

**课程设计报告书**

|  |  |  |
| --- | --- | --- |
| **结果演示成绩60** | **报告成绩40** | **总成绩** |
|  |  |  |

**课程名称 计算机组成原理实验**

**院系名称 计算机科学与工程学院**

**学 号 11703990535**

**姓 名 陈松**

**任课教师 陈媛/石美凤**

**时 间 2019.6**

课程设计报告

**一、课程设计题目**

基于Logisim 软件仿真平台的16位单周期CPU设计。

**二、课程设计的目的与意义**

本课程设计是学完计算机组成原理课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有自定义指令系统的简单计算机系统。所设计的系统能在Logisim仿真平台上或基于FPGA的实验平台上运行一段程序，通过检查程序结果的正确性来判断所设计计算机系统正确性。

本课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行主机系统底层电路的实现、故障分析与定位、系统调式等环节的锻炼，进一步提高学生分析和解决问题的能力。

**二、课程设计的内容**

设计单周期CPU，并调试通过。题目可以根据自己的设计内容、实现方式、所设计的计算机系统的结构从下列备选题中选取：

（1）．基于Logisim软件仿真平台的单周期CPU设计实现。

（2）．基于FPGA的单周期CPU设计与实现。

1.平台介绍：

Logisim是一款数字电路模拟的教育软件，每一位用户都可以通过它来学习如何创建逻辑电路，方便简单。 它是一款基于Java的应用程序，可运行在任何支持JAVA环境的平台，方便学生来学习设计和模仿数字逻辑电路。Logisim中的主要组成部分之一就在于设计并以图示来显示CPU。当然Logisim中还有其他多种组合分析模型来对你进行帮助，如转换电路，表达式，布尔型和真值表等等。同时还可以重新利用小规模的电路来作为大型电路的一部分。

2.单周期CPU的主要技术指标：

1.该单周期CPU实现的指令如下：

表1指令格式

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | 指令 | **15~12** | **11~10** | **9~8** | **7~6** | **5~3** | **2~0** | 指令功能 |
| **1** | **or** | 0 | rs | rt | rd | 0 | 0 | $rd = $rs | $rt（5~3位无用） |
| **2** | **and** | 0 | rs | rt | rd | 0 | 1 | $rd = $rs & $rt |
| **3** | **add** | 0 | rs | rt | rd | 0 | 2 | $rd = $rs + $rt |
| **4** | **sub** | 0 | rs | rt | rd | 0 | 3 | $rd = $rs - $rt |
| **5** | **sllv** | 0 | rs | rt | rd | 0 | 4 | $rd = $rs << $rt |
| **6** | **srlv** | 0 | rs | rt | rd | 0 | 5 | $rd = $rs >> $rt |
| **7** | **srav** | 0 | rs | rt | rd | 0 | 6 | $rd = $rs >> $rt 算术右移 |
| **8** | **slt** | 0 | rs | rt | rd | 0 | 7 | $rd = ($rs < $rt) ? 1 :0 |
| **9** | **DISP** | 1 | rs | rt | immediate-u | | | DISP[imm] = $rs |
| **10** | **lui** | 2 | 0 | rt | immediate-u | | | $rt = imm << 8 |
| **11** | **ori** | 3 | rs | rt | immediate-u | | | $rt = $rs | imm |
| **12** | **andi** | 4 | rs | rt | immediate-u | | | $rt = $rs & imm |
| **13** | **addi** | 5 | rs | rt | **immediate-s** | | | $rt = $rs + imm |
| **14** | **lw** | 6 | rs | rt | **immediate-s** | | | $rt = MEM[$rs + imm] |
| **15** | **sw** | 7 | rs | rt | **immediate-s** | | | MEM[$rs+imm] = $rt |
| **16** | **beq** | 8 | rs | rt | **offset-s** | | | beq =？ |
| **17** | **bne** | 9 | rs | rt | **offset-s** | | | bne != ？ |
| **18** | **bgt** | 10 | rs | rt | **offset-s** | | | bgt >?（有符号比较） |
| **19** | **jump** | 11 | jump address | | | | | jump |
| **20** | **halt** | 12 | 0 | | | | | halt (时钟暂停) |

2．能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖所有指令，程序执行功能正确。

**四、课程设计原理及内容**

**4.1 构造 R 型指令**

R 型指令包括指令表中的前 8 条指令，其指令格式和实现的功能如表 2 所示。

表 2 R 型指令格式表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | 指令 | **15~12** | **11~10** | **9~8** | **7~6** | **5~3** | **2~0** | 指令功能 |
| **1** | **or** | 0 | rs | rt | rd | 0 | 0 | $rd = $rs | $rt（5~3位无用） |
| **2** | **and** | 0 | rs | rt | rd | 0 | 1 | $rd = $rs & $rt |
| **3** | **add** | 0 | rs | rt | rd | 0 | 2 | $rd = $rs + $rt |
| **4** | **sub** | 0 | rs | rt | rd | 0 | 3 | $rd = $rs - $rt |
| **5** | **sllv** | 0 | rs | rt | rd | 0 | 4 | $rd = $rs << $rt |
| **6** | **srlv** | 0 | rs | rt | rd | 0 | 5 | $rd = $rs >> $rt |
| **7** | **srav** | 0 | rs | rt | rd | 0 | 6 | $rd = $rs >> $rt 算术右移 |
| **8** | **slt** | 0 | rs | rt | rd | 0 | 7 | $rd = ($rs < $rt) ? 1 :0 |

