***2017***



**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： | CS1508 |
| 学 号： | U201514699 |
| 姓 名： | 张鑫 |
| 电 话： | 15927282852 |
| 邮 件： | [zhangxinhust@qq.com](mailto:zhangxinhust@qq.com) |
| 完成日期： | 2018-01-03 |



目录

[1 MIPS寄存器文件设计实验 1](#_Toc502750510)

[1.1 设计要求 1](#_Toc502750511)

[1.2 方案设计 2](#_Toc502750512)

[1.3 实验步骤 3](#_Toc502750517)

[1.4 故障与调试 4](#_Toc502750518)

[1.5 测试与分析 5](#_Toc502750520)

[2 MIPS RAM设计实验 7](#_Toc502750521)

[2.1 设计要求 7](#_Toc502750522)

[2.2 方案设计 8](#_Toc502750523)

[2.3 实验步骤 9](#_Toc502750528)

[2.4 故障与调试 11](#_Toc502750529)

[2.5 测试与分析 12](#_Toc502750532)

[3 CPU设计实验 13](#_Toc502750533)

[3.1 设计要求 13](#_Toc502750534)

[3.2 方案设计 14](#_Toc502750535)

[3.3 实验步骤 16](#_Toc502750540)

[3.4 故障与调试 19](#_Toc502750541)

[3.5 测试与分析 24](#_Toc502750547)

[4 总结与心得 27](#_Toc502750548)

[4.1 实验总结 27](#_Toc502750549)

[4.2 实验心得 28](#_Toc502750553)

[参考文献 29](#_Toc502750554)

# 

# MIPS寄存器文件设计实验

## 设计要求

利用logisim平台构建一个MIPS寄存器组，内部包含32个32位寄存器。考虑到画图工作量，将32个寄存器缩减为4个寄存器，对于寄存器的输入编号，只取低2位作为寄存器的片选信号，忽略高位。

表 1.1 MIPS寄存器组芯片引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| R1# | 输入 | 5 | 读寄存器1编号 |
| R2# | 输入 | 5 | 读寄存器2编号 |
| W# | 输入 | 5 | 写入寄存器编号 |
| Din | 输入 | 32 | 写入数据 |
| WE | 输入 | 1 | 写使能，为1时CLK上跳沿将Din数据写入W#寄存器 |
| CLK | 输入 | 1 | 时钟信号，上跳沿有效 |
| R1 | 输出 | 32 | R1#寄存器的值 |
| R2 | 输出 | 32 | R2#寄存器的值 |
| $s0 | 输出 | 32 | 编号为16的寄存器的值 |
| $s1 | 输出 | 32 | 编号为17的寄存器的值 |
| $s2 | 输出 | 32 | 编号为18的寄存器的值 |
| $ra | 输出 | 32 | 编号为31的寄存器的值 |

寄存器文件要实现的主要功能有：同时读取两个寄存器的值、根据使能信号写入一个寄存器值，而且零号寄存器的值永远为0。其具体功能和引脚定义如表1.1。

## 方案设计

### 总体设计

1. 将始终信号接到所有寄存器的始终端，保证写入、读取等操作都是同步的，不存在异步操作。
2. 将待写入的数据直接连到所有寄存器的输入端，最终是否成功写入则由写使能信号控制。
3. 通过写寄存器编号来决定将写使能信号接到哪一个寄存器，使得该寄存器可以写入数据，其他寄存器的写使能为0，不能写入。
4. 通过读寄存器的编号来选择将四个寄存器的的哪一个进行输出。
5. 通过将0号寄存器的清零端置为恒1，使其值永远为0。

### 输入输出处理

由于实验原本要求的是实现32个寄存器组成的寄存器组，需要5位的片选信号作为输入；而为了方便实现，缩减到了4个寄存器，只需要两位片选信号，所以需要对输入信号进行处理，此外还要用隧道将输入和输出信号引出，在实验中多次用到某个信号时，可以用隧道进行连接，而不用把线全部连到输入输出端。具体实现如下：

1. 对于读寄存器R1#和R2#，写寄存器W#的五位输入，均用分线器分出低两位，分别用隧道reg1#、reg2#和writereg#引出，高三位丢弃。
2. 待写入数据Din、写使能WE、时钟信号clk、寄存器输出reg1和reg2，不需要进行截取和忽略操作，所以都分别用其同名的隧道引出，供后续使用。

### 读寄存器

寄存器是没有读使能信号的，所以它的值会在始终到来时没有选择地读出来，我们的任务是从四个寄存器读出的值当中选择我们想要的作为电路输出。

以读寄存器编号1为例，reg1#为00、01、10和11时分别读取0、1、2和3号寄存器的值输出，由此可以想到使用多路选择器进行数据的选择，reg1#作为控制信号。reg2的也与之类似。

### 写寄存器

要根据写寄存器的编号来决定将写使能信号送到哪个寄存器，我们只有一个写使能信号，最终要分送到多个寄存器；而在前面的读寄存器操作中，我们有多个寄存器输出，只需要选择一个，用到了多路选择器，这两者的选择关系是互逆的，所以我们可以通过多路选择器的反用，也就是解码器来实现。将写使能作为解码器DMX的输入，用写寄存器编号W#作为控制信号，四路输出分别接到四个寄存器的使能端，这就实现了对寄存器写入的控制。

## 实验步骤

1. 输入输出端口处理

对R1#、R2#和W#三个寄存器编号，用分线器取低两位作为寄存器片选信号，高位信号不做处理，直接忽略。

为了便于观察四个寄存器的实时输出，以判断数据是否正确写入和读出，增加四个探测引脚$0、$1、$2、$3，分别显示四个寄存器输出。

1. 连接时钟和寄存器输入

把时钟信号连接到所有寄存器的时钟端口，待写入寄存器的数据writedata连接到所有寄存器的数据端。再把寄存器的输出信号端引出，与debug引脚相连，便于观察输出。

1. 读寄存器

选择MUX多路选择器，设置数据位宽为32位，选择数据的宽度为2位，共控制四路输入信号。0~3号输入分别接寄存器0~3的输出值，选择信号接读寄存器编号的低2位reg1#，输出接读寄存器输出reg1，就是读寄存器1的电路。将上述的选择器复制一份，选择信号接reg2#，输出接reg2，就是读寄存器2的电路。读寄存器电路如图1.2，图中两个多路选择器共用了输入信号，从上到下的四路输入分别为寄存器0到3的输入。

1. 写寄存器

前面已经把待写入寄存器的数据连接到了寄存器的输入端，所以任何时候数据都是到达了寄存器的输入端口的，是否写入取决于使能信号。

选择解复用器DMX，设置数据位宽为32位，控制信号为2位，控制四路输出。写使能信号WE接DMX的输入，控制信号为写寄存器编号writereg#，输出编号为0到3，分别接到0到3号寄存器的使能端。

