**Project3 Logisim完成单周期CPU开发**

**一、总体设计**

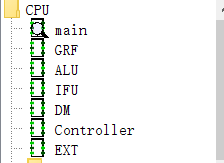


图1 顶层模块

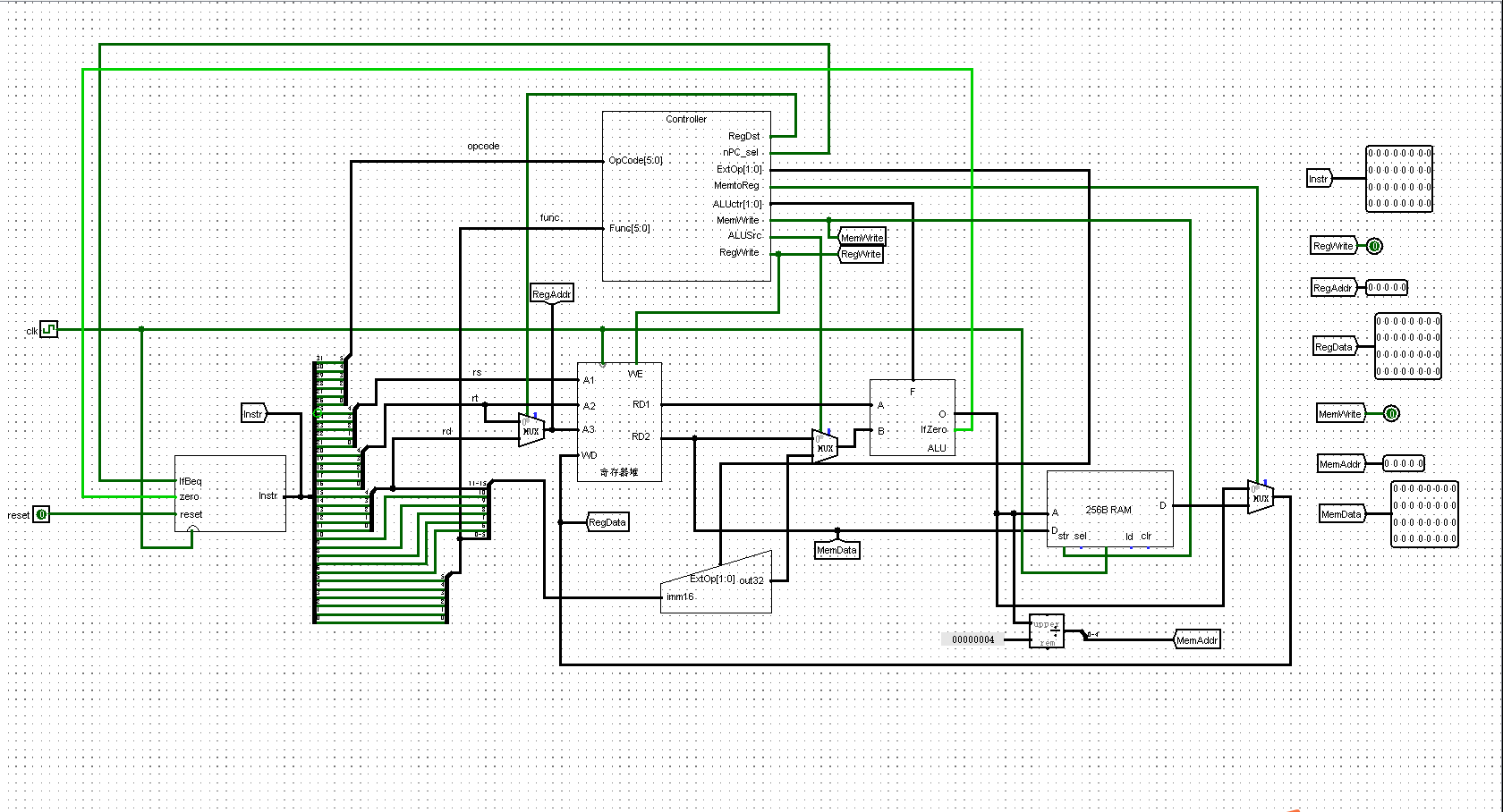


图2 顶层电路图

**二、模块定义**

1. **IFU**

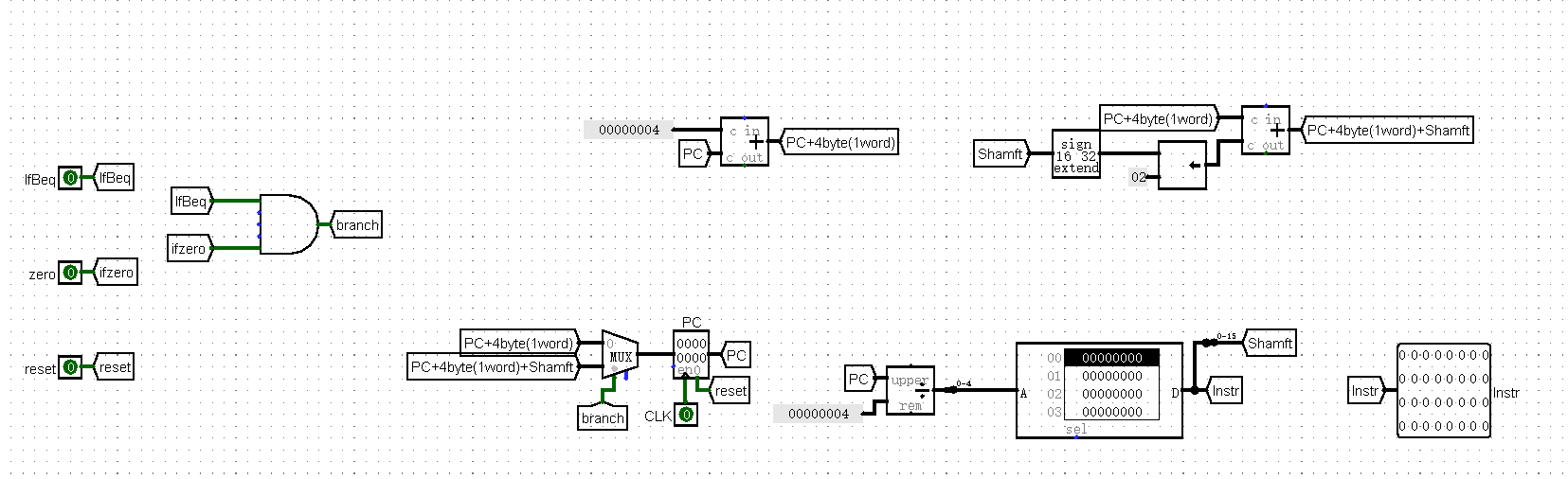


图3 IFU电路图

**（1）基本描述**

IFU主要作用是完成取指令功能。IFU内部包含PC、IM以及其他相关逻辑操作。IFU除了能执行顺序取指令外，还能根据BEQ指令的执行情况决定PC接下来的操作是顺序取指令还是转移取指令。

1. **模块接口**

表1 IFU模块接口

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| IfBeq | I | 判断当前指令是否为beq指令的标志  1：当前指令为beq指令  0：当前指令非beq指令 |
| Zero | I | 判断ALU计算结果是否为0的标志  1：计算结果为0  0：计算结果非0 |
| clk | I | 时钟信号 |
| reset | I | 判断PC是否复位的信号  1：复位  0：无效 |
| Instr | O | 32位MIPS指令 |

1. **功能定义**

表2 IFU功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| 1 | 复位 | 当复位信号有效时PC设置成0x0000\_0000 |
| 2 | 取指令 | 根据PC指定的地址从IM中取出指令 |
| 3 | 计算下一条指令地址 | \*PC取地址为4字节即一个字，所以PC的低2位地址可以省略  如果当前Zero为1且IfBeq为1，则PC←PC+sign\_ext  否则，PC←PC+1 |