这八条 R 型指令的 OP字段和 3~5位均为 0，指令功能由指令的 funct 字段（0~2）

决定，rs（10~11）表示第一源操作数的寄存器号， rt（8~9）表示第二源操作数的寄

存器号，rd（6~7）表示目的操作数的寄存器号。 R 型指令主要负责与、或、加、减、

移位等基本的算术逻辑运算，后面 I 型指令涉及到的算术逻辑运算也由这八条 R 型

指令功能构成的 ALU 执行。

**4.2 构造 I 型指令**

I 型指令是立即数型指令，主要包括一些数据传输、分支、立即数指令，其指 令格式和实现的功能如表 3 所示。

表 3 I 型指令格式表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | 指令 | **15~12** | **11~10** | **9~8** | **7~6** | **5~3** | **2~0** | 指令功能 |
| **10** | **lui** | 2 | 0 | rt | immediate-u | | | $rt = imm << 8 |
| **11** | **ori** | 3 | rs | rt | immediate-u | | | $rt = $rs | imm |
| **12** | **andi** | 4 | rs | rt | immediate-u | | | $rt = $rs & imm |
| **13** | **addi** | 5 | rs | rt | **immediate-s** | | | $rt = $rs + imm |
| **14** | **lw** | 6 | rs | rt | **immediate-s** | | | $rt = MEM[$rs + imm] |
| **15** | **sw** | 7 | rs | rt | **immediate-s** | | | MEM[$rs+imm] = $rt |
| **16** | **beq** | 8 | rs | rt | **offset-s** | | | beq =？ |
| **17** | **bne** | 9 | rs | rt | **offset-s** | | | bne != ？ |
| **18** | **bgt** | 10 | rs | rt | **offset-s** | | | bgt >?（有符号比较） |

这些 I 型指令的功能由每条指令的 OP 字段决定。涉及到基本算术逻辑运算的

操作由 ALU 完成。指令的 0~7 位为立即数或者分支偏移量。若是双目运算，则将

rs 的内容和立即数字段的值分别作为第一操作数和第二操作数，结果送 rt 指定的寄

存器中；若是 Load/Store指令，则将 rs的内容和立即数字段的值经符号位扩展后的

内容相加作为 Load/Store的存储单元地址； 若是分支指令， 则对 rs 和 rt 寄存器中的

数据进行操作码规定的运算，并根据运算的结果决定是否转移。当转移发生时，转移的目标地址采用相对寻址方式获得，即将 PC 的内容和立即数字段经符号位扩展

后的内容相加得到。

**4.3 构造 J 型指令**

J 型指令的格式如表 4 所示。

表 4 J 型指令格式表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | 指令 | **15~12** | **11~10** | **9~8** | **7~6** | **5~3** | **2~0** | 指令功能 |
| **19** | **jump** | 11 | jump address | | | | | jump |

J 型指令主要是无条件转移指令，指令中给出的是 OP 字段和 12 位（0~11）直 接地址，无条件转移的目标地址由 PC 高 4 位与 12 位直接地址作为低 12 位拼接而

得到的。

**4.4 DISP 指令**

DISP 指令的格式及其功能如表 5 所示。

表 5 DISP 指令格式表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | 指令 | **15~12** | **11~10** | **9~8** | **7~6** | **5~3** | **2~0** | 指令功能 |
| **9** | **DISP** | 1 | rs | rt | immediate-u | | | DISP[imm] = $rs |

DISP 指令的立即数字段 immediate 只有 0、1 两个取值，用来选择在 0 号显示

器显示还是在 1 号显示器显示；待显示的内容为 rs对应的寄存器中的数据。

**4.5 halt 指令**

halt 指令格式如表 6 所示。

表 6 halt 指令格式表

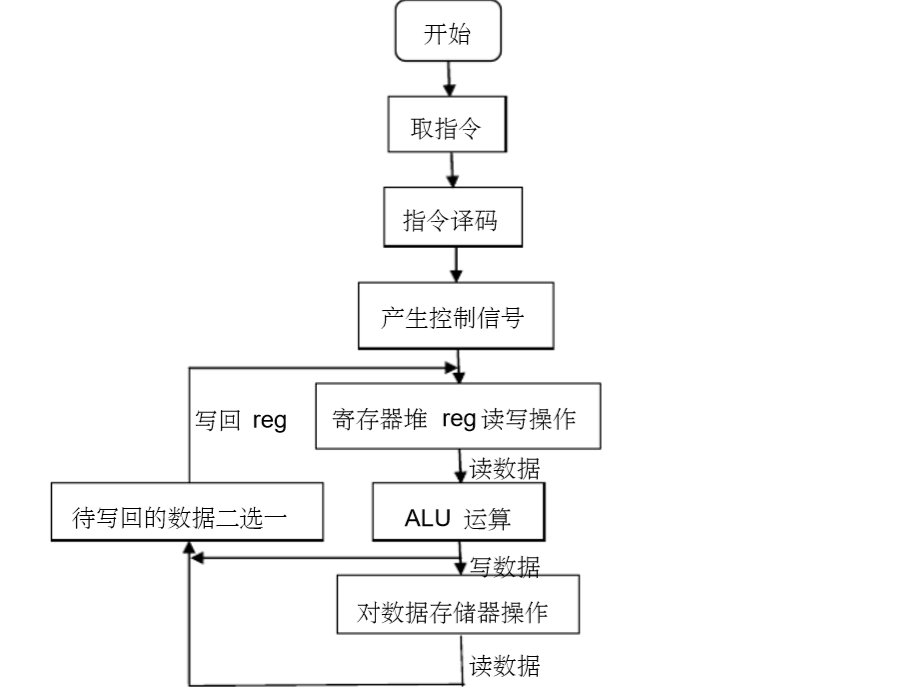
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | 指令 | **15~12** | **11~10** | **9~8** | **7~6** | **5~3** | **2~0** | 指令功能 |
| **20** | **halt** | 12 | 0 | | | | | halt (时钟暂停) |