1. 零号寄存器清零

由于零号寄存器必须保持恒0，所以需要将其清零信号置1，可以用一位常量1接到清零端。至此，电路全部结构连接完成，电路结构如图1.1所示。

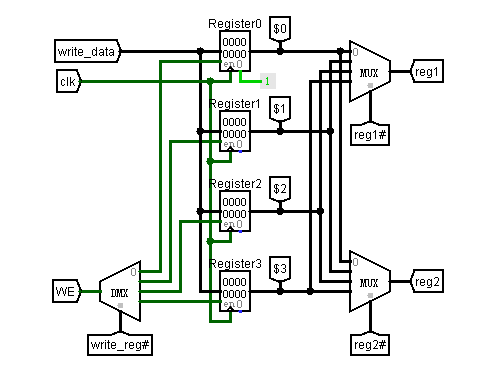


图1.1 寄存器文件总体结构

## 故障与调试

### 写寄存器与读寄存器不能同时进行问题

**故障现象：**将电路提交检查，测试程序自动计分，只得到了62分，说明有部分情况的读寄存器输出错误。

**原因分析：**如图1.1，多路选择器的输出本应该用来当作电路的输出，但是这里对它进行了一定的处理。把写使能信号先取反再进行符号扩展，扩展到了32位，再与多路选择器输出进行与操作，这样的结果是，当写使能为1，也就是写入寄存器数据时，输出全部都被清零了。这个操作是多余的，其实是我对题意理解的错误，我认为写寄存器的时候不能读寄存器，所以在写的时候把输出清零了。

**解决方案：**把图中的与门去掉就是正确的逻辑，多路选择器输出直接作为电路输出，寄存器的写入和读取可以同时进行，而不是写的时候无法读。

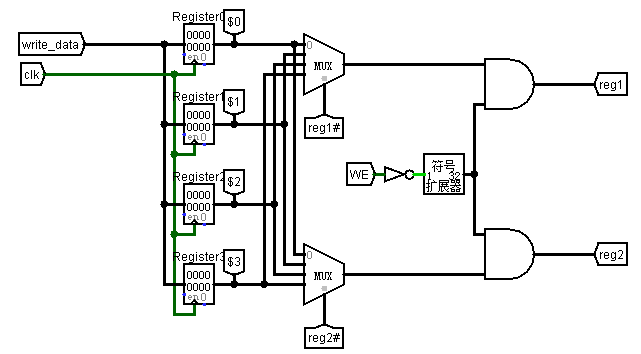


图 1.2 寄存器输出控制

## 测试与分析

测试用例见1.2，各测试用例按照编号顺序执行，测试思路是，先向各个寄存器写数据，然后再从中读出已写的数据。

表 1.2寄存器文件读写测试用例

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | reg1# | reg2# | regw# | WE | Din | reg1 | reg2 |
| 1 | 任意 | 任意 | 00 | 1 | 0x45 | 0 | 0 |
| 2 | 01 | 任意 | 01 | 1 | 0x3c | 0x3c | 0 |
| 3 | 01 | 10 | 10 | 1 | 0x7f | 0x3c | 0x7f |
| 4 | 01 | 11 | 11 | 1 | 0xfa | 0x3c | 0xfa |
| 5 | 00 | 01 | 任意 | 0 | 任意 | 0 | 0x3c |
| 6 | 10 | 11 | 任意 | 0 | 任意 | 0x7f | 0xfa |

**结果分析：**

1. 第一步向寄存器0写数据，结果寄存器的值仍为0，说明寄存器0的值恒为0，不接受任何改变，只可读。
2. 步骤2~4分别向寄存器1、2、3写入数据，写入的成功同时成功读出，验证了寄存器写入数据功能正确。
3. 步骤5和6一次读取两个寄存器的值，共读出了四个寄存器的值，结果与写入的值相同，验证了寄存器读取数据的正确性。

# MIPS RAM设计实验

## 设计要求

利用logisim中的RAM存储器设计完成既能按照 8 位，也能按 16 位，也能按 32 位进行读写访问的 32 位存储器，功能类似于计算机中主存储器，既能按照字节访问也能按照半字访问，还能按照字进行访问，如 MIPS 指令中的 LB/SB 指令（Load/Store byte）、LH/SH 指令（Load/Store Half），LW/SW 指令（Load/Stire Word），X86 指令中 mov eax/ax/ah,[200]。logisim 中 RAM 存储器只能按照一种模式访问，需要对其进行封装改造，最终存储器规格如下：

字节地址12位；数据线宽度32 位；访问Mode：00表示字访问，01表示1字节访问，10表示2字节访问；WE：写使能，1表示写入，0表示读出；Din：32位，写入数据（有效数据均存放在最低位，高位忽略）；Dout：32位，读出数据。

存储器输入输出引脚如表2.1，各访问模式的具体访问方法如表2.2。

表 2.1 芯片引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| addr | 输入 | 12 | 要访问的内存地址 |
| Din | 输入 | 32 | 待写入内存的数据 |
| WE | 输入 | 1 | 写使能信号 |
| mode | 输入 | 2 | 存储访问模式，具体功能见表2.2 |
| clk | 输入 | 1 | 系统时钟信号 |
| Dout | 输出 | 32 | 从内存读取的数据 |

表 2.2 各访问模式功能

|  |  |  |
| --- | --- | --- |
| mode | 访问模式 | 功能 |
| 00 | 字访问 | 忽略低2位，高位作为地址 |
| 01 | 字节访问 | 高位作为地址，低2位片选 |
| 10 | 半字访问 | 高位作为地址，倒数第二位片选，忽略最低位 |

## 方案设计

### 总体设计

1. 输入的地址在不同模式下要按照不同的方式解析，首先要从地址中分出字地址、字节地址和半字地址，还有半字片选和字节片选信号；然后根据访问模式，通过解码器得出字访问、半字访问和字节访问信号。
2. 为了让RAM能够最小按照8位进行访问，我们需要将RAM的数据位宽设置为8，用4个相同的RAM来扩展成为32位，这样就可以按照8、16和32位进行访问。
3. 四块芯片不一定会同时使用到，所以需要用使能信号对芯片进行允许/禁用操作。
4. 不同访问模式下，待写入的数据要送到不同的RAM，应该用mode信号控制解码器来实现写入数据的选择。
5. 不同访问模式下的输出来自不同的RAM的输出的组合，只能选择其中一路，需要用mode控制多路选择器进行输出的选择。

### 生成使能信号

三种访问模式下各自生成四个RAM的使能信号，然后对三组使能信号进行“或”操作，得到总的使能信号。

1. 字访问模式下，四个RAM都要用到，所以四个使能信号都为1；
2. 半字访问模式下，只有两个RAM允许使用，根据半字片选信号来决定选择哪两个，片选信号为0，选择0号和1号RAM，反之选择2和3号RAM；
3. 字节访问模式下，只能有一个RAM被访问，根据字节选择信号来进行选择，被选中的RAM使能信号为1。

### 写入数据选择

待写入RAM的数据是32位的，而单个RAM是8位的，不能直接把数据接到RAM输入端，必须进行分解和选择。首先我们要根据访问模式决定取Din的多少位作为输入，然后再根据访问模式选择其中的一路作为写入的数据。

1. 用mode作为选择信号，通过DMX将Din分成四路。对DMX的输出进行分解送到某几个RAM，每个RAM在每个模式下的数据输入不同，可用或门把这些输入“或”起来。
2. 在进行或操作前先对DMX的四路32位数据进行分解。
3. 第一路是字访问的输入数据，分成四个字节，分送到四个8位的RAM；
4. 第二路是字节访问的输入，只有低字节有效，用分线器分出，送到四个RAM；
5. 第三路是半字访问输入数据，低半字有效，用分线器将其分成两个字节，低字节送到0、2号RAM，高字节送到1、3号RAM。