1. **GRF**

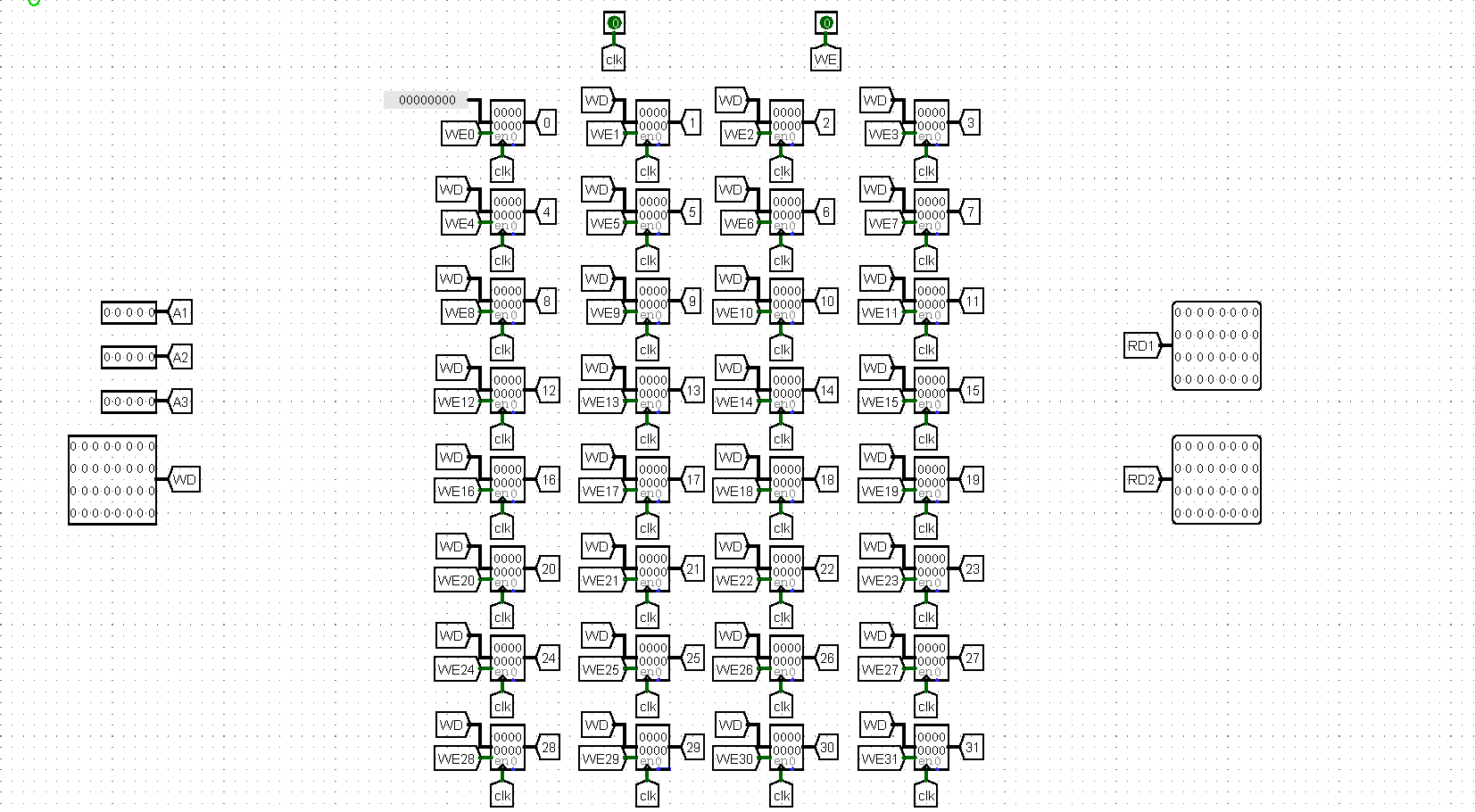


图4.1 GRF电路图

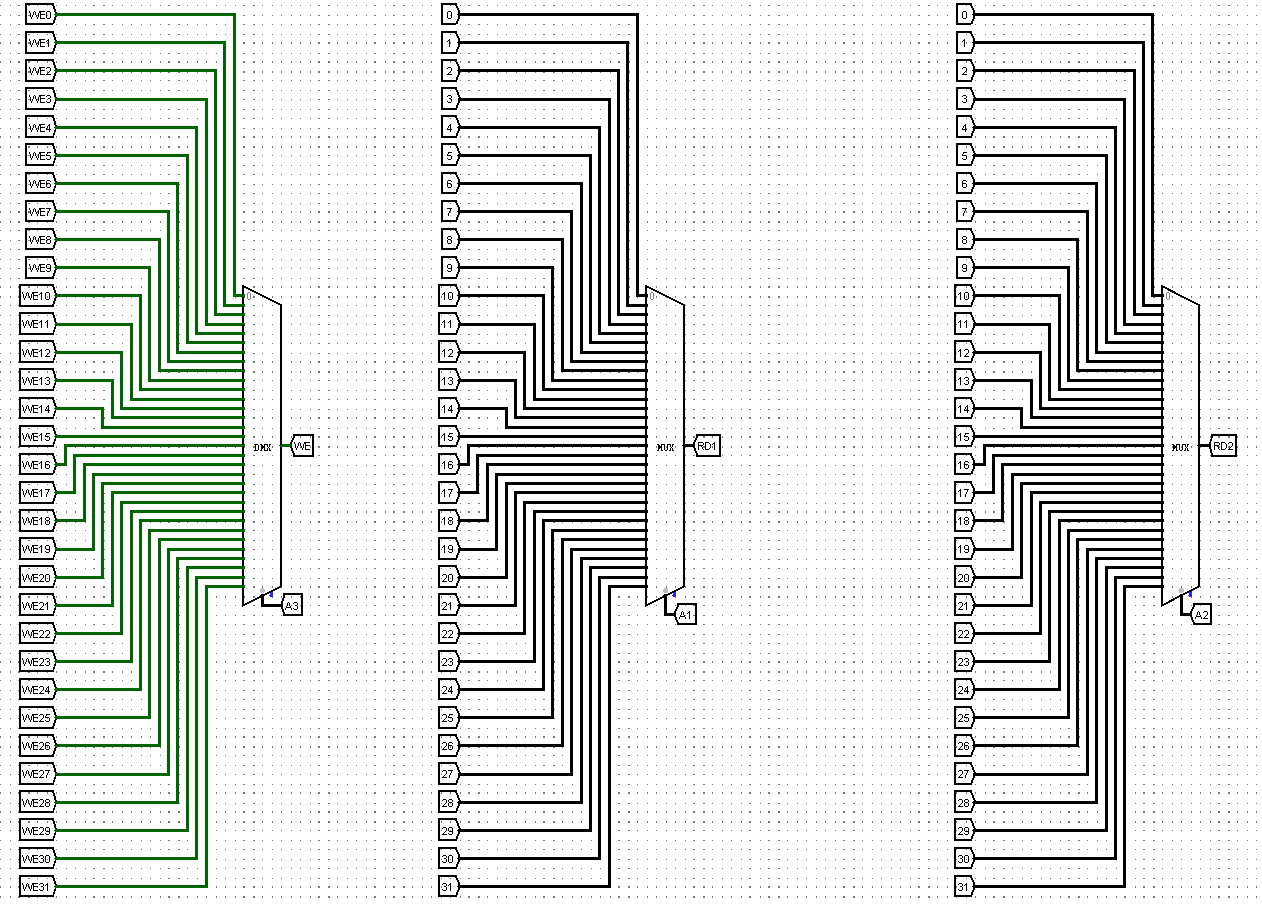


图4.2 GRF电路图

1. **基本描述**

GRF的作用主要是提供相应的寄存器以供指令完成相应的操作。GRF内部包含32个32位寄存器，分别对应0-31号寄存器，其中0号寄存器的读取结果均为0，还包含选择读写寄存器的逻辑操作电路。

1. **模块接口**

表3 GRF模块接口

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| clk | I | 时钟信号 |
| WE | I | 写使能信号  1：可向GRF中写入数据  0：不可向GRF中写入数据 |
| A1 | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD1 |
| A2 | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD2 |
| A3 | I | 5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器 |
| WD | I | 32位数据输入信号 |
| RD1 | O | 输出A1指定的寄存器中的32位数据 |
| RD2 | O | 输出A2指定的寄存器中的32位数据 |

1. **功能定义**

表4 GRF功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **描述** |
| 1 | 读寄存器 | 读出A1、A2地址对应寄存器中所存储的数据到RD1、RD2 |
| 2 | 写寄存器 | 当WE信号有效且时钟上升沿到来时，将WD写入A3所对应的寄存器中 |