halt 指令实现“停机”功能，具体实现方案是： 时钟正常跳转， PC 值不再更新，

因此取出的指令一直是相同的，达到“停机”的目的。

**五、总体设计结构图**

5.1单周期设计流程图



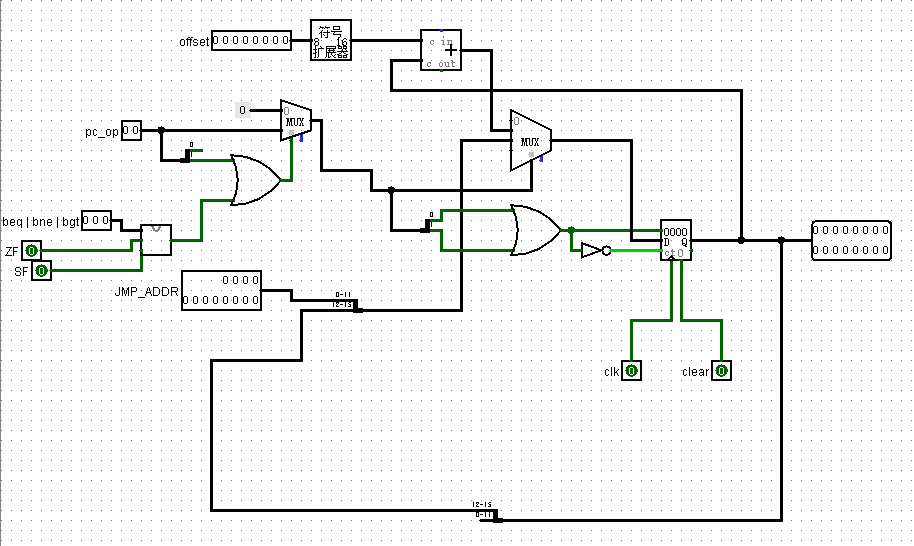
**六、实验过程和步骤**

完整的单周期 CPU 电路由程序计数器 PC、指令存储器 ROM、指令译码器、中央控制器、寄存器堆 Reg、运算器 ALU 、数据存储器 RAM组成。具体设计思路如下:

**6.1程序计数器 PC**

6.1.1 PC模块电路设计如图1所示。

图1 PC模块电路



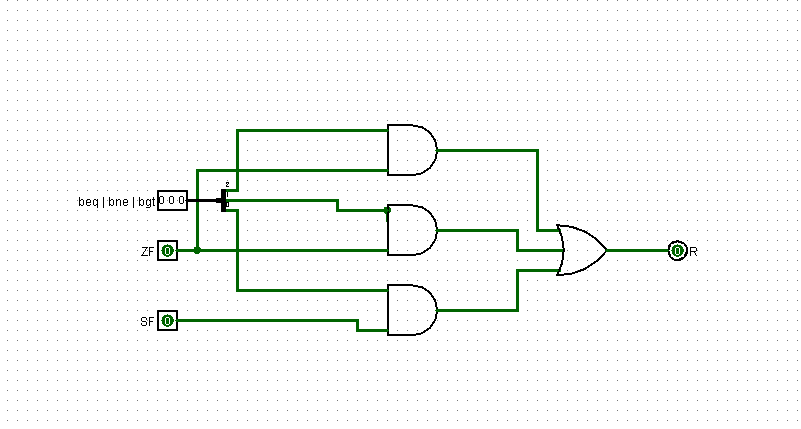
程序计数器的输出 PC有三种选择：当前 PC、jump 指令的目的地址、bne （beq、

bgt）指令的目的地址。这三个地址作为一个数据选择器 MUX 的输入端，数据选择

端信号来自中央控制器产生的控制信号 PC\_OP，数据选择器的输出端接一个计数寄存器存放 PC，然后将寄存器的输出端输出的数据作为指令的地址。

6.1.2偏移满足电路设计图2所示。

图2 判断偏移条件是否满足电路



输入端为由中央控制器产生的分支指令信号，控制ALU进行减法运算产生的0标志位ZF，正数标志位SF.

