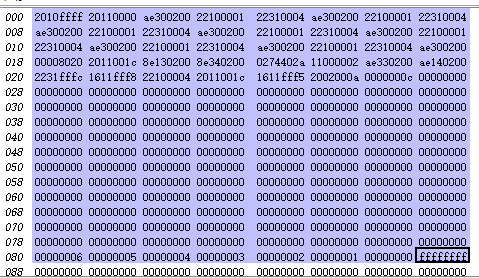


图1.13多周期CPU（硬布线）执行sort.hex后内存布局

1. 时钟周期，执行完毕后，系统停机，sort.hex的时钟周期数为891，如图：

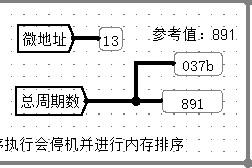


图1.14 多周期CPU（硬布线）执行sort.hex所需时钟周期数

### 多周期CPU（微程序）执行sort.hex

1. 内存布局，在80号单元开始处出现6,5,4,3,2,1,ffff的有符号降序数据，如图：

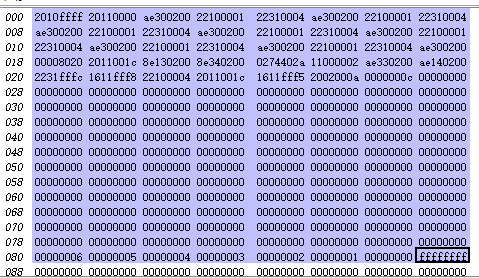


图1.15 多周期CPU（微程序）执行sort.hex后内存布局

1. 时钟周期，执行完毕后，系统停机，sort.hex的时钟周期数为891，如图：

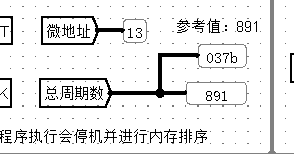


图1.16 多周期CPU（微程序）执行sort.hex所需时钟周期数

# 总结与心得

## 实验总结

本次实验主要完成了如下几点工作：

1. 实现了32位定长指令的译码和指令的解析
2. 实现了对各个寄存器，存储器的读写
3. 实现了对各个操作数的计算，
4. 实现了控制信号的生成
5. 使用多路选择器实现了对多个数据来源的选择和控制。
6. 实现了MIPS单周期CPU硬布线控制器；
7. 实现了MIPS多周期CPU微程序地址转移逻辑；
8. 实现了MIPS多周期CPU硬布线控制器状态机；
9. 完成了单周期和多周期的MIPS CPU的综合数据通路，最终的的CPU能够支持核心指令；

## 实验心得

1. 熟悉了使用logisim的一些基础功能和一些比较高阶的功能。
2. 熟悉的掌握了CPU的各个模块的功能和各个模块的逻辑实现以及具体的电路设计。
3. 熟悉了MIPS的8条核心指令在CPU中的工作中的控制信号的产生过程，清楚了计算机的内存里面具体的指令所转化的信号控制。
4. 在实验中对于秦磊华老师在课上讲述的关于CPU的知识有了更加深刻的理解，秦老师初讲时我只是有个模糊的印象，在实验过程中我才体会到单周期与多周期CPU的差异所在，以及具体分别如何实现它们的微程序控制器和硬布线控制器，对于CPU的知识不再是空中楼阁。
5. 在实验的过程中遇到了不少的问题，不过秦磊华老师给了我们详细的实验指导，以及在许多前辈的经验基础上和与同学的讨论之中，总算是比较顺利地完成了整个实验。
6. 最后总结下来，在逻辑与计算机系统设计的实验课程中确实掌握了一些硬知识，特别是对计算机的硬件底层组成与工作机制有了比较深刻的理解，也克服了我对于硬件学习的畏难心理。

# 参考文献

1. 谭志虎,秦磊华,胡迪青.计算机组成原理实践教程.北京:清华大学出版社，2018年.
2. 谭志虎，秦磊华，吴非，肖亮.计算机组成原理（微课版）. 北京:人民邮电出版社，2021年.

|  |
| --- |
| 一、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: 嵌入签名图片** |
| 二、对课程实验的学术评语（教师填写） |
|  |
| 三、对课程实验的评分（教师填写） |
| |  |  |  |  | | --- | --- | --- | --- | | 评分项目  （分值） | 报告撰写  （30分） | 课设过程  （70分） | 最终评定  （100分） | | 得分 |  |  |  | |
| **指导教师签字:** |