1. **ALU**

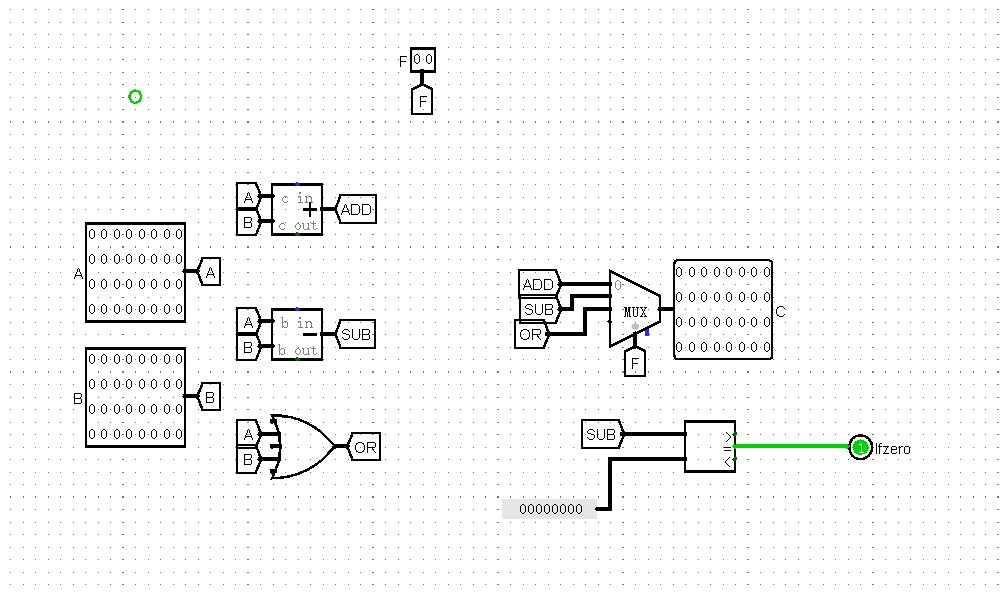


图5 ALU电路图

1. **基本描述**

ALU的主要作用是完成算术逻辑指令或者其他指令所需要的算术逻辑操作。ALU内部包含加法、减法、与或非等基本的算数逻辑部件。

1. **模块接口**

表5 ALU模块接口

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| A | I | 32位输入信号，参与ALU计算的第一个值 |
| B | I | 32位输入信号，参与ALU计算的第二个值 |
| F | I | 2位输入信号，ALU的功能选择信号：  00：ALU进行加法运算  01：ALU进行减法运算  10：ALU进行或运算 |
| C | O | 32位输出信号，ALU的计算结果 |
| Ifzero | O | 通过减运算输出的值和常数0作比较判断输入两值是否相等的信号 |

1. **功能定义**

表6 ALU功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **描述** |
| 1 | 加运算 | C = A + B |
| 2 | 减运算 | C = A - B |
| 3 | 或运算 | C = A | B |

1. **EXT**

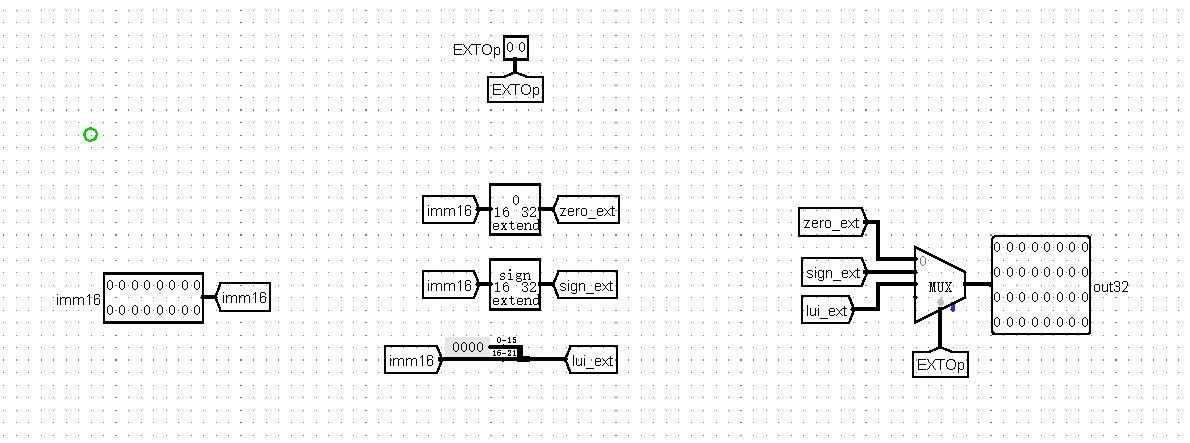


图6 EXT电路图

1. **基本描述**

EXT的主要作用是完成对输入到其中的16位数据的符号扩展、零扩展以及将输入的16位数据加载到高位等操作。其内部包含与完成符号扩展、零扩展以及讲输入的16位数据加载到高位等操作的相关逻辑部件。

1. **模块接口**

表7 EXT模块接口

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| imm16 | I | 输入EXT内部需要被扩展的16位数据 |
| EXTOp | I | 输入数据进行扩展的方式的选择信号：  00：将imm16进行零扩展到32位  01：将imm16进行符号扩展到32位  10：将imm16加载到高位，低位补0 |
| out32 | O | imm16进行扩展后的数据输出 |

1. **功能定义**

表8 EXT功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **描述** |
| 1 | 零扩展 | 将imm16进行高位补0扩展到32位 |
| 2 | 符号扩展 | 将imm16进行符号扩展到32位 |
| 3 | 加载到高位 | 将imm16加载到高位，低位补0 |