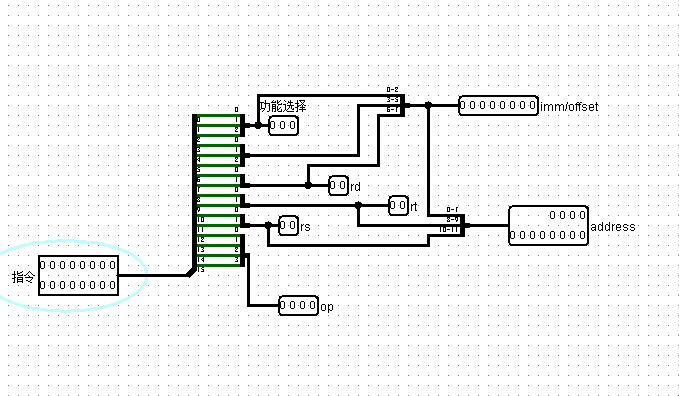
**6.2指令存储器 ROM**

指令存储器 ROM 只由一个地址位宽和数据位宽均为 16的 ROM 组成。

**6.3分离译码器**

6.3.1分离指令译码器电路如图3所示。

图3 分离译码器

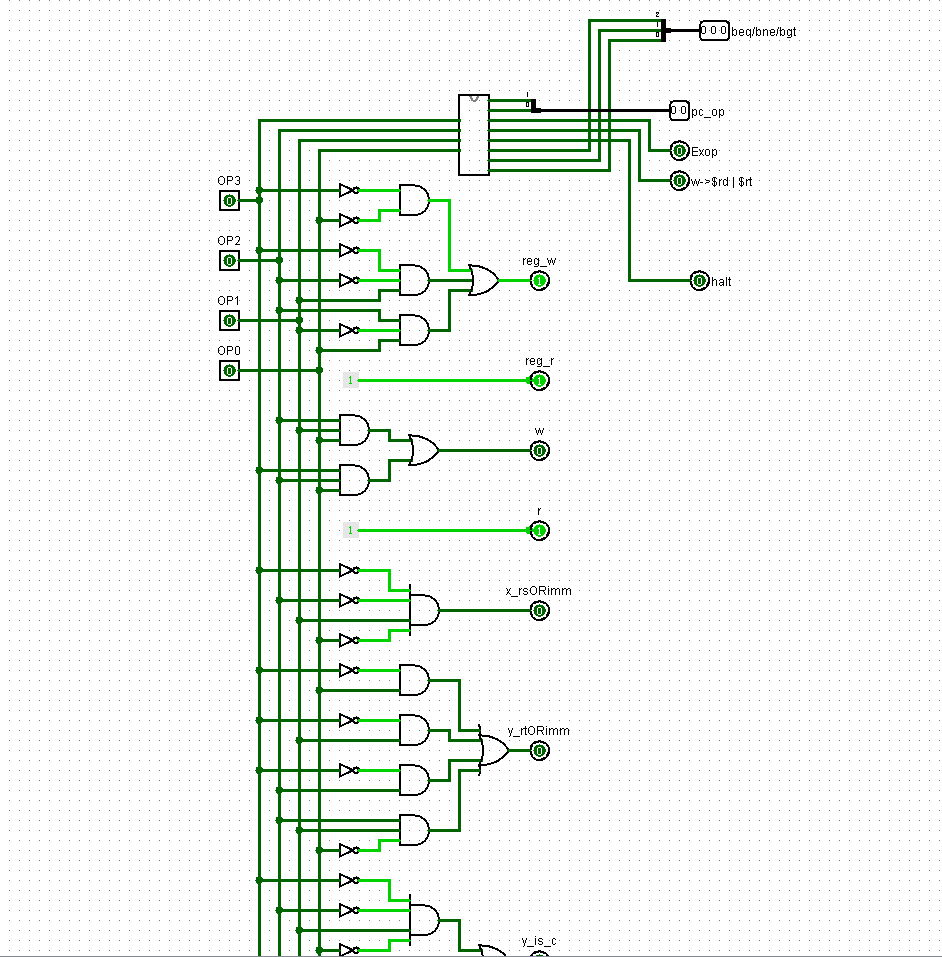


分离指令译码器的输入是从 ROM 中取出的指令，使用一个分离器将指令分成 16位，再根据指令格式表的要求将对应的 bit 位合成为 rs、rt、rd、offset 等。

**6.4 中央控制器**

6.4.1中央控制器设计电路如图4所示。

图4中央控制器



控制中心的输入是每条指令的 OP字段和 funct 字段。由于 R 型指令的操作选择

字段是 funct，I 型和 J型指令的操作选择字段是 OP，为了节省硬件成本和一些意想

不到的错误， 所以想方设法产生一个字段可以对 R、I、J三种指令的操作作出选择。

此外，以每条指令的 OP 字段作为解码器的选择端，这样当执行某条指令时， OP 字

段便会选择解码器的对应输出端为 1，其余输出端为 0；通过观察分许 20 条基本指

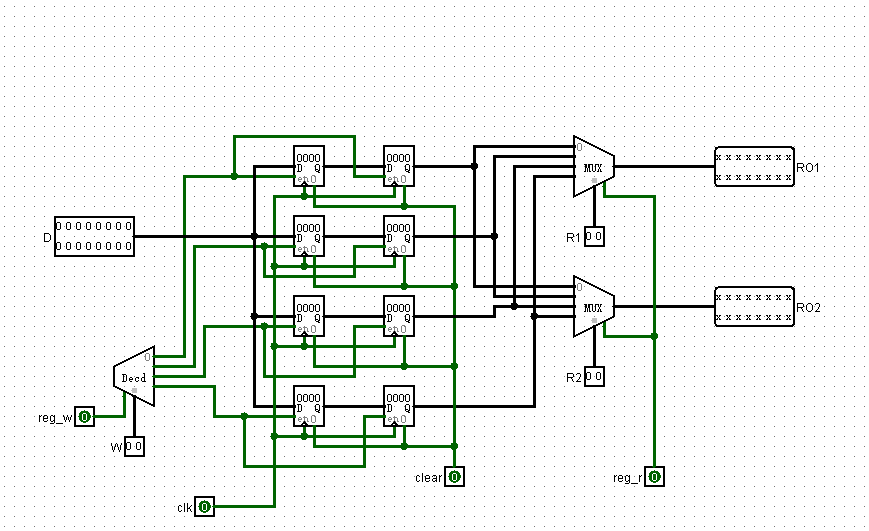
令的特点，将解码器输出端的“ 0”、“1”信号通过适当的逻辑联系在一起生成新的

控制信号，用于控制后续模块的正常工作。

**6.5寄存器堆 Reg**

6.5.1寄存器堆 Reg电路设计如图5所示。

图5 寄存器堆



寄存器堆 Reg由 8 个寄存器组成，第二列的 4 个寄存器主要是为了防止电路震

荡，因为在单周期 CPU 中，同一时刻对于同一个寄存器的读写操作可能会造成冲突。

涉及到寄存器堆的读操作， 需要考虑是要读哪个寄存器。 需要设置两个位宽为 2.

