***2019***



**计算机组成原理 ·实验报告·**

j0242087[1]

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1707 |
| 学 号： | U201714786 |
| 姓 名： | 王占成 |
| 电 话： | 17615921537 |
| 邮 件： | 1748995976@qq.com |
| 完成日期： | 2019/12/12 |



目 录

[1 CPU设计实验 2](#_Toc27074369)

[1.1 设计要求 2](#_Toc27074370)

[1.2 方案设计 2](#_Toc27074371)

[1.3 实验步骤 14](#_Toc27074378)

[1.4 故障与调试 14](#_Toc27074379)

[1.5 调试与分析 15](#_Toc27074380)

[1.6 总结与心得 16](#_Toc27074383)

# CPU设计实验

## 1.1 设计要求

根据实验的要求，构建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CPU应能运行并正确处理核心指令集中列出的所有指令，见表 1‑1。最终设计完成的CPU应能运行标准测试程序。

表 ‑ 核心指令集8条

|  |  |  |  |
| --- | --- | --- | --- |
| # | 指令 | 格式 | 功能描述 |
| 1 | Add | add $rd, $rs, $rt | R[$rd]←R[$rs]+R[$rt] 溢出时产生异常，且不修改R[$rd] |
| 2 | Add Immediate | addi $rt,$rs, imm | R[$rt]←R[$rs]+SignExt16b(imm) 溢出产生异常 |
| 3 | Load Word | lw $rt, offset($rs) | R[$rt]←Mem4B(R[$rs]+SignExt16b(imm)) |
| 4 | Store Word | sw $rt, offset($rs) | Mem4B(R[$rs]+SignExt16b(imm))←R[$rt] |
| 5 | Branch on Equal | beq $rs, $rt, label | if(R[$rs] = R[$rt]) PC ← PC + SignExt18b({imm, 00}) |
| 6 | Branch on Not Equal | bne $rs, $rt, label | if(R[$rs] != R[$rt]) PC ← PC + SignExt18b({imm, 00}) |
| 7 | Set Less Than | slt $rd, $rs, $rt | R[$rd]←R[$rs]<R[$rt] 小于置1，有符号比较 |
| 8 | syscall（display or exit） | syscall | 系统调用，这里用于停机 |

## 1.2 方案设计

首先应该确认不同CPU不同的功能部件构成，以此为根据设计不同的功能部件。然后采用模块化的方式，根据各个CPU的数据通路将它们正确地连接起来。

单周期CPU功能部件

1. 指令计数器PC，用于存放当前指令的地址。
2. 指令存储器IMEM，储存CPU将要执行的指令。
3. 寄存器堆RegFile，提供32个MIPS通用寄存器。
4. 算术逻辑单元ALU，根据运算方式运算数据。
5. 数据存储器DMEM，存取数据。
6. 立即数扩展器，用于将16位立即数扩展为32位。
7. 控制器，产生控制信号，控制数据通路的执行过程。

多周期CPU功能部件

1. 指令计数器PC，用于存放当前指令的地址。
2. 存储器Mem，包括数据存储器和指令存储器。
3. 指令寄存器IR，存放当前要执行的指令。
4. 数据存储器DR，存放当前要操作的数据。
5. 寄存器堆RegFile，提供32个MIPS通用寄存器。
6. 立即数扩展器，用于将16位立即数扩展为32位。
7. 控制器，产生控制信号，控制数据通路的执行过程。
8. 算术逻辑单元ALU，根据运算方式运算数据。
9. 增加三个寄存器A、B、C存储寄存器堆和算术逻辑单元的数据输出。

控制器

不管是单周期MIPS CPU 还是多周期MIPS CPU，控制器的设计是CPU设计的核心。整个实验方案的设计的重心便都集中在了控制器的设计上面。

### 1.2.1 指令译码

根据所给的不同CPU的数据通路，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。但是在设计的过程中要注意到，SysCall指令并不属于R类型的指令，因此在指令译码的时候要特别注意。单周期和多周期指令译码方式相同，如图 1‑1所示：