1. **DM**

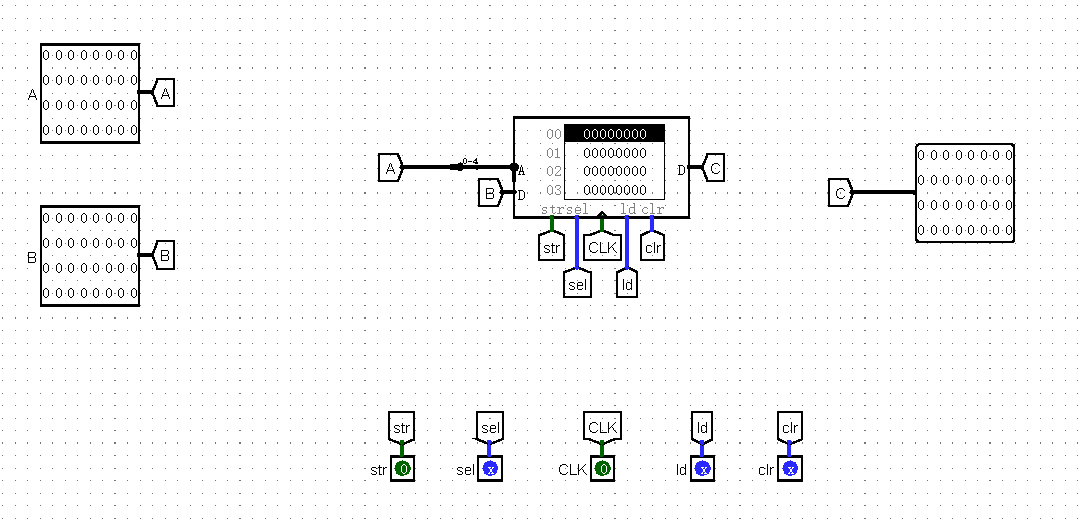


图7 DM电路图

1. **基本描述**

DM的主要作用是作为CPU执行程序时的临时数据存储媒介。内部包含存储芯片及相关逻辑通路。

1. **模块接口**

表9 DM模块接口

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| A | I | 32位输入信号，操作存储器的地址 |
| B | I | 32位输入信号，为写入数据的输入 |
| CLK | I | 时钟信号 |
| str | I | 读写控制信号  1：写信号  0：读信号 |
| C | O | 32位输出信号，输出存储器操作地址输入对应的数据 |

1. **功能定义**

表10 DM功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **描述** |
| 1 | 读 | 根据输入的存储器地址读出数据 |
| 2 | 写 | 根据输入的存储器地址，写入输入的数据 |