### 读出数据选择

RAM有八位输出，用这些输出组合成各模式的输出，选择一路作为电路输出。

1. 四个RAM的输出直接用分线器拼接成32位，就是字访问模式的输出；
2. 四个RAM中的信号选择一路，再把高24位补零，就是字节访问模式的输出。由于字节访问模式下只有一个RAM有效，其他三个被禁用，输出为高阻态，所以可以直接把四路RAM的输出连接起来，不用担心信号冲突。
3. 从四个RAM输出中选择两个拼接成16位，再向高16位补零，可以得到半字访问的输出。
4. 用mode作为选择信号，通过一个4路输入的多路选择器从上述三路输出中选出一路。

## 实验步骤

1. 电路框架构建

根据实验方案设计中的思路可以知道电路的大致逻辑结构，首先可以用各类控制信号把RAM组的主体结构搭建好，然后再生成控制信号。

1. 选择4个随机存储器RAM，数据宽度设为8。
2. 将系统时钟连接到四个RAM的时钟端。
3. RAM的使能端cs接分别接四个使能控制信号CS0到CS3，以达到允许/禁用RAM的目的。
4. 地址和访问信号生成
5. 拖出一个译码器，选择信号接访问模式输入mode，第1到3路输出分别接：字访问、字节访问、半字访问，第四路为无效访问，忽略；
6. 将输入的12位字节地址用分线器分出高10位，接到字地址隧道；低2位接到字节选择信号隧道；倒数第二位接到半字选择信号隧道。
7. 使能信号生成

根据访问模式生成各个RAM的使能信号。

1. 使用四个或门，扇入系数都为3，分别接三种访问模式的使能信号。
2. 字访问信号引出四份，连接到每个或门的一个输入上；
3. 字节访问信号不能直连或门，连接到DMX的输入，字节片选信号作为DMX的选择信号，0到3号输出分别接到4个或门上；
4. 半字访问的信号生成稍微复杂些。将半字访问和半字片选信号用与门运算，输出接到2号和3号或门，表示选中了高位的两片RAM；半字访问和半字片选用异或门运算，结果送到0号和1号或门，表示选中了低位的两片RAM。
5. 输入数据选择
6. 拖出四个或门，数据位宽8位，三路输入，分别来自三种选择模式的输入，其结果对应四个RAM的输入；
7. 选择解复用器DMX，数据位宽32位，选择信号位宽2位。输入接Din，选择信号为模式选择mode。第0号输出用分线器分成4个字节接到四个或门；第1号输出用分线器取低8位，送到四个或门；第2号输出用分线器分成第0字节和第1字节，第0字节接到0号和2号或门，第1字节接到1和3号或门。
8. 输出数据选择
9. 选择一个多路选择器，数据位宽32位，选择信号2位，有四路输入，输出就是电路的数据输出；
10. 四片RAM的输出组合成32位接到MUX的0输入，作为字访问输出；
11. 四片RAM的输出经过缓冲器（防止信号直接相连造成冲突），再直接相连，用扩展器将高24位补零，结果送到MUX第1个输入，作为字节访问输出；
12. 0号和1号RAM的输出用分线器组合成16位，2号和3号的输出也用分线器组合成16位，用扩展器将高16位补零，送到MUX的第2路作为半字访问输出。