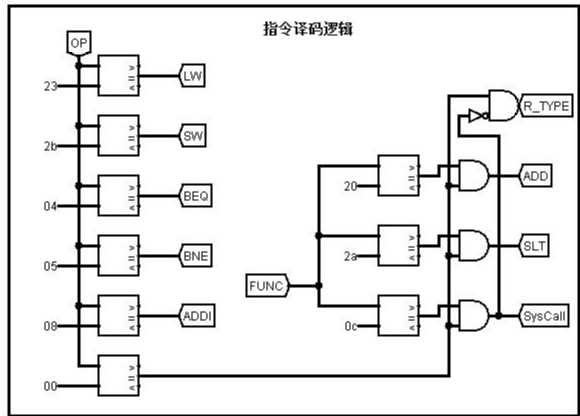


图 ‑ 指令译码

### 1.2.1 ALU控制器逻辑

ALU控制器的设计可以依据如表 1‑2进行设计：

表 ‑ ALU运算功能

|  |  |  |
| --- | --- | --- |
| **ALU\_OP** | **十进制** | **运算功能** |
| 0000 | 0 | Result = X << Y 逻辑左移 （Y取低五位） Result2=0 |
| 0001 | 1 | Result = X >>>Y 算术右移 （Y取低五位） Result2=0 |
| 0010 | 2 | Result = X >> Y 逻辑右移 （Y取低五位） Result2=0 |
| 0011 | 3 | Result = (X \* Y)[31:0]; Result2 = (X \* Y)[63:32] 无符号乘法 |
| 0100 | 4 | Result = X/Y; Result2 = X%Y 无符号除法 |
| 0101 | 5 | Result = X + Y (Set OF/UOF) |
| 0110 | 6 | Result = X - Y (Set OF/UOF) |
| 0111 | 7 | Result = X & Y 按位与 |
| 1000 | 8 | Result = X | Y 按位或 |
| 1001 | 9 | Result = X⊕Y 按位异或 |
| 1010 | 10 | Result = ~(X |Y) 按位或非 |
| 1011 | 11 | Result = (X < Y) ? 1 : 0 符号比较 |
| 1100 | 12 | Result = (X < Y) ? 1 : 0 无符号比较 |

ALU控制器的逻辑设计比较简单，根据不同指令所要执行的运算，来为ALU输入不同的控制信号。ALU控制器逻辑属于简单逻辑，在这里不再赘述。单周期和多周期MIPS CPU的ALU控制器逻辑如下图所示：

单周期：

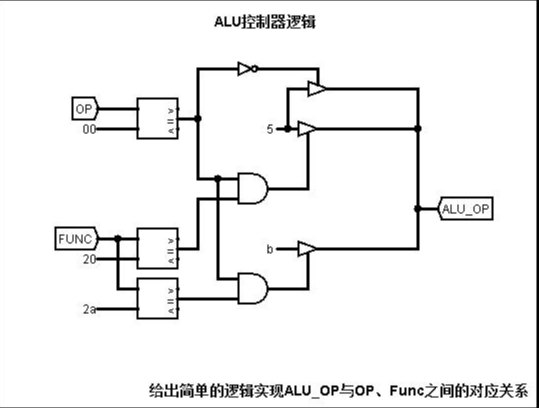


图 ‑ 单周期ALU控制器逻辑

多周期：

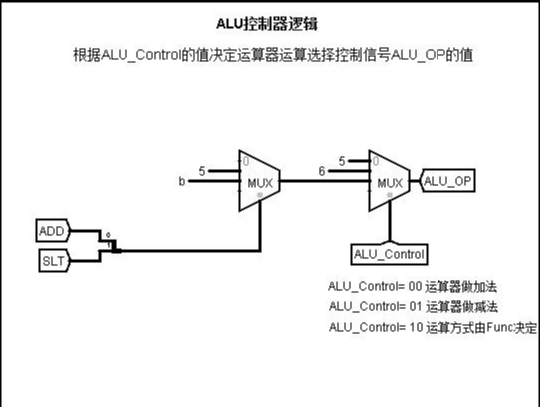


图 ‑ 多周期ALU控制器逻辑

### 1.2.2 单周期硬布线控制器

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。根据不同的指令就算出不同的译码信号，不同的功能需要不同的控制信号来控制。不同控制信号的说明如表 1‑3所示：