**三、Controller设计**

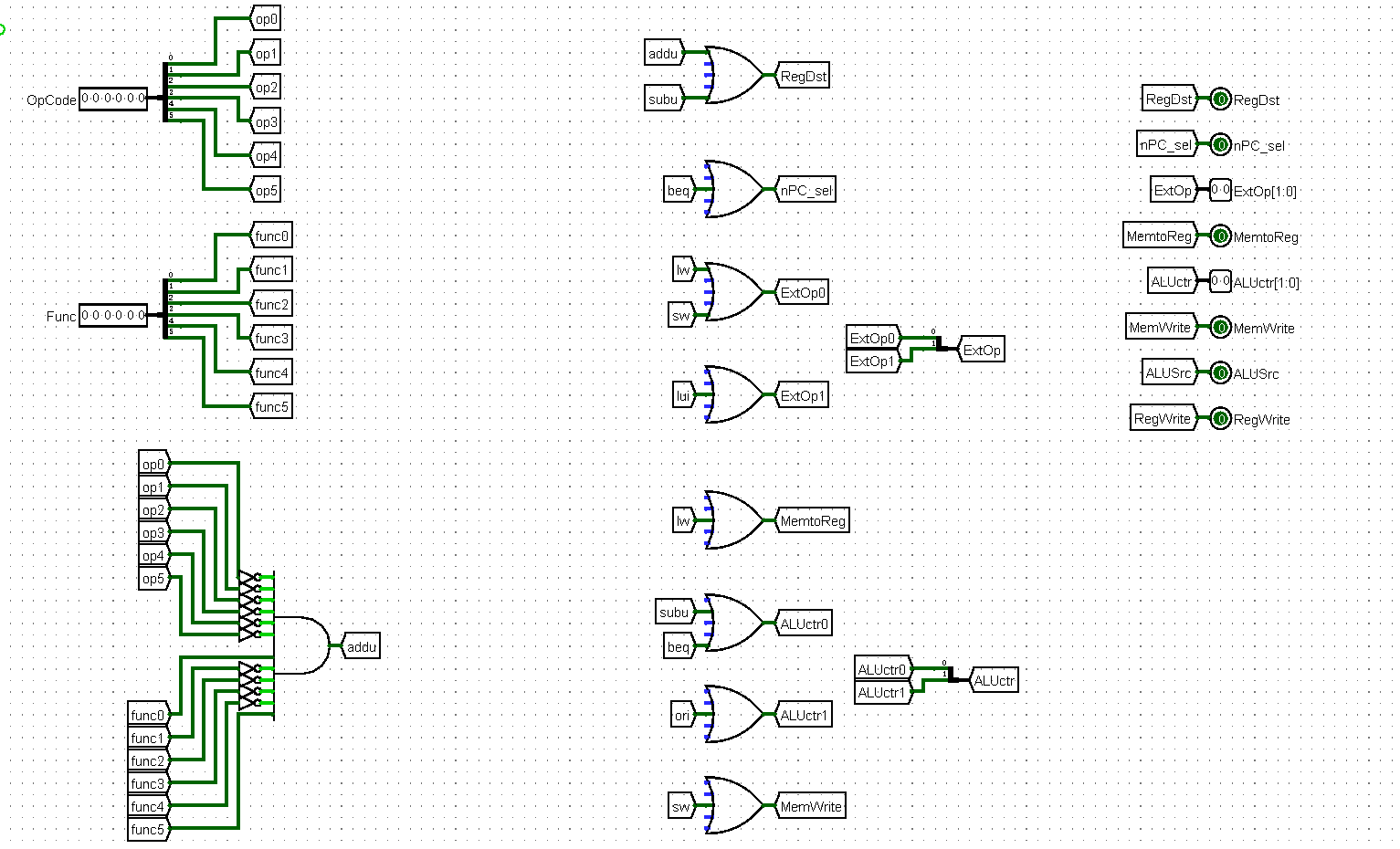


图8.1 Controller电路图

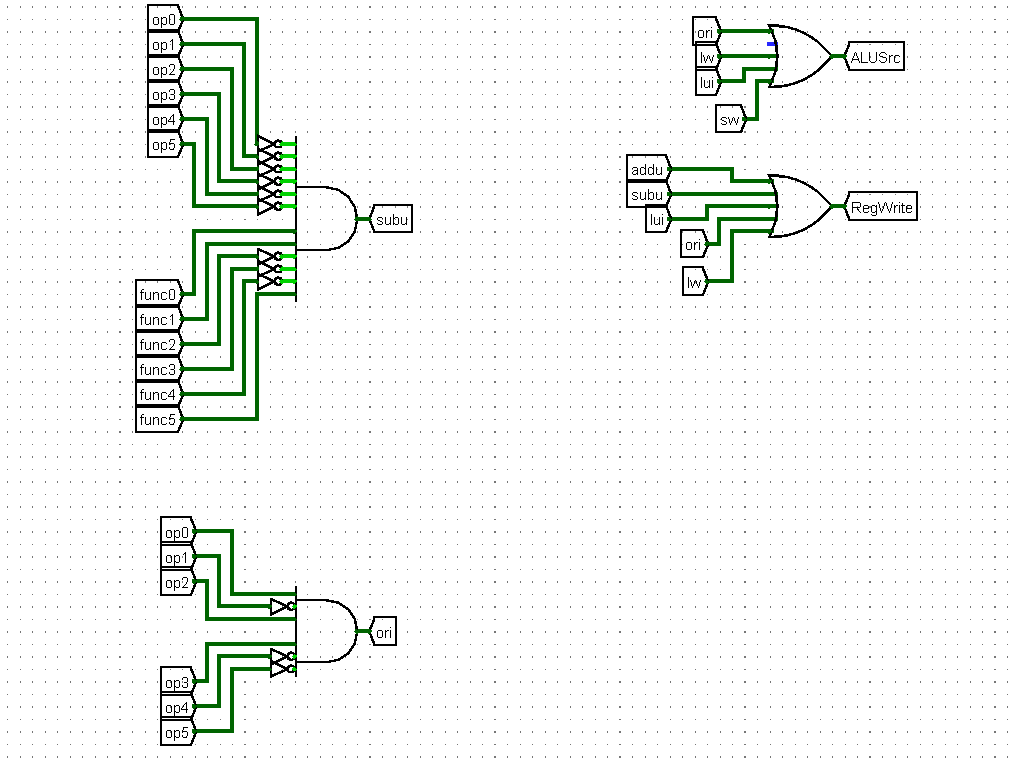


图8.2 Controller电路图

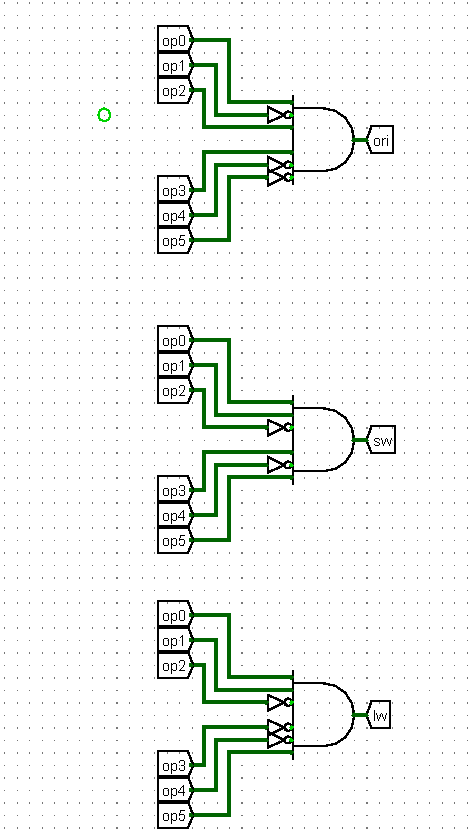


图8.3 Controller电路图

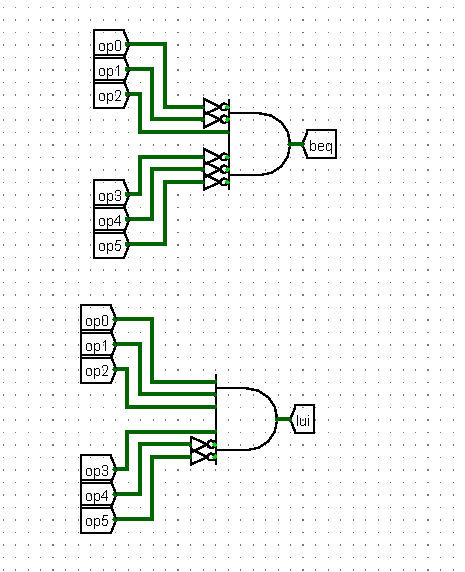


图8.4 Controller电路图

1. **基本描述**

控制器的主要作用是译码，即将每一条机器指令中包含的信息，转化为给CPU各部分的控制信号。其内部主要包含与或门阵列，与逻辑部分的功能是识别，将输入的机器码识别为相应的指令。或逻辑部分的功能是生成，根据输入的指令不同，产生不同的控制信号。

1. **模块接口**

表11 Controller模块接口

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| OpCode | I | 机器指令的操作码部分 |
| Func | I | 机器指令的函数码部分 |
| RegDst | O | GRF写地址控制 |
| nPC\_sel | O | BEQ指令标志 |
| ExtOp[1:0] | O | EXT扩展方式控制标志 |
| MemtoReg | O | DM输出数据控制标志 |
| ALUctr[1:0] | O | ALU运算控制标志 |
| MemWrite | O | DM写入数据控制标志 |
| ALUSrc | O | ALU操作数控制标志 |
| RegWrite | O | GRF写入数据控制标志 |