的字段 R1（即 rs）、R2（rt）分别选择读出哪两个寄存器的值作为 ALU 的两个源操

作数，同时需要设置一个使能端控制位 reg\_r，该控制位由控制中心 产

生。

涉及到寄存器堆的写操作，需要考虑是要写入哪个寄存器。需要设置一个位宽

为 2 的字段 W（即 rt 或者 rd，因指令不同而不同）来选择向哪个寄存器写入数据。

以 W 作为一个解码器的选择端， 选择对应输出端口输出“1”，其余输出端口输出“0”，

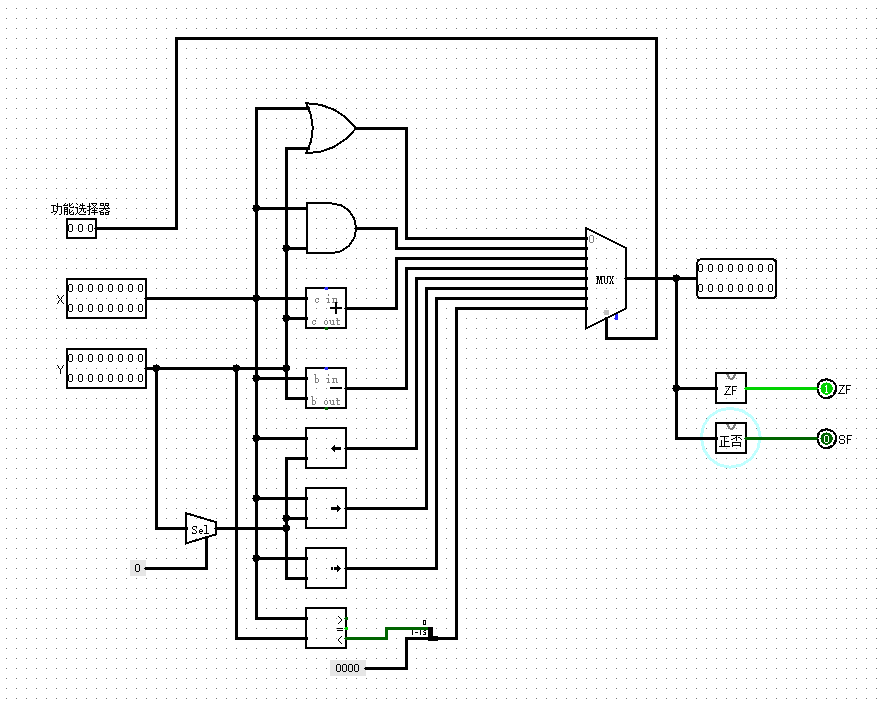
用这四个输出分别控制四个寄存器的使能端；同时，解码器也需要一个使能控制位

reg\_w，否则每条指令都有权限向寄存器中写入数据，该控制位也由控制中心产生。

**6.6运算器 ALU**

6.6.1 ALU电路设计如图6所示。

图6 ALU电路图



运算器 ALU 主要执行算术逻辑运算。运算器有两个输入一个输出，分别表示

两个源操作数和一个目的操作数。运算部件由与门、或门、加法器、减法器、移位

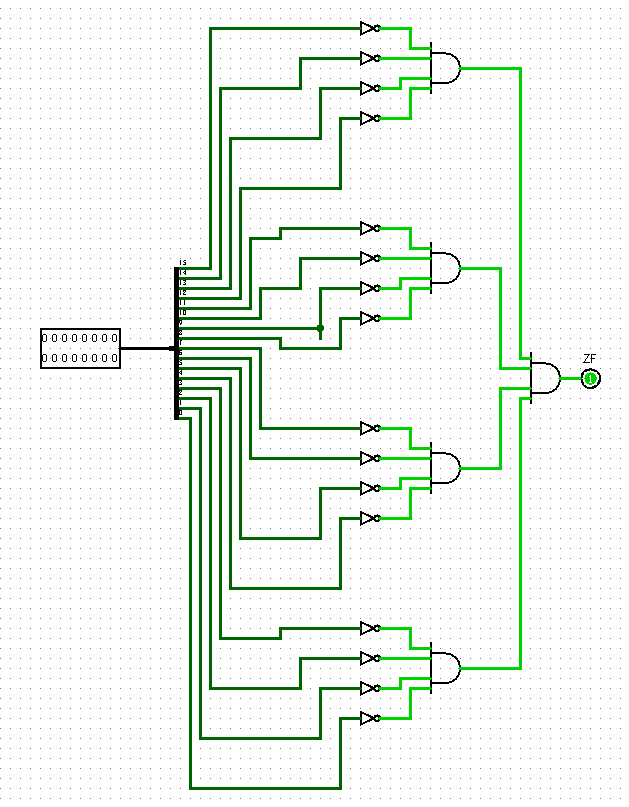
器（逻辑左移、逻辑右移、算术右移） 、比较器组成。两个源操作数均经过这些运算

器运算后得到八个结果，然后用一个数据选择器从中选择一个结果作为输出，数据

选择器的数据选择端为 ALU\_OP，由控制中心产生。

6.6.2 ZF电路模块，检查ALU运算结果是否为0，设计如图7所示:

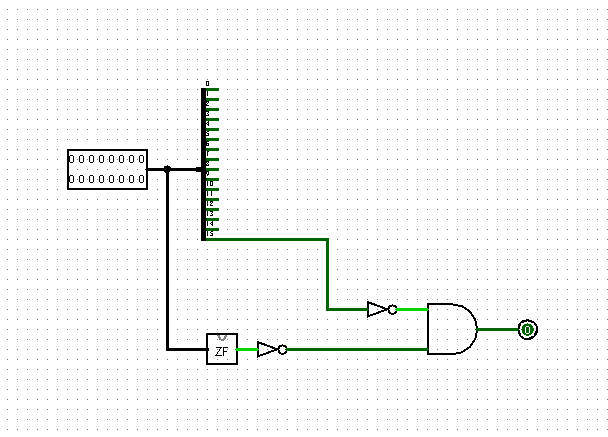
图7 ZF电路



输入端为ALU计算的结果，如果值为0，输出为1，否则输出为0。

6.6.3 SF, 检查ALU运算结果是否大于0,电路设计如图8所示。

图8 SF正标志电路



将输入值接入分离离器，和0标志电路ZF中，将分离器的最高位取反和ZF的输出值取反进行and操作，即输入值为正数，输出值便为1，否则为0。

6.7数据存储器 RAM

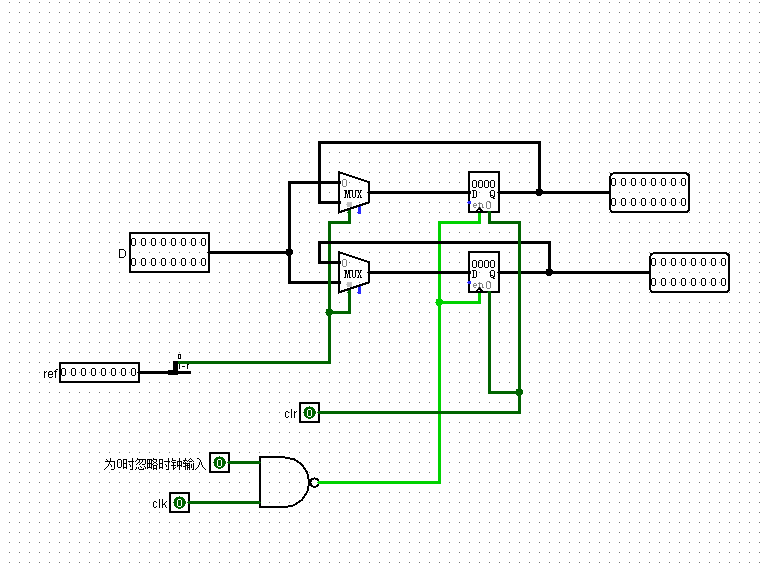