表 ‑ 控制信号功能说明表

|  |  |  |
| --- | --- | --- |
| # | 控制信号 | 信号说明 |
| 1 | MemToReg | 写入寄存器的数据来自存储器 |
| 2 | MemWrite | 写内存控制信号 |
| 3 | Beq | Beq指令译码信号 |
| 4 | Bne | Bne指令译码信号 |
| 5 | AluOP | 运算器操作控制符 |
| 6 | AluSrcB | 运算器第二输入选择 |
| 7 | RegWrite | 寄存器写使能控制信号 |
| 8 | RegDst | 写入寄存器选择控制信号 |
| 9 | Halt | 停机信号，取反后控制PC使能端 |

根据控制信号说明表，我们就可以完成单周期硬布线控制信号表。单周期硬布线控制信号表如表 1‑4所示：

表 ‑ 单周期硬布线控制信号表



根据以上分析，最终形成的单周期硬布线控制器电路如下：

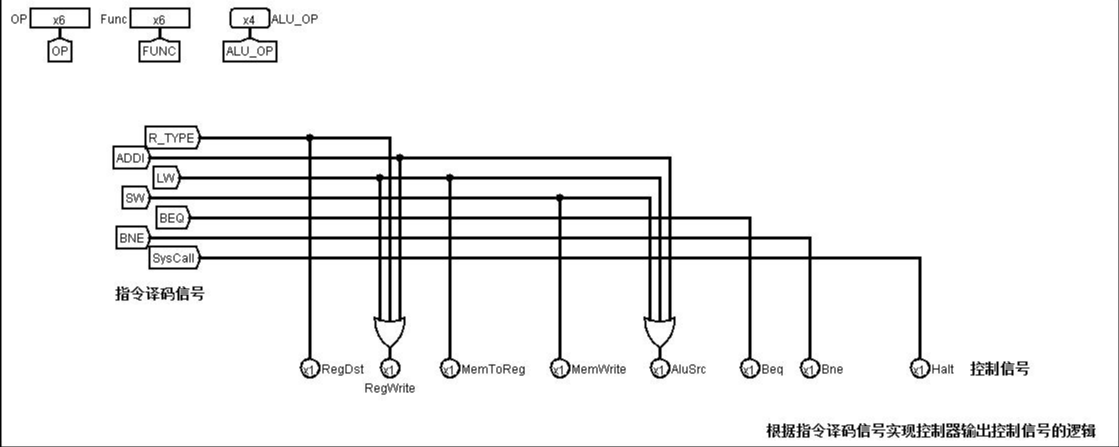


图 ‑ MIPS单周期CPU控制器电路

PC+4接到MUX的0输入端，16位立即数扩展结果左移2位，加上PC+4，接到MUX的1输入端，选择信号为choose。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choose=beq\*equal+bne\*(~equal)形成分支信号。

通过上面的分析，我们已经得到单周期CPU的所有功能部件，并且我们也知道了控制信号的功能以及单周期MIPS CPU的数据通路。所以我们根据自己所得的功能部件和控制信号连接起来，便得到了单周期MIPS CPU的整体结构图。