1. **单周期真值表**

表12 Controller单周期真值表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **addu** | **subu** | **ori** | **lw** | **sw** | **beq** | **lui** |
| Func | 100001 | 100011 | N/A | | | | |
| Op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
| RegDst | 1 | 1 | 0 | 0 | X | X | 0 |
| nPC\_sel | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| MemtoReg | 0 | 0 | 0 | 1 | X | X | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| ExtOp[1:0] | X | X | 0 | 1 | 1 | X | 2 |
| ALUctr[1:0] | 0 | 1 | 2 | 0 | 0 | 1 | X |

1. **测试CPU**

**lui $s1,0x0008 #lui 测试程序要实现：立即数0x0008 加载到s1**

**#（17）寄存器高位**

**ori $s0,$zero,0x00000004 #ori 测试程序要实现：zero（0）寄存器中的数据与 #立即数0x00000004进行或运算，结果存储在s0**

**#(16)寄存器中**

**ori $t0,$zero,0x00003000 #ori 测试程序要实现：zero（0）寄存器中的数据与 #立即数0x00003000进行或运算，结果存储在t0(8) #寄存器中**

**sw $t0,0($s0) #sw 测试程序要实现：把t0(8)寄存器中的数据**

**#（1word），存储到s0(16)的值再加上偏移量0所 #指向的RAM中**

**sw $t0,4($s0) #sw 测试程序要实现：把t0(8)寄存器中的数据 #（1word），存储到s0(16)的值再加上偏移量4所 #指向的RAM中**

**loop:**

**lw $s2,0($s0) #lw 测试程序要实现：把s0(16)寄存器中的值再加上 #偏移量0所指向的RAM中的数据取出来并存到 #s2(18)寄存器中**

**addu $s2,$s2,$s1 #add 测试程序要实现：s1(17)寄存器中的值加上 #s2(18)寄存器中的值后将结果存到s2(18)寄存器中**

**nop #nop 测试程序不用实现任何指令，默认PC <- PC+4**

**sw $s2,0($s0) #sw 测试程序要实现：把s2(18)寄存器中的数据**

**#（1word），存储到s0(16)的值再加上偏移量0所指 #向的RAM中**

**beq $t0,$s2,loop #beq 测试程序要实现：判断t0(8)的值和s2(18)的值 #是否相等，相等跳转loop**

**lw $s2,4($s0) #lw 测试程序要实现：把s0(16)寄存器中的值再加 #上偏移量4所指向的RAM中的数据取出来并存到 #s2(18)寄存器中**

**subu $s2,$s1,$s2 #subu 测试程序要实现：s1(17)的值减去s2(18) #的值后将结果存到s2（18）中**

**sw $s2,4($s0) #sw 测试程序要实现：把s2(18)寄存器中的数据**

**#（1word），存储到s0(16)的值再加上偏移量4所**

**#指向的RAM中**

**机器码：**

**3c110008 34100004 34083000 ae080000 ae080004 8e120000**

**02519021 00000000 ae120000 1112fffb 8e120004 02329023**

**ae120004**

**MARS模拟结果：**

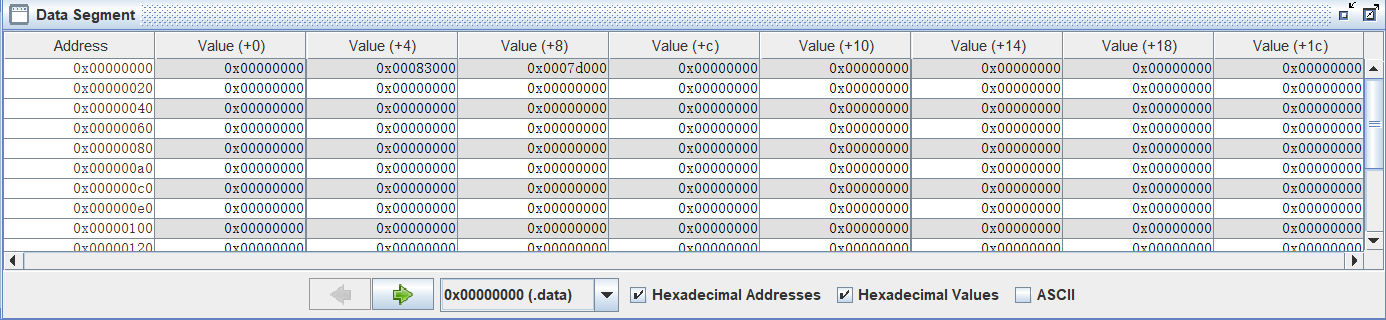


图9 测试程序MARS模拟结果

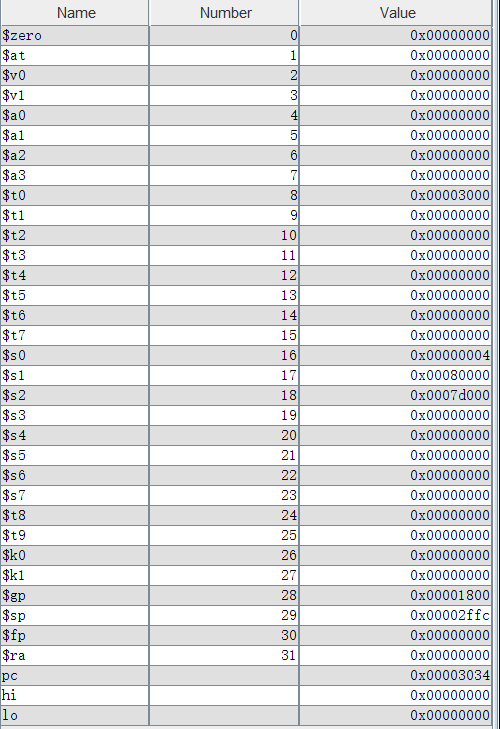


图10 测试程序MARS模拟结果

**Logisim:**

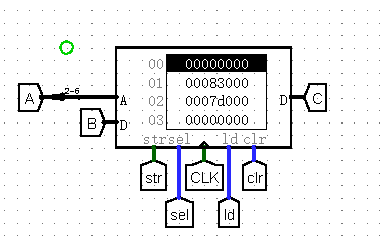