数据选择器由一块 RAM 芯片组成，地址位宽和数据位宽均为 16，采用分离的

加载和存储数据接口。

6.8数据显示器DISP

6.8.1 DISP电路设计如图9所示

图9 DISP

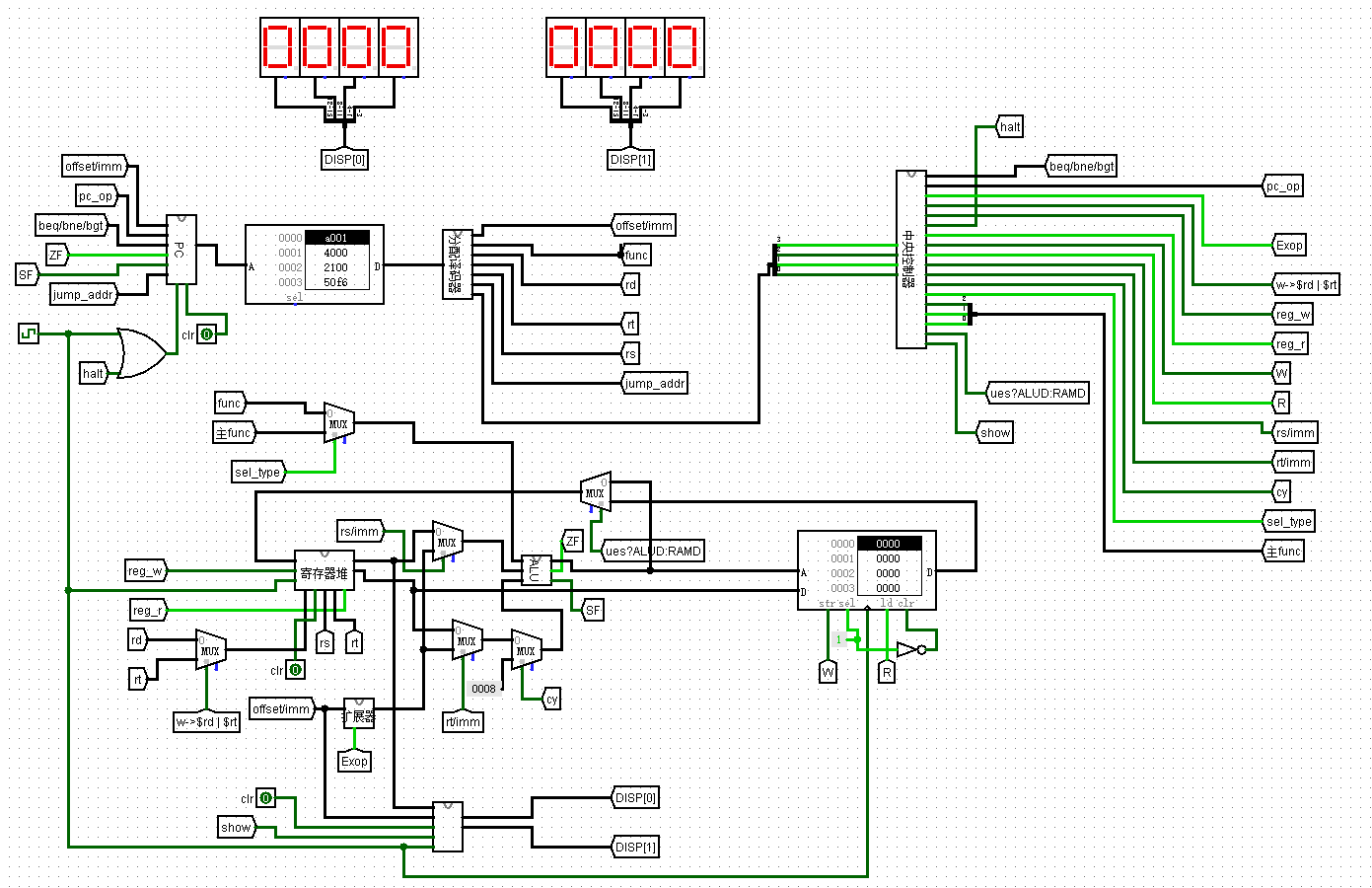


输入端为写入的数据，立即数imm(ref:显示器选择位)，使能端（为0时不写入数据），脉冲信号接受端，显示器的值的改变，通过立即数对数据选择器的选择，确定将那个寄存器的值改变。

6.9主电路设计

6.9.1 主电路，按功能连接上述电路，设计如图10所示。

图10 主电路



**七、实验结果与分析**

7.1实验结果和分析（采样表格形式）

7.1.1数据存储器地址及内容

表3 ROM初始值

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **存储单元** | **存储内容** | | | | | | | |
| 0000 | a001 | 4000 | 2100 | 50f6 | 1000 | 3503 | 1501 | 0180 |
| 0008 | 7e00 | 0181 | 7e01 | 0182 | 7e02 | 0183 | 7e03 | 0184 |
| 0010 | 7e04 | 0185 | 7e05 | 0186 | 7e06 | 0187 | 7e07 | 5505 |
| 0018 | 1c00 | 6c00 | 1001 | 5f01 | 8d02 | b018 | 9dfa | c000 |

表4 RAM 初始值

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 存储单元 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 |
| 存储内容 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