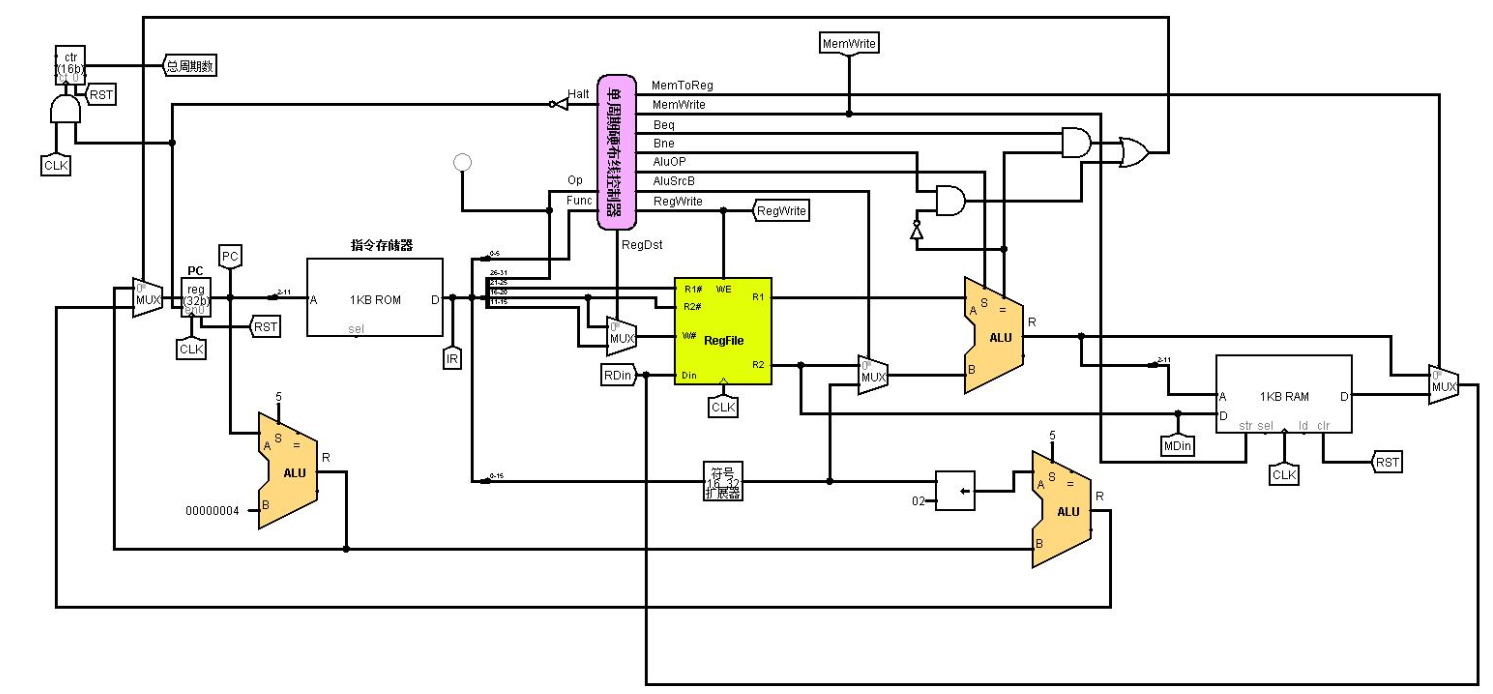


图 ‑单周期MIPS CPU总体结构图

### 1.2.3 多周期微程序控制器

多周期处理器中，指令的执行需要占用多个时钟周期，不同的指令所需的时钟周期数不一定相同，相同指令在不同时钟节拍下产生的控制信号也不同。因此，多周期CPU的整体架构也与单周期CPU也有不同，因此我们简单地总结一下二者之间有何不同的地方。

* 1. 不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。
  2. 部分功能单元，如ALU、RegiFile可以在一条指令执行过程的不同周期中多次使用。
  3. 主要功能单元输出端都增加了寄存器，在后续时钟周期中要用到的所有数据必须存储在相应的寄存器中，增加数据寄存器DR，用于存放从存储器读取的数据，增加指令存储器IR，用于存放从存储器读出的指令，增加三个存储器A、B、C保存RegiFile和ALU的输出。
  4. 更改后ALU输出结果有三种情况：
     1. 分支目标地址，由beq、bne指令给出，该地址将被写到PC中。
     2. 指令的运算结果，将被写入到寄存器文件中。
     3. 为存储器访问指令lw和sw提供存储器地址。
  5. 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。

因为多周期MIPS CPU与单周期MIPS CPU有许多不同的地方，因此它们之间的控制信号也会有所不同。多周期MIPS CPU的控制信号功能说明如表 1‑5所示：

表 ‑ 多周期MIPS CPU控制信号功能表

|  |  |  |
| --- | --- | --- |
| # | 控制信号 | 信号说明 |
| 1 | PCWrite | PC写使能控制 |
| 2 | IorD | 指令还是数据 |
| 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输入来源 |
| 9 | AluOP | 运算器操作控制符 4位 |
| 10 | AluSrcA | 运算器第一输入选择 |
| 11 | AluSrcB | 运算器第二输入选择 |
| 12 | RegWrite | 寄存器写使能控制信号 |
| 13 | RegDst | 写入寄存器选择控制信号 |
| 14 | MemToReg | 写入寄存器的数据来自存储器 |