## 故障与调试

### RAM输出数据不确定问题

**故障现象：**实验提供了MIPS RAM的标准测试程序，将写好的RAM文件用测试程序测试，RAM的输出为红线，表示有信号冲突或者有不确定态。

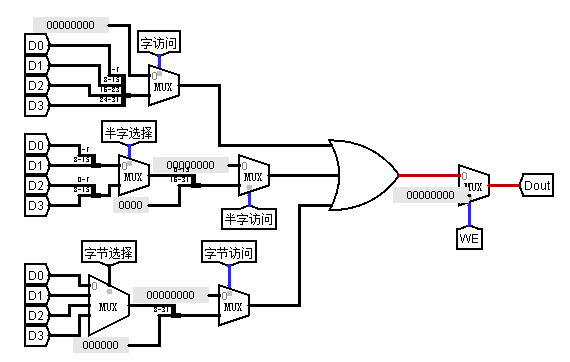


图 2.1 RAM输出冲突

**原因分析：**如图2.1，三路输出信号使用或门进行综合输出，当有效输出为全0，而其他输出项为高阻状态时，或门的结果是不确定态，也就产生了上述的红线。。

**解决方案：**多路选择器可以解决这一问题，将三路输出接到多路选择器的输入端，用mode作选择，MUX的输出就是正确输出。

### 电路结构臃肿问题

**故障现象：**电路功能正确，但是电路总体结构显得臃肿，使用了多达12个多路选择器，成本太高，性能有待提升。

**原因分析：**凡是涉及到多选一的电路，都使用了多路选择器，太过于依赖选择逻辑，而没有用简单的与或非门去实现相同的功能。

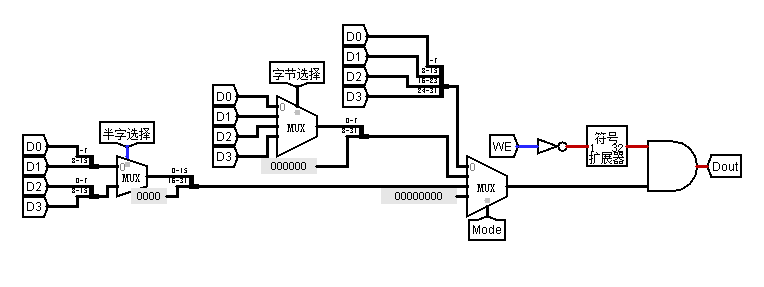
****

图2.7 输出选择逻辑

**解决方案：** 如图2.2的输出选择逻辑，每一路数据输出的生成都用到了MUX，其实是可以直接把数据连接起来的，因为有些数据是高阻态。

## 测试与分析

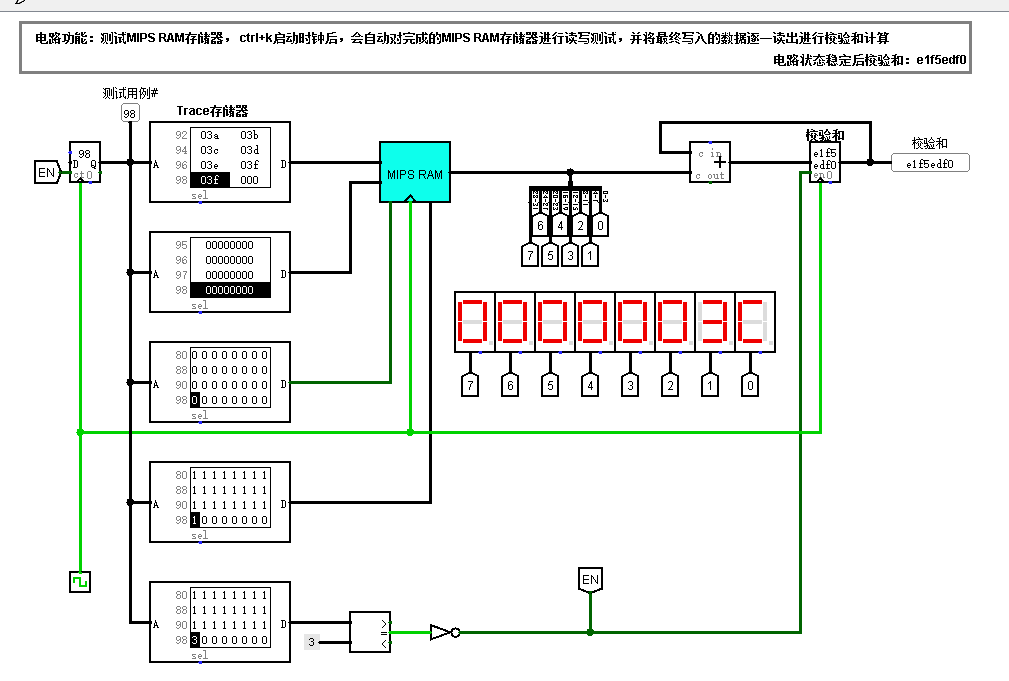


图2.3 MIPS RAM测试结果

实验文件中提供了标准测试文件，如图2.3，程序执行得出的校验和与实验要求中的校验和相同，均为0xe1f5edf0，由此可以得出结论，RAM文件功能正确。

# CPU设计实验

## 设计要求

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中其他功能部件构建一个32位MIPS CPU单周期处理器，该处理器应支持基础指令集中列出的所有指令，见表3.1，另外还必须支持扩展指令集中的2条C类运算指令，1条M类存储指令，1条B类分支指令（详见表3.2）。具体指令功能参见附件中的MIPS标准文档。最终设计完成的CPU应能运行标准测试程序，程序存储在Logisim ROM模块中（指令存储器、数据存储器分开）。

表 3.1 基础指令集

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **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 | Add Immediate Unsigned | addiu $rt, $rs, immediate |
| 4 | Add Unsigned | addu $rd, $rs, $rt |
| 5 | And | and $rd, $rs, $rt |
| 6 | And Immediate | andi $rt, $rs, immediate |
| 7 | Shift Left Logical | sll $rd, $rt, shamt |
| 8 | Shift Right Arithmetic | sra $rd, $rt, shamt |
| 9 | Shift Right Logical | srl $rd, $rt, shamt |
| 10 | Sub | sub $rd, $rs, $rt |
| 11 | Or | or $rd, $rs, $rt |
| 12 | Or Immediate | ori $rt, $rs, immediate |
| 13 | Nor | nor $rd, $rs, $rt |
| 14 | Load Word | lw $rt, offset($rs) |
| 15 | Store Word | sw $rt, offset($rs) |
| 16 | Branch on Equal | beq $rs, $rt, label |
| 17 | Branch on Not Equal | bne $rs, $rt, label |

表 3.1 基础指令集

|  |  |  |  |
| --- | --- | --- | --- |
| # | 指令 | 格式 | 备注 |
| 18 | Set Less Than | slt $rd, $rs, $rt |  |
| 19 | Set Less Than Immediate | slti $rt, $rs, immediate |
| 20 | Set Less Than Unsigned | sltu $rd, $rs, $rt |
| 21 | Jump | j label |
| 22 | Jump and Link | jal label |
| 23 | Jump Register | jr $rs |
| 24 | syscall（display or exit） | syscall | If $v0==10  halt(停机指令)  else  数码管显示$a0值  注意数码管输入数据应该用寄存器锁存 |

表 3.2 运算符功能

|  |  |  |  |
| --- | --- | --- | --- |
| **指令分类** | **指令助记符** | **简单功能描述** | **备注** |
| **运算**  **(C)** | XORI | 异或立即数 | 指令格式参考MIPS32指令集，最终功能以MARS模拟器为准。 |
| SLTIU | 小于立即数置 1(无符号) |
| **存储访问**  **（M）** | LBU | 加载字节(无符号) |
| **跳转**  **(B)** | BLEZ | 小于等于 0 转移 |

## 方案设计

### 系统设计

将系统分成PC寄存器、指令存储器、数据存储器、立即数扩展器、指令解析器、地址转移逻辑、运算器、寄存器文件和操作控制器几大模块：

1. PC寄存器存放当前指令执行位置的地址，程序顺序执行时，PC每执行一条指令加4，当遇到跳转指令时，PC加载目标地址的值，跳到目的地址。
2. 地址转移逻辑实现对PC寄存器的控制，当程序顺序执行时，地址转移逻辑NPC将PC+4的值送到PC的输入端；当程序需要跳转时，NPC将跳转目的地址送到PC输入端。
3. 指令存储器用ROM实现，其中存放了待执行程序的16进制代码。
4. 指令解析器将32位的定长指令解析出操作码、rs、rt、rd等信息。
5. 立即数扩展器将16位和26位的偏移地址扩展成32位。
6. 寄存器文件是32个寄存器组成的，对应mips指令中的32个寄存器。
7. 运算器是运算器实验中实现的ALU。
8. 数据存储器用于存放数据，存储系统实验中实现的RAM可以在这里使用。
9. 操作控制器通过指令的操作码来生成各类控制信号，控制信号决定加载到寄存器、NPC、ALU、数据存储器等部件的数据，使其完成既定的运算。

### 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，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. 遇到无条件分支，直接跳转到目的地址；
3. 遇到有条件分支，跳转到当前地址加上偏移地址；
4. 遇到函数调用返回指令，也是无条件分支，但是它跳转的目的地址保存在31号寄存器里，所以需要从寄存器文件中读取该地址并跳转。

四种可能的跳转，一次只能选择一个值，需要三个控制信号，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. 分支指令中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R型指令中，32位指令中有5位是偏移量，这是移位指令中要用到的，移位运算最终通过alu来完成，要把偏移量扩展成32位。要移的位数都是非负数，用0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，就可以生成电路了。

## 实验步骤

1. 构建指令解析器

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，6-10位为位移量shamt，0-5位为功能码func；取低16位，作为条件转移指令的偏移地址offset；取低26位，作为无条件转移指令的偏移地址。

1. 完成地址转移逻辑
2. PC+4接到第一个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branch。
3. 上一步的选择结果接到第二个MUX的0输入端，26位立即数左移2位，与PC的高4位“或”起来，接到MUX的1输入端，选择信号为jump。
4. 上一步的选择结果接到第三个MUX的0输入端，31号寄存器的值接到1输入端，return作为选择信号。
5. 构建操作控制器

操作控制器的控制信号生成要使用工程化方法。

1. 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3.3，最左侧为指令助记符，第一行为数据项。
2. 输入源合并。计算表3.2中每一个数据项的输入种类，若有多个输入来源，则需要控制信号进行选择，有n个来源时，需要n-1个选择信号（均使用一位选择信号进行二选一）。

表3.3 数据通路表



1. 控制信号综合。列出所有控制信号的产生条件，如表3.3，给出各控制信号逻辑表达式，利用译码电路生成各指令译码信号，然后根据相应逻辑表达式生成控制器，并封装。
2. 根据传入的操作码和功能码，利用比较器和逻辑门，生成各指令的信号，控制信号的生成都依赖这些译码信号。
3. 根据表3.3的控制信号生成表，生成控制信号电路。对于每一个控制信号，都将表项中为1的向用或门“或”起来，输出结果就是控制信号，由于工作量较大，不对每个控制信号的生成步骤详细介绍。

表3.3 cpu控制信号



1. Alu操作码生成。直接在操作控制器中生成操作码。每一类运算都有一个固定的常数作为操作码，可以用三态门来实现这个常数的输出，如图3.5。
2. 访问模式信号生成。在存储系统实验中实现的MIPS RAM文件需要根据2位的访问模式进行字节、半字和字的访问，所以需要在控制器中根据不同的内存操作指令为其生成访问模式位，也用到了三态门。
3. 构建显示模块