7.1.2程序清单以及功能说明

|  |  |  |
| --- | --- | --- |
| 汇编指令 | 机器码（16进制） | 功能说明 |
| Bgt R00 R00 00 | a001 | If($R00> $R00) then pc+1+0 else pc+1;即无意义，为空指令 |
| Andi R00 R00 00 | 4000 | $R00 = $R00 $ 0000;即将编号00的寄存器清零 |
| Lui R00 R01 00 | 2100 | $R01 = 0000<<8; 即将编号01的寄存器清零 |
| Addi R00 R00 f6 | 50f6 | $R00 = $R00 +(-10);即将编号00的寄存器赋值-10 |
| DISP R00 R00 00 | 1000 | DISP[0]=$rs;即将编号00的寄存器的值[-10]取出并在0号显示器显示[fff6] |
| Ori R01 R01 03 | 3503 | $R01=$R01 | 0003; 即将编号01的寄存器赋值3 |
| DISP R01 R01 00 | 1501 | DISP[1]=$rs;即将编号01的寄存器的值[3]取出并在1号显示器显示[0003] |
| Or R00 R01 R10 | 0180 | $R10 = $R00 | $R01;即$R10 = fff6 | 0003 =fff7 |
| Sw R11 R10 00 | 7e00 | MEM[$R11+0000]=$R10;即将编号10的寄存器的值[fff7]放到RAM存储器中地址为0000($R11=0000)的单元中 |
| And R00 R01 R10 | 0181 | $R10 = $R00 $ $R01;即$R10 = fff6 $ 0003 =0002 |
| Sw R11 R10 01 | 7e01 | MEM[$R11+0001]=$R10;即将编号10的寄存器的值[0002]放到RAM存储器中地址为0001($R11=0000)的单元中 |
| Add R00 R01 R10 | 0182 | $R10 = $R00 $ $R01;即$R10 = fff6 + 0003 =fff9 |
| Sw R11 R10 02 | 7e02 | MEM[$R11+0002]=$R10;即将编号10的寄存器的值[fff9]放到RAM存储器中地址为0002($R11=0000)的单元中 |
| Sub R00 R01 R10 | 0183 | $R10 = $R00 $ $R01;即$R10 = fff6 - 0003 =fff3 |
| Sw R11 R10 03 | 7e03 | MEM[$R11+0003]=$R10;即将编号10的寄存器的值[fff3]放到RAM存储器中地址为0003($R11=0000)的单元中 |
| Sllv R00 R01 R10 | 0184 | $R10 = $R00 << $R01;即$R10 = fff6 << =ffb0 |
| Sw R11 R10 04 | 7e04 | MEM[$R11+0004]=$R10;即将编号10的寄存器的值[ffb0]放到RAM存储器中地址为0004($R11=0000)的单元中 |
| Srlv R00 R01 R10 | 0185 | $R10 = $R00 >> $R01;即$R10 = fff6 >> =1ffe |
| Sw R11 R10 05 | 7e05 | MEM[$R11+0005]=$R10;即将编号10的寄存器的值[1ffe]放到RAM存储器中地址为0005($R11=0000)的单元中 |
| Srav R00 R01 R10 | 0186 | $R10 = $R00 >> $R01;即$R10 = fff6 >>(算术) =fffe |
| Sw R11 R10 06 | 7e06 | MEM[$R11+0006]=$R10;即将编号10的寄存器的值[fffe]放到RAM存储器中地址为0006($R11=0000)的单元中 |
| Slt R00 R01 R10 | 0187 | $R10=($R00<$R01)?1:0;即$R10 = (-10<3)?1:0 =0001 |
| Sw R11 R10 07 | 7e07 | MEM[$R11+0007]=$R10;即将编号10的寄存器的值[0001]放到RAM存储器中地址为0007($R11=0000)的单元中 |