不同的指令会有不同的执行过程，不同指令的状态变换过程也不一定相同。因此，我们可以给指令不同执行状态进行编码，8条指令一共14个执行状态，采用4位二进制编码我们可以得到表 1‑6：

表 ‑ 指令执行状态编码表

|  |  |  |  |
| --- | --- | --- | --- |
| # | 微指令 | 状态 | 地址 |
| 1 | 取指令 | S0 | 0 |
| 2 | 译码 | S1 | 1 |
| 3 | LW1 | S2 | 2 |
| 4 | LW2 | S3 | 3 |
| 5 | LW3 | S4 | 4 |
| 6 | SW1 | S5 | 5 |
| 7 | SW2 | S6 | 6 |
| 8 | R型运算 | S7 | 7 |
| 9 | R型运算 | S8 | 8 |
| 10 | Beq | S9 | 9 |
| 11 | Bne | S10 | 10 |
| 12 | ADDI1 | S11 | 11 |
| 13 | ADDI2 | S12 | 12 |
| 14 | SYSCALL | S13 | 13 |

根据不同指令的入口地址，可以完成微程序的地址转移逻辑。该转移逻辑属于简单的组合逻辑。机器译码信号和微程序入口地址如图 1‑6所示：

图 ‑ 译码信号与微程序入口地址关系图

最终的地址转移逻辑根据上表生成的表达式自动绘制，得到的地址转移逻辑和微程序地址转移逻辑图如图 1‑7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

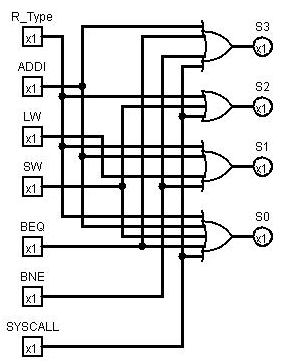


图 ‑ 微程序地址转移逻辑图

控制存储器是由十六进制的微指令组成的，一条微指令是由21位构成的。其中前16位可以根据不同的指令不同的执行状态来产生不同的控制信号。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，即表示下一条指令的地址为下址字段所指。地址位编码4位，代表下一条指令的地址，各条微指令的构成如表 1‑7：

表 ‑ 微指令的构成



最后生成的微指令如表 1‑8：

表 ‑ 微指令译码



### 1.2.4 多周期硬布线控制器

多周期硬布线的状态编码和微程序控制器编码相同。状态转移是通过有限状态机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，因此比较简单。FSM状态转换表如表 1‑9所示：

表 ‑ FSM状态转换表



通过上面的分析，我们已经得到多周期CPU的所有功能部件，并且我们也知道了控制信号的功能以及多周期MIPS CPU的数据通路。所以我们根据自己所得的功能部件和控制信号连接起来，便得到了多周期MIPS CPU的整体结构图。