当执行syscall指令时，假如2号寄存器的值不为10，则要用数码管显示4号寄存器的值。用一个寄存器存放待显示的值，用一个与门将alu相等的输出和syscall控制信号相“与”，作为寄存器使能信号，寄存器文件的输出作为寄存器的加载数据，输出为待显示的值。

1. 构建指令计数器

运行标准测试程序时需要对指令条数进行计数，所以需要增加计数器模块，该模块进行了封装。

1. 用停机指令的信号作为使能信号，控制总周期数的计数。最后一个周期，也就是停机指令到来时不会计数，所以要让停机指令的信号晚一个周期到来，也就是先计数再停机，用一个寄存器将停机指令锁存起来，可达到这一目的。
2. 用无条件跳转的控制信号作为使能信号，控制无条件跳转的计数器，每当有无条件跳转指令时，计数一次。
3. 用有条件分支语句的控制信号作为有条件分支指令的计数器使能信号，每当遇到有条件分支指令时，不论成功跳转与否，都计数一次。
4. 计数有条件分支指令成功跳转的次数，它需要根据alu的运算结果判断两个操作数是否相等，才能判断是否成功跳转，用该信号作为控制信号。
5. 主要数据通路连接

第一步中只构建了基本的部件，由于还缺少控制信号，所以互相之间的关联还没有建立起来，现在已经生成了所有的控制信号，可以将整个主电路中的所有数据通路连接起来，单周期cpu电路如图3.1所示。

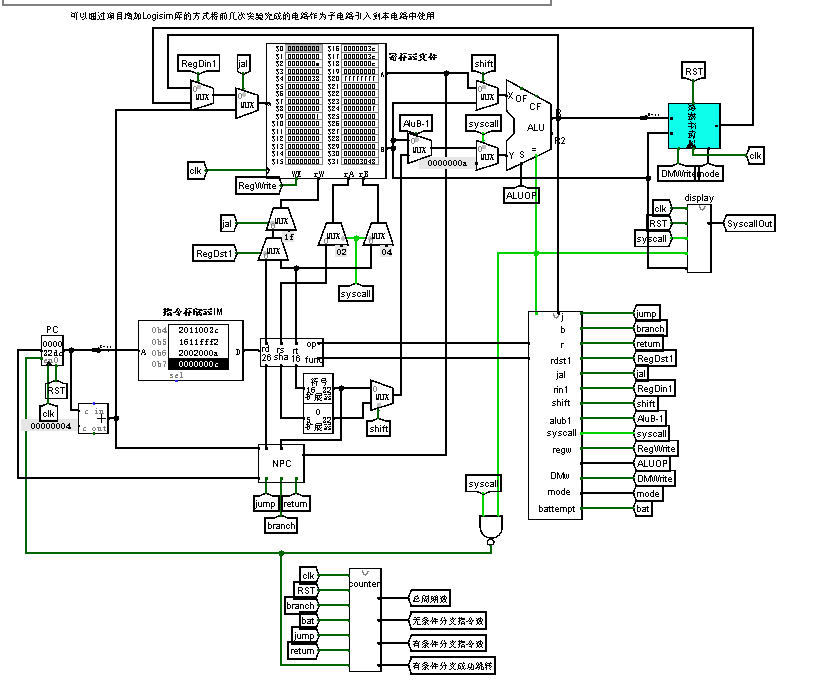


图3.1 单周期cpu全局电路图

## 故障与调试

### B指令测试震荡问题

**故障现象：**初步完成系统后，用B指令测试程序进行测试，遇到第一条syscall指令时，系统发送了明显的震荡，所有与syscall有关的端口都发生了信号冲突。

**原因分析：**如图3.2，一切与syscall有关的线路都是红色的，说明问题一定与syscall指令有关，可能是指令解析出了问题。将带红色，也就是发生震荡的部件逐个右键点击查看，观察震荡发生时，该子电路的内部情况，检查alu时发现，alu的操作码aluop是不确定是值，所以我们可以确定是aluop的生成电路出了问题，也就是操作控制器功能不完备。右键查看操作控制器，发现aluop生成电路中，syscall指令被遗漏了，当syscall指令到来时，控制器没有产生aluop信号，电路无法运算。

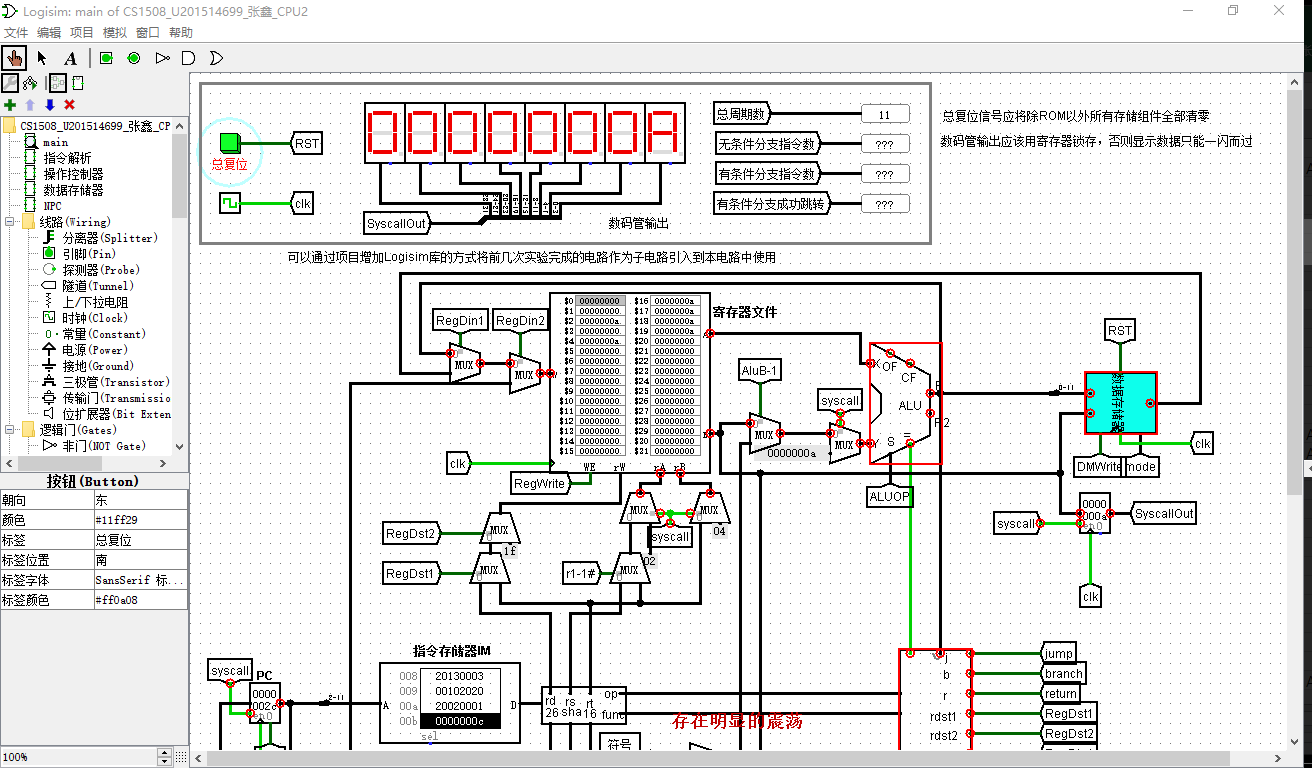


图 3.2 系统测试震荡

**解决方案：**缺少什么就要补充什么，在aluop生成电路中补充syscall的情况，由于syscall指令执行时并不需要alu进行有意义的计算，只要随意给一个操作码让alu不发生故障即可，所以把syscall的aluop码归到了移位指令一项中，为0xc。