**DM：**

图11 测试程序Logisim模拟结果之DM

**GRF：**

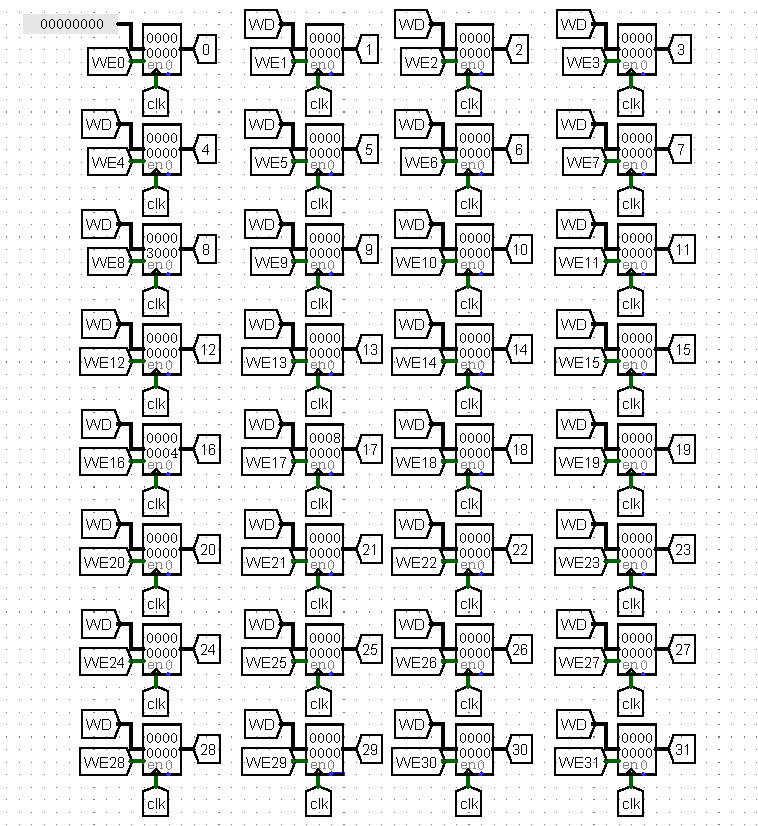


图12 测试程序Logisim模拟结果之GRF

1. **思考题**

**1、若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。**

在计算下一条指令地址时，32位PC可以直接加4，而30位PC需要左移两 位才能加4，即通常情况下30位PC可以省略低两位而直接加1。

**2、现在我们的模块中 IM使用ROM， DM使用RAM， GRF使用寄存器， 这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。**

合理。ROM是只读存储器，作为指令存储器来存储指令可以避免电路运行 时对程 序进行修改，更好的保持程序的稳定性；RAM可读可写，且可以满足DM 用来向 内存中存储数据，需要足够的存储空间而不需要特别快的访存速度的要 求；GRF 用寄存器来实现的话可以满足其需要非常快的存储速度的需求。

1. **结合上文给出的样例真值表，给出RegDst， ALUSrc， MemtoReg，RegWrite, nPC\_Sel, ExtOp与op和func有关的布尔表达式（表达式中只能使用“与、或、 非”3 种基本逻辑运算。）**

****

1. **充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式， 请给 出化简后的形式。**

将X按照方便化简的原则当做0或1，则得最简式如下：

****

5、**事实上，实现nop空指令，我们并不需要将它加入控制信号真值表，为什么？ 请给出你的理由。**

nop指令本身的意义就是对电路中的元件不进行任何操作，且其机器码为 0x00000000，所以不管从实际意义和机器码来看，其存在与否都对电路的工作没 有影响。

1. **前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。**

前文提到的可能需要手工修改指令码中的数据偏移，是因为MARS中指令存储器的起始地址和数据存储器的起始地址是不重叠的，如果MARS中设置指令存储器的起始地址为0，则MARS中的数据存储器的起始地址就有了偏移量如0x00003000,但是此次设置的DM中的RAM很小，地址位宽只有5位，即在此次试验中，我们只截取了地址输入信号的[6:2]位进行工作，所以当偏移量比实验用的RAM容量要大时，不会产生取错数的影响。

但若DM容量很大，需要多片RAM来实现DM时，则需要添加片选信号来解决手工修改指令码中的数据偏移的问题。例如，若设置MARS数据段地址范围为0x30000000-0x3ffffff，即需要256MB×32容量的DM，其存储器地址线为28位，而Logisim中，RAM的地址位最多为24位，即每片RAM的容量最大为16MB×32，所以需要16片RAM去实现满足此例的DM，那么只需要将输入DM的地址信号的[31:28]位与0x3比较，若相等则表明需存取的数据存储器范围为0x30000000-0x3fffffff之间，再取输入DM的地址信号的[27:24]位作为这个区域16片的片选信号即可。

**7、除了编写程序进行测试外，还有一种验证CPU设计正确性的办法——形式 验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法 证明其正确性或非正确性。请搜索“形式验证（Formal Verification)"了解相关 内容后，简要阐述相比与测试，形式验证的优劣。**

形式验证的优点：

1. 测试者不必考虑如何获得测试向量，因为形式验证是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较的。
2. 有效地克服了测试验证的覆盖面不全的缺点，因为形式验证是对指定描述的所有可能的情况进行验证，覆盖率100%，而不仅是对其中的部分进行多次试验。
3. 形式验证可以进行的验证层次广，可以进行从系统级到门级的验证，而且验证时间短，有利于尽早、尽快地发现并改正电路设计中存在的错误，利于缩短设计时间和周期。

形式验证的缺点：

形式验证到目前为止仍不能有效地验证电路的时延和功耗等性能。