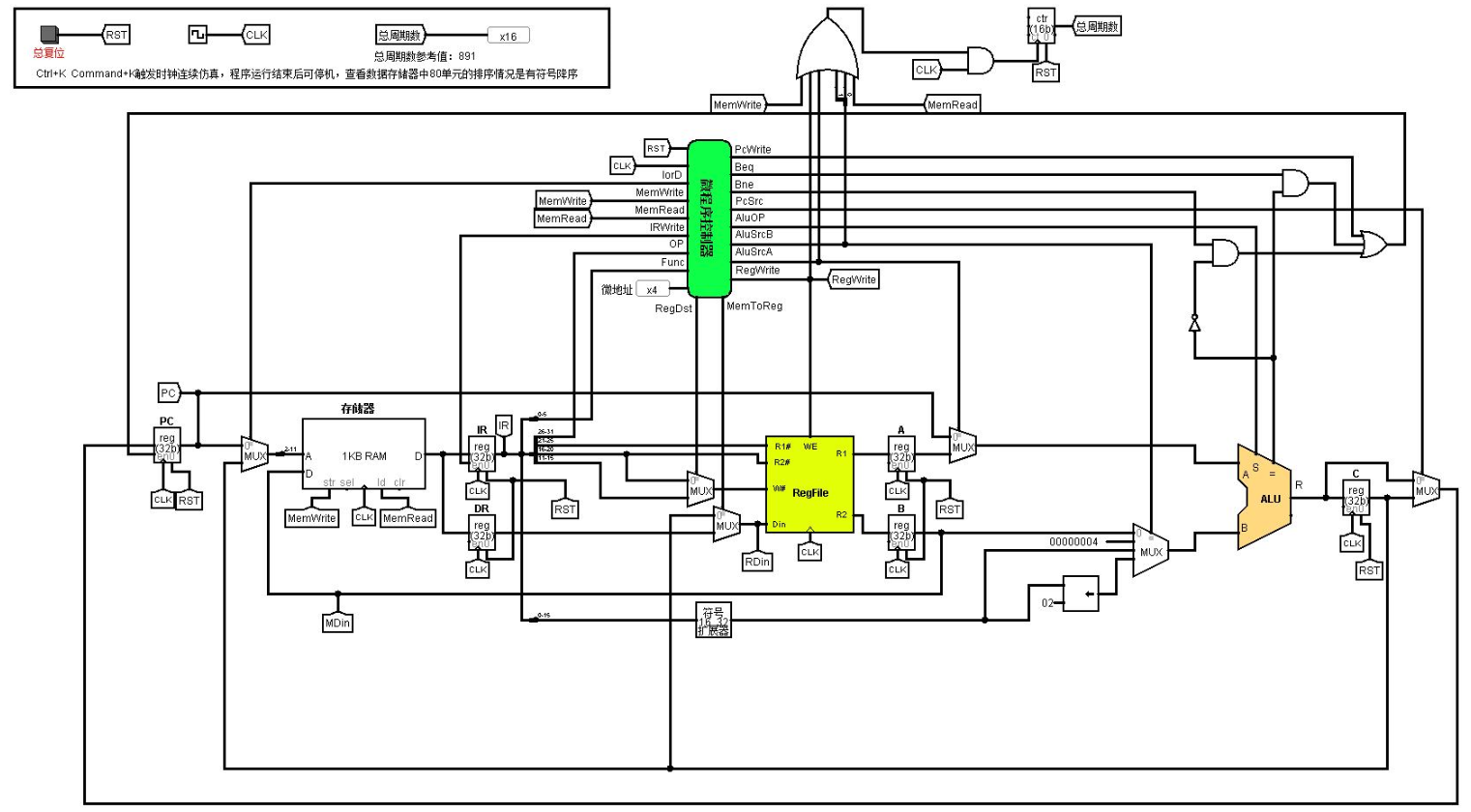


图 ‑ 多周期MIPS CPU的整体结构图

### 1.2.5 细节实现

单周期周期计数模块： 多周期周期计数模块：

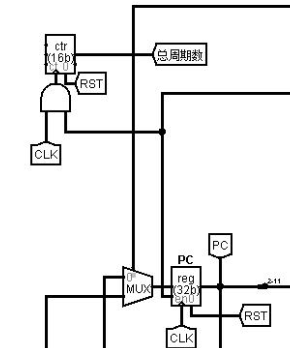
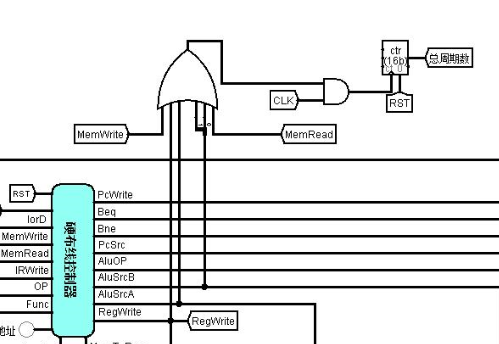
 

图 ‑ 单周期周期计数模块 图 ‑ 多周期周期计数模块

## 1.3 实验步骤

首先按照题目要求对于整个系统的架构进行分析，并了解CPU的整体框架。

根据所得出的数据映射表构建各个模块，并在每个模块构建完成后对其进行单元测试，以保证各个模块独自工作时功能是正常的。

将各个模块按照规划好的模块间关系以及各个信号的走向进行连接，以构建一个完整的CPU。

在构建好的CPU上加载程序并运行，对于执行错误的地方进行单步调试并分析错误原因，然后对于电路进行修改直到没有任何错误出现。

## 1.4 故障与调试

故障1：冒泡排序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。单周期MIPS CPU运算后存储器的内容如图 1‑11所示：