### 测试程序提前结束问题

**故障现象：**运行benchmark测试程序，在运行到第391条的一条syscall指令时，2号寄存器的值应当不为10，此时应该用数码管显示4号寄存器的值，但是系统直接停机了，说明4号寄存器的值等于10，如图3.3。

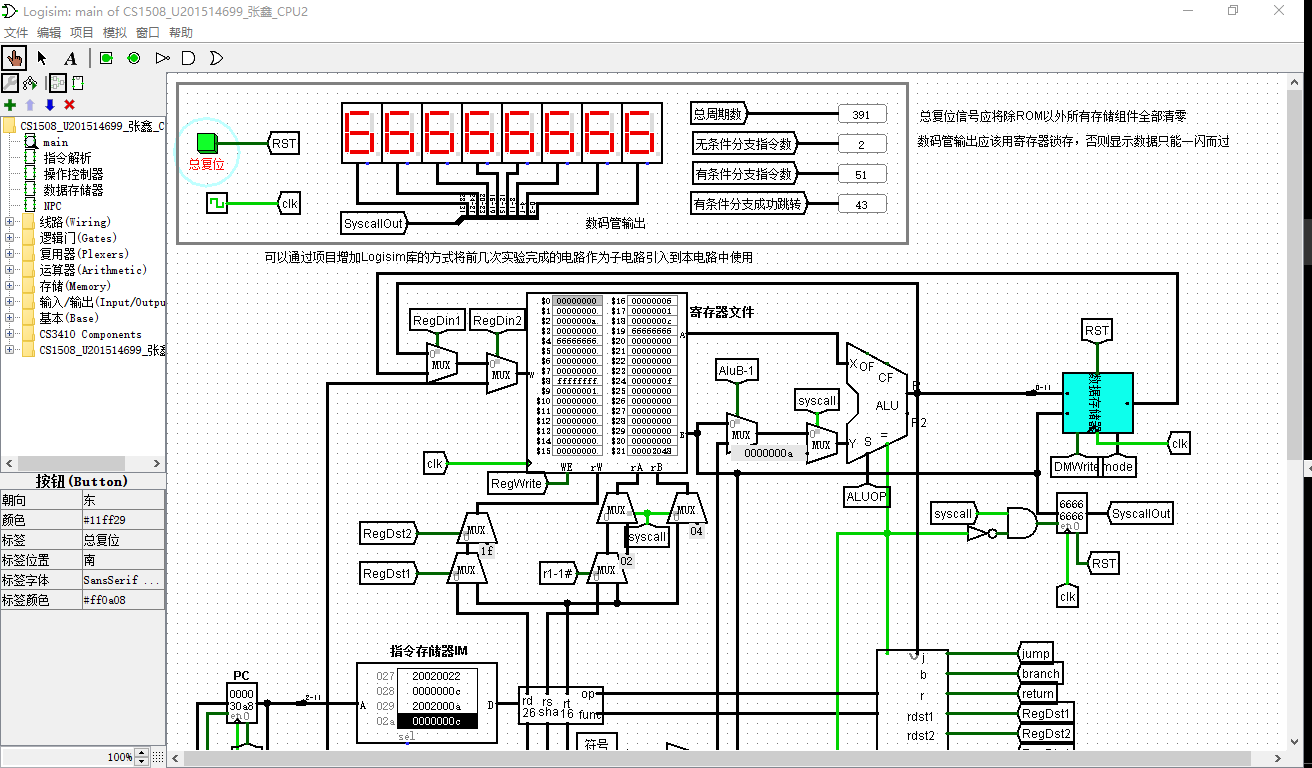


图 3.3 syscall指令提前结束

**原因分析：**虽然程序是在syscall指令处结束的，但是syscall指令并没有问题，而是前面的程序执行过程中写寄存器出现了错误，导致2号寄存器的值被写成了10。

**解决方案：**直接由于程序执行了300多条，所以很难单步执行debug，不妨直接观察写寄存器的多路选择器部分是否选择正确。在观察电路后发现，rs和rt寄存器的编号写到寄存器的r1#和r2#端口时，刚好连反了，将二者对调顺序即可。

### 数据存储器输出不确定问题

**故障现象：**自动运行benchmark测试程序，执行到700到800个周期之间时电路开始有信号冲突，然后冲突逐渐增多，最终整个电路崩溃，如图3.4。这是由一个冲突引发的多个冲突，所以需要找到第一个冲突。从第700个周期开始单步执行，到第718个周期时开始出现信号冲突，数据存储器的访问模式信号mode和数据存储器的输出都为不确定态。

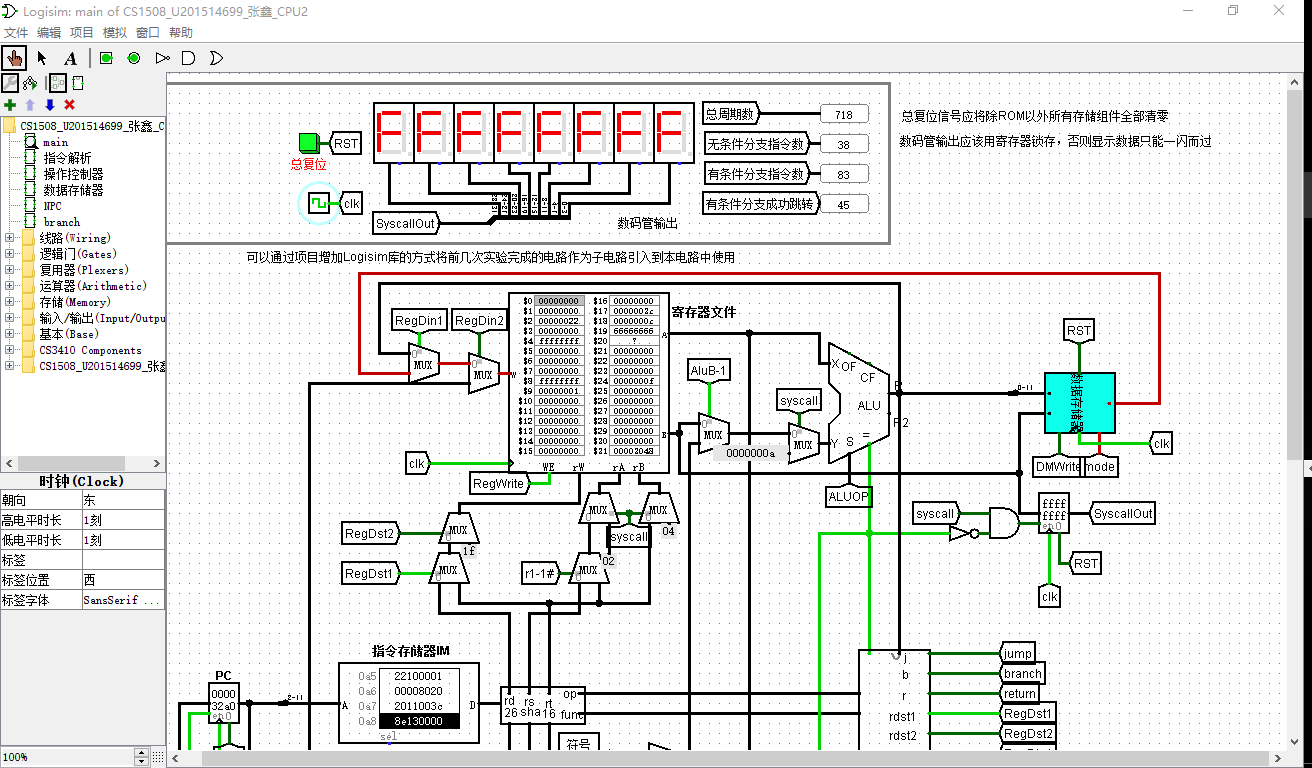


图 3.4 数据存储器输出冲突

**原因分析：**如图3.5，数据存储器的访问模式和数据输出都为不确定态，必定是前者导致了后者，因为访问模式不确定，是不可能得到准确的输出的，所以是访问模式的生成出现了问题，也就是操作控制器的问题。右键查看操作控制器，如图3.13，访问模式信号的生成电路只考虑了加载内存和写入内存指令，对于其他指令，mode信号的输出是高阻状态，所以发生了错误。