| Addi R01 R01 05 | 5505 | $R01 = $R01+0005;即$R01 =3+5 =8=0008 |
| DISP R11 R00 00 | 1c00 | DISP[0]=$R11; 即将编号11的寄存器的值取出并在0号显示器显示 |
| Lw R11 R00 00 | 6c00 | $R00=MEM[$R11+imm];即RAM存储器中地址为$R11 (imm=0000)单元内容取出放到寄存器R00中 |
| DISP R00 R00 00 | 1001 | DISP[1]=$R00; 即将编号11的寄存器的值取出并在1号显示器显示 |
| Addi R11 R11 01 | 5f01 | $R11 = $R11+0001;即值加1 |
| Beq R11 R01 02 | 8d02 | If($R11==$R01[8]) then pc +2 else pc +1;即跳出循环条件 |
| Jump addr[018] | b018 | pc = pc[高4位]拼接018[低12位];即下一次命令为到程序存储器ROM地址为0018单元的内容[即从上数第五个开始] |
| Bne R11 R01 fa | 9dfa | If($R11!=$R01[8]) then pc -6 else pc +1;确定跳出循环条件是否正确 |
| halt | c000 | 停机指令，程序不在运行，即停在此命令上 |

7.1.3运行结果和分析

如下表5所示，它的存储内容为上述测试程序的运行结果，经测试检验，运算，存放，取出，显示结果均正确正确，即20种指令设计正确。

表5 RAM 结果值

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 存储单元 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 |
| 存储内容 | fff7 | 0002 | fff9 | fff3 | ffb0 | 1ffe | fffe | 0001 |

注意：逻辑右移补0，而算术右移：正(最高位0)补0，负(最高位1)补1；即fff6[-10]>>=1ffe[8090], fff6[-10]>>(算术)=fffe[-2].

7.2错误或异常现象分析6

|  |  |
| --- | --- |
| 错误或异常现象 | 分析与解决 |
| 时钟明显震荡 | 一般出现在一个半周器内，对存储数据同时读写；解决:添加一个寄存器作为缓冲，将读写分成一个周期内的两个阶段。 |
| 相对跳转指令向前(offset为负)跳转，地址不对 | 计算机中数据的存储运算都是以补码进行的；解决：将offset改为补码形式表示。 |
| 一次显示命令，显示器更新了两次 | 显示器内的缓冲寄存器使用低电平更新，而显示器的更新使能端在下一个周器开始时(低电平)冲突，即下一个周器刚开始的极短时间内使能端依旧为1(为0忽略更新)，使得显示器数据再次更新了。解决：使用上升沿更新的方式，避开冲突。 |

**八、总结体会**

**8.1总结**

本次课设主要完成了单周期 CPU设计，主要完成了以下几点工作：

1） 进一步了解了 R型指令、 I 型指令和 J 型指令这三种指令的格式和工作

过程。

2） 大概掌握了寄存器堆的设计方法和工作原理。

3） 对数据通路的了解更加深刻。

**8.2 体会**

这次的课程设计，加强了我们动手、思考和解决问题的能力。巩固和加深了对计算机在各个器件中指令执行的理解，提高综合运用所学知识解决问题的能力。培养了我们选用参考书，查阅手册及文献资料的能力。培养独立思考，深入研究，分析问题、解决问题的能力。此外，培养了我严肃认真的工作[作风](http://www.lwlwlw.com/xycl/zfclz/)，逐步建立正确的生产观念、[经济](http://www.lwlwlw.com/xycl/jycla/)观念和全局观念。而且做课程设计同时也是对课本知识的巩固和加强，平时看课本时，有些问题就不是很能理解，做完课程设计，那些问题就迎刃而解了。而且还可以记住很多东西。认识来源于实践，实践是认识的动力和最终目的，实践是检验真理的唯一标准。所以这个期末测试之后的课程设计对我们的作用是非常大的。

这次的课程设计使我懂得了理论与实际相结合是很非常重要的，只有理论知识是远远不够的，只有把所学的理论知识与实践相结合起来，从理论中得出结论，从而提高自己的实际动手能力和独立思考的能力。同时在设计的过程中发现了自己的不足之处，对以前所学过的知识理解得不够深刻，掌握得不够牢固。