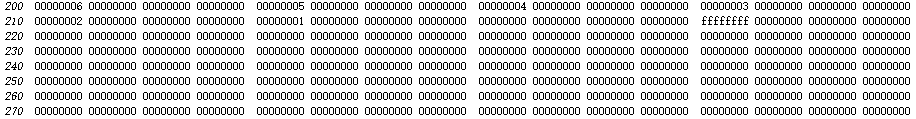


图 ‑ 故障一错误示例

原因：一个数据占用的字节数为4字节，即32位。存储一次性读取或存入的内容也是32位。所以在向存储器中读取或者存入数据时，地址应当除以4之后取址。而我却直接取址，所以导致了错误的发生。错误如图 1‑12：

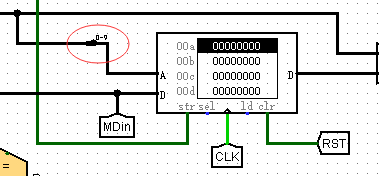


图 ‑ 故障一错误原因

解决方案：字地址除以4，即取第11位到第2位作为地址。

## 1.5 调试与分析

### 单周期CPU执行sort.hex

1. 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 1‑13：

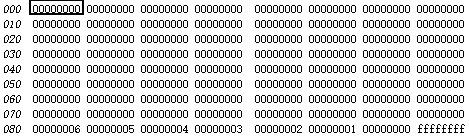


图 ‑ 单周期执行sort.hex后内存显示

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 1‑14：

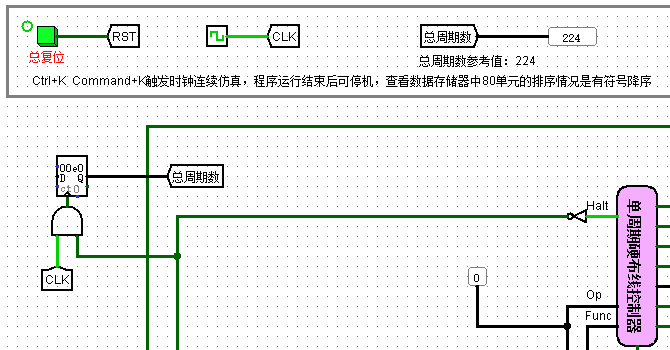


图 ‑ 单周期执行sort.hex所需时钟周期数

### 多周期CPU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 1‑15：

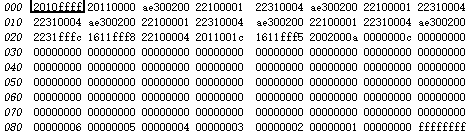


图 ‑ 多周期CPU执行sort.hex后内存显示

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 1‑16：

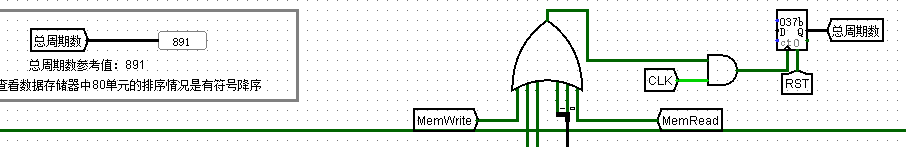


图 ‑ 多周期CPU执行sort.hex所需时钟周期数

# 总结与心得

## 2.1实验总结

本次实验主要完成了如下几点工作：

1. 完成周期MIPS CPU硬布线控制器设计。
2. 完成了多周期MIPS CPU控制存储器设计。
3. 完成了多周期MIPS CPU微程序地址转移逻辑设计。
4. 完成了多周期MIPS CPU硬布线控制器状态机。
5. 完成单周期MIPS CPU和多周期MIPS CPU数据通路综合。

## 2.2 实验心得

1. 熟悉了Logisim的使用以及如何运用逻辑表达式直接完成组合逻辑电路的设计。
2. 对多周期MIPS CPU组合电路的设计有了更深的理解，了解了多周期、单周期的概念。
3. 通过做CPU实验，能对课上遗忘的知识点以及没有听懂的知识点进行及时地复习掌握。
4. 组成原理实验是做起来最舒服的一门实验，相比与其它实验而言，组成原理实验各个知识点以及要求的说明都非常容易理解。