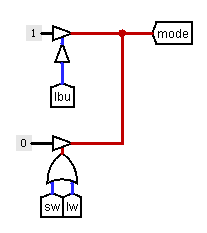


图3.5 mode生成电路

**解决方案：**三态门的缺点造成了输出的不确定态，应该在输出电路上再加一个三态门，对于内存操作指令以外的其他指令，也给出一个明确的的访问模式数值，尽管访问的结果没有意义，但是不会让系统发生冲突。修改后的电路如图3.6。

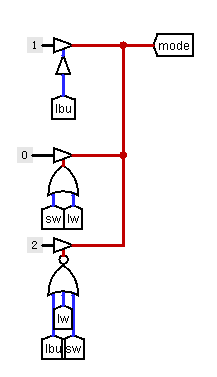


图3.6 修改后的访问模式生成电路

### B指令执行过程信号冲突问题

**故障现象：**执行benchmark测试程序，运行到一条有条件分支指令时发生了信号冲突。

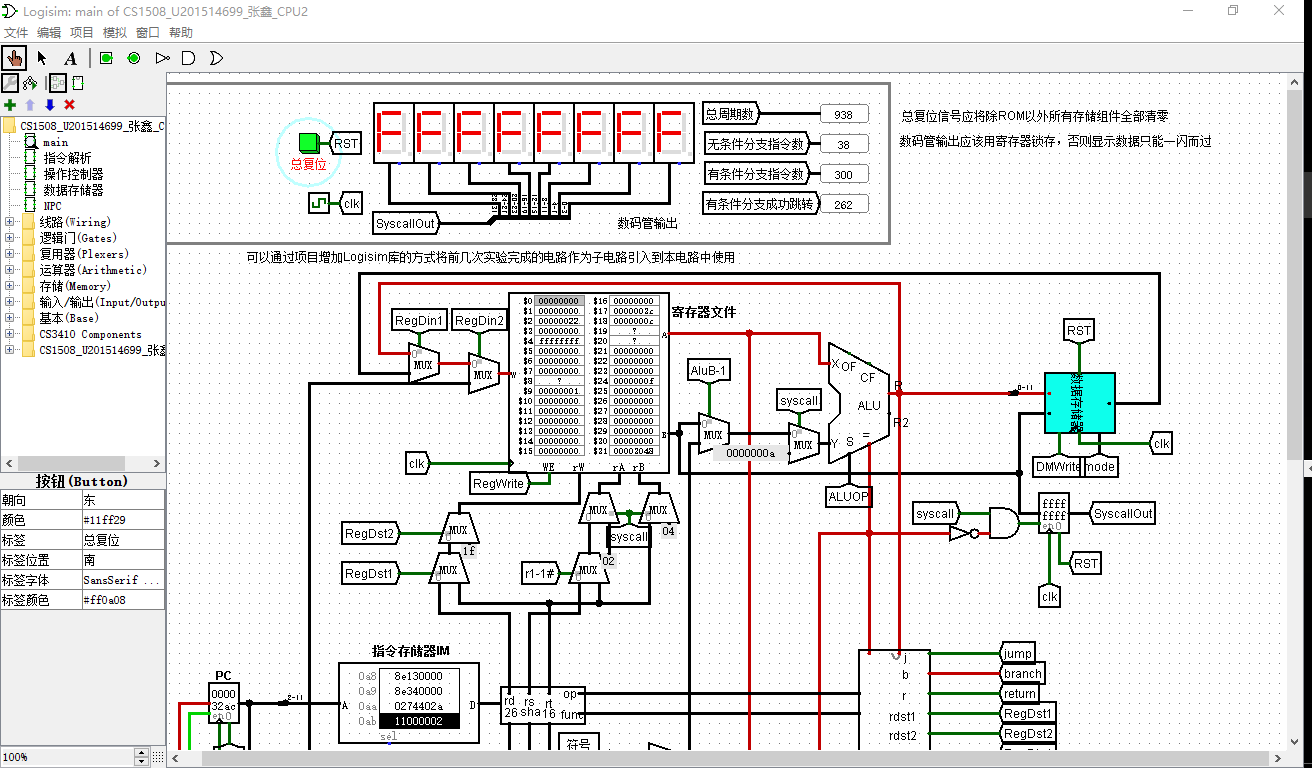


图 3.7 分支指令执行故障

**原因分析：**如图3.7，与分支有关的部件都发生了冲突，分支信号发生了故障。

**解决方案：**branch信号的生成比较复杂，因为它存在着成功跳转和不成功跳转，是否成功跳转需要借助alu的运算结果来判断，操作控制器原则上不允许把alu的运算结果传入，但是这里为了方便生成控制信号，还是将其作为输入传入了。原来的branch信号是手动绘制的电路，存在隐患，而且电路的传播路径长度不一致，采用表达式的方法自动生成电路可以解决这一问题，修改后的电路如图3.8。

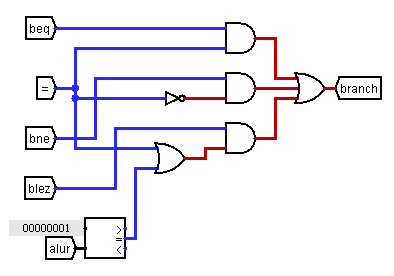


图3.8 有条件分支控制信号生成电路

### ccmb指令测试冲突问题

**故障现象：**测试自己编写的ccmb扩展指令测试程序，中途发生了信号冲突，系统崩溃。

**原因分析：**检查电路未找到原因，再检查ccmb测试程序时发现，xori指令被汇编成了三条指令：lui、ori和xor，如图3.8，这三条指令是精简指令集中没有实现的，所以系统在执行它们时，会得到无法预测的结果，也就崩溃了。至于为何会将一条指令汇编成三条指令，是因为xori指令中使用的立即数过大。

C:\Users\ASUS\Documents\Tencent Files\2452663915\FileRecv\MobileFile\@G8]PYSFI]`PHW)VL50{}EC.png

图3.8 ccmb指令汇编异常

**解决方案：**将测试程序进行修改，改用较小的立即数，就不会发生上述现象。当然也可以扩充实现更多指令，这样就可以执行lui等指令，不过那样太过复杂，超出现阶段的要求。

## 测试与分析

1. 基础指令

运行实验中提供的benchmark标准测试程序，该程序综合了移位、排序、跑马、跳转等等模块，将测试过程数码管的显示情况与标准测试结果样本图进行比较，显示结果一致，程序功能基本正确。最终程序执行指令的计数如图3.9，指令执行的条数也符合预期，所以可以得出结论，单周期CPU的功能达到实验要求。

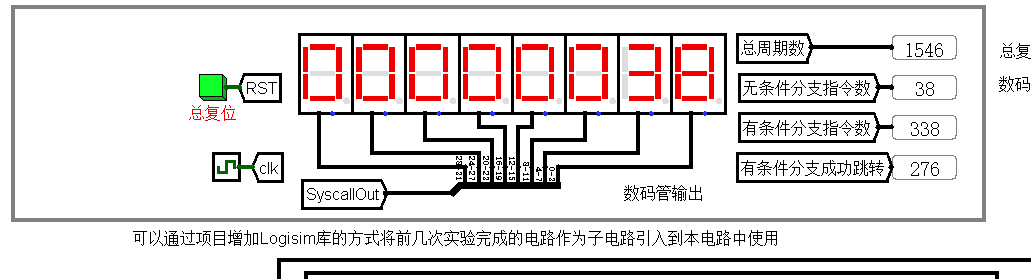


图3.9 benchmark测试程序运行结果

1. CCMB指令

四条扩展指令分别为xori, sltiu, lbu, blez，分四段小程序进行测试，分别如下。

1. xori指令。先将a0寄存器赋初值0xfff，用数码管显示，然后用常量0xedc进行异或操作，就将低4位置为0x123，再用数码管显示。运行结果如图3.10和3.11。

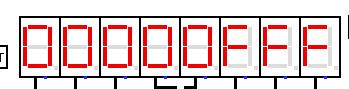


图3.10 xori测试运行结果

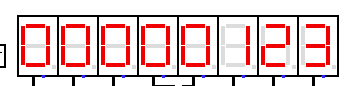


图3.11 xori测试运行结果

1. sltiu指令。设置寄存器s0初值为16，判断它是否小于15，结果不小于，所以a0寄存器被置为0，用数码管显示。将寄存器s0赋值为14，判断它是否小于15，结果小于，所以a0寄存器被置为1，用数码管显示。运行结果如图3.12和3.13。

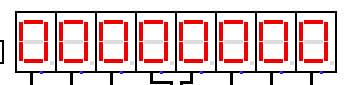


图3.12 sltiu测试运行结果

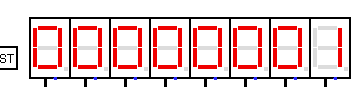


图3.13 sltiu测试运行结果

1. lbu指令。将寄存器a0赋值为0x5678，用数码管显示。用s指令将a0寄存器的值写入4号内存单元，4-7号内存的值分别为0x78，0x56，0x0，0x0。用lbu指令从5号内存单元读取1个字节，其值应为0x56，用数码管显示。运行结果如图3.14和3.15。

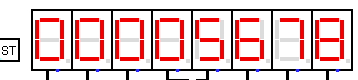


图3.14 lbu测试运行结果

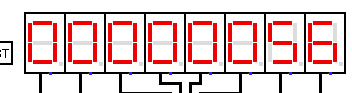


图3.15 lbu测试运行结果

1. blez指令。对a0寄存器赋不同的值，从而执行不同的跳转。先赋值为8，大于0，所以不跳转，继续执行。再赋值为0，满足a0<=0, 跳过几条无关的指令，跳到j\_next处。最后赋值-6，也满足a0<=0，跳转到最后的停机指令。

以上三种跳转满足了所有可能的取值情况，a0大于0，等于0，或小于0。运行结果如图3.16，3.17和3.18。

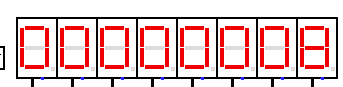


图3.16 blez测试运行结果

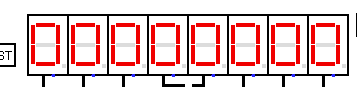


图3.17 blez测试运行结果

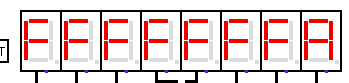


图3.18 blez测试运行结果

# 总结与心得

## 实验总结

### MIPS寄存器文件设计实验

本次实验主要完成了如下几点工作：

1. 完成方案总结：

设计了32位寄存器组，实现了解复用器对写使能信号的分解和多路选择器对于寄存器输出的选择。

1. 功能总结：

实现了通过寄存器编号对任意寄存器进行操作的功能，实现了通过写使能信号向寄存器写入数据，以及通过时钟信号读出寄存器内容。

### MIPS RAM设计实验

1. 完成方案总结：

完成了从字节地址到字地址、字节选择、半字选择的分解，实现了四块RAM存储器的使能控制，通过解复用器实现了电路输入数据到单片RAM的输入数据的分解，通过多路选择器实现了多种模式输出数据的选择。

1. 功能总结：

实现了对8RAM位RAM存储器的扩展，实现了对存储器的多种长度访问：字节访问、半字访问和字访问。

### CPU设计实验

1. 完成方案总结：

实现了地址转移逻辑，实现了32位定长指令的解析，实现了立即数扩展，实现了寄存器内容的显示，实现了程序执行指令数目的记录，实现了寄存器读写，实现了数据存储器的读写，实现了操作数的计算，实现了控制信号的生成，实现了各部件多输入来源的多选一。

1. 功能总结：

完成了单周期CPU，实现了MIPS精简指令集的运行，也实现了4条扩展指令，支持标准测试程序的运行，完成了跑马灯、分支、排序等程序的结果演示。

1. 其他需要总结的内容：

学会了使用工程化方法生成电路，了解了logisim常用debug方法。

## 实验心得

1. 通过数据表示实验了解了汉字区位码和国标码的概念，熟练掌握了海明码的检错纠错原理。
2. 运算器实验中从易到难地实现各类加法器和乘法器，在对比中理解了各种进位方式之间的区别与优劣，也懂得了CPU中数据运算的实质。
3. 存储系统实验主要在于分线器和选择器的灵活使用，学会了如何精简电路，降低成本。
4. 通过CPU实验，首先从宏观上掌握了CPU的运行机理和模块划分，而且还学会了从微观上准确地生成每一个控制信号。
5. 实验最大的感受是对于课程内容理解的加深。在上完课以后往往没有真的学懂知识，在做实验的过程中，自然就会主动地复习上课讲过的内容，而且必须真的弄懂才能完成实验。
6. 即使完成了实验任务书的要求，但是仍有许多未弄懂的问题，实验电路中也可能隐藏着某些未知的漏洞。功能正常的电路，其结构有可能不是最优的，还有优化空间。CPU运行过程中仍然会出现部分不稳定态，尽管已经通过修改电路减少了一些不稳定情况，但是没有彻底消除，这些都是本次实验中做得不够好的地方，以后的学习中仍需要加强。
7. 建议实验难度进行调整，前面的实验过于简单，很多部分都是选做，没有硬性要求，可以改成必做题；而CPU设计实验的难度相比前三次又过难，应该提供更多的指导，这样，四次实验的难度就更均衡。
8. 建议老师在就实验过程中同学反馈的关键点在课程学习群中发布公告，或者以文档形式实时更新，供其他同学参考。

# 参考文献

1. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第4版).北京：机械工业出版社.
2. David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社
3. 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011年.
4. 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011年.
5. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008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:** |
| 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: 2018-01-07** |